Predicate Logic

Predicate logic over integer expressions:

a language of logical assertions, for example

$$\forall x.\ x + 0 = x$$

Why discuss predicate logic?

- It is an example of a simple language
- It has simple denotational semantics
- We will use it later in program specifications

Abstract Syntax

Describes the structure of a phrase

ignoring the details of its representation.

An abstract grammar for predicate logic over integer expressions:

$$
\begin{aligned}
intexp ::= {} & \mathtt{0} \mid \mathtt{1} \mid \ldots \\
\mid {} & var \\
\mid {} & -intexp \mid intexp + intexp \mid intexp - intexp \mid \ldots \\
assert ::= {} & \mathbf{true} \mid \mathbf{false} \\
\mid {} & intexp = intexp \mid intexp < intexp \mid intexp \leq intexp \mid \ldots \\
\mid {} & \neg assert \mid assert \wedge assert \mid assert \vee assert \\
\mid {} & assert \Rightarrow assert \mid assert \Leftrightarrow assert \\
\mid {} & \forall var.\ assert \mid \exists var.\ assert
\end{aligned}
$$

Resolving Notational Ambiguity

- Using parentheses: $(\forall x.\ ((((x) + (0)) + 0) = (x)))$

- Using precedence and parentheses: $\forall x.\ (x + 0) + 0 = x$

  arithmetic operators $(* \ / \ \mathbf{rem} \ldots)$ with the usual precedence
  relational operators $(= \neq < \leq \ \ldots)$
  $\neg$
  $\wedge$
  $\vee$
  $\Rightarrow$
  $\Leftrightarrow$

- The body of a quantified term extends to a delimiter.

Carriers and Constructors

- Carriers: sets of abstract phrases (e.g. $intexp$, $assert$)
- Constructors: specify abstract grammar productions

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

Note: Independent of the concrete pattern of the production:

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

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

Inductive Structure of Carrier Sets

With these properties of constructors and carriers,

carriers can be defined inductively:

$$
\begin{aligned}
intexp^{(0)} &= \{\} \\
intexp^{(j+1)} &= \{c_0\langle\rangle, \ldots\} \cup \{c_+(x_0, x_1) \mid x_0, x_1 \in intexp^{(j)}\} \cup \ldots \\
assert^{(0)} &= \{\} \\
assert^{(j+1)} &= \{c_{\mathbf{true}}\langle\rangle,\ c_{\mathbf{false}}\langle\rangle\} \\
&\quad \cup \{c_=(x_0, x_1) \mid x_0, x_1 \in intexp^{(j)}\} \cup \ldots \\
&\quad \cup \{c_\neg(x_0) \mid x_0 \in assert^{(j)}\} \cup \ldots
\end{aligned}
$$

$$
\begin{aligned}
intexp &= \bigcup_{j=0}^{\infty} intexp^{(j)} \\
assert &= \bigcup_{j=0}^{\infty} assert^{(j)}
\end{aligned}
$$

Denotational Semantics of Predicate Logic

The meaning of term $e \in intexp$ is $[\![e]\!]_{intexp}$

i.e. the function $[\![-]\!]_{intexp}$ maps objects from $intexp$ to their meanings.

What is the set of meanings?

The meaning $[\![5 + 37]\!]_{intexp}$ of the term $5 + 37$ could be the integer 42.

But the term $x + 5$ contains the free variable x...

Environments

. . . hence we need an environment (variable assignment, state)

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

to give meaning to free variables.

The meaning of a term is a function from the states to $\mathbf{Z}$ or $\mathbf{B}$.

$$\begin{array}{lcl} [\![-]\!]_{intexp} & \in & intexp \to \Sigma \to \mathbf{Z} \\ [\![-]\!]_{assert} & \in & assert \to \Sigma \to \mathbf{B} \end{array}$$

if $\sigma = [\mathsf{x} : 3, \mathsf{y} : 4]$, then $[\![\mathsf{x+5}]\!]_{intexp}\ \sigma = 8$

$[\![\exists \mathsf{z}.\ \mathsf{x} < \mathsf{z} \wedge \mathsf{z} < \mathsf{y}]\!]\ \sigma = \mathbf{false}$

