# Ill-Typed Programs Don't Evaluate

Charlie Walpole

University of Bristol

6/7 November 2024

Programming Languages Research Group :: Bristol

# Motivation

"Well-typed programs don't go wrong." (Type Safety)

# Motivation

"Well-typed programs don't go wrong." (Type Safety)

1. Preservation
2. Progress

# Motivation

"Well-typed programs don't go wrong." (Type Safety)

1. Preservation: "Evaluation preserves typing."
2. Progress: "Typed terms don't get stuck."

# Motivation

"Well-typed programs don't go wrong." (Type Safety)

1. Preservation: "Evaluation preserves typing."
2. Progress: "Typed terms don't get stuck."

$$\exists \tau.\ \vdash e : \tau \implies \exists \nu.\ e \Downarrow \nu$$

# Motivation

$$\exists \tau.\ \vdash e : \tau \implies \exists \nu.\ e \Downarrow \nu$$

# Motivation

$$\exists \tau.\ \vdash e : \tau \implies \exists \nu.\ e \Downarrow \nu$$

$$(\forall \alpha) \quad \nvdash \text{if } (\lambda x.\, x) \text{ True then } 1 \text{ else True} : \alpha$$
$$(\forall \alpha) \quad \nvdash \text{if } (\lambda x.\, x)\ 1 \text{ then } 1 \text{ else } 1 : \alpha$$

# Language

| | | | | |
|---|---|---|---|---|
| pre-terms | $e$ | $::=$ | $x$ | variables |
| | | | $e_1 \ e_2$ | function application |
| | | | $\lambda x.\, e$ | function abstraction |
| | | | $c(e_1, \cdots, e_n)$ | constructor application |
| | | | match $e_0$ with $\{|_{i=0}^{n} p_i \mapsto e_i\}$ | pattern matching |
| types | $\tau$ | $::=$ | $\alpha$ | type variables |
| | | | $\tau_1 \rightarrow \tau_2$ | function arrow |
| | | | $\tau_1 + \tau_2$ | sum types |
| | | | $c(\tau_1, \cdots, \tau_n)$ | constructor types |
| | | | Ok | top-type |

# Language

| | | | | |
|---|---|---|---|---|
| pre-terms | $e$ | $::=$ | $x$ | variables |
| | | | $e_1 \; e_2$ | function application |
| | | | $\lambda x.\, e$ | function abstraction |
| | | | $c(e_1, \cdots, e_n)$ | constructor application |
| | | | $\mathsf{match}\ e_0\ \mathsf{with}\ \{\vert_{i=0}^{n} p_i \mapsto e_i\}$ | pattern matching |
| types | $\tau$ | $::=$ | $\alpha$ | type variables |
| | | | $\tau_1 \to \tau_2$ | function arrow |
| | | | $\tau_1 + \tau_2$ | sum types |
| | | | $c(\tau_1, \cdots, \tau_n)$ | constructor types |
| | | | $\mathsf{Ok}$ | top-type |

Language:

$$e := \lambda x.\, e \mid e_1 \; e_2 \mid x \mid c(e_1, \cdots, e_n) \mid \text{match } e_0 \text{ with } \{|_{i=1}^{n} p_i \mapsto e_i\}$$

Types:

$$\tau := \alpha \mid \tau_1 \to \tau_2 \mid \tau_1 + \tau_2 \mid c(\tau_1, \cdots, \tau_n) \mid \text{Ok}$$

# Constructors - Terms & Types

$$\text{zero} \quad :: \quad \text{zero}$$

# Constructors - Terms & Types

$$
\begin{aligned}
\mathsf{zero} &\;::\; \mathsf{zero} \\
\mathsf{succ}(\mathsf{zero}) &\;::\; \mathsf{succ}(\mathsf{zero})
\end{aligned}
$$

# Constructors - Terms & Types

$$\begin{aligned}
\mathsf{zero} &\;::\; \mathsf{zero} \\
\mathsf{succ}(\mathsf{zero}) &\;::\; \mathsf{succ}(\mathsf{zero}) \\
\mathsf{succ}(\mathsf{zero}) &\;::\; \mathsf{succ}(\mathsf{Nat}) \\
\mathsf{succ}(\mathsf{zero}) &\;::\; \mathsf{Nat}
\end{aligned}$$

# Constructors - Terms & Types

$$\vdash \mathsf{zero} : \mathsf{zero}$$
$$\vdash \mathsf{succ}(\mathsf{zero}) : \mathsf{succ}(\mathsf{zero})$$
$$\vdash \mathsf{succ}(\mathsf{zero}) : \mathsf{succ}(\mathsf{Nat})$$
$$\vdash \mathsf{succ}(\mathsf{zero}) : \mathsf{Nat}$$

# Match Expressions

$$\text{match } e_0 \text{ with } \{|_{i=1}^{n} p_i \mapsto e_i\}$$

# Match Expressions

$$\mathsf{match}\ e_0\ \mathsf{with}\ \{|_{i=1}^{n} p_i \mapsto e_i\}$$

$$\begin{aligned}\mathsf{match}\ \mathsf{succ}(\mathsf{zero})\ \mathsf{with}\ \{&|\ \mathsf{zero} \mapsto \mathsf{zero}\\ &|\ \mathsf{succ}(n) \mapsto n\}\end{aligned}$$

# Top - Type

$$(\forall \tau) \vdash e : \tau \implies \vdash e : \mathsf{Ok}$$

# Goal

$$\neg(\vdash \mathsf{if}\ (\lambda x.\, x)\ 1\ \mathsf{then}\ 1\ \mathsf{else}\ 1 : \mathsf{Ok})$$

# Typing Judgements

The typing judgement takes the form of:

$$x_1 : A_1, \cdots, x_n : A_n \vdash e : B$$

# Typing Judgements

The typing judgement takes the form of:

$$x_1 : A_1, \cdots, x_n : A_n \vdash e : B$$

We can intuitively read this as:

$$x_1 \in A_1 \wedge \cdots \wedge x_n \in A_n \Rightarrow e \in B$$

# Typing Judgements

The typing judgement takes the form of:

$$x_1 : A_1, \cdots, x_n : A_n \vdash e : B$$

We can intuitively read this as:

$$x_1 \in A_1 \land \cdots \land x_n \in A_n \Rightarrow e \in B$$

$$\neg(x_1 : A_1 \vdash e : B) \implies ???$$

$$\neg(x_1 \in A_1 \Rightarrow e \in B) \implies x_1 \in A_1 \land \neg(e \in B)$$

# Denotational Semantics

We can define an operation, $[\![\cdot]\!]$, for both terms and types.
Intuitively, $e \Downarrow \nu \implies \nu \in [\![e]\!]$.

