# Mathematical Background[1]

[1] Slides modified from the notes created by Xinyu Feng

# Outline

# Outline

# Sets – Basic Notations

| | |
|---|---|
| $x \in S$ | membership |
| $S \subseteq T$ | subset |
| $S \subset T$ | proper subset |
| $S \subseteq^{\text{fin}} T$ | finite subset |
| $S = T$ | equivalence |
| $\emptyset$ | the empty set |
| $\mathbf{N}$ | natural numbers |
| $\mathbf{Z}$ | integers |
| $\mathbf{B}$ | $\{\mathbf{true}, \mathbf{false}\}$ |

# Sets – Basic Notations

| | | |
|---|---|---|
| $S \cap T$ | intersection | $\stackrel{\text{def}}{=} \{x \mid x \in S \text{ and } x \in T\}$ |
| $S \cup T$ | union | $\stackrel{\text{def}}{=} \{x \mid x \in S \text{ or } x \in T\}$ |
| $S - T$ | difference | $\stackrel{\text{def}}{=} \{x \mid x \in S \text{ and } x \notin T\}$ |
| $\mathcal{P}(S)$ | powerset | $\stackrel{\text{def}}{=} \{T \mid T \subseteq S\}$ |
| $[m, n]$ | integer range | $\stackrel{\text{def}}{=} \{x \mid m \leq x \leq n\}$ |

# Generalized Unions of Sets

$$\bigcup \mathcal{S} \stackrel{\text{def}}{=} \{x \mid \exists T \in \mathcal{S}.\, x \in T\}$$

$$\bigcup_{i \in I} S(i) \stackrel{\text{def}}{=} \bigcup \{S(i) \mid i \in I\}$$

$$\bigcup_{i=m}^{n} S(i) \stackrel{\text{def}}{=} \bigcup_{i \in [m,n]} S(i)$$

Here $\mathcal{S}$ is a set of sets. $S(i)$ is a set whose definition depends on $i$. For instance, we may have

$$S(i) = \{x \mid x > i + 3\}$$

Given $i = 1, 2, \ldots, n$, we know the corresponding $S(i)$.

# Generalized Unions of Sets

### Example (1)

$$A \cup B = \bigcup \{A, B\}$$

Proof?

### Example (2)

Let $S(i) = [i, i+1]$ and $I = \{j^2 \mid j \in [1,3]\}$, then

$$\bigcup_{i \in I} S(i) = \{1, 2, 4, 5, 9, 10\}$$

# Generalized Intersections of Sets

$$\begin{aligned}
\bigcap \mathcal{S} &\stackrel{\text{def}}{=} \{x \mid \forall T \in \mathcal{S}.\, x \in T\} \\
\bigcap_{i \in I} S(i) &\stackrel{\text{def}}{=} \bigcap \{S(i) \mid i \in I\} \\
\bigcap_{i=m}^{n} S(i) &\stackrel{\text{def}}{=} \bigcap_{i \in [m,n]} S(i)
\end{aligned}$$

# Generalized Unions and Intersections of Empty Sets

From

$$\bigcup \mathcal{S} \stackrel{\text{def}}{=} \{x \mid \exists T \in \mathcal{S}.\, x \in T\}$$

$$\bigcap \mathcal{S} \stackrel{\text{def}}{=} \{x \mid \forall T \in \mathcal{S}.\, x \in T\}$$

we know

$$\bigcup \emptyset = \emptyset$$

$$\bigcap \emptyset \quad \text{meaningless}$$

$\bigcap \emptyset$ is meaningless, since it denotes the paradoxical "set of everything" (see Russell's paradox).

# Outline

# RELATIONS

We need to first define the *Cartesian product* of two sets $A$ and $B$:
$A \times B = \{(x, y) \mid x \in A \text{ and } y \in B\}$
Here $(x, y)$ is called a *pair*.

Projections over pairs:
$\pi_0(x, y) = x$ and $\pi_1(x, y) = y$.

Then, $\rho$ is a *relation from A to B* if $\rho \subseteq A \times B$.
Or, written as $\rho \in \mathcal{P}(A \times B)$.

# RELATIONS

$\rho$ is a *relation from A to B* if $\rho \subseteq A \times B$, or $\rho \in \mathcal{P}(A \times B)$.

$\rho$ is a *relation on S* if $\rho \subseteq S \times S$.

We say *$\rho$ relates $x$ and $y$* if $(x, y) \in \rho$. Sometimes we write it as $x \, \rho \, y$.

$\rho$ is an *identity relation* if $\forall (x, y) \in \rho.\ x = y$.