Direct Semantics Equations for Predicate Logic

$v \in var$ $\quad e \in intexp$ $\quad p \in assert$

$$[\![\mathsf{0}]\!]_{intexp}\sigma = 0$$

(really $[\![c_0\langle\rangle]\!]_{intexp}\sigma = 0$)

$$[\![v]\!]_{intexp}\sigma = \sigma v$$

$$[\![e_0\mathsf{+}e_1]\!]_{intexp}\sigma = [\![e_0]\!]_{intexp}\sigma + [\![e_1]\!]_{intexp}\sigma$$

$$[\![\mathbf{true}]\!]_{assert}\sigma = \mathbf{true}$$

$$[\![e_0\mathsf{=}e_1]\!]_{assert}\sigma = [\![e_0]\!]_{intexp}\sigma = [\![e_1]\!]_{intexp}\sigma$$

$$[\![\neg p]\!]_{assert}\sigma = \neg([\![p]\!]_{assert}\sigma)$$

$$[\![p_0 \wedge p_1]\!]_{assert}\sigma = [\![p_0]\!]_{assert}\sigma \wedge [\![p_1]\!]_{assert}\sigma$$

$$[\![\forall v.\, p]\!]_{assert}\sigma = \forall n \in \mathbf{Z}.\, [\![p]\!]_{assert}[\sigma|v : n]$$

Example: The Meaning of a Term

$$
\begin{aligned}
&[\![\forall \mathsf{x}.\ \mathsf{x+0=x}]\!]_{assert}\sigma \\
&\quad = \forall n \in \mathbf{Z}.\ [\![\mathsf{x+0=x}]\!]_{assert}[\sigma|\mathsf{x} : n] \\
&\quad = \forall n \in \mathbf{Z}.\ [\![\mathsf{x+0}]\!]_{intexp}[\sigma|\mathsf{x} : n] = [\![\mathsf{x}]\!]_{intexp}[\sigma|\mathsf{x} : n] \\
&\quad = \forall n \in \mathbf{Z}.\ [\![\mathsf{x}]\!]_{intexp}[\sigma|\mathsf{x} : n] + [\![\mathsf{0}]\!]_{intexp}[\sigma|\mathsf{x} : n] = [\![\mathsf{x}]\!]_{intexp}[\sigma|\mathsf{x} : n] \\
&\quad = \forall n \in \mathbf{Z}.\ [\sigma|\mathsf{x} : n](\mathsf{x}) + 0 = [\sigma|\mathsf{x} : n](\mathsf{x}) \\
&\quad = \forall n \in \mathbf{Z}.\ n + 0 = n \\
&\quad = \mathbf{true}
\end{aligned}
$$

Properties of the Semantic Equations

- They are syntax-directed (homomorphic):
  - exactly one equation for each abstract grammar production (constructor)
  - result expressed using functions (meanings) of subterms only (arguments of constructor)

  $\Rightarrow$ they have exactly one solution $\langle [\![-]\!]_{intexp}, [\![-]\!]_{assert} \rangle$ (proof by induction on the structure of terms).

- They define compositional semantic functions

  (depending only on the meaning of the subterms)

  $\Rightarrow$ "equivalent" subterms can be substituted

Remaining Topics

- Formal Proofs
  - Inference Rules
  - Soundness and Completeness
- Binding
  - Free Variables
  - Finite-State Semantics of Predicate Logic
- Substitutions

Validity of Assertions

p holds/is true in σ $\iff$ σ satisfies p $\iff$ $[\![p]\!]_{assert}\sigma = \mathbf{true}$

p is valid $\iff \forall \sigma \in \Sigma.\ p$ holds in σ

p is unsatisfiable $\iff \forall \sigma \in \Sigma.\ [\![p]\!]_{assert}\sigma = \mathbf{false}$

$\iff \neg p$ is valid

p is stronger than p' $\iff \forall \sigma \in \Sigma.\ (p'$ holds if p holds $)$