# Denotational Semantics

We can define an operation, $[\![\cdot]\!]$, for both terms and types.
Intuitively, $e \Downarrow \nu \implies \nu \in [\![e]\!]$.
Term semantics:

$$[\![\text{zero}]\!] = \{0\} \subseteq \mathbb{Z}$$
$$[\![\text{zero} + \text{succ}(\text{zero})]\!] = \{1\}$$

# Denotational Semantics - (Partial) Definition

Type Semantics:

$$
\begin{aligned}
[\![\mathsf{Nat}]\!] &:= \mathbb{Z} \\
[\![\tau_1 + \tau_2]\!] &:= [\![\tau_1]\!] \cup [\![\tau_2]\!]
\end{aligned}
$$

# Denotational Semantics - (Partial) Definition

Type Semantics:

$$
\begin{aligned}
[\![\mathsf{Nat}]\!] &:= \mathbb{Z} \\
[\![\tau_1 + \tau_2]\!] &:= [\![\tau_1]\!] \cup [\![\tau_2]\!] \\
[\![\mathsf{Ok}]\!] &:= \mathcal{U}
\end{aligned}
$$

# Denotational Semantics - (Partial) Definition

Type Semantics:

$$
\begin{aligned}
[\![\mathsf{Nat}]\!] &:= \mathbb{Z} \\
[\![\tau_1 + \tau_2]\!] &:= [\![\tau_1]\!] \cup [\![\tau_2]\!] \\
[\![\mathsf{Ok}]\!] &:= \mathcal{U} = \bigcup_{\tau} [\![\tau]\!]
\end{aligned}
$$

# Denotational Semantics - (Partial) Definition

$$[\![\tau_1 \to \tau_2]\!] := \{f \mid f : \mathcal{U} \to \mathcal{U},\ x \in [\![\tau_1]\!] \implies f(x) \in [\![\tau_2]\!]\}$$

# Denotational Semantics - (Partial) Definition

$$
\begin{aligned}
[\![\tau_1 \to \tau_2]\!] &:= \{f \mid f : \mathcal{U} \to \mathcal{U},\ x \in [\![\tau_1]\!] \implies f(x) \in [\![\tau_2]\!]\} \\
&\neq [\![\tau_1]\!] \to [\![\tau_2]\!]
\end{aligned}
$$

# Logical Typing Judgements

Now, we can write 'the meaning' of a typing judgment:

$$x_1 : A_1, \cdots, x_n : A_n \vdash e : B$$

As a logical statement:

$$[\![x_1]\!] \subseteq [\![A_1]\!] \wedge \cdots \wedge [\![x_n]\!] \subseteq [\![A_n]\!] \implies [\![e]\!] \subseteq [\![B]\!]$$

# Negation

Our goal:

$$\neg(\vdash \mathsf{if}\ (\lambda x.\, x)\ 1\ \mathsf{then}\ 1\ \mathsf{else}\ 1 : \mathsf{Ok})$$

# Negation

Our goal:

$$\neg(\vdash \text{if } (\lambda x.\, x)\ 1 \text{ then } 1 \text{ else } 1 : \mathsf{Ok})$$

Negation fact:

$$(\neg A) \iff (A \implies \bot)$$

# Negation

Our goal:

$$\neg(\vdash \text{if } (\lambda x.\, x)\ 1 \text{ then } 1 \text{ else } 1 : \mathsf{Ok})$$

Negation fact:

$$(\neg A) \iff (A \implies \bot)$$

Thus:

$$\begin{aligned}\neg(\vdash \text{if } (\lambda x.\, x)\ 1 \text{ then } 1 \text{ else } 1 : \mathsf{Ok}) &\iff \neg(\top \implies [\![\text{if } (\lambda x.\, x)\ 1 \text{ then } 1 \text{ else } 1]\!] \subseteq [\![\mathsf{Ok}]\!]) \\ &\iff (([\![\text{if } 1 \text{ then } (\lambda x.\, x)\ 1 \text{ else } 1]\!] \subseteq [\![\mathsf{Ok}]\!]) \implies \bot)\end{aligned}$$

# Negation

Our goal:

$$\neg(\vdash \text{if } (\lambda x.\, x)\ 1 \text{ then } 1 \text{ else } 1 : \mathsf{Ok})$$

Negation fact:

$$(\neg A) \iff (A \implies \bot)$$

Thus:

$$\begin{aligned}
\neg(\vdash \text{if } (\lambda x.\, x)\ 1 \text{ then } 1 \text{ else } 1 : \mathsf{Ok}) &\iff \neg(\top \implies [\![\text{if } (\lambda x.\, x)\ 1 \text{ then } 1 \text{ else } 1]\!] \subseteq [\![\mathsf{Ok}]\!]) \\
&\iff (([\![\text{if } 1 \text{ then } (\lambda x.\, x)\ 1 \text{ else } 1]\!] \subseteq [\![\mathsf{Ok}]\!]) \implies \bot) \\
&\iff (\text{if } (\lambda x.\, x)\ 1 \text{ then } 1 \text{ else } 1 : \mathsf{Ok} \vdash \bot)
\end{aligned}$$

# Examaining Typing Judgements

Take a general typing judgement:

$$x_1 : A_1, \cdots, x_n : A_n \vdash e : B$$

Notable restrictions:

# Examaining Typing Judgements

Take a general typing judgement:

$$x_1 : A_1, \cdots, x_n : A_n \vdash e : B$$

Notable restrictions:

1. Only variable typings may appear in the context.

# Examaining Typing Judgements

Take a general typing judgement:

$$x_1 : A_1, \cdots, x_n : A_n \vdash e : B$$

Notable restrictions:

1. Only variable typings may appear in the context.
2. There must be exactly 1 typing as the conclusion.

# Examaining Typing Judgements

Take a general typing judgement:

$$x_1 : A_1, \cdots, x_n : A_n \vdash e : B$$

Notable restrictions:

1. Only variable typings may appear in the context.
2. There must be exactly 1 typing as the conclusion.

Lets relax these restrictions.

