COMP2111 Week 5
Term 1, 2024
First-Order Predicate Logic

Summary of topics

- Re-introduction to Predicate Logic
- Syntax of Predicate Logic
- Semantics of Predicate Logic
- Natural Deduction for Predicate Logic

Summary of topics

- Re-introduction to Predicate Logic
- Syntax of Predicate Logic
- Semantics of Predicate Logic
- Natural Deduction for Predicate Logic

Motivating example

Consider the statement:

$$\text{For all } x, y \in X : (y = x{+}1) \rightarrow (x \leq y)$$

Can we encode this statement in propositional logic?

Motivating example

Consider the statement:

$$\text{For all } x, y \in X : (y = x+1) \rightarrow (x \leq y)$$

$X = \{1, 2, 3\}$: 18 propositional variables:

$$\begin{array}{llll} P_{11} & = \text{“}1 = 1 + 1\text{”} & S_{11} & = \text{“}1 \leq 1\text{”} \\ P_{12} & = \text{“}2 = 1 + 1\text{”} & S_{12} & = \text{“}1 \leq 2\text{”} \\ \vdots & \vdots & \vdots & \vdots \end{array}$$

Final result: $(P_{11} \rightarrow S_{11}) \wedge (P_{12} \rightarrow S_{12}) \wedge \cdots \wedge (P_{33} \rightarrow S_{33})$

NB

“Normal arithmetic”, where P_{11} is false, P_{12} is true, etc is just one of many possibilities.

Motivating example

Consider the statement:

$$\text{For all } x, y \in X : (y = x{+}1) \rightarrow (x \leq y)$$

$X = \mathbb{N}$: ∞ propositional variables:

$$\begin{array}{llll} P_{00} & = \text{“}0 = 0 + 0\text{”} & S_{00} & = \text{“}0 \leq 0\text{”} \\ P_{01} & = \text{“}1 = 0 + 1\text{”} & S_{01} & = \text{“}0 \leq 1\text{”} \\ \vdots & \vdots & \vdots & \vdots \end{array}$$

Final result: $(P_{00} \rightarrow S_{00}) \wedge (P_{01} \rightarrow S_{01}) \wedge \cdots$

Motivating example

Consider the statement:

$$\text{For all } x, y \in X : (y = x{+}1) \rightarrow (x \leq y)$$

$X = \mathbb{N}$: ∞ propositional variables:

$$\begin{array}{llll} P_{00} & = \text{“}0 = 0 + 0\text{”} & S_{00} & = \text{“}0 \leq 0\text{”} \\ P_{01} & = \text{“}1 = 0 + 1\text{”} & S_{01} & = \text{“}0 \leq 1\text{”} \\ \vdots & \vdots & \vdots & \vdots \end{array}$$

Final result: $(P_{00} \rightarrow S_{00}) \wedge (P_{01} \rightarrow S_{01}) \wedge \cdots$ Not permitted!

Motivating example

Consider the statement:

$$\text{For all } x, y \in X : (y = x{+}1) \rightarrow (x \leq y)$$

Predicate logic introduces:

- Predicates

Motivating example

Consider the statement:

$$\text{For all } x, y \in X : (y = x + 1) \rightarrow (x \leq y)$$

Predicate logic introduces:

- Predicates
- Functions

Motivating example

Consider the statement:

$$\text{For all } x, y \in X : (y = x + 1) \rightarrow (x \leq y)$$

Predicate logic introduces:

- Predicates
- Functions
- Constants

Motivating example

Consider the statement:

$$\text{For all } x, y \in X : (y = x+1) \rightarrow (x \leq y)$$

Predicate logic introduces:

- Predicates
- Functions
- Constants
- Variables, and

Motivating example

Consider the statement:

$$\text{For all } x, y \in X : (y = x + 1) \rightarrow (x \leq y)$$

Predicate logic introduces:

- Predicates
- Functions
- Constants
- Variables, and
- Quantifiers

Domain of discourse

Q: Is this a true statement?

$$\forall x\, y.\ x = y$$

Domain of discourse

Q: Is this a true statement?

$$\forall x\, y.\; x = y$$

A: depends on what the domain of discourse is.

Domain of discourse

Fundamental to interpreting formulas is the **domain of discourse**: the set of "ground objects" that we are referring to.

Domain of discourse

Fundamental to interpreting formulas is the **domain of discourse**: the set of "ground objects" that we are referring to.

- Predicates: Relations on the domain

Domain of discourse

Fundamental to interpreting formulas is the **domain of discourse**: the set of "ground objects" that we are referring to.