$\iff (p \Rightarrow p')$ is valid

p and p' are equivalent $\iff p$ is stronger than p'

and p' is stronger than p

Inference Rules

Class	*Examples*
$\vdash p$ (Axiom)	$\vdash \mathsf{x} + 0 = \mathsf{x}$ (xPlusZero)
$\dfrac{}{\vdash p}$(Axiom Schema)	$\dfrac{}{\vdash e_1 = e_0 \Rightarrow e_0 = e_1}$(SymmObjEq)
$\dfrac{\vdash p_0 \quad \ldots \quad \vdash p_{n-1}}{\vdash p}$(Rule)	$\dfrac{\vdash p \quad \vdash p \Rightarrow p'}{\vdash p'}$(ModusPonens)
	$\dfrac{\vdash p}{\vdash \forall v.\, p}$(Generalization)

Formal Proofs

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

A formal proof (in a logical theory):

a sequence of instances of the inference rules, where
the premisses of each rule occur as conclusions earlier in the sequence.

1.	$\vdash \mathsf{x} + 0 = \mathsf{x}$	(xPlusZero)
2.	$\vdash \mathsf{x} + 0 = \mathsf{x} \Rightarrow \mathsf{x} = \mathsf{x} + 0$	(SymmObjEq) $[e_0 : \mathsf{x} \mid e_1 : \mathsf{x} + 0]$
3.	$\vdash \mathsf{x} = \mathsf{x} + 0$	(ModusPonens, 1, 2) $[p : \mathsf{x} + 0 = \mathsf{x} \mid p' : \mathsf{x} = \mathsf{x} + 0]$
4.	$\vdash \forall \mathsf{x}.\, \mathsf{x} = \mathsf{x} + 0$	(Generalization, 3) $[v : \mathsf{x} \mid p : \mathsf{x} = \mathsf{x} + 0]$

Tree Representation of Formal Proofs

$$\dfrac{\dfrac{\dfrac{\vdash x + 0 = x \quad \dfrac{}{\vdash x + 0 = x \Rightarrow x = x + 0}(\text{SymmObjEq})}{\vdash x = x + 0}(\text{MP})}{\vdash \forall x.\, x = x + 0}(\text{Gen})$$

Soundness of a Logical Theory

An inference rule is sound if in every instance of the rule the conclusion is valid if all the premisses are.

A logical theory $\vdash$ is sound if all inference rules in it are sound.

If $\vdash$ is sound and there is a formal proof of $\vdash p$, then p is valid.

Object vs Meta implication:

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

Completeness of a Logical Theory

A logical theory $\vdash$ is complete if

for every valid p there is a formal proof of $\vdash p$.

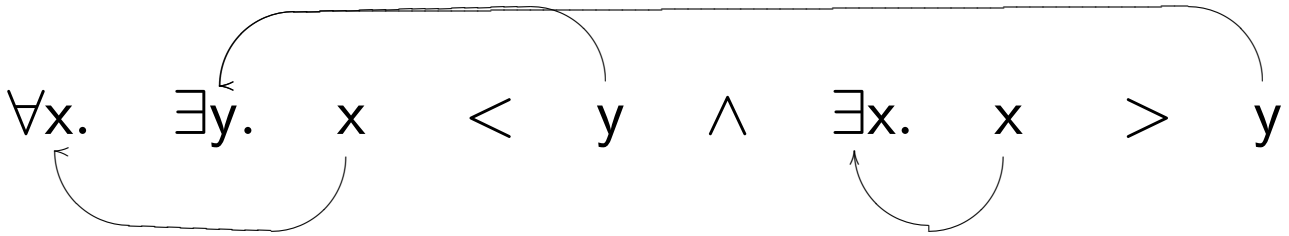
A logical theory $\vdash$ is axiomatizable if