# Relations – Basic Notations

the *identity on* $S$ $\qquad \mathrm{Id}_S \stackrel{\text{def}}{=} \{(x,x) \mid x \in S\}$

the *domain* of $\rho$ $\qquad \mathrm{dom}(\rho) \stackrel{\text{def}}{=} \{x \mid \exists y.\,(x,y) \in \rho\}$

the *range* of $\rho$ $\qquad \mathrm{ran}(\rho) \stackrel{\text{def}}{=} \{y \mid \exists x.\,(x,y) \in \rho\}$

*composition* of $\rho$ and $\rho'$ $\qquad \rho' \circ \rho \stackrel{\text{def}}{=}$

$$\{(x,z) \mid \exists y.\,(x,y) \in \rho \wedge (y,z) \in \rho'\}$$

*inverse* of $\rho$ $\qquad \rho^{-1} \stackrel{\text{def}}{=} \{(y,x) \mid (x,y) \in \rho\}$

# Relations – Properties and Examples

$$(\rho_3 \circ \rho_2) \circ \rho_1 = \rho_3 \circ (\rho_2 \circ \rho_1)$$

$$\rho \circ \mathbf{Id}_S = \rho = \mathbf{Id}_T \circ \rho, \quad \text{if } \rho \subseteq S \times T$$

$$\mathbf{dom}(\mathbf{Id}_S) = S = \mathbf{ran}(\mathbf{Id}_S)$$

$$\mathbf{Id}_T \circ \mathbf{Id}_S = \mathbf{Id}_{T \cap S}$$

$$\mathbf{Id}_S{}^{-1} = \mathbf{Id}_S$$

$$\left(\rho^{-1}\right)^{-1} = \rho$$

$$(\rho_2 \circ \rho_1)^{-1} = \rho_1{}^{-1} \circ \rho_2{}^{-1}$$

$$\rho \circ \emptyset = \emptyset = \emptyset \circ \rho$$

$$\mathbf{Id}_\emptyset = \emptyset = \emptyset^{-1}$$

$$\mathbf{dom}(\rho) = \emptyset \iff \rho = \emptyset$$

# Relations – Properties and Examples

$$
\begin{aligned}
< \;&\subseteq\; \leq \\
< \cup \mathrm{Id}_{\mathbf{N}} \;&=\; \leq \\
\leq \cap \geq \;&=\; \mathrm{Id}_{\mathbf{N}} \\
< \cap \geq \;&=\; \emptyset \\
< \circ \leq \;&=\; < \\
\leq \circ \leq \;&=\; \leq \\
\geq \;&=\; \leq^{-1}
\end{aligned}
$$

# Equivalence Relations

$\rho$ is an *equivalence relation* on $S$ if it is reflexive, symmetric and transitive.

Reflexivity: $\mathrm{Id}_S \subseteq \rho$

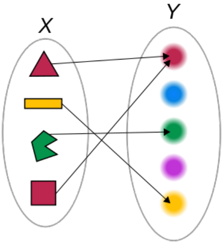
Symmetry: $\rho^{-1} = \rho$

Transitivity: $\rho \circ \rho \subseteq \rho$

# Outline

# FUNCTIONS

A function $f$ from $A$ to $B$ is a special relation from $A$ to $B$. A relation $\rho$ is a function if, for all $x$, $y$ and $y'$, $(x, y) \in \rho$ and $(x, y') \in \rho$ imply $y = y'$.




Function application $f(x)$ can also be written as $f\ x$.

# Functions

$\emptyset$ and $\mathrm{Id}_S$ are functions.

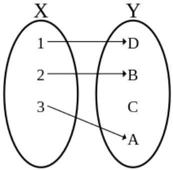
If $f$ and $g$ are functions, then $g \circ f$ is a function.

$$(g \circ f)\, x = g(f\, x)$$

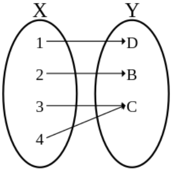
If $f$ is a function, $f^{-1}$ is *not* necessarily a function. ($f^{-1}$ is a function if $f$ is an injection.)

# Functions – Injection, Surjection and Bijection

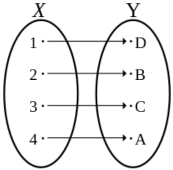
Injective and non-surjective:



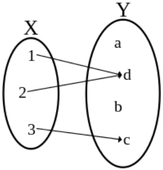

Surjective and non-injective:




Bijective:




Non-injective and non-surjective:



# Functions – Denoted by Typed Lambda Expressions

$\lambda x \in S.\, E$ denotes the function $f$ with domain $S$ such that $f(x) = E$ for all $x \in S$.

**Example**

$\lambda x \in \mathbf{N}.\, x + 3$ denotes the function $\{(x, x + 3) \mid x \in \mathbf{N}\}$.

# Functions – Variation

Variation of a function at a single argument:

$$f\{x \rightsquigarrow n\} \stackrel{\text{def}}{=} \lambda z.\begin{cases} f\,z & \text{if } z \neq x \\ n & \text{if } z = x \end{cases}$$

Note that $x$ does not have to be in $\text{dom}(f)$.

$$\text{dom}(f\{x \rightsquigarrow n\}) = \text{dom}(f) \cup \{x\}$$
$$\text{ran}(f\{x \rightsquigarrow n\}) \quad = \text{ran}(f - \{(x, n') \mid (x, n') \in f\}) \cup \{n\}$$

**Example**

$(\lambda x \in [0..2].\, x + 1)\{2 \rightsquigarrow 7\} = \{(0, 1), (1, 2), (2, 7)\}$
$(\lambda x \in [0..1].\, x + 1)\{2 \rightsquigarrow 7\} = \{(0, 1), (1, 2), (2, 7)\}$

# Function Types

We use $A \to B$ to represent the set of all functions from $A$ to $B$.