# Two Sides of Typing

Typing:

$$e_1 : A_1, \cdots, e_n : A_n \vdash e'_1 : B_1, \cdots, e'_m : B_m$$

# Two Sides of Typing

Typing:

$$e_1 : A_1, \cdots, e_n : A_n \vdash e'_1 : B_1, \cdots, e'_m : B_m$$

Logic:

$$[\![e_1]\!] \subseteq [\![A_1]\!] \wedge \cdots \wedge [\![e_n]\!] \subseteq [\![A_n]\!] \implies [\![e'_1]\!] \subseteq [\![B_1]\!] \vee \cdots \vee [\![e'_m]\!] \subseteq [\![B_m]\!]$$

# Two Sides of Typing

Typing:

$$e_1 : A_1, \cdots, e_n : A_n \vdash e'_1 : B_1, \cdots, e'_m : B_m$$

Logic:

$$[\![e_1]\!] \subseteq [\![A_1]\!] \wedge \cdots \wedge [\![e_n]\!] \subseteq [\![A_n]\!] \implies [\![e'_1]\!] \subseteq [\![B_1]\!] \vee \cdots \vee [\![e'_m]\!] \subseteq [\![B_m]\!]$$

# Sequent Calculus

$$P \wedge Q \wedge \top \iff P \wedge Q$$
$$P \wedge \top \iff P$$
$$(P_1 \wedge \cdots \wedge P_n) \wedge \top \iff P_1 \wedge \cdots \wedge P_n$$

## Sequent Calculus

$$
\begin{aligned}
P \wedge Q \wedge \top &\iff P \wedge Q \\
P \wedge \top &\iff P \\
(P_1 \wedge \cdots \wedge P_n) \wedge \top &\iff P_1 \wedge \cdots \wedge P_n
\end{aligned}
$$

$$
\begin{aligned}
P \vee Q \vee \bot &\iff P \vee Q \\
P \vee \bot &\iff P \\
(P_1 \vee \cdots \vee P_n) \vee \bot &\iff P_1 \vee \cdots \vee P_n
\end{aligned}
$$

# False

$$e_1 : A_1, \cdots, e_n : A_n \vdash$$

# False

$$e_1 : A_1, \cdots, e_n : A_n \vdash$$

$$[\![e_1]\!] \subseteq [\![A_1]\!] \wedge \cdots \wedge [\![e_n]\!] \subseteq [\![A_n]\!] \implies \bot$$

# False

$$e_1 : A_1, \cdots, e_n : A_n \vdash \bot$$

$$[\![e_1]\!] \subseteq [\![A_1]\!] \wedge \cdots \wedge [\![e_n]\!] \subseteq [\![A_n]\!] \implies \bot$$

# Progress

Goal:

$$\neg(\vdash \text{if } (\lambda x.\, x)\ 1 \text{ then } 1 \text{ else } 1 : \text{Ok})$$

Translation:

$$\text{if } (\lambda x.\, x)\ 1 \text{ then } 1 \text{ else } 1 : \text{Ok} \vdash$$

# Examples of Note

1. $\vdash \mathsf{map} : (\mathsf{Nat} \geq 3 \to \mathsf{Nat} \geq 8) \to [\mathsf{Nat} \geq 3] \to [\mathsf{Nat} \geq 8]$

# Examples of Note

1. $\vdash \mathsf{map} : (\mathsf{Nat} \geq 3 \to \mathsf{Nat} \geq 8) \to [\mathsf{Nat} \geq 3] \to [\mathsf{Nat} \geq 8]$
2. $x + 1 : \mathsf{Nat} \vdash (\lambda y.\, y)\; x : \mathsf{Nat}$

# Examples of Note

1. $\vdash \mathsf{map} : (\mathsf{Nat} \geq 3 \to \mathsf{Nat} \geq 8) \to [\mathsf{Nat} \geq 3] \to [\mathsf{Nat} \geq 8]$
2. $x + 1 : \mathsf{Nat} \vdash (\lambda y.\, y)\ x : \mathsf{Nat}$
3. $\mathsf{Nil} : \mathsf{Nat} \vdash$

# Examples of Note

1. $\vdash \mathsf{map} : (\mathsf{Nat} \geq 3 \to \mathsf{Nat} \geq 8) \to [\mathsf{Nat} \geq 3] \to [\mathsf{Nat} \geq 8]$
2. $x + 1 : \mathsf{Nat} \vdash (\lambda y.\, y)\ x : \mathsf{Nat}$
3. $\mathsf{Nil} : \mathsf{Nat} \vdash$
4. $1 + \mathsf{Nil} : \mathsf{Ok} \vdash$

# Examples of Note

1. $\vdash \mathsf{map} : (\mathsf{Nat} \geq 3 \to \mathsf{Nat} \geq 8) \to [\mathsf{Nat} \geq 3] \to [\mathsf{Nat} \geq 8]$
2. $x + 1 : \mathsf{Nat} \vdash (\lambda y.\, y)\ x : \mathsf{Nat}$
3. $\mathsf{Nil} : \mathsf{Nat} \vdash$
4. $1 + \mathsf{Nil} : \mathsf{Ok} \vdash$
5. $\mathsf{match}\ \mathsf{Nil}\ \mathsf{with}\ \{\mathsf{Cons}(y, ys) \mapsto y\} : \mathsf{Ok} \vdash$

# Examples of Note

1. $\vdash \mathsf{map} : (\mathsf{Nat} \geq 3 \to \mathsf{Nat} \geq 8) \to [\mathsf{Nat} \geq 3] \to [\mathsf{Nat} \geq 8]$
2. $x + 1 : \mathsf{Nat} \vdash (\lambda y.\, y)\ x : \mathsf{Nat}$
3. $\mathsf{Nil} : \mathsf{Nat} \vdash$
4. $1 + \mathsf{Nil} : \mathsf{Ok} \vdash$
5. $\mathsf{match}\ \mathsf{Nil}\ \mathsf{with}\ \{\mathsf{Cons}(y, ys) \mapsto y\} : \mathsf{Ok} \vdash$

How are these derived/proven?