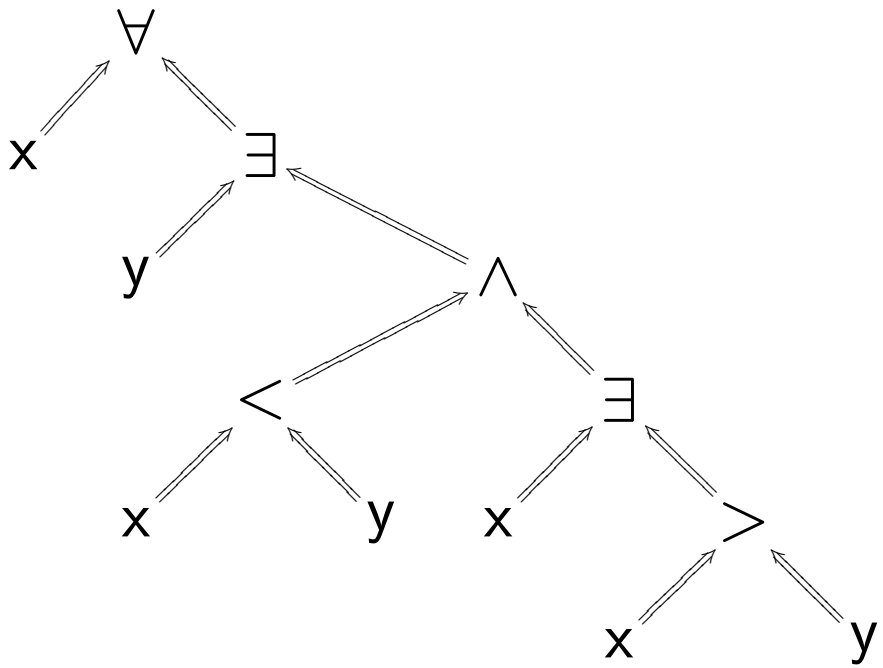
there exists a finite set of inference rules

from which can be constructed formal proofs of all assertions in $\vdash$.

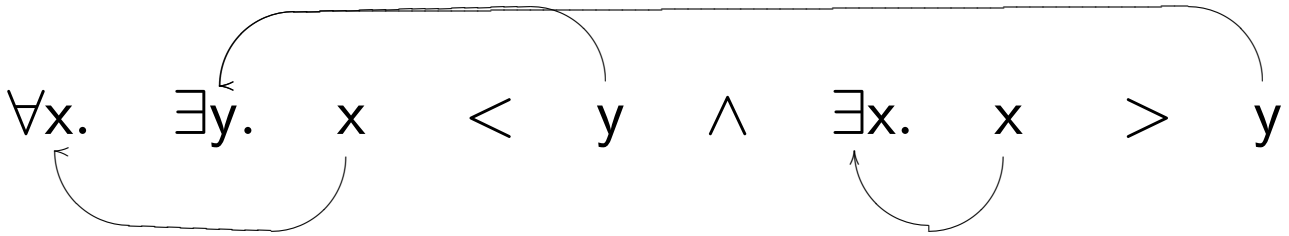
No first-order theory of arithmetic is complete and axiomatizable.

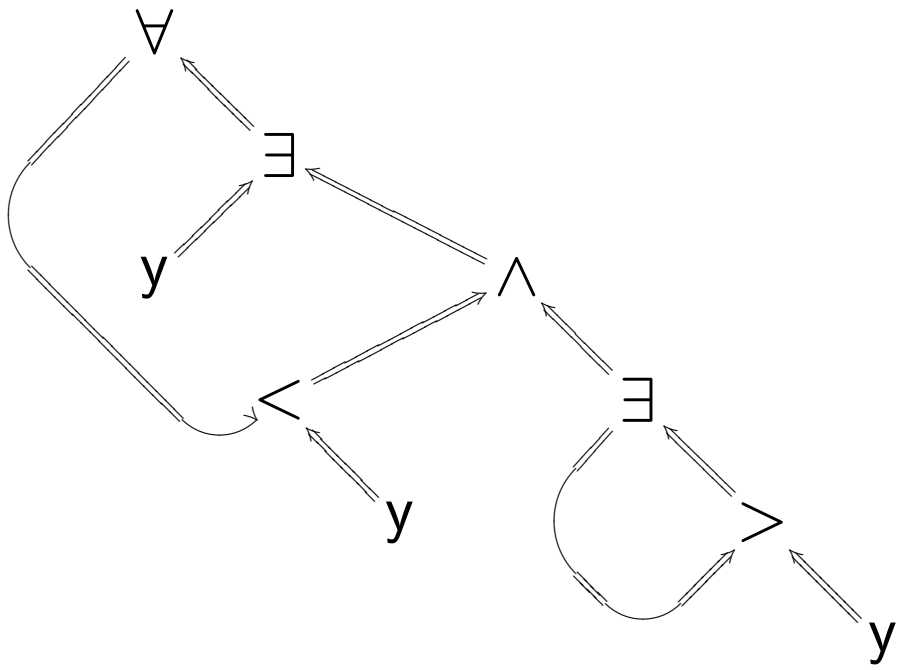
Variable Binding





Variable Binding





Bound and Free Variables

In $\forall v.\, p$, v is the binding occurrence (binder) and p is its scope.

