Introduction to Proving Stuff™ with Logical Relations

Jesse Sigal

November 7, 2024

Attribution

- Slides with $*$ at the end of their title were written with the help of GPT 4o (for lazy LaTeX'ing).
- Most things for the calculus are in line with Crole 1994.

Overview

- What do we want to prove?
- Lambda calculus (review?)
  - Types
  - Signatures
  - Syntax
  - Typing judgments
  - Denotational semantics
- Logical relations
  - Types
  - Signatures
  - Terms
  - Fundamental theorem
  - Application

What do we want to prove?

What do we want to prove?

- Proving something about *all* programs in a language?

What do we want to prove?

- Proving something about *all* programs in a language?
- If we have $+$ and $\times$, how can we prove even in $\Rightarrow$ even out?

What do we want to prove?

- Proving something about *all* programs in a language?
- If we have $+$ and $\times$, how can we prove even in $\Rightarrow$ even out?
- Logical relations!

What do we want to prove?

- Proving something about *all* programs in a language?
- If we have $+$ and $\times$, how can we prove even in $\Rightarrow$ even out?
- Logical relations!
- Also can prove more complicated and interesting theorems.

What do we want to prove?

- Proving something about *all* programs in a language?
- If we have $+$ and $\times$, how can we prove even in $\Rightarrow$ even out?
- Logical relations!
- Also can prove more complicated and interesting theorems.
- For example:

What do we want to prove?

- Proving something about *all* programs in a language?
- If we have $+$ and $\times$, how can we prove even in $\Rightarrow$ even out?
- Logical relations!
- Also can prove more complicated and interesting theorems.
- For example:
    - Termination: do your programs stop?

What do we want to prove?

- Proving something about *all* programs in a language?
- If we have $+$ and $\times$, how can we prove even in $\Rightarrow$ even out?
- Logical relations!
- Also can prove more complicated and interesting theorems.
- For example:
  - Termination: do your programs stop?
  - Type safety: do your programs keep going?

What do we want to prove?

- Proving something about *all* programs in a language?
- If we have $+$ and $\times$, how can we prove even in $\Rightarrow$ even out?
- Logical relations!
- Also can prove more complicated and interesting theorems.
- For example:
  - Termination: do your programs stop?
  - Type safety: do your programs keep going?
  - Optimizations: why can I rewrite my program?

What do we want to prove?

- Proving something about *all* programs in a language?
- If we have $+$ and $\times$, how can we prove even in $\Rightarrow$ even out?
- Logical relations!
- Also can prove more complicated and interesting theorems.
- For example:
  - Termination: do your programs stop?
  - Type safety: do your programs keep going?
  - Optimizations: why can I rewrite my program?
  - Representation independence: internals don't matter if you hide them.

What do we want to prove?

- Proving something about *all* programs in a language?
- If we have $+$ and $\times$, how can we prove even in $\Rightarrow$ even out?
- Logical relations!
- Also can prove more complicated and interesting theorems.
- For example:
  - Termination: do your programs stop?
  - Type safety: do your programs keep going?
  - Optimizations: why can I rewrite my program?
  - Representation independence: internals don't matter if you hide them.
  - Security: show the output doesn't depend on secure information.

Types*

Types*

$$\alpha, \beta ::= \tau \mid 1 \mid \alpha_1 \times \alpha_2 \mid \alpha \to \beta$$

Types*

$$\alpha, \beta ::= \tau \mid 1 \mid \alpha_1 \times \alpha_2 \mid \alpha \to \beta$$

Where:

- τ is a ground type from a fixed set of symbols, e.g. $\{\mathsf{Int}, \mathsf{Bool}, \ldots\}$,

Types*

$$\alpha, \beta ::= \tau \mid 1 \mid \alpha_1 \times \alpha_2 \mid \alpha \to \beta$$

Where:

- τ is a ground type from a fixed set of symbols, e.g. $\{\mathsf{Int}, \mathsf{Bool}, \ldots\}$,
- 1 is the unit type,

Types*

$$\alpha, \beta ::= \tau \mid 1 \mid \alpha_1 \times \alpha_2 \mid \alpha \to \beta$$

Where:

- τ is a ground type from a fixed set of symbols, e.g. $\{\mathsf{Int}, \mathsf{Bool}, \ldots\}$,
- 1 is the unit type,
- $\alpha_1 \times \alpha_2$ is a product type,

Types*

$$\alpha, \beta ::= \tau \mid 1 \mid \alpha_1 \times \alpha_2 \mid \alpha \to \beta$$

Where:

- τ is a ground type from a fixed set of symbols, e.g. $\{\mathsf{Int}, \mathsf{Bool}, \ldots\}$,
- 1 is the unit type,
- $\alpha_1 \times \alpha_2$ is a product type,
- $\alpha \to \beta$ is a function type.

Signatures

Signatures

A signature $\Sigma = (\Sigma_{\text{const}}, \Sigma_{\text{func}})$ is composed of two sets, namely

Signatures

A signature $\Sigma = (\Sigma_{\mathsf{const}}, \Sigma_{\mathsf{func}})$ is composed of two sets, namely

- Σ_{const} whose elements are constant symbols $c : \tau$; and

Signatures

A signature $\Sigma = (\Sigma_{\mathrm{const}}, \Sigma_{\mathrm{func}})$ is composed of two sets, namely

- Σ_{const} whose elements are constant symbols $c : \tau$; and
- Σ_{func} whose elements are function symbols $f : (\tau_1, \ldots, \tau_n) \rightarrow \tau$.

Signatures

A signature $\Sigma = (\Sigma_{\text{const}}, \Sigma_{\text{func}})$ is composed of two sets, namely

- Σ_{const} whose elements are constant symbols $c : \tau$; and
- Σ_{func} whose elements are function symbols $f : (\tau_1, \ldots, \tau_n) \rightarrow \tau$.

Everything is defined with respect to a signature Σ.

Signatures

A signature $\Sigma = (\Sigma_{\mathrm{const}}, \Sigma_{\mathrm{func}})$ is composed of two sets, namely

- Σ_{const} whose elements are constant symbols $c : \tau$; and
- Σ_{func} whose elements are function symbols $f : (\tau_1, \dots, \tau_n) \to \tau$.

Everything is defined with respect to a signature Σ.

For example, assume that we have Int as ground type. Then we could defined $\Sigma = (\{\underline{n} : n \in \mathbb{Z}, \}, \{\underline{+}, \underline{\times}\})$.

Syntax*

Syntax*

$$M, N ::= x \mid \langle\rangle \mid c \mid f(M_1, \ldots, M_n) \mid \lambda(x : \alpha).M \mid M\,N \mid \langle M_1, M_2 \rangle \mid \pi_1(M) \mid \pi_2(M)$$

Syntax*

$$M, N ::= x \mid \langle\rangle \mid c \mid f(M_1, \ldots, M_n) \mid \lambda(x : \alpha).M \mid M\,N \mid \langle M_1, M_2\rangle \mid \pi_1(M) \mid \pi_2(M)$$

Where:

- x is a variable from a countably infinite set $\{x, y, z, \ldots\}$,

Syntax*

$$M, N ::= x \mid \langle\rangle \mid c \mid f(M_1, \ldots, M_n) \mid \lambda(x : \alpha).M \mid M\,N \mid \langle M_1, M_2\rangle \mid \pi_1(M) \mid \pi_2(M)$$

Where:

- x is a variable from a countably infinite set $\{x, y, z, \ldots\}$,
- $\langle\rangle$ is the term of type 1,

Syntax*

$$M, N ::= x \mid \langle\rangle \mid c \mid f(M_1, \dots, M_n) \mid \lambda(x : \alpha).M \mid M\,N \mid \langle M_1, M_2\rangle \mid \pi_1(M) \mid \pi_2(M)$$

Where:

- x is a variable from a countably infinite set $\{x, y, z, \dots\}$,
- $\langle\rangle$ is the term of type 1,
- c is a constant symbol in Σ_{const},

Syntax*

$$M, N ::= x \mid \langle\rangle \mid c \mid f(M_1, \ldots, M_n) \mid \lambda(x : \alpha).M \mid M\,N \mid \langle M_1, M_2 \rangle \mid \pi_1(M) \mid \pi_2(M)$$

Where:

- x is a variable from a countably infinite set $\{x, y, z, \ldots\}$,
- $\langle\rangle$ is the term of type 1,
- c is a constant symbol in Σ_{const},
- f is a function symbol in Σ_{func},

Syntax*

$$M, N ::= x \mid \langle\rangle \mid c \mid f(M_1, \ldots, M_n) \mid \lambda(x : \alpha).M \mid M\,N \mid \langle M_1, M_2\rangle \mid \pi_1(M) \mid \pi_2(M)$$

Where:

- x is a variable from a countably infinite set $\{x, y, z, \ldots\}$,
- $\langle\rangle$ is the term of type 1,
- c is a constant symbol in Σ_{const},
- f is a function symbol in Σ_{func},
- $\lambda(x : \alpha).M$ is lambda abstraction with x of type α,

Syntax*

$$M, N ::= x \mid \langle\rangle \mid c \mid f(M_1, \ldots, M_n) \mid \lambda(x : \alpha).M \mid M\,N \mid \langle M_1, M_2\rangle \mid \pi_1(M) \mid \pi_2(M)$$

Where:

- x is a variable from a countably infinite set $\{x, y, z, \ldots\}$,
- $\langle\rangle$ is the term of type 1,
- c is a constant symbol in Σ_{const},
- f is a function symbol in Σ_{func},
- $\lambda(x : \alpha).M$ is lambda abstraction with x of type α,
- $M\,N$ is application,

Syntax*

$$M, N ::= x \mid \langle\rangle \mid c \mid f(M_1, \ldots, M_n) \mid \lambda(x : \alpha).M \mid M\,N \mid \langle M_1, M_2\rangle \mid \pi_1(M) \mid \pi_2(M)$$

Where:

- x is a variable from a countably infinite set $\{x, y, z, \ldots\}$,
- $\langle\rangle$ is the term of type 1,
- c is a constant symbol in Σ_{const},
- f is a function symbol in Σ_{func},
- $\lambda(x : \alpha).M$ is lambda abstraction with x of type α,
- $M\,N$ is application,
- $\langle M_1, M_2\rangle$ is a product,

Syntax*

$$M, N ::= x \mid \langle\rangle \mid c \mid f(M_1, \ldots, M_n) \mid \lambda(x : \alpha).M \mid M\,N \mid \langle M_1, M_2\rangle \mid \pi_1(M) \mid \pi_2(M)$$

Where:

- x is a variable from a countably infinite set $\{x, y, z, \ldots\}$,
- $\langle\rangle$ is the term of type 1,
- c is a constant symbol in Σ_{const},
- f is a function symbol in Σ_{func},
- $\lambda(x : \alpha).M$ is lambda abstraction with x of type α,
- $M\,N$ is application,
- $\langle M_1, M_2\rangle$ is a product,
- $\pi_1(M)$ and $\pi_2(M)$ are projections.

Typing Judgments*

Typing Judgments*

$$\frac{(x : \alpha) \in \Gamma}{\Gamma \vdash x : \alpha}$$

Typing Judgments*

$$\frac{(x : \alpha) \in \Gamma}{\Gamma \vdash x : \alpha} \qquad \frac{}{\Gamma \vdash \langle\rangle : 1}$$

Typing Judgments*

$$\frac{(x : \alpha) \in \Gamma}{\Gamma \vdash x : \alpha} \qquad \frac{}{\Gamma \vdash \langle\rangle : 1} \qquad \frac{c : \tau \in \Sigma_{\mathsf{const}}}{\Gamma \vdash c : \tau}$$

Typing Judgments*

$$\frac{(x : \alpha) \in \Gamma}{\Gamma \vdash x : \alpha} \qquad \frac{}{\Gamma \vdash \langle\rangle : 1} \qquad \frac{c : \tau \in \Sigma_{\mathsf{const}}}{\Gamma \vdash c : \tau}$$

$$\frac{\Gamma \vdash M_1 : \tau_1 \quad \cdots \quad \Gamma \vdash M_n : \tau_n \quad f : (\tau_1, \ldots, \tau_n) \to \tau \in \Sigma_{\mathsf{func}}}{\Gamma \vdash f(M_1, \ldots, M_n) : \tau}$$

Typing Judgments*

$$\frac{(x : \alpha) \in \Gamma}{\Gamma \vdash x : \alpha} \qquad \frac{}{\Gamma \vdash \langle\rangle : 1} \qquad \frac{c : \tau \in \Sigma_{\mathsf{const}}}{\Gamma \vdash c : \tau}$$

$$\frac{\Gamma \vdash M_1 : \tau_1 \quad \cdots \quad \Gamma \vdash M_n : \tau_n \quad f : (\tau_1, \ldots, \tau_n) \to \tau \in \Sigma_{\mathsf{func}}}{\Gamma \vdash f(M_1, \ldots, M_n) : \tau}$$

$$\frac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda(x : \alpha).M : \alpha \to \beta}$$

Typing Judgments*

$$\frac{(x : \alpha) \in \Gamma}{\Gamma \vdash x : \alpha} \qquad \frac{}{\Gamma \vdash \langle\rangle : 1} \qquad \frac{c : \tau \in \Sigma_{\mathsf{const}}}{\Gamma \vdash c : \tau}$$

$$\frac{\Gamma \vdash M_1 : \tau_1 \quad \cdots \quad \Gamma \vdash M_n : \tau_n \quad f : (\tau_1, \dots, \tau_n) \to \tau \in \Sigma_{\mathsf{func}}}{\Gamma \vdash f(M_1, \dots, M_n) : \tau}$$

$$\frac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda(x : \alpha).M : \alpha \to \beta} \qquad \frac{\Gamma \vdash M : \alpha \to \beta \quad \Gamma \vdash N : \alpha}{\Gamma \vdash M N : \beta}$$

Typing Judgments*

$$\frac{(x : \alpha) \in \Gamma}{\Gamma \vdash x : \alpha} \qquad \frac{}{\Gamma \vdash \langle\rangle : 1} \qquad \frac{c : \tau \in \Sigma_{\text{const}}}{\Gamma \vdash c : \tau}$$

$$\frac{\Gamma \vdash M_1 : \tau_1 \quad \cdots \quad \Gamma \vdash M_n : \tau_n \quad f : (\tau_1, \ldots, \tau_n) \to \tau \in \Sigma_{\text{func}}}{\Gamma \vdash f(M_1, \ldots, M_n) : \tau}$$

$$\frac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda(x : \alpha).M : \alpha \to \beta} \qquad \frac{\Gamma \vdash M : \alpha \to \beta \quad \Gamma \vdash N : \alpha}{\Gamma \vdash M N : \beta}$$

$$\frac{\Gamma \vdash M_1 : \alpha_1 \quad \Gamma \vdash M_2 : \alpha_2}{\Gamma \vdash \langle M_1, M_2 \rangle : \alpha_1 \times \alpha_2}$$

Typing Judgments*

$$\frac{(x : \alpha) \in \Gamma}{\Gamma \vdash x : \alpha} \qquad \frac{}{\Gamma \vdash \langle\rangle : 1} \qquad \frac{c : \tau \in \Sigma_{\mathsf{const}}}{\Gamma \vdash c : \tau}$$

$$\frac{\Gamma \vdash M_1 : \tau_1 \quad \cdots \quad \Gamma \vdash M_n : \tau_n \quad f : (\tau_1, \ldots, \tau_n) \to \tau \in \Sigma_{\mathsf{func}}}{\Gamma \vdash f(M_1, \ldots, M_n) : \tau}$$

$$\frac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda(x : \alpha).M : \alpha \to \beta} \qquad \frac{\Gamma \vdash M : \alpha \to \beta \quad \Gamma \vdash N : \alpha}{\Gamma \vdash M\,N : \beta}$$

$$\frac{\Gamma \vdash M_1 : \alpha_1 \quad \Gamma \vdash M_2 : \alpha_2}{\Gamma \vdash \langle M_1, M_2\rangle : \alpha_1 \times \alpha_2} \qquad \frac{\Gamma \vdash M : \alpha_1 \times \alpha_2}{\Gamma \vdash \pi_1(M) : \alpha_1}$$

Typing Judgments*

$$\frac{(x : \alpha) \in \Gamma}{\Gamma \vdash x : \alpha} \qquad \frac{}{\Gamma \vdash \langle\rangle : 1} \qquad \frac{c : \tau \in \Sigma_{\mathsf{const}}}{\Gamma \vdash c : \tau}$$

$$\frac{\Gamma \vdash M_1 : \tau_1 \quad \cdots \quad \Gamma \vdash M_n : \tau_n \quad f : (\tau_1, \ldots, \tau_n) \to \tau \in \Sigma_{\mathsf{func}}}{\Gamma \vdash f(M_1, \ldots, M_n) : \tau}$$

$$\frac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda(x : \alpha).M : \alpha \to \beta} \qquad \frac{\Gamma \vdash M : \alpha \to \beta \quad \Gamma \vdash N : \alpha}{\Gamma \vdash M\,N : \beta}$$

$$\frac{\Gamma \vdash M_1 : \alpha_1 \quad \Gamma \vdash M_2 : \alpha_2}{\Gamma \vdash \langle M_1, M_2 \rangle : \alpha_1 \times \alpha_2} \qquad \frac{\Gamma \vdash M : \alpha_1 \times \alpha_2}{\Gamma \vdash \pi_1(M) : \alpha_1} \qquad \frac{\Gamma \vdash M : \alpha_1 \times \alpha_2}{\Gamma \vdash \pi_2(M) : \alpha_2}$$

Set-Theoretic Denotational Semantics for Types*

Set-Theoretic Denotational Semantics for Types*

Let ρ be a function that assigns a set $\rho(\tau)$ to each ground type τ, e.g., $\rho(\mathsf{Int}) = \mathbb{Z}$, $\rho(\mathsf{Bool}) = \{\mathsf{true}, \mathsf{false}\}$.