- Predicates: Relations on the domain
- Functions: Operators on the domain

Domain of discourse

Fundamental to interpreting formulas is the **domain of discourse**: the set of "ground objects" that we are referring to.

- Predicates: Relations on the domain
- Functions: Operators on the domain
- Constants: "Named" elements of the domain
- Variables: "Unnamed" elements of the domain (placeholders for elements)

Domain of discourse

Fundamental to interpreting formulas is the **domain of discourse**: the set of "ground objects" that we are referring to.

- Predicates: Relations on the domain
- Functions: Operators on the domain
- Constants: "Named" elements of the domain
- Variables: "Unnamed" elements of the domain (placeholders for elements)
- Quantifiers: Range over domain elements

Domain of discourse

Fundamental to interpreting formulas is the **domain of discourse**: the set of "ground objects" that we are referring to.

- Predicates: Relations on the domain
- Functions: Operators on the domain
- Constants: "Named" elements of the domain
- Variables: "Unnamed" elements of the domain (placeholders for elements)
- Quantifiers: Range over domain elements

Example

Consider: $\forall x C(x)$ where $C(x)$ represents "x studies COMP2111"
It is `true` if the domain of discourse is the set of students in this room.

Domain of discourse

Fundamental to interpreting formulas is the **domain of discourse**: the set of "ground objects" that we are referring to.

- Predicates: Relations on the domain
- Functions: Operators on the domain
- Constants: "Named" elements of the domain
- Variables: "Unnamed" elements of the domain (placeholders for elements)
- Quantifiers: Range over domain elements

Example

Consider: $\forall x C(x)$ where $C(x)$ represents "x studies COMP2111"
It is `false` if the domain of discourse is the set of students at UNSW.

Multiple domains of discourse

Multiple domains can be combined into one as follows.

For example: the predicate studies(x, y) representing "x (a student) studies y (a subject)".

Multiple domains of discourse

Multiple domains can be combined into one as follows.

For example: the predicate studies(x, y) representing "x (a student) studies y (a subject)".

- Take $\textsc{Students} \cup \textsc{Subjects}$ as the domain.
- Use unary predicates, e.g. isStudent(x), to restrict the domain.

Multiple domains of discourse

Multiple domains can be combined into one as follows.

For example: the predicate $\text{studies}(x, y)$ representing "x (a student) studies y (a subject)".

- Take $\textsc{Students} \cup \textsc{Subjects}$ as the domain.
- Use unary predicates, e.g. $\text{isStudent}(x)$, to restrict the domain.
- To restrict quantifiers (applies to any subset of the domain defined by a unary predicate):
  - $\exists x \in \textsc{Students} : \varphi$ is equivalent to: $\exists x(\text{isStudent}(x) \wedge \varphi)$
  - $\forall x \in \textsc{Students} : \varphi$ is equivalent to: $\forall x(\text{isStudent}(x) \rightarrow \varphi)$

Domain of discourse

Function outputs, constants, and variables are interpreted as elements of the domain.

Predicates are truth-functional: they map elements of the domain to `true` or `false`.

Quantifiers (and the Boolean connectives) are predicate operators: they transform predicates into other predicates.

Example

Consider the following predicates and constants:

$K(x, y)$:	x knows y
$S(x, y)$:	x is not the son of y

J:	Jon Snow
N:	Ned Stark
B:	Bran Stark

Domain of discourse: PEOPLE

The following are OK:

- $S(B, J)$: Bran is not the son of Jon
- $K(N, J)$: Ned knows Jon
- $\forall x \neg K(J, x)$: Jon Snow knows nothing.

Example

Consider the following predicates and constants:

$K(x, y)$:	x knows y
$S(x, y)$:	x is not the son of y
J:	Jon Snow
N:	Ned Stark
B:	Bran Stark

Domain of discourse: PEOPLE

The following are OK:

- $S(B, J)$: Bran is not the son of Jon
- $K(N, J)$: Ned knows Jon
- $\forall x \neg K(J, x)$: Jon Snow knows no-one.

Example

Consider the following predicates and constants:

$K(x, y)$:	x knows y
$S(x, y)$:	x is not the son of y

J:	Jon Snow
N:	Ned Stark
B:	Bran Stark

Domain of discourse: PEOPLE

The following are OK:

- $S(B, J)$: Bran is not the son of Jon
- $K(N, J)$: Ned knows Jon
- $\forall x \neg K(J, x)$: Jon Snow knows no-one.

