

MAT 7003 : Mathematical Foundations

(for Software Engineering)

J Paul Gibson, A207

paul.gibson@it-sudparis.eu

<http://www-public.it-sudparis.eu/~gibson/Teaching/MAT7003/>

Logic

<http://www-public.it-sudparis.eu/~gibson/Teaching/MAT7003/L4-Logic.pdf>

What is (a) logic?

Hundreds of different logics have been studied by philosophers, computer scientists, software engineers, mathematicians, ...

Any ‘formal system’ can be considered a logic if it has:

- a *well-defined* **syntax**,
defines the syntactically acceptable objects of the language, which are properly called well-formed formulae (wff). (We shall just call them formulae.)
- a *well-defined* **semantics**,
associates each formula with a meaning
- a *well-defined* **proof-theory**,
for manipulating formulae according to certain rules

What is propositional logic?

Perhaps, the simplest, and most abstract logic we can study

It allows us to reason about *propositions*

Definition: A *proposition* is a statement that can be either **true** or **false**; it must be one or the other, and it cannot be both

It is possible to determine whether any given statement is a proposition by prefixing it with:

« It is true that . . . »

and seeing whether the result makes grammatical sense.

What is propositional logic?

We now define our fundamental building blocks – the *atomic propositions*. Intuitively, these are the set of « smallest » propositions.

Definition: An *atomic proposition* is one whose truth or falsity does not depend on the truth or falsity of any other proposition.

Thus, its meaning is directly either **true** or **false**.

What is propositional logic?

Now, rather than write out propositions in full, we will abbreviate them by using *propositional variables*.

It is standard practice to use the lower-case roman letters

$p; q; r; \dots$

to stand for propositions.

If we do this, we must define what we mean by writing something like:

Let p be « *Paul Gibson likes chocolate* ».

Another alternative is to write something like:

PaulGibson_likes-chocolate,

so that the interpretation of the propositional variable is obvious.

What is propositional logic?

Connectives

The study of atomic propositions is trivial.

We introduce a number of **connectives** which will allow us to build up non-trivial propositions (sometimes called **formulae**):

\wedge	<i>and</i>	(& or .)
\vee	<i>or</i>	(or +)
\neg	<i>not</i>	(~)
\Rightarrow	<i>implies</i>	(\supset)
\Leftrightarrow	<i>iff</i>	

What is propositional logic?

\vee *or*

Definition: If p and q are arbitrary propositions, then the *disjunction* of p and q is written

$$p \vee q$$

and will be **false** *iff* both p and q are **false**.

The **truth table** for *disjunction* is as follows:

p	q	$p \vee q$
F	F	F
F	T	T
T	F	T
T	T	T

What is propositional logic?

\wedge *and*

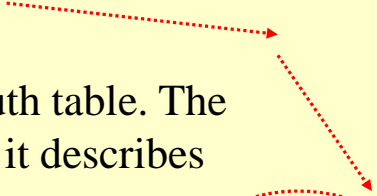
Definition: If p and q are arbitrary propositions, then the *conjunction* of p and q is written

$$p \wedge q$$

and will be **true** *iff* both p and q are **true**.

We can summarise the operation of \wedge in a truth table. The idea of a truth table for some formula is that it describes the behaviour of a formula under all possible interpretations of the primitive propositions the are included in the formula.

The **truth table** for *conjunction* is as follows:



p	q	$p \wedge q$
F	F	F
F	T	F
T	F	F
T	T	T

What is propositional logic?

\Rightarrow *implies*

Definition: If p and q are arbitrary propositions, then the *conditional (implication)* of p and q is written

$$p \Rightarrow q$$

and will be true *iff* the truth of p guarantees the truth of q

The **truth table** for *implication* is as follows:

p	q	$p \Rightarrow q$
F	F	T
F	T	T
T	F	F
T	T	T

NOTE: when p is **false** then the implication is sure to be **true**. (This is critically important)

Terminology: if Φ is the formula $p \Rightarrow q$, then:

p is the *antecedent* of Φ and q is the *consequent*

What is propositional logic?

\Leftrightarrow *iff*

Definition: If p and q are arbitrary propositions, then the *biconditional* of p and q is written

$$p \Leftrightarrow q$$

and will be true *iff* the meaning of p is the same as that of q

The **truth table** for *biconditional* is as follows:

p	q	$p \Leftrightarrow q$
F	F	T
F	T	F
T	F	F
T	T	T

If $p \Leftrightarrow q$ is **true**, then p and q are said to be *logically equivalent*: they will be true under exactly the same circumstances

What is propositional logic?

All of the connectives we have considered so far have been binary: they have taken two arguments.

The final connective we consider here is unary. It only takes one argument.

Any proposition can be prefixed by the word 'not' to form a second proposition called the *negation* of the original.

Definition: If p is an arbitrary proposition then the *negation* of p is written $\neg p$ and will be **true** iff p is **false**.

p	$\neg p$
F	T
T	F

The **truth table** for *negation* is as follows:

What is propositional logic?

Well-defined syntax

We can nest complex formulae as deeply as we want.

We can use parentheses - i.e. $()$ and $(-$ to disambiguate formulae.

If p, q, r, s and t are atomic propositions, then all of the following are *well formed formulae* (or just *formulae*):

$$\begin{aligned} p \wedge q \Rightarrow r \\ p \wedge (q \Rightarrow r) \\ (p \wedge (q \Rightarrow r)) \vee s \\ ((p \wedge (q \Rightarrow r)) \vee s) \wedge t \end{aligned}$$

But,
these
are not:

$$\begin{aligned} p \wedge \\ p \wedge q) \\ p \neg \end{aligned}$$

What is propositional logic?