$\to$ is right associative. That is,

$$A \to B \to C \; = A \to (B \to C)\,.$$

If $f \in A \to B \to C$, $a \in A$ and $b \in B$, then $f\,a\,b = (f(a))b \in C$.

# Functions with multiple arguments

$$f \in A_1 \times A_2 \times \cdots \times A_n \to A$$

$$f = \lambda x \in A_1 \times A_2 \times \cdots \times A_n.\, E$$

$$f\,(a_1, a_2, \ldots, a_n)$$

*Currying* it gives us a function

$$g \in A_1 \to A_2 \to \cdots \to A_n \to A$$

$$g = \lambda x_1 \in A_1.\, \lambda x_2 \in A_2.\, \ldots \lambda x_n \in A_n.\, E$$

$$g\, a_1\, a_2\, \ldots\, a_n$$

# Outline

# Cartesian Products

Recall $A \times B = \{(x, y) \mid x \in A \text{ and } y \in B\}$.
Projections over pairs: $\pi_0(x, y) = x$ and $\pi_1(x, y) = y$.

Generalize to $n$ sets:
$S_0 \times S_1 \times \cdots \times S_{n-1} = \{(x_0, \ldots, x_{n-1}) \mid \forall i \in [0, n-1].\, x_i \in S_i\}$
We say $(x_0, \ldots, x_{n-1})$ is an *n-tuple*.

Then we have $\pi_i(x_0, \ldots, x_{n-1}) = x_i$.

# Tuples as Functions

We can view a pair $(x, y)$ as a function

$$\lambda i \in \mathbf{2}.\begin{cases} x & \text{if } i = 0 \\ y & \text{if } i = 1 \end{cases}$$

where $\mathbf{2} = \{0, 1\}$.

$$A \times B \stackrel{\text{def}}{=} \{f \mid \text{dom}(f) = \mathbf{2},\ \text{and } f\,0 \in A \text{ and } f\,1 \in B\}$$

(We re-define $A \times B$ in this way, in order to generalize $\times$ later. Functions and relations are still defined based on the old definitions of $\times$.)

# Tuples as Functions

Similarly, we can view an $n$-tuple $(x_0, \ldots, x_{n-1})$ as a function

$$\lambda i \in \mathbf{n}. \begin{cases} x_0 & \text{if } i = 0 \\ \ldots & \ldots \\ x_{n-1} & \text{if } i = n-1 \end{cases}$$

where $\mathbf{n} = \{0, 1, \ldots, n-1\}$.

$$S_0 \times \cdots \times S_{n-1} \stackrel{\text{def}}{=} \{f \mid \mathrm{dom}(f) = \mathbf{n},\ \text{and}\ \forall i \in \mathbf{n}.\, f\, i \in S_i\}$$

# Generalized Products

From

$$S_0 \times \cdots \times S_{n-1} \stackrel{\text{def}}{=} \{f \mid \text{dom}(f) = \mathbf{n}, \text{ and } \forall i \in \mathbf{n}.\, f\, i \in S_i\}$$

we can generalize $S_0 \times \cdots \times S_{n-1}$ to an infinite number of sets.

$$\prod_{i \in I} S(i) \stackrel{\text{def}}{=} \{f \mid \text{dom}(f) = I, \text{ and } \forall i \in I.\, f\, i \in S(i)\}$$

$$\prod_{i=m}^{n} S(i) \stackrel{\text{def}}{=} \prod_{i \in [m,n]} S(i)$$

# GENERALIZED PRODUCTS

Let $\theta$ is a function from a set of indices to a set of sets, i.e., $\theta$ is an indexed family of sets. We can define $\Pi\,\theta$ as follows.

$$\Pi\,\theta \stackrel{\text{def}}{=} \{f \mid \mathrm{dom}(f) = \mathrm{dom}(\theta),\ \text{and}\ \forall i \in \mathrm{dom}(\theta).\, f\,i \in \theta\,i\}$$

### EXAMPLE

Let $\theta = \lambda i \in I.\, S(i)$. Then

$$\Pi\,\theta = \prod_{i \in I} S(i)$$

# Generalized Products – Examples

$$\Pi\,\theta \;\stackrel{\text{def}}{=}\; \{f \mid \mathrm{dom}(f) = \mathrm{dom}(\theta),\ \text{and}\ \forall i \in \mathrm{dom}(\theta).\, f\,i \in \theta\,i\}$$

**Example (1)**

Let $\theta = \lambda i \in \mathbf{2}.\mathbf{B}$. Then

$$\begin{aligned}\Pi\,\theta = \{\ & \{(0, \mathbf{true}), (1, \mathbf{true})\},\\ & \{(0, \mathbf{true}), (1, \mathbf{false})\},\\ & \{(0, \mathbf{false}), (1, \mathbf{true})\},\\ & \{(0, \mathbf{false}), (1, \mathbf{false})\}\ \}\end{aligned}$$

That is, $\Pi\,\theta = \mathbf{B} \times \mathbf{B}$.