Set-Theoretic Denotational Semantics for Types*

Let ρ be a function that assigns a set $\rho(\tau)$ to each ground type τ, e.g., $\rho(\mathsf{Int}) = \mathbb{Z}$, $\rho(\mathsf{Bool}) = \{\mathsf{true}, \mathsf{false}\}$.

Then to type α, we are going to assign a set $[\![\alpha]\!]_\rho$ as follows:

Set-Theoretic Denotational Semantics for Types*

Let ρ be a function that assigns a set $\rho(\tau)$ to each ground type τ, e.g., $\rho(\mathsf{Int}) = \mathbb{Z}$, $\rho(\mathsf{Bool}) = \{\mathsf{true}, \mathsf{false}\}$.

Then to type α, we are going to assign a set $[\![\alpha]\!]_\rho$ as follows:

- $[\![\tau]\!]_\rho = \rho(\tau)$

Set-Theoretic Denotational Semantics for Types*

Let ρ be a function that assigns a set $\rho(\tau)$ to each ground type τ, e.g., $\rho(\mathsf{Int}) = \mathbb{Z}$, $\rho(\mathsf{Bool}) = \{\mathsf{true}, \mathsf{false}\}$.

Then to type α, we are going to assign a set $[\![\alpha]\!]_\rho$ as follows:

- $[\![\tau]\!]_\rho = \rho(\tau)$
- $[\![1]\!]_\rho = \{\star\}$

Set-Theoretic Denotational Semantics for Types*

Let ρ be a function that assigns a set $\rho(\tau)$ to each ground type τ, e.g., $\rho(\mathsf{Int}) = \mathbb{Z}$, $\rho(\mathsf{Bool}) = \{\mathsf{true}, \mathsf{false}\}$.

Then to type α, we are going to assign a set $[\![\alpha]\!]_\rho$ as follows:

- $[\![\tau]\!]_\rho = \rho(\tau)$
- $[\![1]\!]_\rho = \{\star\}$
- $[\![\alpha_1 \times \alpha_2]\!]_\rho = [\![\alpha_1]\!]_\rho \times [\![\alpha_2]\!]_\rho$

Set-Theoretic Denotational Semantics for Types*

Let ρ be a function that assigns a set $\rho(\tau)$ to each ground type τ, e.g., $\rho(\mathsf{Int}) = \mathbb{Z}$, $\rho(\mathsf{Bool}) = \{\mathsf{true}, \mathsf{false}\}$.

Then to type α, we are going to assign a set $[\![\alpha]\!]_\rho$ as follows:

- $[\![\tau]\!]_\rho = \rho(\tau)$
- $[\![1]\!]_\rho = \{\star\}$
- $[\![\alpha_1 \times \alpha_2]\!]_\rho = [\![\alpha_1]\!]_\rho \times [\![\alpha_2]\!]_\rho$
- $[\![\alpha \to \beta]\!]_\rho = [\![\alpha]\!]_\rho \to [\![\beta]\!]_\rho$

Set-Theoretic Denotational Semantics for Signatures

Set-Theoretic Denotational Semantics for Signatures

For a fixed ρ assigning ground types to sets, we can give an interpretation σ to the constants and functions of a signature $\Sigma = (\Sigma_{\text{const}}, \Sigma_{\text{func}})$:

Set-Theoretic Denotational Semantics for Signatures

For a fixed ρ assigning ground types to sets, we can give an interpretation σ to the constants and functions of a signature $\Sigma = (\Sigma_{\mathsf{const}}, \Sigma_{\mathsf{func}})$:

- for each $c : \tau \in \Sigma_{\mathsf{const}}$, an element $\sigma(c) \in \rho(\tau)$; and

Set-Theoretic Denotational Semantics for Signatures

For a fixed ρ assigning ground types to sets, we can give an interpretation σ to the constants and functions of a signature $\Sigma = (\Sigma_{\mathsf{const}}, \Sigma_{\mathsf{func}})$:

- for each $c : \tau \in \Sigma_{\mathsf{const}}$, an element $\sigma(c) \in \rho(\tau)$; and
- for each $f : (\tau_1, \ldots, \tau_n) \to \tau \in \Sigma_{\mathsf{func}}$, a function $\sigma(f) \in \rho(\tau_1) \times \cdots \times \rho(\tau_n) \to \rho(\tau)$.

Set-Theoretic Denotational Semantics for Signatures

For a fixed ρ assigning ground types to sets, we can give an interpretation σ to the constants and functions of a signature $\Sigma = (\Sigma_{\text{const}}, \Sigma_{\text{func}})$:

- for each $c : \tau \in \Sigma_{\text{const}}$, an element $\sigma(c) \in \rho(\tau)$; and
- for each $f : (\tau_1, \ldots, \tau_n) \to \tau \in \Sigma_{\text{func}}$, a function $\sigma(f) \in \rho(\tau_1) \times \cdots \times \rho(\tau_n) \to \rho(\tau)$.

Note that $\sigma(c) \in [\![\tau]\!]_\rho$ and $\sigma(f) \in [\![\tau_1 \times \cdots \times \tau_n \to \tau]\!]_\rho$.

Set-Theoretic Denotational Semantics for Terms in Context*

Set-Theoretic Denotational Semantics for Terms in Context*

Fix a fixed ρ and σ, we can define the meaning of a lambda term. In a context $\Gamma = x_1 : \alpha_1, \ldots, x_n : \alpha_n$, the denotational semantics of a term M is a function:

$$[\![\Gamma \vdash M : \alpha]\!]_{\rho,\sigma} : [\![\alpha_1 \times \cdots \times \alpha_n]\!]_\rho \to [\![\alpha]\!]_\rho$$

For a Γ as above, we will write γ for an element of $[\![\alpha_1 \times \cdots \times \alpha_n]\!]_\rho$ and write $\gamma(x_i)$ for the i^{th} component of the tuple.

Set-Theoretic Denotational Semantics for Terms in Context*

Fix a fixed ρ and σ, we can define the meaning of a lambda term. In a context $\Gamma = x_1 : \alpha_1, \dots, x_n : \alpha_n$, the denotational semantics of a term M is a function:

$$[\![\Gamma \vdash M : \alpha]\!]_{\rho,\sigma} : [\![\alpha_1 \times \cdots \times \alpha_n]\!]_\rho \to [\![\alpha]\!]_\rho$$

For a Γ as above, we will write γ for an element of $[\![\alpha_1 \times \cdots \times \alpha_n]\!]_\rho$ and write $\gamma(x_i)$ for the i^{th} component of the tuple.

We write $\gamma[x \mapsto v]$ to denote the extension of γ mapping x to v. E.g. for $\Gamma = x : \mathsf{Int}, y : \mathsf{Int}$ if $\{x \mapsto 1, y \mapsto 2\} \in [\![\mathsf{Int} \times \mathsf{Int}]\!]_\rho$ then

$$\{x \mapsto 1, y \mapsto 2\}[z \mapsto 3] := \{x \mapsto 1, y \mapsto 2, z \mapsto 3\} \in [\![\mathsf{Int} \times \mathsf{Int} \times \mathsf{Int}]\!]_\rho$$

for $\Gamma = x : \mathsf{Int}, y : \mathsf{Int}, z : \mathsf{Int}$.