# Deriving Typing Rules

Let's derive a typing rule for $(e_1, e_2) : A_1 \times A_2 \vdash$:

# Deriving Typing Rules

Let's derive a typing rule for $(e_1, e_2) : A_1 \times A_2 \vdash$:

$$(e_1, e_2) : A_1 \times A_2 \vdash$$

# Deriving Typing Rules

Let's derive a typing rule for $(e_1, e_2) : A_1 \times A_2 \vdash$:

$$\begin{aligned} & (e_1, e_2) : A_1 \times A_2 \vdash \\ \iff \quad & \neg([\![(e_1, e_2)]\!] \subseteq [\![A_1 \times A_2]\!]) \end{aligned}$$

# Deriving Typing Rules

Let's derive a typing rule for $(e_1, e_2) : A_1 \times A_2 \vdash$:

$$
\begin{aligned}
& (e_1, e_2) : A_1 \times A_2 \vdash \\
\iff \quad & \neg([\![(e_1, e_2)]\!] \subseteq [\![A_1 \times A_2]\!]) \\
\iff \quad & \neg(\forall p.\ p \in [\![(e_1, e_2)]\!] \implies p \in [\![A_1 \times A_2]\!])
\end{aligned}
$$

# Deriving Typing Rules

Let's derive a typing rule for $(e_1, e_2) : A_1 \times A_2 \vdash$:

$$
\begin{aligned}
& (e_1, e_2) : A_1 \times A_2 \vdash \\
\iff \quad & \neg(\llbracket (e_1, e_2) \rrbracket \subseteq \llbracket A_1 \times A_2 \rrbracket) \\
\iff \quad & \neg(\forall p.\ p \in \llbracket (e_1, e_2) \rrbracket \implies p \in \llbracket A_1 \times A_2 \rrbracket) \\
\iff \quad & \neg(\forall x, y.\ (x, y) \in \llbracket e_1 \rrbracket \times \llbracket e_2 \rrbracket \implies (x, y) \in \llbracket A_1 \rrbracket \times \llbracket A_2 \rrbracket)
\end{aligned}
$$

# Deriving Typing Rules

Let's derive a typing rule for $(e_1, e_2) : A_1 \times A_2 \vdash$:

$$
\begin{aligned}
& (e_1, e_2) : A_1 \times A_2 \vdash \\
\iff \quad & \neg(\llbracket (e_1, e_2) \rrbracket \subseteq \llbracket A_1 \times A_2 \rrbracket) \\
\iff \quad & \neg(\forall p.\ p \in \llbracket (e_1, e_2) \rrbracket \implies p \in \llbracket A_1 \times A_2 \rrbracket) \\
\iff \quad & \neg(\forall x, y.\ (x, y) \in \llbracket e_1 \rrbracket \times \llbracket e_2 \rrbracket \implies (x, y) \in \llbracket A_1 \rrbracket \times \llbracket A_2 \rrbracket) \\
\iff \quad & \neg(\forall x, y.\ x \in \llbracket e_1 \rrbracket \wedge y \in \llbracket e_2 \rrbracket \implies x \in \llbracket A_1 \rrbracket \wedge y \in \llbracket A_2 \rrbracket)
\end{aligned}
$$

# Deriving Typing Rules

Let's derive a typing rule for $(e_1, e_2) : A_1 \times A_2 \vdash$:

$$
\begin{aligned}
& (e_1, e_2) : A_1 \times A_2 \vdash \\
\iff \quad & \neg(\llbracket (e_1, e_2) \rrbracket \subseteq \llbracket A_1 \times A_2 \rrbracket) \\
\iff \quad & \neg(\forall p.\ p \in \llbracket (e_1, e_2) \rrbracket \implies p \in \llbracket A_1 \times A_2 \rrbracket) \\
\iff \quad & \neg(\forall x, y.\ (x, y) \in \llbracket e_1 \rrbracket \times \llbracket e_2 \rrbracket \implies (x, y) \in \llbracket A_1 \rrbracket \times \llbracket A_2 \rrbracket) \\
\iff \quad & \neg(\forall x, y.\ x \in \llbracket e_1 \rrbracket \wedge y \in \llbracket e_2 \rrbracket \implies x \in \llbracket A_1 \rrbracket \wedge y \in \llbracket A_2 \rrbracket) \\
\iff \quad & \exists x, y.\ x \in \llbracket e_1 \rrbracket \wedge y \in \llbracket e_2 \rrbracket \wedge \neg(x \in \llbracket A_1 \rrbracket \wedge y \in \llbracket A_2 \rrbracket)
\end{aligned}
$$

# Deriving Typing Rules

Let's derive a typing rule for $(e_1, e_2) : A_1 \times A_2 \vdash$:

$$\begin{aligned}
& (e_1, e_2) : A_1 \times A_2 \vdash \\
\iff\quad & \neg(\llbracket (e_1, e_2) \rrbracket \subseteq \llbracket A_1 \times A_2 \rrbracket) \\
\iff\quad & \neg(\forall p.\ p \in \llbracket (e_1, e_2) \rrbracket \implies p \in \llbracket A_1 \times A_2 \rrbracket) \\
\iff\quad & \neg(\forall x, y.\ (x, y) \in \llbracket e_1 \rrbracket \times \llbracket e_2 \rrbracket \implies (x, y) \in \llbracket A_1 \rrbracket \times \llbracket A_2 \rrbracket) \\
\iff\quad & \neg(\forall x, y.\ x \in \llbracket e_1 \rrbracket \wedge y \in \llbracket e_2 \rrbracket \implies x \in \llbracket A_1 \rrbracket \wedge y \in \llbracket A_2 \rrbracket) \\
\iff\quad & \exists x, y.\ x \in \llbracket e_1 \rrbracket \wedge y \in \llbracket e_2 \rrbracket \wedge \neg(x \in \llbracket A_1 \rrbracket \wedge y \in \llbracket A_2 \rrbracket) \\
\iff\quad & \exists x, y.\ x \in \llbracket e_1 \rrbracket \wedge y \in \llbracket e_2 \rrbracket \wedge (x \notin \llbracket A_1 \rrbracket \vee y \notin \llbracket A_2 \rrbracket)
\end{aligned}$$