(Here $\mathbf{B} \times \mathbf{B}$ uses the new definition of $\times$. If we use its old definition, we will see an elegant correspondence between $\Pi\,\theta$ and $\mathbf{B} \times \mathbf{B}$.)

# Generalized Products – Examples

$$\Pi\,\theta \;\stackrel{\text{def}}{=}\; \{f \mid \mathrm{dom}(f) = \mathrm{dom}(\theta),\ \text{and } \forall i \in \mathrm{dom}(\theta).\, f\, i \in \theta\, i\}$$

**Example (2)**

$\Pi\,\emptyset \;=\; \{\emptyset\}$.

**Example (3)**

If $\exists i \in \mathrm{dom}(\theta).\ \theta\, i = \emptyset$, then $\Pi\,\theta \;=\; \emptyset$.

# Exponentiation

Recall $\prod_{x \in T} S(x) = \Pi \lambda x \in T.\, S(x)$.

We write $S^T$ for $\prod_{x \in T} S$ if $S$ is independent of $x$.

$$\begin{aligned} S^T \;&=\; \prod_{x \in T} S \;=\; \Pi \lambda x \in T.\, S \\ &=\; \{f \mid \mathrm{dom}(f) = T, \text{ and } \forall x \in T.\, f\,x \in S\} \;=\; (T \to S) \end{aligned}$$

Recall that $T \to S$ is the set of all functions from T to S.

# Exponentiation – Example

We sometimes use $\mathbf{2}^S$ for powerset $\mathcal{P}(S)$. Why?

# Exponentiation – Example

We sometimes use $\mathbf{2}^S$ for powerset $\mathcal{P}(S)$. Why?

$$\mathbf{2}^S \;=\; (S \to \mathbf{2})$$

For any subset $T$ of $S$, we can define

$$f \;=\; \lambda x \in S.\begin{cases} 1 & \text{if } x \in T \\ 0 & \text{if } x \in S - T \end{cases}$$

Then $f \in (S \to \mathbf{2})$.

On the other hand, for any $f \in (S \to \mathbf{2})$, we can construct a subset of $S$.

# Outline

# Sums (or Disjoint Unions)

**Example**

Let $A = \{1, 2, 3\}$ and $B = \{2, 3\}$.
To define the disjoint union of $A$ and $B$, we need to index the elements according to which set they originated in:

$$\begin{aligned} A' &= \{(0,1), (0,2), (0,3)\} \\ B' &= \{(1,2), (1,3)\} \end{aligned}$$

$$A + B = A' \cup B'$$

# Sums (or Disjoint Unions)

$$A + B \stackrel{\text{def}}{=} \{(i, x) \mid i = 0 \text{ and } x \in A, \text{ or } i = 1 \text{ and } x \in B\}$$

Injection operations:

$$\iota^0_{A+B} \in A \to A + B$$
$$\iota^1_{A+B} \in B \to A + B$$

The sum can be generalized to $n$ sets:

$$S_0 + S_1 + \cdots + S_{n-1} \stackrel{\text{def}}{=} \{(i, x) \mid i \in \mathbf{n} \text{ and } x \in S_i\}$$

# Generalized Sums (or Disjoint Unions)

It can also be generalized to an infinite number of sets.

$$\sum_{i \in I} S(i) \stackrel{\text{def}}{=} \{(i, x) \mid i \in I \text{ and } x \in S(i)\}$$

$$\sum_{i=m}^{n} S(i) \stackrel{\text{def}}{=} \sum_{i \in [m,n]} S(i)$$

The sum of $\theta$ is

$$\Sigma\, \theta \stackrel{\text{def}}{=} \{(i, x) \mid i \in \mathrm{dom}(\theta) \text{ and } x \in \theta\, i\}$$

So

$$\sum_{i \in I} S(i) = \Sigma \lambda i \in I . S(i)$$

# Generalized Sums (or Disjoint Unions) – Examples

$$\Sigma\,\theta \stackrel{\text{def}}{=} \{(i,x) \mid i \in \mathrm{dom}(\theta) \text{ and } x \in \theta\, i\}$$

### Example (1)

$$\sum_{i\in\mathbf{n}} S(i) \;=\; \Sigma\lambda i \in \mathbf{n}.S(i) \;=\; \{(i,x) \mid i \in \mathbf{n} \text{ and } x \in S(i)\}$$

### Example (2)

Let $\theta = \lambda i \in \mathbf{2}.\mathbf{B}$. Then

$$\Sigma\,\theta = \{\ (0,\mathbf{true}),(0,\mathbf{false}),(1,\mathbf{true}),(1,\mathbf{false})\ \}$$

That is, $\Sigma\,\theta = \mathbf{2} \times \mathbf{B}$.