This is not:

- $K(B, S(J, N))$: Bran knows that Jon is not the son of Ned

Example

Consider the following predicates and constants:

$K(x, y)$:	x knows y
$S(x, y)$:	x is not the son of y
$F(x, y)$:	the fact that x is not the son of y (functional)
J:	Jon Snow
N:	Ned Stark
B:	Bran Stark

Domain of discourse: $\textsc{People} \cup \textsc{Facts}$

The following are OK:

- $S(B, J)$: Bran is not the son of Jon
- $K(N, J)$: Ned knows Jon
- $\forall x \, \neg K(J, x)$: Jon Snow knows no-one.

This is OK:

- $K(B, F(J, N))$: Bran knows that Jon is not the son of Ned

Summary of topics

- Re-introduction to Predicate Logic
- Syntax of Predicate Logic
- Semantics of Predicate Logic
- Natural Deduction for Predicate Logic

Vocabulary

A **vocabulary** indicates what predicates, functions and constants we can use to build up our formulas. Very similar to C header files, or Java interfaces, or database schemas.

A vocabulary V is a set of:

- Predicate symbols P, Q, . . . , each with an associated *arity* (number of arguments)
- Function symbols f, g, . . . , each with an associated *arity*
- Constant symbols c, d, . . . (also known as 0-arity functions)

Vocabulary

A **vocabulary** indicates what predicates, functions and constants we can use to build up our formulas. Very similar to C header files, or Java interfaces, or database schemas.

A vocabulary V is a set of:

- Predicate symbols P, Q, . . . , each with an associated *arity* (number of arguments)
- Function symbols f, g, . . . , each with an associated *arity*
- Constant symbols c, d, . . . (also known as 0-arity functions)

Example

$V = \{\leq, +, 1\}$ where $\leq$ is a binary predicate symbol, $+$ is a binary function symbol, and 1 is a constant symbol.

Vocabulary: example (databases)

Example

A database schema identifies the various tables, their attributes, and their attributes' types. For example:

Person	
Name:	String
Surname:	String
Address:	String

Employee	
ID:	int
Surname:	String

Vocabulary: example (databases)

Example

A database schema identifies the various tables, their attributes, and their attributes' types. For example:

Person	
Name:	String
Surname:	String
Address:	String

Employee	
ID:	int
Surname:	String

Tables *relate* a number of attributes
The above schema would be represented by the vocabulary:

$$DB = \{\text{Person}, \text{Employee}\}$$

where Person is a ternary predicate symbol and Employee is a binary predicate symbol

Vocabulary: example (databases)

Example

A database schema identifies the various tables, their attributes, and their attributes' types. For example:

Person	
Name:	String
Surname:	String
Address:	String

Employee	
ID:	int
Surname:	String

Tables *relate* a number of attributes (over several domains). The above schema would be represented by the vocabulary:

$$DB = \{\text{Person}, \text{Employee}, \text{isString}, \text{isInteger}\}$$

where Person is a ternary predicate symbol and Employee is a binary predicate symbol and isString and isInteger are unary predicate symbols.

Terms

A **term** is defined inductively as follows:

- A variable is a term
- A constant symbol is a term
- If f is a function symbol with arity k, and t_1, ..., t_k are terms, then $f(t_1, t_2, \ldots, t_k)$ is a term.

NB

Terms will be interpreted as elements of the domain of discourse.

Terms: examples

Example

Over $V = \{\leq, +, 1\}$, the following are all terms:

- x
- 1
- $+(y, 1)$
- $+(y, +(x, 1))$

Formulas

A **formula of Predicate Logic** is defined inductively as follows:

- If P is a predicate symbol with arity k, and t_1, ..., t_k are terms, then $P(t_1, t_2, \ldots, t_k)$ is a formula
- If t_1 and t_2 are terms then $(t_1 = t_2)$ is a formula
- If φ, ψ are a formulas then the following are formulas:
  - $\neg\varphi$
  - $(\varphi \wedge \psi)$
  - $(\varphi \vee \psi)$
  - $(\varphi \rightarrow \psi)$
  - $(\varphi \leftrightarrow \psi)$
  - $\forall x \varphi$
  - $\exists x \varphi$

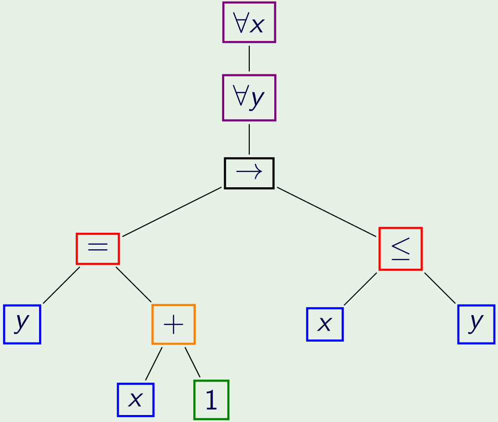
NB

The base cases are known as **atomic** *formulas: they play a similar role in the parse tree as propositional variables.*