Recursive Definition of Denotation for Terms*

Recursive Definition of Denotation for Terms*

The denotational semantics $[\![\Gamma \vdash M : \alpha]\!]_{\rho,\sigma}(\gamma)$ is defined recursively as follows:

Recursive Definition of Denotation for Terms*

The denotational semantics $[\![\Gamma \vdash M : \alpha]\!]_{\rho,\sigma}(\gamma)$ is defined recursively as follows:

- $[\![\Gamma \vdash x : \alpha]\!]_{\rho,\sigma}(\gamma) = \gamma(x)$

Recursive Definition of Denotation for Terms*

The denotational semantics $[\![\Gamma \vdash M : \alpha]\!]_{\rho,\sigma}(\gamma)$ is defined recursively as follows:

- $[\![\Gamma \vdash x : \alpha]\!]_{\rho,\sigma}(\gamma) = \gamma(x)$
- $[\![\Gamma \vdash \langle\rangle : 1]\!]_{\rho,\sigma}(\gamma) = \star$

Recursive Definition of Denotation for Terms*

The denotational semantics $[\![\Gamma \vdash M : \alpha]\!]_{\rho,\sigma}(\gamma)$ is defined recursively as follows:

- $[\![\Gamma \vdash x : \alpha]\!]_{\rho,\sigma}(\gamma) = \gamma(x)$
- $[\![\Gamma \vdash \langle\rangle : 1]\!]_{\rho,\sigma}(\gamma) = \star$
- $[\![\Gamma \vdash c : \tau]\!]_{\rho,\sigma}(\gamma) = \sigma(c)$

Recursive Definition of Denotation for Terms*

The denotational semantics $[\![\Gamma \vdash M : \alpha]\!]_{\rho,\sigma}(\gamma)$ is defined recursively as follows:

- $[\![\Gamma \vdash x : \alpha]\!]_{\rho,\sigma}(\gamma) = \gamma(x)$
- $[\![\Gamma \vdash \langle\rangle : 1]\!]_{\rho,\sigma}(\gamma) = \star$
- $[\![\Gamma \vdash c : \tau]\!]_{\rho,\sigma}(\gamma) = \sigma(c)$
- $[\![\Gamma \vdash f(M_1, \ldots, M_n) : \sigma]\!]_{\rho,\sigma}(\gamma) =$
  $\sigma(f)([\![\Gamma \vdash M_1 : \tau_1]\!]_{\rho,\sigma}(\gamma), \ldots, [\![\Gamma \vdash M_n : \tau_n]\!]_{\rho,\sigma}(\gamma))$

Recursive Definition of Denotation for Terms*

The denotational semantics $[\![\Gamma \vdash M : \alpha]\!]_{\rho,\sigma}(\gamma)$ is defined recursively as follows:

- $[\![\Gamma \vdash x : \alpha]\!]_{\rho,\sigma}(\gamma) = \gamma(x)$
- $[\![\Gamma \vdash \langle\rangle : 1]\!]_{\rho,\sigma}(\gamma) = \star$
- $[\![\Gamma \vdash c : \tau]\!]_{\rho,\sigma}(\gamma) = \sigma(c)$
- $[\![\Gamma \vdash f(M_1, \dots, M_n) : \sigma]\!]_{\rho,\sigma}(\gamma) =$
  $\sigma(f)([\![\Gamma \vdash M_1 : \tau_1]\!]_{\rho,\sigma}(\gamma), \dots, [\![\Gamma \vdash M_n : \tau_n]\!]_{\rho,\sigma}(\gamma))$
- $[\![\Gamma \vdash \lambda(x : \alpha).M : \alpha \to \beta]\!]_{\rho,\sigma}(\gamma) = \lambda v.[\![\Gamma, x : \alpha \vdash M : \beta]\!]_{\rho,\sigma}(\gamma[x \mapsto v])$

Recursive Definition of Denotation for Terms*

The denotational semantics $[\![\Gamma \vdash M : \alpha]\!]_{\rho,\sigma}(\gamma)$ is defined recursively as follows:

- $[\![\Gamma \vdash x : \alpha]\!]_{\rho,\sigma}(\gamma) = \gamma(x)$
- $[\![\Gamma \vdash \langle\rangle : 1]\!]_{\rho,\sigma}(\gamma) = \star$
- $[\![\Gamma \vdash c : \tau]\!]_{\rho,\sigma}(\gamma) = \sigma(c)$
- $[\![\Gamma \vdash f(M_1, \dots, M_n) : \sigma]\!]_{\rho,\sigma}(\gamma) =$
  $\sigma(f)([\![\Gamma \vdash M_1 : \tau_1]\!]_{\rho,\sigma}(\gamma), \dots, [\![\Gamma \vdash M_n : \tau_n]\!]_{\rho,\sigma}(\gamma))$
- $[\![\Gamma \vdash \lambda(x : \alpha).M : \alpha \to \beta]\!]_{\rho,\sigma}(\gamma) = \lambda v.[\![\Gamma, x : \alpha \vdash M : \beta]\!]_{\rho,\sigma}(\gamma[x \mapsto v])$
- $[\![\Gamma \vdash M\,N : \beta]\!]_{\rho,\sigma}(\gamma) = [\![M]\!]_{\rho,\sigma}(\gamma)([\![N]\!]_{\rho,\sigma}(\gamma))$

Recursive Definition of Denotation for Terms*

The denotational semantics $[\![\Gamma \vdash M : \alpha]\!]_{\rho,\sigma}(\gamma)$ is defined recursively as follows:

- $[\![\Gamma \vdash x : \alpha]\!]_{\rho,\sigma}(\gamma) = \gamma(x)$
- $[\![\Gamma \vdash \langle\rangle : 1]\!]_{\rho,\sigma}(\gamma) = \star$
- $[\![\Gamma \vdash c : \tau]\!]_{\rho,\sigma}(\gamma) = \sigma(c)$
- $[\![\Gamma \vdash f(M_1, \dots, M_n) : \sigma]\!]_{\rho,\sigma}(\gamma) =$
  $\sigma(f)([\![\Gamma \vdash M_1 : \tau_1]\!]_{\rho,\sigma}(\gamma), \dots, [\![\Gamma \vdash M_n : \tau_n]\!]_{\rho,\sigma}(\gamma))$
- $[\![\Gamma \vdash \lambda(x : \alpha).M : \alpha \to \beta]\!]_{\rho,\sigma}(\gamma) = \lambda v.[\![\Gamma, x : \alpha \vdash M : \beta]\!]_{\rho,\sigma}(\gamma[x \mapsto v])$
- $[\![\Gamma \vdash M\,N : \beta]\!]_{\rho,\sigma}(\gamma) = [\![M]\!]_{\rho,\sigma}(\gamma)([\![N]\!]_{\rho,\sigma}(\gamma))$
- $[\![\Gamma \vdash \langle M_1, M_2\rangle : \alpha_1 \times \alpha_2]\!]_{\rho,\sigma}(\gamma) = ([\![M_1]\!]_{\rho,\sigma}(\gamma), [\![M_2]\!]_{\rho,\sigma}(\gamma))$

Recursive Definition of Denotation for Terms*

The denotational semantics $[\![\Gamma \vdash M : \alpha]\!]_{\rho,\sigma}(\gamma)$ is defined recursively as follows:

- $[\![\Gamma \vdash x : \alpha]\!]_{\rho,\sigma}(\gamma) = \gamma(x)$
- $[\![\Gamma \vdash \langle\rangle : 1]\!]_{\rho,\sigma}(\gamma) = \star$
- $[\![\Gamma \vdash c : \tau]\!]_{\rho,\sigma}(\gamma) = \sigma(c)$
- $[\![\Gamma \vdash f(M_1, \ldots, M_n) : \sigma]\!]_{\rho,\sigma}(\gamma) =$
  $\sigma(f)([\![\Gamma \vdash M_1 : \tau_1]\!]_{\rho,\sigma}(\gamma), \ldots, [\![\Gamma \vdash M_n : \tau_n]\!]_{\rho,\sigma}(\gamma))$
- $[\![\Gamma \vdash \lambda(x : \alpha).M : \alpha \to \beta]\!]_{\rho,\sigma}(\gamma) = \lambda v.[\![\Gamma, x : \alpha \vdash M : \beta]\!]_{\rho,\sigma}(\gamma[x \mapsto v])$
- $[\![\Gamma \vdash M\,N : \beta]\!]_{\rho,\sigma}(\gamma) = [\![M]\!]_{\rho,\sigma}(\gamma)([\![N]\!]_{\rho,\sigma}(\gamma))$
- $[\![\Gamma \vdash \langle M_1, M_2\rangle : \alpha_1 \times \alpha_2]\!]_{\rho,\sigma}(\gamma) = ([\![M_1]\!]_{\rho,\sigma}(\gamma), [\![M_2]\!]_{\rho,\sigma}(\gamma))$
- $[\![\Gamma \vdash \pi_1(M) : \alpha_1]\!]_{\rho,\sigma}(\gamma) = \pi_1([\![M]\!]_{\rho,\sigma}(\gamma))$

Recursive Definition of Denotation for Terms*

The denotational semantics $[\![\Gamma \vdash M : \alpha]\!]_{\rho,\sigma}(\gamma)$ is defined recursively as follows:

- $[\![\Gamma \vdash x : \alpha]\!]_{\rho,\sigma}(\gamma) = \gamma(x)$
- $[\![\Gamma \vdash \langle\rangle : 1]\!]_{\rho,\sigma}(\gamma) = \star$
- $[\![\Gamma \vdash c : \tau]\!]_{\rho,\sigma}(\gamma) = \sigma(c)$
- $[\![\Gamma \vdash f(M_1, \dots, M_n) : \sigma]\!]_{\rho,\sigma}(\gamma) =$
  $\sigma(f)([\![\Gamma \vdash M_1 : \tau_1]\!]_{\rho,\sigma}(\gamma), \dots, [\![\Gamma \vdash M_n : \tau_n]\!]_{\rho,\sigma}(\gamma))$
- $[\![\Gamma \vdash \lambda(x : \alpha).M : \alpha \to \beta]\!]_{\rho,\sigma}(\gamma) = \lambda v.[\![\Gamma, x : \alpha \vdash M : \beta]\!]_{\rho,\sigma}(\gamma[x \mapsto v])$
- $[\![\Gamma \vdash M\,N : \beta]\!]_{\rho,\sigma}(\gamma) = [\![M]\!]_{\rho,\sigma}(\gamma)([\![N]\!]_{\rho,\sigma}(\gamma))$
- $[\![\Gamma \vdash \langle M_1, M_2\rangle : \alpha_1 \times \alpha_2]\!]_{\rho,\sigma}(\gamma) = ([\![M_1]\!]_{\rho,\sigma}(\gamma), [\![M_2]\!]_{\rho,\sigma}(\gamma))$
- $[\![\Gamma \vdash \pi_1(M) : \alpha_1]\!]_{\rho,\sigma}(\gamma) = \pi_1([\![M]\!]_{\rho,\sigma}(\gamma))$
- $[\![\Gamma \vdash \pi_2(M) : \alpha_2]\!]_{\rho,\sigma}(\gamma) = \pi_2([\![M]\!]_{\rho,\sigma}(\gamma))$

Logical Relations Semantics for Types

We can view logical relations as a new type of denotational semantics. We begin with the types. For each type α, we're going to define *two sets*, P and A, such that $P \subseteq A$. The set P is our *predicate*.

Logical Relations Semantics for Types

We can view logical relations as a new type of denotational semantics. We begin with the types. For each type α, we're going to define *two sets*, P and A, such that $P \subseteq A$. The set P is our *predicate*.

Let ρ be a function that assigns a pair of sets $(\rho_{\mathcal{P}}(\tau), \rho_{\mathcal{A}}(\tau))$ to each ground type τ such that $\rho_{\mathcal{P}}(\tau) \subseteq \rho_{\mathcal{A}}(\tau)$, e.g., $\rho(\mathsf{Int}) = (\{2m : m \in \mathbb{Z}\}, \mathbb{Z})$.

Logical Relations Semantics for Types

We can view logical relations as a new type of denotational semantics. We begin with the types. For each type α, we're going to define *two sets*, P and A, such that $P \subseteq A$. The set P is our *predicate*.

Let ρ be a function that assigns a pair of sets $(\rho_{\mathcal{P}}(\tau), \rho_{\mathcal{A}}(\tau))$ to each ground type τ such that $\rho_{\mathcal{P}}(\tau) \subseteq \rho_{\mathcal{A}}(\tau)$, e.g., $\rho(\mathsf{Int}) = (\{2m : m \in \mathbb{Z}\}, \mathbb{Z})$.

Then to type α, we are going to assign a pair of sets $\{\!\![\alpha]\!\!\}_\rho = \left(\mathcal{P}\{\!\![\alpha]\!\!\}_\rho, \mathcal{A}\{\!\![\alpha]\!\!\}_\rho\right)$ as follows:

Logical Relations Semantics for Types

We can view logical relations as a new type of denotational semantics. We begin with the types. For each type α, we're going to define *two sets*, P and A, such that $P \subseteq A$. The set P is our *predicate*.

Let ρ be a function that assigns a pair of sets $(\rho_{\mathcal{P}}(\tau), \rho_{\mathcal{A}}(\tau))$ to each ground type τ such that $\rho_{\mathcal{P}}(\tau) \subseteq \rho_{\mathcal{A}}(\tau)$, e.g., $\rho(\text{Int}) = (\{2m : m \in \mathbb{Z}\}, \mathbb{Z})$.

Then to type α, we are going to assign a pair of sets $\llbracket \alpha \rrbracket_\rho = \left(\mathcal{P}\llbracket \alpha \rrbracket_\rho, \mathcal{A}\llbracket \alpha \rrbracket_\rho\right)$ as follows:

- $\llbracket \tau \rrbracket_\rho = (\rho_{\mathcal{P}}(\tau), \rho_{\mathcal{A}}(\tau))$

Logical Relations Semantics for Types

We can view logical relations as a new type of denotational semantics. We begin with the types. For each type α, we're going to define *two sets*, P and A, such that $P \subseteq A$. The set P is our *predicate*.

Let ρ be a function that assigns a pair of sets $(\rho_{\mathcal{P}}(\tau), \rho_{\mathcal{A}}(\tau))$ to each ground type τ such that $\rho_{\mathcal{P}}(\tau) \subseteq \rho_{\mathcal{A}}(\tau)$, e.g., $\rho(\mathsf{Int}) = (\{2m : m \in \mathbb{Z}\}, \mathbb{Z})$.

Then to type α, we are going to assign a pair of sets $\llbracket\alpha\rrbracket_\rho = \left(\mathcal{P}\llbracket\alpha\rrbracket_\rho, \mathcal{A}\llbracket\alpha\rrbracket_\rho\right)$ as follows:

- $\llbracket\tau\rrbracket_\rho = (\rho_{\mathcal{P}}(\tau), \rho_{\mathcal{A}}(\tau))$
- $\llbracket 1\rrbracket_\rho = (\{\star\}, \{\star\})$

Logical Relations Semantics for Types

We can view logical relations as a new type of denotational semantics. We begin with the types. For each type α, we're going to define *two sets*, P and A, such that $P \subseteq A$. The set P is our *predicate*.

Let ρ be a function that assigns a pair of sets $(\rho_{\mathcal{P}}(\tau), \rho_{\mathcal{A}}(\tau))$ to each ground type τ such that $\rho_{\mathcal{P}}(\tau) \subseteq \rho_{\mathcal{A}}(\tau)$, e.g., $\rho(\mathsf{Int}) = (\{2m : m \in \mathbb{Z}\}, \mathbb{Z})$.

Then to type α, we are going to assign a pair of sets $\llbracket\alpha\rrbracket_\rho = \left(\mathcal{P}\llbracket\alpha\rrbracket_\rho, \mathcal{A}\llbracket\alpha\rrbracket_\rho\right)$ as follows:

- $\llbracket\tau\rrbracket_\rho = (\rho_{\mathcal{P}}(\tau), \rho_{\mathcal{A}}(\tau))$
- $\llbracket 1\rrbracket_\rho = (\{\star\}, \{\star\})$
- $\llbracket\alpha_1 \times \alpha_2\rrbracket_\rho = \left(\mathcal{P}\llbracket\alpha_1\rrbracket_\rho \times \mathcal{P}\llbracket\alpha_2\rrbracket_\rho, \mathcal{A}\llbracket\alpha_1\rrbracket_\rho \times \mathcal{A}\llbracket\alpha_2\rrbracket_\rho\right)$

Logical Relations Semantics for Types

We can view logical relations as a new type of denotational semantics. We begin with the types. For each type α, we're going to define *two sets*, P and A, such that $P \subseteq A$. The set P is our *predicate*.

Let ρ be a function that assigns a pair of sets $(\rho_{\mathcal{P}}(\tau), \rho_{\mathcal{A}}(\tau))$ to each ground type τ such that $\rho_{\mathcal{P}}(\tau) \subseteq \rho_{\mathcal{A}}(\tau)$, e.g., $\rho(\mathsf{Int}) = (\{2m : m \in \mathbb{Z}\}, \mathbb{Z})$.

Then to type α, we are going to assign a pair of sets $\llbracket\alpha\rrbracket_\rho = \left(\mathcal{P}\llbracket\alpha\rrbracket_\rho, \mathcal{A}\llbracket\alpha\rrbracket_\rho\right)$ as follows:

- $\llbracket\tau\rrbracket_\rho = (\rho_{\mathcal{P}}(\tau), \rho_{\mathcal{A}}(\tau))$
- $\llbracket 1\rrbracket_\rho = (\{\star\}, \{\star\})$
- $\llbracket\alpha_1 \times \alpha_2\rrbracket_\rho = \left(\mathcal{P}\llbracket\alpha_1\rrbracket_\rho \times \mathcal{P}\llbracket\alpha_2\rrbracket_\rho, \mathcal{A}\llbracket\alpha_1\rrbracket_\rho \times \mathcal{A}\llbracket\alpha_2\rrbracket_\rho\right)$
- $\llbracket\alpha \to \beta\rrbracket_\rho = \left(\left\{f : \forall x \in \mathcal{P}\llbracket\alpha\rrbracket_\rho . f(x) \in \mathcal{P}\llbracket\beta\rrbracket_\rho\right\}, \mathcal{A}\llbracket\alpha\rrbracket_\rho \to \mathcal{A}\llbracket\beta\rrbracket_\rho\right)$

Logical Relations Semantics for Types

We can view logical relations as a new type of denotational semantics. We begin with the types. For each type α, we're going to define *two sets*, P and A, such that $P \subseteq A$. The set P is our *predicate*.