# Deriving Typing Rules

Let's derive a typing rule for $(e_1, e_2) : A_1 \times A_2 \vdash$:

$$
\begin{aligned}
& (e_1, e_2) : A_1 \times A_2 \vdash \\
\iff\quad & \neg(\llbracket(e_1, e_2)\rrbracket \subseteq \llbracket A_1 \times A_2\rrbracket) \\
\iff\quad & \neg(\forall p.\ p \in \llbracket(e_1, e_2)\rrbracket \implies p \in \llbracket A_1 \times A_2\rrbracket) \\
\iff\quad & \neg(\forall x, y.\ (x, y) \in \llbracket e_1\rrbracket \times \llbracket e_2\rrbracket \implies (x, y) \in \llbracket A_1\rrbracket \times \llbracket A_2\rrbracket) \\
\iff\quad & \neg(\forall x, y.\ x \in \llbracket e_1\rrbracket \wedge y \in \llbracket e_2\rrbracket \implies x \in \llbracket A_1\rrbracket \wedge y \in \llbracket A_2\rrbracket) \\
\iff\quad & \exists x, y.\ x \in \llbracket e_1\rrbracket \wedge y \in \llbracket e_2\rrbracket \wedge \neg(x \in \llbracket A_1\rrbracket \wedge y \in \llbracket A_2\rrbracket) \\
\iff\quad & \exists x, y.\ x \in \llbracket e_1\rrbracket \wedge y \in \llbracket e_2\rrbracket \wedge (x \notin \llbracket A_1\rrbracket \vee y \notin \llbracket A_2\rrbracket) \\
\iff\quad & \exists i.\exists x.\ x \in \llbracket e_i\rrbracket \wedge x \notin \llbracket A_i\rrbracket
\end{aligned}
$$

# Deriving Typing Rules

Let's derive a typing rule for $(e_1, e_2) : A_1 \times A_2 \vdash$:

$$
\begin{aligned}
& (e_1, e_2) : A_1 \times A_2 \vdash \\
\iff\quad & \neg(\llbracket (e_1, e_2) \rrbracket \subseteq \llbracket A_1 \times A_2 \rrbracket) \\
\iff\quad & \neg(\forall p.\ p \in \llbracket (e_1, e_2) \rrbracket \implies p \in \llbracket A_1 \times A_2 \rrbracket) \\
\iff\quad & \neg(\forall x, y.\ (x, y) \in \llbracket e_1 \rrbracket \times \llbracket e_2 \rrbracket \implies (x, y) \in \llbracket A_1 \rrbracket \times \llbracket A_2 \rrbracket) \\
\iff\quad & \neg(\forall x, y.\ x \in \llbracket e_1 \rrbracket \wedge y \in \llbracket e_2 \rrbracket \implies x \in \llbracket A_1 \rrbracket \wedge y \in \llbracket A_2 \rrbracket) \\
\iff\quad & \exists x, y.\ x \in \llbracket e_1 \rrbracket \wedge y \in \llbracket e_2 \rrbracket \wedge \neg(x \in \llbracket A_1 \rrbracket \wedge y \in \llbracket A_2 \rrbracket) \\
\iff\quad & \exists x, y.\ x \in \llbracket e_1 \rrbracket \wedge y \in \llbracket e_2 \rrbracket \wedge (x \notin \llbracket A_1 \rrbracket \vee y \notin \llbracket A_2 \rrbracket) \\
\iff\quad & \exists i. \exists x.\ x \in \llbracket e_i \rrbracket \wedge x \notin \llbracket A_i \rrbracket \\
\iff\quad & \exists i.\ \neg(\forall x.\ x \in \llbracket e_i \rrbracket \implies x \in \llbracket A_i \rrbracket)
\end{aligned}
$$

# Deriving Typing Rules

Let's derive a typing rule for $(e_1, e_2) : A_1 \times A_2 \vdash$:

$$
\begin{aligned}
& (e_1, e_2) : A_1 \times A_2 \vdash \\
\iff \quad & \neg(\llbracket (e_1, e_2) \rrbracket \subseteq \llbracket A_1 \times A_2 \rrbracket) \\
\iff \quad & \neg(\forall p.\ p \in \llbracket (e_1, e_2) \rrbracket \implies p \in \llbracket A_1 \times A_2 \rrbracket) \\
\iff \quad & \neg(\forall x, y.\ (x, y) \in \llbracket e_1 \rrbracket \times \llbracket e_2 \rrbracket \implies (x, y) \in \llbracket A_1 \rrbracket \times \llbracket A_2 \rrbracket) \\
\iff \quad & \neg(\forall x, y.\ x \in \llbracket e_1 \rrbracket \wedge y \in \llbracket e_2 \rrbracket \implies x \in \llbracket A_1 \rrbracket \wedge y \in \llbracket A_2 \rrbracket) \\
\iff \quad & \exists x, y.\ x \in \llbracket e_1 \rrbracket \wedge y \in \llbracket e_2 \rrbracket \wedge \neg(x \in \llbracket A_1 \rrbracket \wedge y \in \llbracket A_2 \rrbracket) \\
\iff \quad & \exists x, y.\ x \in \llbracket e_1 \rrbracket \wedge y \in \llbracket e_2 \rrbracket \wedge (x \notin \llbracket A_1 \rrbracket \vee y \notin \llbracket A_2 \rrbracket) \\
\iff \quad & \exists i. \exists x.\ x \in \llbracket e_i \rrbracket \wedge x \notin \llbracket A_i \rrbracket \\
\iff \quad & \exists i.\ \neg(\forall x.\ x \in \llbracket e_i \rrbracket \implies x \in \llbracket A_i \rrbracket) \\
\iff \quad & \exists i.\ \neg(\llbracket e_i \rrbracket \subseteq \llbracket A_i \rrbracket)
\end{aligned}
$$