Parse trees

Example

$$\forall x \forall y ((y = x + 1) \rightarrow (x \leq y))$$



Formulas: examples

Example

Over $V = \{\leq, +, 1\}$, the following are all formulas:

- $\leq(x, y)$
- $\leq(1, 1)$
- $x = +(y, 1)$
- $\leq(x, y) \rightarrow (x = +(y, 1))$
- $\exists x(1 = +(1, 1))$
- $\forall x \forall y \leq(x, y) \rightarrow (x = +(y, 1))$

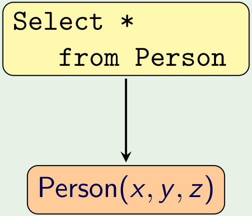
Feel free to write predicates and functions in infix for readability.

Formulas: example (databases)

In relational databases, formulas correspond to (select-)queries.

Example

For the vocabulary
$DB = \{\text{Person}, \text{Employee}, \text{isString}, \text{isInteger}\}$:



Formulas: example (databases)

In relational databases, formulas correspond to (select-)queries.

Example

For the vocabulary
$DB = \{\text{Person}, \text{Employee}, \text{isString}, \text{isInteger}, \text{Alice}\}$:

```
Select *
   from Person
   where Person.name = "Alice"
```

↓

$\text{Person}(x, y, z) \wedge (x = \text{Alice})$

Formulas: example (databases)

In relational databases, formulas correspond to (select-)queries.

Example

For the vocabulary
$DB = \{\text{Person}, \text{Employee}, \text{isString}, \text{isInteger}, \text{Alice}\}$:

```
Select *
   from Person inner join Employee
   on Person.surname = Employee.surname
```

↓

$\text{Person}(x, y, z) \vee \text{Employee}(w, y)$

Free and Bound variables

A variable is **bound** to the closest matching quantifier that lies above it in the parse tree. A variable that is not bound is **free**.

Free and Bound variables

A variable is **bound** to the closest matching quantifier that lies above it in the parse tree. A variable that is not bound is **free**.

Example

In $(\forall x \exists z \exists x P(x, y, z)) \wedge Q(x)$:

Free and Bound variables

A variable is **bound** to the closest matching quantifier that lies above it in the parse tree. A variable that is not bound is **free**.

Example

In $(\forall x \exists z \exists x P(x, y, z)) \wedge Q(x)$:

- z is bound by $\exists z$

Free and Bound variables

A variable is **bound** to the closest matching quantifier that lies above it in the parse tree. A variable that is not bound is **free**.

Example

In $(\forall x \exists z \exists x P(x, y, z)) \land Q(x)$:

- z is bound by $\exists z$
- y is free

Free and Bound variables

A variable is **bound** to the closest matching quantifier that lies above it in the parse tree. A variable that is not bound is **free**.

Example

In $(\forall x \exists z \exists x P(x, y, z)) \wedge Q(x)$:

- z is bound by $\exists z$
- y is free
- First x is bound by $\exists x$

Free and Bound variables

A variable is **bound** to the closest matching quantifier that lies above it in the parse tree. A variable that is not bound is **free**.

Example

In $(\forall x \exists z \exists x P(x, y, z)) \wedge Q(x)$:

- z is bound by $\exists z$
- y is free
- First x is bound by $\exists x$
- Second x is free

Free and Bound variables

A variable is **bound** to the closest matching quantifier that lies above it in the parse tree. A variable that is not bound is **free**.

Example

In $(\forall x \exists z \exists x P(x, y, z)) \wedge Q(x)$:

- z is bound by $\exists z$
- y is free
- First x is bound by $\exists x$
- Second x is free

A formula with no free variables is a **sentence**.

Free variables formally

We can define the set of free variables recursively on the structure of a formula:

- $FV(x) = \{x\}$ for all variables x
- $FV(c) = \emptyset$ for all constants c
- $FV(f(t_1, \ldots, t_k)) = FV(t_1) \cup \cdots \cup FV(t_k)$ for all k-ary functions f

Free variables formally

We can define the set of free variables recursively on the structure of a formula:

- $FV(x) = \{x\}$ for all variables x
- $FV(c) = \emptyset$ for all constants c
- $FV(f(t_1, \ldots, t_k)) = FV(t_1) \cup \cdots \cup FV(t_k)$ for all k-ary functions f
- $FV(P(t_1, \ldots, t_k)) = FV(t_1) \cup \cdots \cup FV(t_k)$ for all k-ary predicates P
- $FV(t_1 = t_2) = FV(t_1) \cup FV(t_2)$