# Generalized Sums (or Disjoint Unions) – Examples

$$\Sigma\,\theta \stackrel{\text{def}}{=} \{(i,x) \mid i \in \mathrm{dom}(\theta) \text{ and } x \in \theta\, i\}$$

**Example (3)**

$\Sigma\,\emptyset \;=\; \emptyset.$

**Example (4)**

If $\forall i \in \mathrm{dom}(\theta).\ \theta\, i = \emptyset$, then $\Sigma\,\theta \;=\; \emptyset.$

**Example (5)**

Let $\theta \;=\; \lambda i \in \mathbf{2}.\ \begin{cases} \mathbf{B} & \text{if } i = 0 \\ \emptyset & \text{if } i = 1 \end{cases}$,
then $\Sigma\,\theta \;=\; \{(0,\mathbf{true}),(0,\mathbf{false})\}.$

# More on Generalized Sums (or Disjoint Unions)

$$\Sigma\,\theta \stackrel{\text{def}}{=} \{(i, x) \mid i \in \operatorname{dom}(\theta) \text{ and } x \in \theta\, i\}$$

$$\sum_{x \in T} S(x) \stackrel{\text{def}}{=} \Sigma\lambda x \in T.S(x)$$

We can prove $\sum_{x \in T} S = T \times S$ if $S$ is independent of $x$.

$$\begin{aligned}\sum_{x \in T} S &= \Sigma\lambda x \in T.\,S \\ &= \{(x, y) \mid x \in T \text{ and } y \in S\} = (T \times S)\end{aligned}$$

# Outline

# Predicate logic

Predicate logic over integer expressions is a language of logical assertions, e.g.,

$$\forall \mathsf{x}.\, \mathsf{x} + 0 = \mathsf{x}$$

We introduce 4 concepts that pervade the study of PL:

1. Abstract syntax
2. Denotational semantics
3. Inference rules
4. Binding

By using them to describe predicate logic.

# Abstract syntax

Describes the *structure* of a phrase,
ignoring the details of its *presentation*.

An abstract grammar for predicate logic over integer arithmetic:

$$
\begin{array}{rcl}
\langle intexp \rangle & ::= & 0 \mid 1 \mid 2 \mid \ldots \\
& \mid & \langle var \rangle \mid -\langle intexp \rangle \mid \langle intexp \rangle + \langle intexp \rangle \mid \ldots \\
\langle assert \rangle & ::= & \mathbf{true} \mid \mathbf{false} \\
& \mid & \langle intexp \rangle = \langle intexp \rangle \mid \langle intexp \rangle > \langle intexp \rangle \mid \ldots \\
& \mid & \neg\langle assert \rangle \mid \langle assert \rangle \vee \langle assert \rangle \mid \langle assert \rangle \Rightarrow \langle assert \rangle \\
& \mid & \forall\langle var \rangle.\, \langle assert \rangle \\
& \mid & \ldots
\end{array}
$$

# Resolving ambiguity

- Parenthesizing each production.

  E.g., $\forall(\mathsf{x}).\,(((\mathsf{x}) + (0)) = (\mathsf{x}))$.
- Conventions about precedence.

  E.g., $\forall \mathsf{x}.\,(\mathsf{x} + 0 = \mathsf{x})$.

  Precedence list: $(\times, \div, \%)(+\ -)(=\ <\ \ \ldots)\neg \wedge \vee \Rightarrow \Leftrightarrow$.
- Extend quantified term to a stopping symbol.

  E.g., $\forall x.\, x + 0 = x \wedge \forall y.\, x + y = x + y$.
  - closing delimiters: ) ] } $:\,|\ \ \ldots$
  - other stopping symbols: ; $\rightarrow\ \ \Box\ \ \ldots$

# Carriers and Constructors

- Carriers: sets of abstract phrases (e.g. $\langle intexp \rangle$)
- Constructors: specify abstract grammar productions

$$
\begin{array}{lcl}
\langle intexp \rangle ::= 0 & \longrightarrow & c_0 \in \{\langle\rangle\} \to \langle intexp \rangle \\
\langle intexp \rangle ::= \langle intexp \rangle + \langle intexp \rangle & \longrightarrow & c_+ \in \langle intexp \rangle \times \langle intexp \rangle \to \langle intexp \rangle
\end{array}
$$

Note: independent of the concrete pattern of the production.

$$\langle intexp \rangle ::= \mathbf{plus}\ \langle intexp \rangle \langle intexp \rangle \quad \longrightarrow \quad c_+ \in \langle intexp \rangle \times \langle intexp \rangle \to \langle intexp \rangle$$

- Constructors must be injective and have disjoint ranges
- Carriers must be predefined or their elements must be constructible in finitely many constructor applications

# Inductive Structure of Carrier Sets

Carriers can be defined inductively:

$$
\begin{array}{rcl}
\langle intexp \rangle^{(0)} & = & \emptyset \\
\langle assert \rangle^{(0)} & = & \emptyset \\
\langle intexp \rangle^{(j+1)} & = & \{c_0(), c_1() \ldots\} \\
 & & \cup \{c_+(x_0, x_1) \mid x_0, x_1 \in \langle intexp \rangle^{(j)}\} \cup \ldots \\
\langle assert \rangle^{(j+1)} & = & \{c_{\mathbf{true}}(), c_{\mathbf{false}}()\} \\
 & & \cup \{c_=(x_0, x_1) \mid x_0, x_1 \in \langle intexp \rangle^{(j)}\} \\
 & & \cup \{c_\forall(x_0, x_1) \mid x_0 \in \langle var \rangle, x_1 \in \langle assert \rangle^{(j)}\} \cup \ldots \\
\langle intexp \rangle & = & \bigcup_{j=0}^{\infty} \langle intexp \rangle^{(j)} \\
\langle assert \rangle & = & \bigcup_{j=0}^{\infty} \langle assert \rangle^{(j)}
\end{array}
$$

Intuitively, these sets are constructed inductively over the depth of each carrier.