# Deriving Typing Rules

Let's derive a typing rule for $(e_1, e_2) : A_1 \times A_2 \vdash$:

$$
\begin{aligned}
& (e_1, e_2) : A_1 \times A_2 \vdash \\
\iff \quad & \neg([\![(e_1, e_2)]\!] \subseteq [\![A_1 \times A_2]\!]) \\
\iff \quad & \neg(\forall p.\ p \in [\![(e_1, e_2)]\!] \implies p \in [\![A_1 \times A_2]\!]) \\
\iff \quad & \neg(\forall x, y.\ (x, y) \in [\![e_1]\!] \times [\![e_2]\!] \implies (x, y) \in [\![A_1]\!] \times [\![A_2]\!]) \\
\iff \quad & \neg(\forall x, y.\ x \in [\![e_1]\!] \wedge y \in [\![e_2]\!] \implies x \in [\![A_1]\!] \wedge y \in [\![A_2]\!]) \\
\iff \quad & \exists x, y.\ x \in [\![e_1]\!] \wedge y \in [\![e_2]\!] \wedge \neg(x \in [\![A_1]\!] \wedge y \in [\![A_2]\!]) \\
\iff \quad & \exists x, y.\ x \in [\![e_1]\!] \wedge y \in [\![e_2]\!] \wedge (x \notin [\![A_1]\!] \vee y \notin [\![A_2]\!]) \\
\iff \quad & \exists i. \exists x.\ x \in [\![e_i]\!] \wedge x \notin [\![A_i]\!] \\
\iff \quad & \exists i.\ \neg(\forall x.\ x \in [\![e_i]\!] \implies x \in [\![A_i]\!]) \\
\iff \quad & \exists i.\ \neg([\![e_i]\!] \subseteq [\![A_i]\!]) \\
\iff \quad & \exists i.\ e_i : A_i \vdash
\end{aligned}
$$

# Deriving Typing Rules

Let's derive a typing rule for $(e_1, e_2) : A_1 \times A_2 \vdash$:

$$\begin{aligned} & (e_1, e_2) : A_1 \times A_2 \vdash \\ \iff \quad & \exists i.\ e_i : A_i \vdash \end{aligned}$$

# Deriving Typing Rules

Let's derive a typing rule for $(e_1, e_2) : A_1 \times A_2 \vdash$:

$$
\begin{aligned}
& (e_1, e_2) : A_1 \times A_2 \vdash \\
\iff \quad & \exists i.\ e_i : A_i \vdash
\end{aligned}
$$

$$
(\text{PairL})\ \frac{(\exists i.)\ \Gamma, e_i : A_i \vdash \Delta}{\Gamma, (e_1, e_2) : A_1 \times A_2 \vdash \Delta}
$$

# Deriving Typing Rules

Let's derive a typing rule for $(e_1, e_2) : A_1 \times A_2 \vdash$:

$$\begin{aligned} & (e_1, e_2) : A_1 \times A_2 \vdash \\ \iff \quad & \exists i.\ e_i : A_i \vdash \end{aligned}$$

$$(\text{PairL})\,\frac{(\exists i.)\ \Gamma, e_i : A_i \vdash \Delta}{\Gamma, (e_1, e_2) : A_1 \times A_2 \vdash \Delta} \qquad (\text{PairR})\,\frac{(\forall i.)\ \Gamma \vdash e_i : A_i, \Delta}{\Gamma \vdash (e_1, e_2) : A_1 \times A_2, \Delta}$$

# Deriving Typing Rules

So we have derived a new typing rule:

$$\text{(PairL)}\ \frac{(\exists i.)\ \Gamma, e_i : A_i \vdash \Delta}{\Gamma, (e_1, e_2) : A_1 \times A_2 \vdash \Delta} \qquad \text{(PairR)}\ \frac{(\forall i.)\ \Gamma \vdash e_i : A_i, \Delta}{\Gamma \vdash (e_1, e_2) : A_1 \times A_2, \Delta}$$

And the key point in its derivation is:

$$x \notin [\![A_1 \times A_2]\!] \iff (x \notin [\![A_1]\!] \vee x \notin [\![A_2]\!])$$

Alternatively:

$$[\![A_1 \times A_2]\!]^c = [\![A_1]\!]^c \cup [\![A_2]\!]^c$$

# Deriving Typing Rules - If

$$(\text{IfR})\ \frac{\Gamma \vdash e_0 : \text{Bool} \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e_0 \text{ then } e_1 \text{ else } e_2 : \tau}$$

$$\frac{?}{\text{if } e_0 \text{ then } e_1 \text{ else } e_2 : \tau \vdash}$$

# Deriving Typing Rules - App

$$(\text{AppR})\ \frac{\Gamma \vdash e_0 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_1 : \tau_1}{\Gamma \vdash e_0\ e_1 : \tau_2}$$

$$\frac{?}{e_0\ e_1 : \tau_2 \vdash}$$

# Why does AppR work?

To prove the (AppR) rule correct, we show:

Given:

$$\Gamma \vdash e_0 : \tau_1 \rightarrow \tau_2$$
$$\Gamma \vdash e_1 : \tau_1$$

We want to show:

$$\Gamma \vdash (e_0 \;\, e_1) : \tau_2$$

# Why does AppR work?

To prove the (AppR) rule correct, we show:

Given:

$$\llbracket \Gamma \rrbracket \implies \llbracket e_0 \rrbracket \subseteq \llbracket \tau_1 \to \tau_2 \rrbracket$$
$$\llbracket \Gamma \rrbracket \implies \llbracket e_1 \rrbracket \subseteq \llbracket \tau_1 \rrbracket$$

We want to show:

$$\llbracket \Gamma \rrbracket \implies \llbracket e_0 \ e_1 \rrbracket \subseteq \llbracket \tau_2 \rrbracket$$

# Why does AppR work?

To prove the (AppR) rule correct, we show:

Given:

$$\llbracket e_0 \rrbracket \subseteq \llbracket \tau_1 \to \tau_2 \rrbracket$$
$$\llbracket e_1 \rrbracket \subseteq \llbracket \tau_1 \rrbracket$$

We want to show:

$$\llbracket e_0\ e_1 \rrbracket \subseteq \llbracket \tau_2 \rrbracket$$

# Why does AppR work?

To prove the (AppR) rule correct, we show:

Given:

$$\llbracket e_0 \rrbracket \subseteq \{f \mid f : \mathcal{U} \to \mathcal{U},\ x \in \llbracket \tau_1 \rrbracket \implies f(x) \in \llbracket \tau_2 \rrbracket\}$$
$$\llbracket e_1 \rrbracket \subseteq \llbracket \tau_1 \rrbracket$$

We want to show:

$$\llbracket e_0\ e_1 \rrbracket \subseteq \llbracket \tau_2 \rrbracket$$

# Why does AppR work?

To prove the (AppR) rule correct, we show:

Given:

$$
\begin{aligned}
&[\![e_0]\!] \subseteq \{f \mid f : \mathcal{U} \to \mathcal{U},\ x \in [\![\tau_1]\!] \implies f(x) \in [\![\tau_2]\!]\} \\
&[\![e_1]\!] \subseteq [\![\tau_1]\!]
\end{aligned}
$$

We want to show:

$$\{f(x) \mid f \in [\![e_0]\!],\ x \in [\![e_1]\!]\} \subseteq [\![\tau_2]\!]$$

# Why doesn't this AppL work?

$$\frac{\Gamma \vdash e_0 : \tau_1 \to \tau_2 \quad \Gamma,\ e_1 : \tau_1 \vdash}{\Gamma,\ (e_0\ e_1) : \tau_2 \vdash}$$

Given:

$$[\![e_0]\!] \subseteq \{f \mid f : \mathcal{U} \to \mathcal{U},\ x \in [\![\tau_1]\!] \implies f(x) \in [\![\tau_2]\!]\}$$
$$\neg([\![e_1]\!] \subseteq [\![\tau_1]\!])$$

We want to show:

$$\neg(\{f(x) \mid f \in [\![e_0]\!],\ x \in [\![e_1]\!]\} \subseteq [\![\tau_2]\!])$$

# What do we want?

$$[\![?]\!] := \{f \mid f : \mathcal{U} \to \mathcal{U},\ f(x) \in [\![\tau_2]\!] \implies x \in [\![\tau_1]\!]\}$$

# What do we want?

$$[\![\tau_1 \succ\!\!- \tau_2]\!] := \{f \mid f : \mathcal{U} \to \mathcal{U},\ f(x) \in [\![\tau_2]\!] \implies x \in [\![\tau_1]\!]\}$$

# Necessity Function Type

$$A \rightarrowtail B$$

"A only to B"

# Necessity Function Type

**Sufficiency**

$$\frac{\Gamma,\ x : A \vdash e : B,\ \Delta}{\Gamma \vdash \lambda x.\, e : A \to B,\ \Delta}\,(\text{AbsR})$$

**Necessity**

$$(\text{AbnR})\,\frac{\Gamma,\ e : B \vdash x : A,\ \Delta}{\Gamma \vdash \lambda x.\, e : A \succ\!\!- B,\ \Delta}$$

# Necessity Function Type

## Sufficiency

$$\frac{\Gamma,\ x : A \vdash e : B,\ \Delta}{\Gamma \vdash \lambda x.\, e : A \to B,\ \Delta}\,(\mathsf{AbsR})$$

$$\vdash \lambda x.\, 42 : \mathsf{Nat} \to \mathsf{Nat}$$

## Necessity

$$(\mathsf{AbnR})\,\frac{\Gamma,\ e : B \vdash x : A,\ \Delta}{\Gamma \vdash \lambda x.\, e : A \succ\!\!- B,\ \Delta}$$

# Necessity Function Type

## Sufficiency

$$\frac{\Gamma,\ x : A \vdash e : B,\ \Delta}{\Gamma \vdash \lambda x.\, e : A \to B,\ \Delta}\,(\mathsf{AbsR})$$

$\vdash \lambda x.\, 42 : \mathsf{Nat} \to \mathsf{Nat}$

$\vdash \mathsf{map}\ f\ : \mathsf{Nil} \to \mathsf{Nil}$

## Necessity

$$(\mathsf{AbnR})\,\frac{\Gamma,\ e : B \vdash x : A,\ \Delta}{\Gamma \vdash \lambda x.\, e : A \succ\!\!- B,\ \Delta}$$

# Necessity Function Type

## Sufficiency

$$\frac{\Gamma,\ x : A \vdash e : B,\ \Delta}{\Gamma \vdash \lambda x.\, e : A \to B,\ \Delta}\ (\mathsf{AbsR})$$

$\vdash \lambda x.\, 42 : \mathsf{Nat} \to \mathsf{Nat}$

$\vdash \mathsf{map}\ f\ : \mathsf{Nil} \to \mathsf{Nil}$

$\vdash \mathsf{map}\ f\ : \mathsf{Nil} \to \mathsf{List}\ \alpha$

## Necessity

$$(\mathsf{AbnR})\ \frac{\Gamma,\ e : B \vdash x : A,\ \Delta}{\Gamma \vdash \lambda x.\, e : A \succ\!\!- B,\ \Delta}$$

# Necessity Function Type

## Sufficiency

$$\frac{\Gamma,\ x : A \vdash e : B,\ \Delta}{\Gamma \vdash \lambda x.\, e : A \to B,\ \Delta}\ (\mathsf{AbsR})$$

$\vdash \lambda x.\, 42 : \mathsf{Nat} \to \mathsf{Nat}$

$\vdash \mathsf{map}\ f\ : \mathsf{Nil} \to \mathsf{Nil}$

$\vdash \mathsf{map}\ f\ : \mathsf{Nil} \to \mathsf{List}\ \alpha$

$\nvdash \mathsf{map}\ f\ : \mathsf{List}(\alpha) \to \mathsf{Nil}$

## Necessity

$$(\mathsf{AbnR})\ \frac{\Gamma,\ e : B \vdash x : A,\ \Delta}{\Gamma \vdash \lambda x.\, e : A \succ\!\!- B,\ \Delta}$$

# Necessity Function Type

## Sufficiency

$$\frac{\Gamma,\ x : A \vdash e : B,\ \Delta}{\Gamma \vdash \lambda x.\, e : A \to B,\ \Delta}\ (\mathsf{AbsR})$$