Free variables formally

We can define the set of free variables recursively on the structure of a formula:

- $FV(x) = \{x\}$ for all variables x
- $FV(c) = \emptyset$ for all constants c
- $FV(f(t_1, \ldots, t_k)) = FV(t_1) \cup \cdots \cup FV(t_k)$ for all k-ary functions f
- $FV(P(t_1, \ldots, t_k)) = FV(t_1) \cup \cdots \cup FV(t_k)$ for all k-ary predicates P
- $FV(t_1 = t_2) = FV(t_1) \cup FV(t_2)$
- $FV(\neg\varphi) = FV(\varphi)$
- $FV(\psi \wedge \varphi) = FV(\psi \vee \varphi) = FV(\psi \rightarrow \varphi) = FV(\psi \leftrightarrow \varphi) = FV(\psi) \cup FV(\varphi)$

Free variables formally

We can define the set of free variables recursively on the structure of a formula:

- $FV(x) = \{x\}$ for all variables x
- $FV(c) = \emptyset$ for all constants c
- $FV(f(t_1, \ldots, t_k)) = FV(t_1) \cup \cdots \cup FV(t_k)$ for all k-ary functions f
- $FV(P(t_1, \ldots, t_k)) = FV(t_1) \cup \cdots \cup FV(t_k)$ for all k-ary predicates P
- $FV(t_1 = t_2) = FV(t_1) \cup FV(t_2)$
- $FV(\neg\varphi) = FV(\varphi)$
- $FV(\psi \wedge \varphi) = FV(\psi \vee \varphi) = FV(\psi \to \varphi) = FV(\psi \leftrightarrow \varphi) = FV(\psi) \cup FV(\varphi)$
- $FV(\forall x\varphi) = FV(\exists x\varphi) = FV(\varphi) \setminus \{x\}$

Substitution

If t is a term, φ a formula, and $x \in FV(\varphi)$, then the **substitution of** t **for** x **in** φ (denoted $\varphi[t/x]$) is the formula obtained by replacing every free occurrence of x with t.

Substitution

If t is a term, φ a formula, and $x \in FV(\varphi)$, then the **substitution of t for x in φ** (denoted $\varphi[t/x]$) is the formula obtained by replacing every free occurrence of x with t.

It can be useful to have "access" to the free variables of a formula. So if $x_1, \ldots, x_k$ are the free variables of φ, we may denote this as $\varphi(x_1, \ldots, x_k)$. Substitution can be easily presented: $\varphi(t)$ for $\varphi(x)[t/x]$.

Note

Variable names matter: $\varphi(x)$ and $\varphi(y)$ are different formulas!

Summary of topics

- Re-introduction to Predicate Logic
- Syntax of Predicate Logic
- Semantics of Predicate Logic
- Natural Deduction for Predicate Logic

Models

Q: Is this a true statement?

There is nothing going on between us.

Models

Q: Is this a true statement?

There is nothing going on between us.

A: It depends upon what the meaning of the word 'is' is.

Models

Q: Is this a true statement?

$$1 + 1 = 2$$

Models

Q: Is this a true statement?

$$1 + 1 = 2$$

A: It depends on what the model of 1, 2 and $+$ is.

Models

$\{\forall, \exists, =, \wedge, \vee, \neg, \rightarrow, \leftrightarrow\}$ have a fixed meaning in first-order logic.

All other symbols are meaningless, unless we specify a *model*.

Models

Predicate formulas are interpreted in **Models**.

Given a vocabulary V a model $\mathcal{M}$ defines:

- A (non-empty) domain $D = \text{dom}(\mathcal{M})$
- For every predicate symbol $P \in V$ with arity k: a k-ary relation $P^{\mathcal{M}}$ on D
- For every function symbol $f \in V$ with arity k: a function $f^{\mathcal{M}} : D^k \rightarrow D$
- For every constant symbol $c \in V$: an element, $c^{\mathcal{M}}$ of D

Models

Predicate formulas are interpreted in **Models**.

Given a vocabulary V a model $\mathcal{M}$ defines:

- A (non-empty) domain $D = \mathrm{dom}(\mathcal{M})$
- For every predicate symbol $P \in V$ with arity k: a k-ary relation $P^{\mathcal{M}}$ on D
- For every function symbol $f \in V$ with arity k: a function $f^{\mathcal{M}} : D^k \rightarrow D$
- For every constant symbol $c \in V$: an element, $c^{\mathcal{M}}$ of D

Example

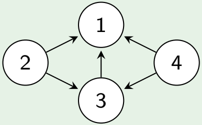
For the vocabulary $V = \{\leq, +, 1\}$: one model could be $\mathbb{N}$ with the standard definitions.