Let ρ be a function that assigns a pair of sets $(\rho_{\mathcal{P}}(\tau), \rho_{\mathcal{A}}(\tau))$ to each ground type τ such that $\rho_{\mathcal{P}}(\tau) \subseteq \rho_{\mathcal{A}}(\tau)$, e.g., $\rho(\mathsf{Int}) = (\{2m : m \in \mathbb{Z}\}, \mathbb{Z})$.

Then to type α, we are going to assign a pair of sets $\{\!\![\alpha]\!\!\}_\rho = \left(\mathcal{P}\{\!\![\alpha]\!\!\}_\rho, \mathcal{A}\{\!\![\alpha]\!\!\}_\rho\right)$ as follows:

- $\{\!\![\tau]\!\!\}_\rho = (\rho_{\mathcal{P}}(\tau), \rho_{\mathcal{A}}(\tau))$
- $\{\!\![1]\!\!\}_\rho = (\{\star\}, \{\star\})$
- $\{\!\![\alpha_1 \times \alpha_2]\!\!\}_\rho = \left(\mathcal{P}\{\!\![\alpha_1]\!\!\}_\rho \times \mathcal{P}\{\!\![\alpha_2]\!\!\}_\rho, \mathcal{A}\{\!\![\alpha_1]\!\!\}_\rho \times \mathcal{A}\{\!\![\alpha_2]\!\!\}_\rho\right)$
- $\{\!\![\alpha \to \beta]\!\!\}_\rho = \left(\left\{f : \forall x \in \mathcal{P}\{\!\![\alpha]\!\!\}_\rho . f(x) \in \mathcal{P}\{\!\![\beta]\!\!\}_\rho\right\}, \mathcal{A}\{\!\![\alpha]\!\!\}_\rho \to \mathcal{A}\{\!\![\beta]\!\!\}_\rho\right)$

Note that $\mathcal{A}\{\!\![\alpha]\!\!\}_\rho = [\![\alpha]\!]_{\rho_{\mathcal{A}}}$.

Logical Relations Semantics for Signatures

Logical Relations Semantics for Signatures

For a fixed ρ assigning ground types to sets, we can give an interpretation σ to the constants and functions of a signature $\Sigma = (\Sigma_{\text{const}}, \Sigma_{\text{func}})$:

Logical Relations Semantics for Signatures

For a fixed ρ assigning ground types to sets, we can give an interpretation σ to the constants and functions of a signature $\Sigma = (\Sigma_{\mathsf{const}}, \Sigma_{\mathsf{func}})$:

- for each $c : \tau \in \Sigma_{\mathsf{const}}$, an element $\sigma(c) \in \rho_{\mathcal{P}}(\tau)$; and

Logical Relations Semantics for Signatures

For a fixed ρ assigning ground types to sets, we can give an interpretation σ to the constants and functions of a signature $\Sigma = (\Sigma_{\text{const}}, \Sigma_{\text{func}})$:

- for each $c : \tau \in \Sigma_{\text{const}}$, an element $\sigma(c) \in \rho_{\mathcal{P}}(\tau)$; and
- for each $f : (\tau_1, \dots, \tau_n) \to \tau \in \Sigma_{\text{func}}$, a function

$$\sigma(f) \in \rho_{\mathcal{A}}(\tau_1) \times \cdots \times \rho_{\mathcal{A}}(\tau_n) \to \rho_{\mathcal{A}}(\tau)$$

such that

$$(x_1, \dots, x_n) \in \rho_{\mathcal{P}}(\tau_1) \times \cdots \times \rho_{\mathcal{P}}(\tau_n) \Rightarrow \sigma(f)(x_1, \dots, x_n) \in \rho_{\mathcal{P}}(\tau).$$

Logical Relations Semantics for Signatures

For a fixed ρ assigning ground types to sets, we can give an interpretation σ to the constants and functions of a signature $\Sigma = (\Sigma_{\mathsf{const}}, \Sigma_{\mathsf{func}})$:

- for each $c : \tau \in \Sigma_{\mathsf{const}}$, an element $\sigma(c) \in \rho_{\mathcal{P}}(\tau)$; and
- for each $f : (\tau_1, \dots, \tau_n) \to \tau \in \Sigma_{\mathsf{func}}$, a function

$$\sigma(f) \in \rho_{\mathcal{A}}(\tau_1) \times \cdots \times \rho_{\mathcal{A}}(\tau_n) \to \rho_{\mathcal{A}}(\tau)$$

such that

$$(x_1, \dots, x_n) \in \rho_{\mathcal{P}}(\tau_1) \times \cdots \times \rho_{\mathcal{P}}(\tau_n) \Rightarrow \sigma(f)(x_1, \dots, x_n) \in \rho_{\mathcal{P}}(\tau).$$

Note that $\sigma(c) \in \mathcal{P}\{\!|\tau|\!\}_{\rho}$ and $\sigma(f) \in \mathcal{P}\{\!|\tau_1 \times \cdots \times \tau_n \to \tau|\!\}_{\rho}$, as well as that $\sigma(c) \in [\![\tau]\!]_{\rho_{\mathcal{A}}}$ and $\sigma(f) \in [\![\tau_1 \times \cdots \times \tau_n \to \tau]\!]_{\rho_{\mathcal{A}}}$.

Logical Relations Semantics for Signatures

For a fixed ρ assigning ground types to sets, we can give an interpretation σ to the constants and functions of a signature $\Sigma = (\Sigma_{\mathsf{const}}, \Sigma_{\mathsf{func}})$:

- for each $c : \tau \in \Sigma_{\mathsf{const}}$, an element $\sigma(c) \in \rho_{\mathcal{P}}(\tau)$; and
- for each $f : (\tau_1, \ldots, \tau_n) \to \tau \in \Sigma_{\mathsf{func}}$, a function

$$\sigma(f) \in \rho_{\mathcal{A}}(\tau_1) \times \cdots \times \rho_{\mathcal{A}}(\tau_n) \to \rho_{\mathcal{A}}(\tau)$$

  such that

$$(x_1, \ldots, x_n) \in \rho_{\mathcal{P}}(\tau_1) \times \cdots \times \rho_{\mathcal{P}}(\tau_n) \Rightarrow \sigma(f)(x_1, \ldots, x_n) \in \rho_{\mathcal{P}}(\tau).$$

Note that $\sigma(c) \in \mathcal{P}\{\!|\tau|\!\}_\rho$ and $\sigma(f) \in \mathcal{P}\{\!|\tau_1 \times \cdots \times \tau_n \to \tau|\!\}_\rho$, as well as that $\sigma(c) \in [\![\tau]\!]_{\rho_{\mathcal{A}}}$ and $\sigma(f) \in [\![\tau_1 \times \cdots \times \tau_n \to \tau]\!]_{\rho_{\mathcal{A}}}$.

If we want to forget that σ preserves our predicates, we will write $\sigma_{\mathcal{A}}$.

Logical Relations Semantics for Terms in Context

Logical Relations Semantics for Terms in Context

Fix a fixed ρ and σ, we want to define the meaning of a lambda term. In a context $\Gamma = x_1 : \alpha_1, \ldots, x_n : \alpha_n$, we want an interpretation of type

$$\{\!\!\{\Gamma \vdash M : \alpha\}\!\!\}_{\rho,\sigma} : \mathcal{A}\{\!\!\{\alpha_1 \times \cdots \times \alpha_n\}\!\!\}_\rho \to \mathcal{A}\{\!\!\{\alpha\}\!\!\}_\rho$$

such that for all $\gamma \in \mathcal{P}\{\!\!\{\alpha_1 \times \cdots \times \alpha_n\}\!\!\}_\rho$ we have $\{\!\!\{\Gamma \vdash M : \alpha\}\!\!\}_{\rho,\sigma}(\gamma) \in \mathcal{P}\{\!\!\{\alpha\}\!\!\}_\rho$. I.e., we map values satisfying our predicate to values satisfying our predicate.

How do we define this semantics?

Fundamental Theorem of Logical Relations

Recall that $\mathcal{A}\{\!\!\{\alpha\}\!\!\}_\rho = [\![\alpha]\!]_{\rho_\mathcal{A}}$. Thus, we can define

$$\{\!\!\{\Gamma \vdash M : \alpha\}\!\!\}_{\rho,\sigma} : \mathcal{A}\{\!\!\{\alpha_1 \times \cdots \times \alpha_n\}\!\!\}_\rho \to \mathcal{A}\{\!\!\{\alpha\}\!\!\}_\rho$$

as $\{\!\!\{\Gamma \vdash M : \alpha\}\!\!\}_{\rho,\sigma} := [\![\Gamma \vdash M : \alpha]\!]_{\rho_\mathcal{A},\sigma_\mathcal{A}}$ if it actually preserves our predicates.