# Denotational Semantics of Predicate Logic

The meaning of a term $e \in \langle intexp \rangle$ is denoted by $[\![e]\!]_{intexp}$
that is, the function $[\![\cdot]\!]_{intexp}$ maps $\langle intexp \rangle$ to their meanings

# Denotational Semantics of Predicate Logic

The meaning of a term $e \in \langle intexp \rangle$ is denoted by $[\![e]\!]_{intexp}$
that is, the function $[\![\cdot]\!]_{intexp}$ maps $\langle intexp \rangle$ to their meanings

*What is the set of meanings?*

# Denotational Semantics of Predicate Logic

The meaning of a term $e \in \langle intexp \rangle$ is denoted by $[\![e]\!]_{intexp}$
that is, the function $[\![\cdot]\!]_{intexp}$ maps $\langle intexp \rangle$ to their meanings

*What is the set of meanings?*

$\langle intexp \rangle$ have integer values and $\langle assert \rangle$ have boolean values
E.g., $[\![5 + 37]\!]_{intexp}$ could be the integer 42.

# Denotational Semantics of Predicate Logic

The meaning of a term $e \in \langle \mathit{intexp} \rangle$ is denoted by $[\![e]\!]_{intexp}$
that is, the function $[\![\cdot]\!]_{intexp}$ maps $\langle \mathit{intexp} \rangle$ to their meanings

*What is the set of meanings?*

$\langle \mathit{intexp} \rangle$ have integer values and $\langle \mathit{assert} \rangle$ have boolean values
E.g., $[\![5 + 37]\!]_{intexp}$ could be the integer 42.

However, value of a term can depend on its *free variables*.
E.g., $x + 37$ contains the free variable $x$, its value depends on a *state* (or environment / variable assignment)

# States and Denotation

A *state* maps each variable into its integer value

$$\sigma \in \Sigma \stackrel{\text{def}}{=} \langle \mathit{var} \rangle \to \mathbf{Z}$$

to give meaning to free variables.

The meaning or denotation of a term is a function from the state to **Z** or **B**.

$$\begin{array}{rcl} [\![\cdot]\!]_{\mathit{intexp}} & \in & \langle \mathit{intexp} \rangle \to \Sigma \to \mathbf{Z} \\ [\![\cdot]\!]_{\mathit{assert}} & \in & \langle \mathit{assert} \rangle \to \Sigma \to \mathbf{B} \end{array}$$

E.g., if $\sigma = \{x \rightsquigarrow 3, y \rightsquigarrow 4\}$ then $[\![x + 5]\!]_{\mathit{intexp}}\sigma = 8$, and $[\![\exists z.\, x < z \land z < y]\!]\sigma = \mathbf{false}$

# Semantic Equations for Predicate Logic

$$
\begin{aligned}
[\![0]\!]_{intexp}\sigma &= 0 \\
[\![v]\!]_{intexp}\sigma &= \sigma v \\
[\![e_0 + e_1]\!]_{intexp}\sigma &= [\![e_0]\!]_{intexp}\sigma + [\![e_1]\!]_{intexp}\sigma \\
[\![\mathbf{true}]\!]_{assert}\sigma &= \mathbf{true} \\
[\![e_0 = e_1]\!]_{assert}\sigma &= [\![e_0]\!]_{intexp}\sigma = [\![e_1]\!]_{intexp}\sigma \\
[\![\forall v.\, p]\!]_{assert}\sigma &= \forall n \in \mathbf{Z}.\, [\![p]\!]_{assert}\sigma\{v \rightsquigarrow n\} \\
\ldots
\end{aligned}
$$

# Example

$$
\begin{aligned}
& [\![\forall x.\, x + 0 = x]\!]_{assert}\sigma \\
=& \forall n \in \mathbf{Z}.\, [\![x + 0 = x]\!]_{assert}\sigma\{x \leadsto n\} \\
=& \forall n \in \mathbf{Z}.\, [\![x + 0]\!]_{assert}\sigma\{x \leadsto n\} = [\![x]\!]_{assert}\sigma\{x \leadsto n\} \\
=& \forall n \in \mathbf{Z}.\, [\![x + 0]\!]_{assert}\sigma\{x \leadsto n\} = n \\
=& \forall n \in \mathbf{Z}.\, [\![x]\!]_{assert}\sigma\{x \leadsto n\} + [\![0]\!]_{assert}\sigma\{x \leadsto n\} = n \\
=& \forall n \in \mathbf{Z}.\, n + 0 = n \\
=& \mathbf{true}
\end{aligned}
$$

# Properties of the Semantic Equations

- *Syntax-directed*
  - exactly 1 equation for each constructor
  - result expressed using meanings of its immediate subterms only
  - syntax-directed $\wedge$ abstract phrases
    $\Rightarrow$ have unique solution ($[\![\cdot]\!]_{intexp}$ and $[\![\cdot]\!]_{assert}$)
- Define *compositional* semantic functions

  depend only on the *meanings* of subterms, irrelevant to any other properties.
  $\Rightarrow$ subterms can be substituted by equivalent terms