Models: examples

Example

For the vocabulary $V = \{\leq, +, 1\}$ the following are models:

- $\mathbb{N}$ with the standard definitions of $\leq$, $+$, and 1.
- $\{0, 1, 2, 3, 4\}$ with the standard definition of $\leq$ and 1, and $m + n$ defined as $m + n \pmod 5$.
- The directed graph $G = (V, E)$ shown below with $\leq = E$; and $v + w$ defined to be w.



Models: example (databases)

Example

For the vocabulary $DB = \{\text{Person}, \text{Employee}, \text{isString}, \text{isInteger}\}$, the following **database** is a model:

Person		
Name	**Surname**	**Address**
Rapunzel	-	Tower
Cinderella	-	c/o Stepmum
Snow	White	Cottage

Employee	
ID	**Surname**
31415	Psmith
27182	Ukridge
16180	Wooster

isString and isInteger are defined by what values are permitted in each of the columns (*sanitizing* the input).

Environments

Given a model $\mathcal{M}$, an **environment** (or **lookup table**), η, is a function from the set of variables to $\text{dom}(\mathcal{M})$.

Environments

Given a model $\mathcal{M}$, an **environment** (or **lookup table**), η, is a function from the set of variables to $\mathsf{dom}(\mathcal{M})$.

Given an environment η, we denote by $\eta[x \mapsto c]$ the environment that agrees with η everywhere except possibly at x (where it has value c).

Interpretations

An **interpretation** is a pair $(\mathcal{M}, \eta)$ where $\mathcal{M}$ is a model and η is an environment.

Interpretations

An **interpretation** is a pair $(\mathcal{M}, \eta)$ where $\mathcal{M}$ is a model and η is an environment.

An interpretation $(\mathcal{M}, \eta)$ maps terms to elements of $\mathrm{dom}(\mathcal{M})$ recursively as follows:

- $[\![x]\!]^\eta_\mathcal{M} = \eta(x)$
- $[\![c]\!]^\eta_\mathcal{M} = c^\mathcal{M}$
- $[\![f(t_1, \ldots, t_k)]\!]^\eta_\mathcal{M} = f^\mathcal{M}([\![t_1]\!]^\eta_\mathcal{M}, \ldots, [\![t_k]\!]^\eta_\mathcal{M})$

Interpretations

An **interpretation** is a pair $(\mathcal{M}, \eta)$ where $\mathcal{M}$ is a model and η is an environment.

An interpretation $(\mathcal{M}, \eta)$ maps formulas to $\mathbb{B}$ recursively as follows:

- $[\![P(t_1, \ldots, t_k)]\!]^{\eta}_{\mathcal{M}} = \texttt{true}$ if $P^{\mathcal{M}}([\![t_1]\!]^{\eta}_{\mathcal{M}}, \ldots, [\![t_k]\!]^{\eta}_{\mathcal{M}})$ holds.
- $[\![t_1 = t_2]\!]^{\eta}_{\mathcal{M}} = \texttt{true}$ if $[\![t_1]\!]^{\eta}_{\mathcal{M}} = [\![t_2]\!]^{\eta}_{\mathcal{M}}$
- $[\![\forall x \varphi]\!]^{\eta}_{\mathcal{M}} = \texttt{true}$ if $[\![\varphi]\!]^{\eta[x \mapsto c]}_{\mathcal{M}} = \texttt{true}$ for all $c \in \mathrm{dom}(\mathcal{M})$
- $[\![\exists x \varphi]\!]^{\eta}_{\mathcal{M}} = \texttt{true}$ if $[\![\varphi]\!]^{\eta[x \mapsto c]}_{\mathcal{M}} = \texttt{true}$ for some $c \in \mathrm{dom}(\mathcal{M})$
- $[\![\varphi]\!]^{\eta}_{\mathcal{M}}$ defined in the same way as Propositional Logic for all other formulas φ.