$\vdash \lambda x.\, 42 : \mathsf{Nat} \to \mathsf{Nat}$

$\vdash \mathsf{map}\ f : \mathsf{Nil} \to \mathsf{Nil}$

$\vdash \mathsf{map}\ f : \mathsf{Nil} \to \mathsf{List}\ \alpha$

$\nvdash \mathsf{map}\ f : \mathsf{List}(\alpha) \to \mathsf{Nil}$

## Necessity

$$(\mathsf{AbnR})\ \frac{\Gamma,\ e : B \vdash x : A,\ \Delta}{\Gamma \vdash \lambda x.\, e : A \succ\!\!- B,\ \Delta}$$

$\vdash \mathsf{head} : \mathsf{Cons}(\alpha, \mathsf{Ok}) \succ\!\!- \alpha$

# Necessity Function Type

## Sufficiency

$$\frac{\Gamma,\ x : A \vdash e : B,\ \Delta}{\Gamma \vdash \lambda x.\, e : A \to B,\ \Delta}\ (\mathsf{AbsR})$$

$\vdash \lambda x.\, 42 : \mathsf{Nat} \to \mathsf{Nat}$

$\vdash \mathsf{map}\ f : \mathsf{Nil} \to \mathsf{Nil}$

$\vdash \mathsf{map}\ f : \mathsf{Nil} \to \mathsf{List}\ \alpha$

$\nvdash \mathsf{map}\ f : \mathsf{List}(\alpha) \to \mathsf{Nil}$

## Necessity

$$(\mathsf{AbnR})\ \frac{\Gamma,\ e : B \vdash x : A,\ \Delta}{\Gamma \vdash \lambda x.\, e : A \succ\!\!- B,\ \Delta}$$

$\vdash \mathsf{head} : \mathsf{Cons}(\alpha, \mathsf{Ok}) \succ\!\!- \alpha$

$\vdash \mathsf{map}\ f : \mathsf{Nil} \succ\!\!- \mathsf{Nil}$

# Necessity Function Type

## Sufficiency

$$\frac{\Gamma,\ x : A \vdash e : B,\ \Delta}{\Gamma \vdash \lambda x.\, e : A \to B,\ \Delta}\ (\mathsf{AbsR})$$

$\vdash \lambda x.\, 42 : \mathsf{Nat} \to \mathsf{Nat}$

$\vdash \mathsf{map}\ f : \mathsf{Nil} \to \mathsf{Nil}$

$\vdash \mathsf{map}\ f : \mathsf{Nil} \to \mathsf{List}\ \alpha$

$\nvdash \mathsf{map}\ f : \mathsf{List}(\alpha) \to \mathsf{Nil}$

## Necessity

$$(\mathsf{AbnR})\ \frac{\Gamma,\ e : B \vdash x : A,\ \Delta}{\Gamma \vdash \lambda x.\, e : A \succ\!\!- B,\ \Delta}$$

$\vdash \mathsf{head} : \mathsf{Cons}(\alpha, \mathsf{Ok}) \succ\!\!- \alpha$

$\vdash \mathsf{map}\ f : \mathsf{Nil} \succ\!\!- \mathsf{Nil}$

$\vdash \mathsf{map}\ f : \mathsf{List}\ \alpha \succ\!\!- \mathsf{Nil}$

# Necessity Function Type

## Sufficiency

$$\frac{\Gamma,\ x : A \vdash e : B,\ \Delta}{\Gamma \vdash \lambda x.\, e : A \to B,\ \Delta}\,(\text{AbsR})$$

$\vdash \lambda x.\, 42 : \mathsf{Nat} \to \mathsf{Nat}$

$\vdash \mathsf{map}\ f : \mathsf{Nil} \to \mathsf{Nil}$

$\vdash \mathsf{map}\ f : \mathsf{Nil} \to \mathsf{List}\ \alpha$

$\nvdash \mathsf{map}\ f : \mathsf{List}(\alpha) \to \mathsf{Nil}$

## Necessity

$$(\text{AbnR})\,\frac{\Gamma,\ e : B \vdash x : A,\ \Delta}{\Gamma \vdash \lambda x.\, e : A \succ\!\!- B,\ \Delta}$$

$\vdash \mathsf{head} : \mathsf{Cons}(\alpha, \mathsf{Ok}) \succ\!\!- \alpha$

$\vdash \mathsf{map}\ f : \mathsf{Nil} \succ\!\!- \mathsf{Nil}$

$\vdash \mathsf{map}\ f : \mathsf{List}\ \alpha \succ\!\!- \mathsf{Nil}$

$\nvdash \lambda x.\, 42 : \mathsf{Nat} \succ\!\!- \mathsf{Nat}$

# Application on the Left

Now we have:

$$
(\mathrm{AppL})\ \frac{\Gamma \vdash e_1 : A \succ\!\!- B,\ \Delta \quad \Gamma,\ e_2 : A \vdash \Delta}{\Gamma,\ e_1\ e_2 : B \vdash \Delta}
$$

# Application on the Left

Now we have:

$$(\mathrm{AppL})\ \frac{\Gamma \vdash e_1 : A \succ\!\!- B,\ \Delta \quad \Gamma,\ e_2 : A \vdash \Delta}{\Gamma,\ e_1\ e_2 : B \vdash \Delta}$$

$$(\mathrm{IfL1})\ \dfrac{(\mathrm{AppL})\ \dfrac{\vdash \lambda x.\,x : \mathrm{Bool} \succ\!\!- \mathrm{Bool} \quad 1 : \mathrm{Bool} \vdash}{(\lambda x.\,x)\ 1 : \mathrm{Bool} \vdash}}{\text{if } (\lambda x.\,x)\ 1 \text{ then } 1 \text{ else } 1 : \mathrm{Ok} \vdash}$$

"(if $(\lambda x.\,x)$ 1 then 1 else 1) does not evaluate (reach a value)"

## Soundness Results

$$\vdash e : \mathsf{Ok}$$

"Well-typed programs don't go wrong."

No execution of a well-typed program can crash.

$$e : \mathsf{Ok} \vdash$$

"Ill-typed programs don't evaluate."

No execution of an ill-typed program can reach a value.