# TERMINOLOGIES FOR $[\![p]\!]_{assert}\sigma = \textbf{true}$

- $[\![p]\!]_{assert}\sigma = \textbf{true}$
  $p$ *is true* in / *holds* for / *describes* $\sigma$, or $\sigma$ *satisfies* $p$.
- $\forall\sigma \in \Sigma.\, p$ holds for $\sigma$
  $p$ is *valid*.
- $\neg p$ is valid
  $p$ is *unsatisfiable*.
- $p \Rightarrow p'$ is valid
  $p$ is *stronger* than $p'$; $p'$ is *weaker* than $p$.
- $p$ is stronger and weaker than $p'$
  $p$ and $p'$ are *equivalent*.

# Inference Rules

$$\textsc{Rule name}\quad \frac{\textit{premises}}{\textit{conclusion}}$$

- Premises and conclusion may contain *metavariables*, each range over some type of phrase.

$$\textsc{Axiom}\quad \mathsf{x} + \mathsf{0} = \mathsf{x} \qquad \textsc{Axiom Schema}\quad \frac{}{e_1 = e_0 \Rightarrow e_0 = e_1} \qquad \textsc{Rule}\quad \frac{p_0 \qquad p_0 \Rightarrow p_1}{p_1} \qquad \textsc{Rule}'\quad \frac{p}{\forall v.\, p}$$

# Inference Rules

$$\textsc{xPlusZero}\quad \overline{\mathsf{x} + \mathsf{0} = \mathsf{x}}$$

$$\textsc{SymmObjEq}\quad \frac{}{\mathsf{x} + \mathsf{0} = \mathsf{x} \Rightarrow \mathsf{x} = \mathsf{x} + \mathsf{0}}$$

$$\textsc{ModusPonens}\quad \frac{\mathsf{x} + \mathsf{0} = \mathsf{x} \qquad \mathsf{x} + \mathsf{0} = \mathsf{x} \Rightarrow \mathsf{x} = \mathsf{x} + \mathsf{0}}{\mathsf{x} = \mathsf{x} + \mathsf{0}}$$

$$\textsc{Generalization}\quad \frac{\mathsf{x} = \mathsf{x} + \mathsf{0}}{\forall \mathsf{x}.\, \mathsf{x} = \mathsf{x} + \mathsf{0}}$$

- An *instance* of an inference rule is obtained by replacing all metavariables by phrases.

# FORMAL PROOFS

A set of inference rules defines a *logical theory* $\vdash$.

A *formal proof* in a logical theory is a *sequence of assertions*, each of which

- is the conclusion of some instance of an inference rule
- whose promises occur earlier in the sequence

$$
\begin{array}{lll}
1. & \mathsf{x} + \mathsf{0} = \mathsf{x} & (\textsc{xPlusZero}) \\
2. & \mathsf{x} + \mathsf{0} = \mathsf{x} \Rightarrow \mathsf{x} = \mathsf{x} + \mathsf{0} & (\textsc{SymmObjEq}) \\
3. & \mathsf{x} = \mathsf{x} + \mathsf{0} & (\textsc{ModusPonens}, 1, 2) \\
4. & \forall \mathsf{x}.\, \mathsf{x} = \mathsf{x} + \mathsf{0} & (\textsc{Generalization}, 3)
\end{array}
$$

# Tree Repr. of Formal Proofs

$$
\dfrac{\dfrac{}{x + 0 = x}\ \textsc{xPlusZero} \qquad \dfrac{}{x + 0 = x \Rightarrow x = x + 0}\ \textsc{SymmObjEq}}{\dfrac{x = x + 0}{\forall x.\, x = x + 0}\ \textsc{Gen}}\ \textsc{MP}
$$

# Soundness and Completeness

An inference rule is *sound* if all of its instances are sound, i.e., the conclusion is valid if all the premises are valid.

A logical theory is *sound* if all its inference rules are sound.

Note the differences between object and meta implication:

$$p \Rightarrow \forall v.\, p \text{ is not a sound rule, although } \frac{p}{\forall v.\, p} \text{ is.}$$

A logical theory is *complete* if all valid $p$ has a formal proof

No first-order theory of arithmetic with finite inference rules is complete (Gödel's incompleteness theorem)

# BINDING

Occurrences of variables:

- *binding occurrences*, or *binders*
  first occurrences of $v$ in $\forall v.\, p$ or $\exists v.\, p$
  $p$ is called the *scope* of the binder $v$
- *bounded occurrence*
  nonbinding occurrences of $v$ within the scope of a binder of $v$
- *free occurrence*
  otherwise
  a phrase with no free occurrence of variables is said to be *closed*

EXAMPLE

$$\forall \mathsf{x}.\, (\mathsf{x} \neq \mathsf{y} \vee \forall \mathsf{y}.\, (\mathsf{x} = \mathsf{y} \vee \forall \mathsf{x}.\, \mathsf{x} + \mathsf{y} \neq \mathsf{x}))$$