If a non-binding occurrence of v is within the scope of a binder for v, then it is a bound occurrence; otherwise it's a free one.

$$
\begin{array}{rcl@{\qquad}rcl}
FV_{intexp}(\mathsf{0}) & = & \{\} & FV_{assert}(\mathbf{true}) & = & \{\} \\
FV(v) & = & \{v\} & FV(e_0 \mathsf{=} e_1) & = & FV(e_0) \cup FV(e_1) \\
FV(-e) & = & FV(e) & FV(\neg p) & = & FV(p) \\
FV(e_0 + e_1) & = & FV(e_0) \cup FV(e_1) & FV(p_0 \wedge p_1) & = & FV(p_0) \cup FV(p_1) \\
 & & & FV(\forall v.\, p) & = & FV(p) - \{v\}
\end{array}
$$

Example:

$$FV(\exists \mathsf{y}.\, \mathsf{x} < \mathsf{y} \wedge \exists \mathsf{x}.\, \mathsf{x} > \mathsf{y}) = \{\mathsf{x}\}$$

Only Assignment of Free Variables Matters

Coincidence Theorem:

If $\sigma v = \sigma' v$ for all $v \in FV_\theta(p)$, then $[\![p]\!]_\theta \sigma = [\![p]\!]_\theta \sigma'$

(where p is a phrase of type θ).

Proof: By structural induction.

Inductive hypothesis:
The statement of the theorem holds
for all phrases of depth less than that of the phrase p'.

Base cases:

$p' = \mathsf{0} \Rightarrow [\![\mathsf{0}]\!]_{intexp} \sigma = \mathsf{0} = [\![\mathsf{0}]\!]_{intexp} \sigma'$

$p' = v \Rightarrow [\![v]\!]_{intexp} \sigma = \sigma\, v = \sigma' v = [\![v]\!]_{intexp} \sigma'$, since $FV(v) = \{v\}$.

Proof of Concidence Theorem, cont'd

Coincidence Theorem:

If $\sigma v = \sigma' v$ for all $v \in FV_\theta(p)$, then $[\![p]\!]_\theta \sigma = [\![p]\!]_\theta \sigma'$.

Inductive cases:

$p' = e_0 + e_1$: by IH $[\![e_i]\!]_{intexp}\sigma = [\![e_i]\!]_{intexp}\sigma'$, $i \in \{1, 2\}$.

$$\begin{aligned}[\![p']\!]_{intexp}\sigma &= [\![e_0]\!]_{intexp}\sigma + [\![e_1]\!]_{intexp}\sigma \\ &= [\![e_0]\!]_{intexp}\sigma' + [\![e_1]\!]_{intexp}\sigma' = [\![p']\!]_{intexp}\sigma'\end{aligned}$$

$p' = \forall u.\, q$: $\sigma v = \sigma' v$, $\forall v \in FV(p') = FV(q) - \{u\}$

then $[\sigma|u : n]v = [\sigma'|u : n]v$, $\forall v \in FV(q)$, $n \in \mathbf{Z}$

Then by IH $[\![q]\!]_{assert}[\sigma|u : n] = [\![q]\!]_{assert}[\sigma'|u : n]$ for all $n \in \mathbf{Z}$,

hence $\forall n \in \mathbf{Z}.\, [\![q]\!]_{assert}[\sigma|u : n] = \forall n \in \mathbf{Z}.\, [\![q]\!]_{assert}[\sigma'|u : n]$

$$[\![\forall u.\, q]\!]_{assert}\sigma = [\![\forall u.\, q]\!]_{assert}\sigma'.$$

Substitution

The set of substitution maps: $\Delta \stackrel{\text{def}}{=} var \to intexp$.