Fundamental Theorem of Logical Relations

Recall that $\mathcal{A}\{\![\alpha]\!\}_\rho = [\![\alpha]\!]_{\rho_\mathcal{A}}$. Thus, we can define

$$\{\![\Gamma \vdash M : \alpha]\!\}_{\rho,\sigma} : \mathcal{A}\{\![\alpha_1 \times \cdots \times \alpha_n]\!\}_\rho \to \mathcal{A}\{\![\alpha]\!\}_\rho$$

as $\{\![\Gamma \vdash M : \alpha]\!\}_{\rho,\sigma} := [\![\Gamma \vdash M : \alpha]\!]_{\rho_\mathcal{A},\sigma_\mathcal{A}}$ if it actually preserves our predicates.

Theorem

Fix ρ and σ for logical relations. For all $\gamma \in \mathcal{P}\{\![\alpha_1 \times \cdots \times \alpha_n]\!\}_\rho$ we have $[\![\Gamma \vdash M : \alpha]\!]_{\rho_\mathcal{A},\sigma_\mathcal{A}}(\gamma) \in \mathcal{P}\{\![\alpha]\!\}_\rho$.

Proof.

Induction on the structure of M. □

Fundamental Theorem of Logical Relations

Recall that $\mathcal{A}\{\!\![\alpha]\!\!\}_\rho = [\![\alpha]\!]_{\rho_\mathcal{A}}$. Thus, we can define

$$\{\!\![\Gamma \vdash M : \alpha]\!\!\}_{\rho,\sigma} : \mathcal{A}\{\!\![\alpha_1 \times \cdots \times \alpha_n]\!\!\}_\rho \to \mathcal{A}\{\!\![\alpha]\!\!\}_\rho$$

as $\{\!\![\Gamma \vdash M : \alpha]\!\!\}_{\rho,\sigma} := [\![\Gamma \vdash M : \alpha]\!]_{\rho_\mathcal{A},\sigma_\mathcal{A}}$ if it actually preserves our predicates.

Theorem

Fix ρ and σ for logical relations. For all $\gamma \in \mathcal{P}\{\!\![\alpha_1 \times \cdots \times \alpha_n]\!\!\}_\rho$ we have $[\![\Gamma \vdash M : \alpha]\!]_{\rho_\mathcal{A},\sigma_\mathcal{A}}(\gamma) \in \mathcal{P}\{\!\![\alpha]\!\!\}_\rho$.

Proof.

Induction on the structure of M. □

This is known as the *Fundamental Theorem of Logical Relations*, or the *Basic Lemma of Logical Relations*. Note that we had to choose the interpretation σ of our constants and built-in functions to respect $\rho_\mathcal{P}$.

Example Application: Preserving Evenness

Example Application: Preserving Evenness

- Fix the ground types to be {Int}.

Example Application: Preserving Evenness

- Fix the ground types to be {Int}.
- Fix a signature $\Sigma = \left(\{\underline{n} : n \in \mathbb{Z}, n \text{ even}\}, \{\underline{+}, \underline{\times}\}\right)$, everything using Int.

Example Application: Preserving Evenness

- Fix the ground types to be $\{\mathsf{Int}\}$.
- Fix a signature $\Sigma = \left(\{\underline{n} : n \in \mathbb{Z},\ n \text{ even}\}, \{\underline{+}, \underline{\times}\}\right)$, everything using Int.
- Interpret our ground types with $\rho(\mathsf{Int}) = (\{2m : m \in \mathbb{Z}\}, \mathbb{Z})$.

Example Application: Preserving Evenness

- Fix the ground types to be {Int}.
- Fix a signature $\Sigma = \left(\{\underline{n} : n \in \mathbb{Z}, n \text{ even}\}, \{\underline{+}, \underline{\times}\}\right)$, everything using Int.
- Interpret our ground types with $\rho(\text{Int}) = (\{2m : m \in \mathbb{Z}\}, \mathbb{Z})$.
- Interpret our signature with $\sigma\left(\underline{n}\right) = n, \sigma\left(\underline{+}\right) = +, \sigma\left(\underline{\times}\right) = \times$ and check that our functions preserve even numbers.

Example Application: Preserving Evenness

- Fix the ground types to be {Int}.
- Fix a signature $\Sigma = (\{\underline{n} : n \in \mathbb{Z}, n \text{ even}\}, \{\underline{+}, \underline{\times}\})$, everything using Int.
- Interpret our ground types with $\rho(\mathsf{Int}) = (\{2m : m \in \mathbb{Z}\}, \mathbb{Z})$.
- Interpret our signature with $\sigma(\underline{n}) = n, \sigma(\underline{+}) = +, \sigma(\underline{\times}) = \times$ and check that our functions preserve even numbers.
- Apply the theorem! For example, for all terms $x : \mathsf{Int} \vdash M : \mathsf{Int}$, we have

$$n \in \mathcal{P}\{\!|\mathsf{Int}|\!\}_\rho \Rightarrow [\![x : \mathsf{Int} \vdash M : \mathsf{Int}]\!]_{\rho_{\mathcal{A}},\sigma}(n) \in \mathcal{P}\{\!|\mathsf{Int}|\!\}_\rho$$

which is equivalent to

$$n \text{ even} \Rightarrow [\![x : \mathsf{Int} \vdash M : \mathsf{Int}]\!]_{\rho_{\mathcal{A}},\sigma}(n) \text{ even.}$$

Example Application: Preserving Evenness

- Fix the ground types to be {Int}.
- Fix a signature $\Sigma = (\{\underline{n} : n \in \mathbb{Z}, n \text{ even}\}, \{\underline{+}, \underline{\times}\})$, everything using Int.
- Interpret our ground types with $\rho(\mathsf{Int}) = (\{2m : m \in \mathbb{Z}\}, \mathbb{Z})$.
- Interpret our signature with $\sigma(\underline{n}) = n, \sigma(\underline{+}) = +, \sigma(\underline{\times}) = \times$ and check that our functions preserve even numbers.
- Apply the theorem! For example, for all terms $x : \mathsf{Int} \vdash M : \mathsf{Int}$, we have

$$n \in \mathcal{P}\{\!\{\mathsf{Int}\}\!\}_\rho \Rightarrow [\![x : \mathsf{Int} \vdash M : \mathsf{Int}]\!]_{\rho_{\mathcal{A}},\sigma}(n) \in \mathcal{P}\{\!\{\mathsf{Int}\}\!\}_\rho$$

  which is equivalent to

$$n \text{ even} \Rightarrow [\![x : \mathsf{Int} \vdash M : \mathsf{Int}]\!]_{\rho_{\mathcal{A}},\sigma}(n) \text{ even}.$$

- Can also show that all polynomials with even coefficients preserve evenness.
- Importantly, the theorem also applies to contexts with function types. If we have $f : \mathsf{Int} \to \mathsf{Int}$, we are forced to feed in a function from $\mathcal{P}\{\!\{\mathsf{Int} \to \mathsf{Int}\}\!\}_\rho$, which are exactly even preserving functions!

Conclusion

Conclusion

- We saw the
  - syntax,
  - typing rules, and
  - set-theoretic denotational semantics

  of simply typed lambda calculus with products, ground types, constants, and built-in functions.

Conclusion

- We saw the
  - syntax,
  - typing rules, and
  - set-theoretic denotational semantics

  of simply typed lambda calculus with products, ground types, constants, and built-in functions.
- We then extended the denotational semantics to include a predicate at each type, and observed that all of our constructions respected these predicates.

Conclusion

- We saw the
  - syntax,
  - typing rules, and
  - set-theoretic denotational semantics

  of simply typed lambda calculus with products, ground types, constants, and built-in functions.
- We then extended the denotational semantics to include a predicate at each type, and observed that all of our constructions respected these predicates.
- The preservation gave use the Fundamental Theorem of Logical Relations, which we then applied to an example.

Conclusion

- We saw the
  - syntax,
  - typing rules, and
  - set-theoretic denotational semantics

  of simply typed lambda calculus with products, ground types, constants, and built-in functions.
- We then extended the denotational semantics to include a predicate at each type, and observed that all of our constructions respected these predicates.
- The preservation gave use the Fundamental Theorem of Logical Relations, which we then applied to an example.

References

Crole, Roy L. (1994). *Categories for Types*. Cambridge University Press.