Interpretations: examples

Example

$$\forall x \forall y((y = x + 1) \to (x \leq y))$$

- $\mathbb{N}$ with the standard definitions of $\leq$, $+$, and 1:

Interpretations: examples

Example

$$\forall x \forall y((y = x + 1) \rightarrow (x \leq y))$$

- $\mathbb{N}$ with the standard definitions of $\leq$, $+$, and 1: `true`

Interpretations: examples

Example

$$\forall x \forall y((y = x + 1) \rightarrow (x \leq y))$$

- $\mathbb{N}$ with the standard definitions of $\leq$, $+$, and 1: `true`
- $\{0, 1, 2, 3, 4\}$ with the standard definition of $\leq$ and 1, and $m + n$ defined as $m + n \pmod 5$:

Interpretations: examples

Example

$$\forall x \forall y((y = x + 1) \rightarrow (x \leq y))$$

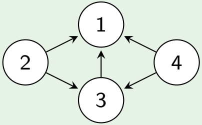
- $\mathbb{N}$ with the standard definitions of $\leq$, $+$, and 1: `true`
- $\{0, 1, 2, 3, 4\}$ with the standard definition of $\leq$ and 1, and $m + n$ defined as $m + n \pmod 5$: `false`

Interpretations: examples

Example

$$\forall x \forall y ((y = x + 1) \to (x \le y))$$

- $\mathbb{N}$ with the standard definitions of $\le$, $+$, and 1: `true`
- $\{0, 1, 2, 3, 4\}$ with the standard definition of $\le$ and 1, and $m + n$ defined as $m + n \pmod 5$: `false`
- The directed graph $G = (V, E)$ shown below with $\le = E$, 1 be the vertex 1, and $v + w$ defined to be w.

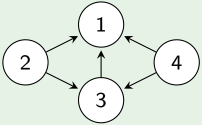


Interpretations: examples

Example

$$\forall x \forall y ((y = x + 1) \to (x \leq y))$$

- $\mathbb{N}$ with the standard definitions of $\leq$, $+$, and 1: `true`
- $\{0, 1, 2, 3, 4\}$ with the standard definition of $\leq$ and 1, and $m + n$ defined as $m + n \pmod 5$: `false`
- The directed graph $G = (V, E)$ shown below with $\leq = E$, 1 be the vertex 1, and $v + w$ defined to be w.



`true`

Why separate the environment from the model?

In the definition of $[\![\varphi]\!]^{\eta}_{\mathcal{M}}$, η is only used to define values for the free variables. In particular, if φ is a sentence then $[\![\varphi]\!]^{\eta}_{\mathcal{M}}$ is independent of η.

Why separate the environment from the model?

In the definition of $[\![\varphi]\!]^{\eta}_{\mathcal{M}}$, η is only used to define values for the free variables. In particular, if φ is a sentence then $[\![\varphi]\!]^{\eta}_{\mathcal{M}}$ is independent of η.

Define $[\![\cdot]\!]_{\mathcal{M}}$ by "delaying" the assigning of values to free variables, and propagating them out.

Why separate the environment from the model?

In the definition of $[\![\varphi]\!]^{\eta}_{\mathcal{M}}$, η is only used to define values for the free variables. In particular, if φ is a sentence then $[\![\varphi]\!]^{\eta}_{\mathcal{M}}$ is independent of η.

Define $[\![\cdot]\!]_{\mathcal{M}}$ by "delaying" the assigning of values to free variables, and propagating them out. That is, define:

$$[\![\varphi(x_1, x_2, \ldots, x_n)]\!]_{\mathcal{M}} = [\![\varphi]\!]_{\mathcal{M}}(x_1, x_2, \ldots, x_n)$$

where $[\![\varphi]\!]_{\mathcal{M}}$:

Why separate the environment from the model?

In the definition of $[\![\varphi]\!]^{\eta}_{\mathcal{M}}$, η is only used to define values for the free variables. In particular, if φ is a sentence then $[\![\varphi]\!]^{\eta}_{\mathcal{M}}$ is independent of η.

Define $[\![\cdot]\!]_{\mathcal{M}}$ by "delaying" the assigning of values to free variables, and propagating them out. That is, define:

$$[\![\varphi(x_1, x_2, \ldots, x_n)]\!]_{\mathcal{M}} = [\![\varphi]\!]_{\mathcal{M}}(x_1, x_2, \ldots, x_n)$$

where $[\![\varphi]\!]_{\mathcal{M}} : \mathrm{dom}(\mathcal{M})^n \to \mathbb{B}$;

Why separate the environment from the model?

In the definition of $[\![\varphi]\!]^{\eta}_{\mathcal{M}}$, η is only used to define values for the free variables. In particular, if φ is a sentence then $[\![\varphi]\!]^{\eta}_{\mathcal{M}}$ is independent of η.

Define $[\![\cdot]\!]_{\mathcal{M}}$ by "delaying" the assigning of values to free variables, and propagating them out. That is, define:

$$[\![\varphi(x_1, x_2, \ldots, x_n)]\!]_{\mathcal{M}} = [\![\varphi]\!]_{\mathcal{M}}(x_1, x_2, \ldots, x_n)$$

where $[\![\varphi]\!]_{\mathcal{M}} : \mathsf{dom}(\mathcal{M})^n \to \mathbb{B}$; that is, $[\![\varphi]\!]_{\mathcal{M}}$ is an n-ary relation on $\mathsf{dom}(\mathcal{M})$.