Substitution: $-/\delta \in intexp \to intexp,\ assert \to assert$ when $\delta \in \Delta$.

$$\begin{array}{rcl@{\qquad}rcl} 0/\delta & = & 0 & v/\delta & = & \delta v \\ (-e)/\delta & = & -(e/\delta) & (p_0 \wedge p_1)/\delta & = & (p_0/\delta) \wedge (p_1/\delta) \end{array}$$

$$(\forall v.\, p)/\delta = \forall v_{new}.\, (p/[\delta | v : v_{new}]),$$

where $v_{new} \notin \cup\{FV(\delta u) \mid u \in FV(p) - \{v\}\}$

Example:

$$(\exists \mathsf{x}.\, \mathsf{x} \leq \mathsf{y})/[\mathsf{x} : \mathsf{y}] = \exists \mathsf{x}.\, \mathsf{x} \leq \mathsf{y}$$

Substitution

$$\left.\begin{array}{l} -/\delta \in intexp \to intexp \\ -/\delta \in assert \to assert \end{array}\right\} \text{when } \delta \in var \to intexp$$

$$\begin{array}{rcl} \mathsf{0}/\delta & = & \mathsf{0} \\ (\mathsf{-}e)/\delta & = & \mathsf{-}(e/\delta) \\ (e_0\mathsf{+}e_1)/\delta & = & (e_0/\delta)\mathsf{+}(e_1/\delta) \\ \ldots & & \end{array} \qquad \begin{array}{rcl} v/\delta & = & \delta v \\ (p_0 \wedge p_1)/\delta & = & (p_0/\delta) \wedge (p_1/\delta) \\ (\forall v.\, p)/\delta & = & \forall v'.\, (p/[\delta | v : v']), \\ \multicolumn{3}{r}{\text{where } v' \notin \bigcup_{u \in FV(p)-\{v\}} FV(\delta u)} \end{array}$$

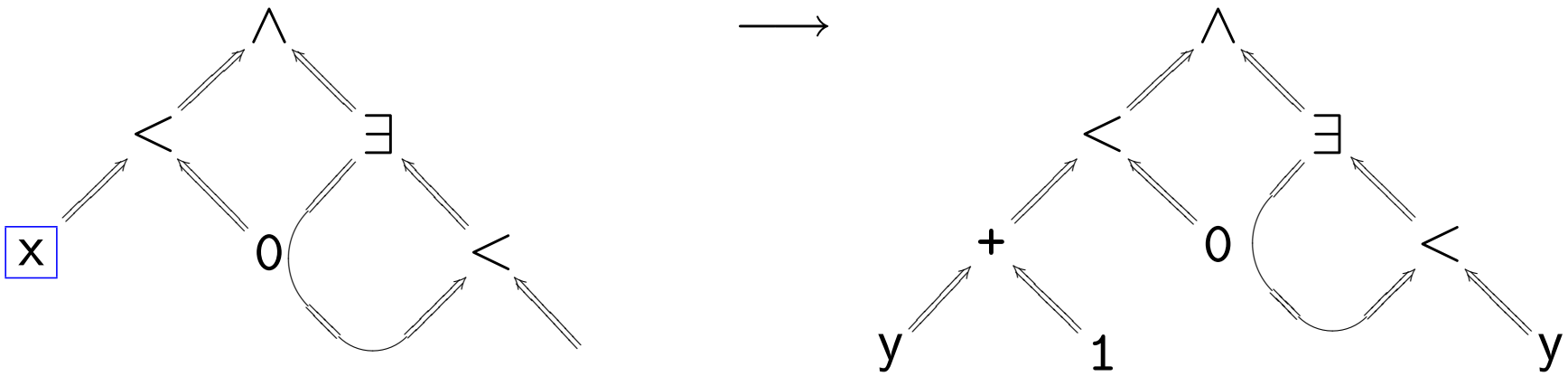
Examples:

$$(\mathsf{x} < \mathsf{0} \wedge \exists \mathsf{x}.\, \mathsf{x} \leq \mathsf{y})/[\mathsf{x} : \mathsf{y+1}] = \mathsf{y+1} < \mathsf{0} \wedge \exists \mathsf{x}.\, \mathsf{x} \leq \mathsf{y}$$

$$(\mathsf{x} < \mathsf{0} \wedge \exists \mathsf{x}.\, \mathsf{x} \leq \mathsf{y})/[\mathsf{y} : \mathsf{x+1}] = \mathsf{x} < \mathsf{0} \wedge \exists \mathsf{z}.\, \mathsf{z} \leq \mathsf{x+1}$$