# Binding

$$\forall \mathsf{x}.\, (\mathsf{x} \neq \mathsf{y} \vee \forall \mathsf{y}.\, (\mathsf{x} = \mathsf{y} \vee \forall \mathsf{x}.\, \mathsf{x} + \mathsf{y} \neq \mathsf{x}))$$

# FREE VARIABLES

A syntax-directed definition of free variables.

$$
\begin{array}{rcl@{\qquad}rcl}
fv(\mathsf{0}) & = & \emptyset & fv(\mathbf{true}) & = & \emptyset \\
fv(\mathsf{v}) & = & \mathsf{v} & fv(e_0 = e_1) & = & fv(e_0) \cup fv(e_1) \\
fv(-\mathsf{e}) & = & fv(\mathsf{e}) & fv(\neg p) & = & fv(p) \\
fv(\mathsf{e}_1 + \mathsf{e}_2) & = & fv(\mathsf{e}_1) \cup fv(\mathsf{e}_2) & fv(p \vee p') & = & fv(p) \cup fv(p') \\
fv(p \Rightarrow p') & = & fv(p) \cup fv(p') & fv(\forall v.\, p) & = & fv(p) - \{v\} \\
 & \cdots & & & \cdots &
\end{array}
$$

E.g.,

$$
\begin{array}{l}
fv(\forall \mathsf{x}.\, (\mathsf{x} \neq \mathsf{y} \vee \forall \mathsf{y}.\, (\mathsf{x} = \mathsf{y} \vee \forall \mathsf{x}.\, \mathsf{x} + \mathsf{y} \neq \mathsf{x}))) \\
=
\end{array}
$$

# The Significance of Free Variables

**Proposition (Coincidence Theorem)**

*For any phrase $p$ and states $\sigma$, $\sigma'$, we have*

$$(\forall v \in fv(p).\, \sigma v = \sigma' v) \Rightarrow [\![p]\!]\sigma = [\![p]\!]\sigma'$$

**Proof.**

By structural induction over $p$.

**Inductive Hypothesis:** The statement of the proposition holds for all phrases of depth less than that of $p$.

... $\square$

# The Significance of Free Variables

**Proof.**

...

**Base cases:**

- $p = \mathsf{0}$: $[\![\mathsf{0}]\!]\sigma = 0 = [\![\mathsf{0}]\!]\sigma'$
- $p = \mathsf{v}$: $[\![\mathsf{v}]\!]\sigma = \sigma v = \sigma' v = [\![\mathsf{v}]\!]\sigma'$
- ...

**Inductive cases:**

- $p = e_1 + e_2$: By IH, $[\![e_i]\!]\sigma = [\![e_i]\!]\sigma'$. Thus we have
$[\![e_1 + e_2]\!]\sigma = [\![e_1]\!]\sigma + [\![e_2]\!]\sigma = [\![e_1]\!]\sigma' + [\![e_2]\!]\sigma' = [\![e_1 + e_2]\!]\sigma'$

...

- $p = \forall v.\, p'$: By IH, $[\![p]\!]\sigma\{v \rightsquigarrow n\} = [\![p']\!]\sigma'\{v \rightsquigarrow n\}$ for any $n \in \mathbf{Z}$.
Thus we have
$[\![\forall v.\, p]\!]\sigma = \forall n.\, [\![p]\!]\sigma\{v \rightsquigarrow n\} = \forall n.\, [\![p]\!]\sigma'\{v \rightsquigarrow n\} = [\![\forall v.\, p]\!]\sigma'$.

$\square$

# SUBSTITUTION

$$\frac{}{\forall v.\, p \Rightarrow p[e/v]}$$

where $p[e/v]$ denotes the result of substituting $e$ for $v$ in $p$.

Naïve substituting every occurrence will cause problems. E.g., consider the substitution $[y + 1/x]$:

$$(\forall x.\, \exists y.\, y > x) \not\Rightarrow \exists y.\, y > y + 1$$

The substitution cause $y$ in expression $y + 1$ bounded by $\exists y.$. This problem is called *unintended name capture*.

# SUBSTITUTION – AVOID NAME CAPTURE

**Solution**: rename bound variables before substitution.

$$
\begin{aligned}
&(\exists y.\, y > x)[y+1/x] \\
=&(\exists v.\, v > x)[y+1/x] \\
&\qquad \text{where } v \notin fv(y > x) \cup fv(y+1) \\
=&\exists v.\, v > y+1
\end{aligned}
$$

That is, before substitution, find a new variable $v$, replace the binder $\exists y$ with $\exists v$, and replace *free* occurrences of $y$ in $y > x$ with $v$.

**Exercise**: define substitution in a syntax-directed way.

# Substitution Theorems

Denote a substitution function by $\delta$, and substitution by $p[\delta]$.

**Proposition (Substitution theorem)**

*If* $\forall w \in fv(p).\, \sigma w = [\![\delta w]\!]\sigma'$, *then* $[\![p[\delta]]\!]\sigma' = [\![p]\!]\sigma$.

**Proposition (Renaming theorem)**

*If* $v_{new} \notin fv(q) - \{v\}$, *then* $[\![\forall v_{new}.\, q[v_{new}/v]]\!] = [\![\forall v.\, q]\!]$