Free Variables*

The function $\mathsf{FV}(M)$ is defined recursively as follows:

- $\mathsf{FV}(x) = \{x\}$
- $\mathsf{FV}(\langle\rangle) = \emptyset$
- $\mathsf{FV}(c) = \emptyset$
- $\mathsf{FV}(f(M_1, \ldots, M_n)) = \bigcup_{i=1}^{n} \mathsf{FV}(M_i)$

Free Variables*

The function $\mathsf{FV}(M)$ is defined recursively as follows:

- $\mathsf{FV}(x) = \{x\}$
- $\mathsf{FV}(\langle\rangle) = \varnothing$
- $\mathsf{FV}(c) = \varnothing$
- $\mathsf{FV}(f(M_1, \dots, M_n)) = \bigcup_{i=1}^{n} \mathsf{FV}(M_i)$
- $\mathsf{FV}(\lambda(x : \alpha).M) = \mathsf{FV}(M) \setminus \{x\}$

Free Variables*

The function $\mathsf{FV}(M)$ is defined recursively as follows:

- $\mathsf{FV}(x) = \{x\}$
- $\mathsf{FV}(\langle\rangle) = \varnothing$
- $\mathsf{FV}(c) = \varnothing$
- $\mathsf{FV}(f(M_1, \ldots, M_n)) = \bigcup_{i=1}^{n} \mathsf{FV}(M_i)$
- $\mathsf{FV}(\lambda(x : \alpha).M) = \mathsf{FV}(M) \setminus \{x\}$
- $\mathsf{FV}(M\,N) = \mathsf{FV}(M) \cup \mathsf{FV}(N)$
- $\mathsf{FV}(\langle M_1, M_2\rangle) = \mathsf{FV}(M_1) \cup \mathsf{FV}(M_2)$
- $\mathsf{FV}(\pi_1(M)) = \mathsf{FV}(M)$
- $\mathsf{FV}(\pi_2(M)) = \mathsf{FV}(M)$

Free Variables*

The function $\mathsf{FV}(M)$ is defined recursively as follows:

- $\mathsf{FV}(x) = \{x\}$
- $\mathsf{FV}(\langle\rangle) = \varnothing$
- $\mathsf{FV}(c) = \varnothing$
- $\mathsf{FV}(f(M_1, \dots, M_n)) = \bigcup_{i=1}^{n} \mathsf{FV}(M_i)$
- $\mathsf{FV}(\lambda(x : \alpha).M) = \mathsf{FV}(M) \setminus \{x\}$
- $\mathsf{FV}(M\,N) = \mathsf{FV}(M) \cup \mathsf{FV}(N)$
- $\mathsf{FV}(\langle M_1, M_2\rangle) = \mathsf{FV}(M_1) \cup \mathsf{FV}(M_2)$
- $\mathsf{FV}(\pi_1(M)) = \mathsf{FV}(M)$
- $\mathsf{FV}(\pi_2(M)) = \mathsf{FV}(M)$

For example, $\mathsf{FV}(\lambda(x : \alpha).y\,x) = \{y\}$.

Capture-avoiding Substitution*

Substitution of N for x in M in a capture-avoiding way, denoted $M[x := N]$, is defined recursively as follows:

- $x[x := N] = N$
- $y[x := N] = y$, for $y \neq x$

Capture-avoiding Substitution*

Substitution of N for x in M in a capture-avoiding way, denoted $M[x := N]$, is defined recursively as follows:

- $x[x := N] = N$
- $y[x := N] = y$, for $y \neq x$
- $\langle\rangle[x := N] = \langle\rangle$
- $c[x := N] = c$

Capture-avoiding Substitution*

Substitution of N for x in M in a capture-avoiding way, denoted $M[x := N]$, is defined recursively as follows:

- $x[x := N] = N$
- $y[x := N] = y$, for $y \neq x$
- $\langle\rangle[x := N] = \langle\rangle$
- $c[x := N] = c$
- $f(M_1, \dots, M_n)[x := N] = f(M_1[x := N], \dots, M_n[x := N]))$
- $(M\,P)[x := N] = (M[x := N])\,(P[x := N])$
- $\langle M_1, M_2\rangle[x := N] = \langle M_1[x := N], M_2[x := N]\rangle$
- $\pi_1(M)[x := N] = \pi_1(M[x := N])$
- $\pi_2(M)[x := N] = \pi_2(M[x := N])$
- ...

Capture-avoiding Substitution (continued)*

Most importantly, we have the rule for abstraction:

- $(\lambda(y : \alpha).M)[x := N] = \begin{cases} \lambda(y : \alpha).M[x := N] & \text{if } y \neq x \text{ and } y \notin \mathrm{FV}(N) \\ \lambda(z : \alpha).M[y := z][x := N] & \text{if } y = x \text{ or } y \in \mathrm{FV}(N) \end{cases}$

Capture-avoiding Substitution (continued)*

Most importantly, we have the rule for abstraction:

- $(\lambda(y : \alpha).M)[x := N] = \begin{cases} \lambda(y : \alpha).M[x := N] & \text{if } y \neq x \text{ and } y \notin \mathrm{FV}(N) \\ \lambda(z : \alpha).M[y := z][x := N] & \text{if } y = x \text{ or } y \in \mathrm{FV}(N) \end{cases}$

Here are two examples:

- $(\lambda(y : \alpha).x)[x := z] = \lambda(y : \alpha).z$
- $(\lambda(y : \alpha).y\,x)[x := y] = \lambda(z : \alpha).z\,y$

Equations-in-Context*

An equation-in-context is expressed as:

$$\Gamma \vdash M = N : \alpha$$

The judgments means that in the type context Γ, the terms M and N are considered equal and both have type α.

Equations-in-contexts allow us to perform equational reasoning while respecting the types assigned to the variables involved.

Equational Reasoning Rules*

$$\frac{\Gamma \vdash M : \alpha}{\Gamma \vdash M = M : \alpha} \text{ (Refl)}$$

$$\frac{\Gamma \vdash M = N : \alpha}{\Gamma \vdash N = M : \alpha} \text{ (Sym)}$$

$$\frac{\Gamma \vdash M = N : \alpha \quad \Gamma \vdash N = P : \alpha}{\Gamma \vdash M = P : \alpha} \text{ (Trans)}$$

Weakening and Substitution Rules*

$$\frac{\Gamma \vdash M = N : \alpha}{\Gamma, x : \beta \vdash M = N : \alpha} \text{ (Weak)}$$

$$\frac{\Gamma, x : \beta \vdash M = N : \alpha \quad \Gamma \vdash P : \beta}{\Gamma \vdash M[x := P] = N[x := P] : \alpha} \text{ (Subs)}$$

Rules for Unit and Binary Products*

$$\frac{\Gamma \vdash M : 1}{\Gamma \vdash M = \langle\rangle : 1} \text{ (Unit-Eq)}$$

$$\frac{\Gamma \vdash M_1 : \alpha_1 \quad \Gamma \vdash M_2 : \alpha_2}{\Gamma \vdash \pi_1(\langle M_1, M_2\rangle) = M_1 : \alpha_1} \text{ (Proj1)}$$

$$\frac{\Gamma \vdash M_1 : \alpha_1 \quad \Gamma \vdash M_2 : \alpha_2}{\Gamma \vdash \pi_2(\langle M_1, M_2\rangle) = M_2 : \alpha_2} \text{ (Proj2)}$$

$$\frac{\Gamma \vdash P : \alpha_1 \times \alpha_2}{\Gamma \vdash \langle \pi_1(P), \pi_2(P)\rangle = P : \alpha_1 \times \alpha_2} \text{ (η-Prod)}$$

Rules for Functions*

$$\frac{\Gamma, x : \alpha \vdash M : \beta \quad \Gamma \vdash N : \alpha}{\Gamma \vdash (\lambda(x : \alpha).M)\,N = M[x := N] : \beta} \ (\beta\text{-Eq})$$

$$\frac{x \notin \mathsf{FV}(M)}{\Gamma \vdash \lambda(x : \alpha).(M\,x) = M : \alpha \to \beta} \ (\eta\text{-Eq})$$

$$\frac{\Gamma, x : \alpha \vdash M = N : \beta}{\Gamma \vdash \lambda(x : \alpha).M = \lambda(x : \alpha).N : \alpha \to \beta} \ (\lambda\text{-Cong})$$

Axioms

We also want axioms in order to reason about elements of our signature Σ. An axiom is a pair of terms $(\Gamma \vdash M : \alpha, \Gamma \vdash N : \alpha)$ of terms of the same type in the same context. For a set of axioms Ω, we have the rule

$$\frac{(\Gamma \vdash M : \alpha, \Gamma \vdash N : \alpha) \in \Omega}{\Gamma \vdash M = N : \alpha} \text{ (Axiom)}$$

Note that it is possible to prove everything equals everything else if you choose your axioms wrong!

Soundness

Theorem

Let Ω be a set of axioms in a signature Σ. Let ρ and σ be assignments such that, for all $(\Gamma \vdash M : \alpha, \Gamma \vdash N : \alpha) \in \Omega$, we have $[\![\Gamma \vdash M : \alpha]\!]_{\rho,\sigma} = [\![\Gamma \vdash N : \alpha]\!]_{\rho,\sigma}$. Then, for all valid equations $\Gamma \vdash M = N : \alpha$ we have

$$[\![\Gamma \vdash M : \alpha]\!]_{\rho,\sigma} = [\![\Gamma \vdash N : \alpha]\!]_{\rho,\sigma}$$

Proof.

By induction on the proof of $\Gamma \vdash M = N : \alpha$. □

Thus, if we respect the axioms, then equivalent terms have equal denotational semantics. This is the minimum we expect from denotational semantics.