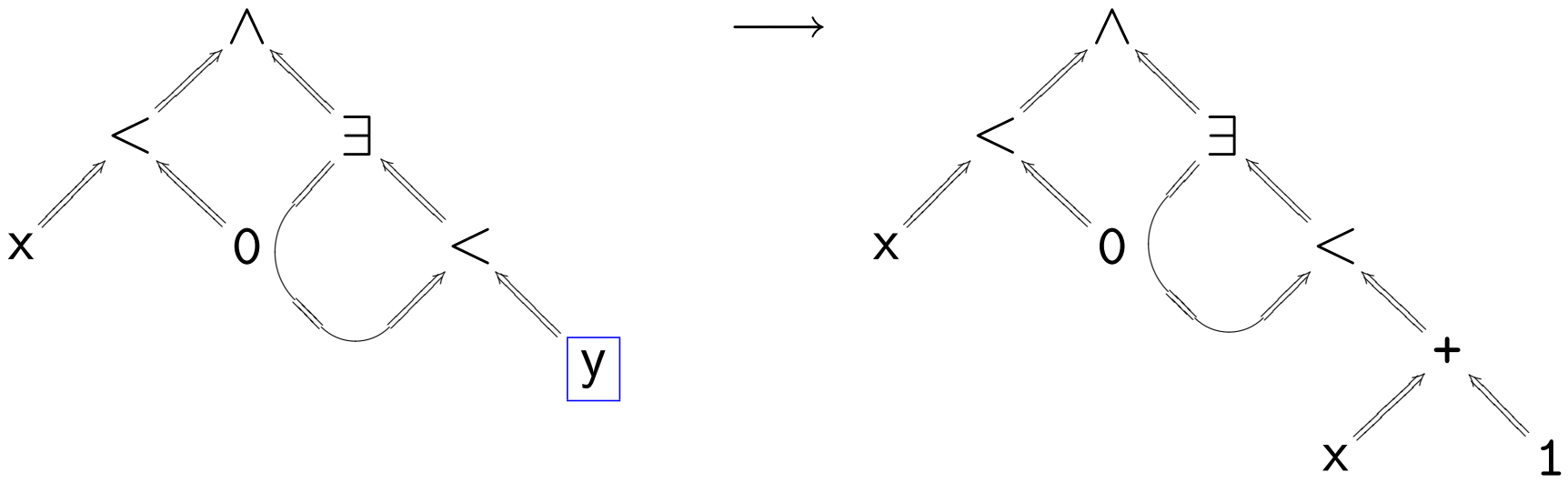
Preserving Binding Structure

$(x < 0 \wedge \exists x.\, x \leq y)/[\boxed{x} : y{+}1] \quad = \quad y{+}1 < 0 \wedge \exists x.\, x \leq y$



Avoiding Variable Capture

$(x < 0 \wedge \exists x.\, x \leq y)/[y : x{+}1] \quad = \quad x < 0 \wedge \exists z.\, z \leq x{+}1$



Substitution Theorems

Substitution Theorem:

If $\sigma = [\![-]\!]_{intexp}\sigma' \cdot \delta$ on $FV(p)$, then $([\![-]\!]\sigma)p = ([\![-]\!]\sigma' \cdot (-/\delta))p$.

Finite Substitution Theorem:

$[\![p/v_0 \to e_0, \ldots v_{n-1} \to e_{n-1}]\!]\sigma = [\![p]\!][\sigma|v_0 : [\![e_0]\!]\sigma, \ldots]$.

where

$p/v_0 \to e_0, \ldots v_{n-1} \to e_{n-1} \stackrel{\text{def}}{=} p/[c\text{var}|v_0 : e_0| \ldots |v_{n-1} : e_{n-1}]$.

Renaming:

If $u \notin FV(q) - \{v\}$, then $[\![\forall u.\, (q/v \to u)]\!]_{boolexp} = [\![\forall v.\, q]\!]_{boolexp}$.