Interpretations: example (databases)

Example

- Vocabulary: database schema
- Formulas: queries (φ)
- Models: databases ($\mathcal{D}$)
- Interpretation:

Interpretations: example (databases)

Example

- Vocabulary: database schema
- Formulas: queries (φ)
- Models: databases ($\mathcal{D}$)
- Interpretation: $[\![\varphi]\!]_{\mathcal{D}}$ is a relation on $\mathrm{dom}(\mathcal{D})$, i.e. a (derived) table in $\mathcal{D}$

Interpretations: example (databases)

Example

- Vocabulary: database schema
- Formulas: queries (φ)
- Models: databases ($\mathcal{D}$)
- Interpretation: $[\![\varphi]\!]_{\mathcal{D}}$ is a relation on $\mathrm{dom}(\mathcal{D})$, i.e. a (derived) table in $\mathcal{D}$
- Environment:

Interpretations: example (databases)

Example

- Vocabulary: database schema
- Formulas: queries (φ)
- Models: databases ($\mathcal{D}$)
- Interpretation: $[\![\varphi]\!]_{\mathcal{D}}$ is a relation on $\mathrm{dom}(\mathcal{D})$, i.e. a (derived) table in $\mathcal{D}$
- Environment: looks up an entry in a (derived) table and returns whether the lookup was successful

Interpretations: example (databases)

Example

- Vocabulary: database schema
- Formulas: queries (φ)
- Models: databases ($\mathcal{D}$)
- Interpretation: $[\![\varphi]\!]_{\mathcal{D}}$ is a relation on $\mathsf{dom}(\mathcal{D})$, i.e. a (derived) table in $\mathcal{D}$
- Environment: looks up an entry in a (derived) table and returns whether the lookup was successful
- $[\![\varphi]\!]^{\eta}_{\mathcal{D}}$: Success/fail outcome of looking up a specific entry in a query result on $\mathcal{D}$.

Satisfiability, truth, validity

A formula φ of predicate logic is:

- **satisfiable** if there is some model $\mathcal{M}$ and some environment η such that $[\![\varphi]\!]^{\eta}_{\mathcal{M}} = \mathtt{true}$.
- **true in a model** $\mathcal{M}$ if for all environments η we have $[\![\varphi]\!]^{\eta}_{\mathcal{M}} = \mathtt{true}$
- a **logical validity** if it is true in all models.

NB

For sentences the first two definitions coincide.

Satisfiability, truth, validity

A formula φ of predicate logic is:

- **satisfiable** if there is some model $\mathcal{M}$ and some environment η such that $[\![\varphi]\!]^{\eta}_{\mathcal{M}} = \texttt{true}$.
- **true in a model** $\mathcal{M}$ if for all environments η we have $[\![\varphi]\!]^{\eta}_{\mathcal{M}} = \texttt{true}$
- a **logical validity** if it is true in all models.

NB

For sentences the first two definitions coincide.

Example

The sentence $\forall x \forall y((y = x + 1) \to (x \leq y))$ is satisfiable but it is not a logical validity.

Entailment, Logical equivalence

- A theory T **entails** a formula φ, $T \models \varphi$, if φ is satisfied by any interpretation that satisfies all formulas in T.
- φ is **logically equivalent** to ψ, $\varphi \equiv \psi$, if $[\![\varphi]\!]^{\eta}_{\mathcal{M}} = [\![\psi]\!]^{\eta}_{\mathcal{M}}$ for all interpretations $(\mathcal{M}, \eta)$.

Entailment, Logical equivalence

- A theory T **entails** a formula φ, $T \models \varphi$, if φ is satisfied by any interpretation that satisfies all formulas in T.
- φ is **logically equivalent** to ψ, $\varphi \equiv \psi$, if $[\![\varphi]\!]^{\eta}_{\mathcal{M}} = [\![\psi]\!]^{\eta}_{\mathcal{M}}$ for all interpretations $(\mathcal{M}, \eta)$.

Theorem

- $\varphi_1, \ldots, \varphi_n \models \psi$ *if, and only if,* $(\varphi_1 \wedge \cdots \wedge \varphi_n) \rightarrow \psi$ *is a logical validity.*
- $\varphi \equiv \psi$ *if, and only if,* $\varphi \leftrightarrow \psi$ *is a logical validity.*

Summary of topics

- Re-introduction to Predicate Logic
- Syntax of Predicate Logic
- Semantics of Predicate Logic
- Natural Deduction for Predicate Logic (not today)