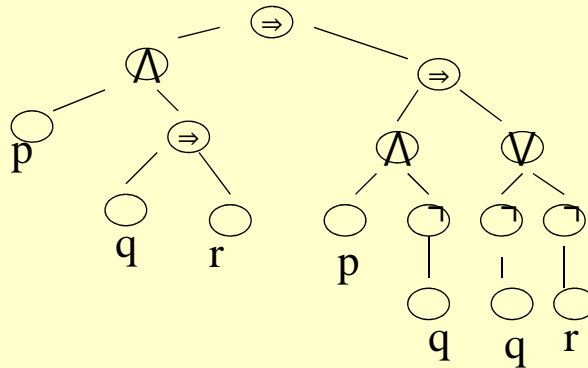
Well-defined syntax

We can nest complex formulae as deeply as we want.

We can use parentheses - i.e.) and (- to disambiguate formulae.

We can visualise these formulae using syntax trees.

$$((p \wedge (q \Rightarrow r)) \Rightarrow ((p \wedge (\neg q)) \Rightarrow ((\neg q) \vee (\neg r))))$$

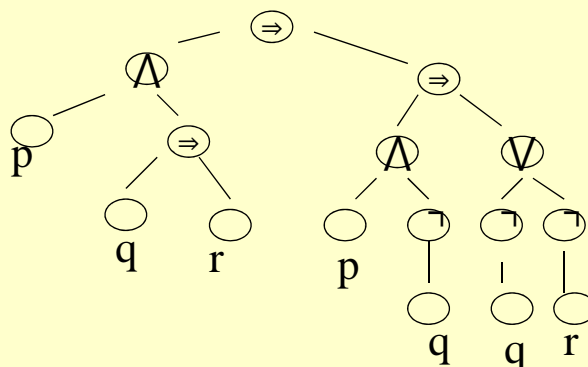


What is propositional logic?

Well-defined syntax

Using « Polish notation » we can dispense with brackets by writing connectives in front of the expressions (sub-formulae) they connect

For example: $\Rightarrow \wedge p \Rightarrow q r \Rightarrow \wedge p \neg q \vee \neg q \neg r$



Note: this reduces the original expression from 34 to 16 symbols

What is propositional logic?

Well-defined semantics

Given a particular formula, can you tell if it is true or not?

No — you usually need to know the truth values of the component atomic propositions in order to be able to tell whether a formula is true.

Definition: A *valuation* (or *interpretation* or *assignment*) is a function which assigns a truth value to each primitive proposition.

Given an *interpretation* (I) of the atomic propositions, any formula (built from these atomic propositions, using the previously defined connectives) can be given a meaning under that *interpretation*.

What is propositional logic?

Well-defined semantics

For example:

$$v(p) = F$$

$$v(q) = T$$

$$v(r) = F$$

$$(v(p) \vee v(q)) \Rightarrow v(r) \quad (1)$$

$$= (F \vee T) \Rightarrow F \quad (2)$$

$$= T \Rightarrow F \quad (3)$$

$$= F \quad (4)$$

What is propositional logic?

Well-defined semantics

Definition(s):

1. A formula is a *tautology* (valid) iff it is true under every valuation;
2. A formula is *consistent* (*satisfiable*) iff it is true under at least one valuation;
3. A formula is *inconsistent* (*a contradiction*) iff it is not made true under any valuation.
4. A formula is *contingent* iff it is *consistent* but not a *tautology*.

Now, each line in the truth table of a formula corresponds to a valuation.

So, we can use truth tables to determine whether or not formulae are *tautologies/consistent/inconsistent/contingent*

What is propositional logic?

Well-defined semantics

Definition(s):

•Two formulae are said to be *logically equivalent* if they have the same truth tables.
In other words, if the **sets** of *interpretations/valuations/assignments* for which the formulae are true are the same.

•If ζ is a (possibly infinite) set of formulae and Φ is a single formula, then

Φ is a *logical consequence* of ζ , written

$$\zeta \models \Phi$$

if, any assignment making all members of ζ true also makes Φ true

Thus, two formulae are *logically equivalent* iff each is a *logical consequence* of the other

What is propositional logic?

Well-defined proof-theory

Proof methods

Proof methods divide into (roughly) two kinds:

- *Model checking*:
 - Truth table enumeration (sound and complete for propositional logic)
 - Heuristic search in model space (sound but incomplete)
- *Application of inference rules*:
 - Legitimate (sound) generation of new sentences from old
 - A *proof* is a sequence of inference rule applications
 - Inference rules can be used as operators in a standard search algorithm!

What is propositional logic?

Well-defined proof-theory allows us to reason as follows:

If ζ is a set of formulae and Φ is a single formula, then Φ is a *logical consequence* of ζ , written

$$\zeta \models \Phi$$

if, any assignment making all members of ζ true also makes Φ true

To test/prove/demonstrate whether

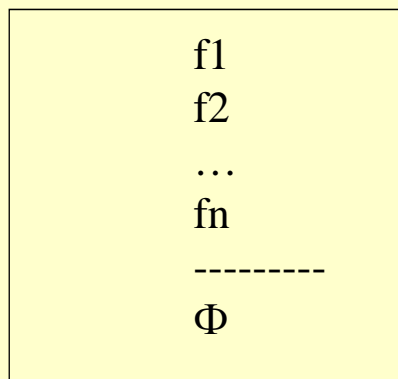
$$\zeta \models \Phi$$

•Either, find an assignment of truth values in which all members of ζ are true but Φ is false; thus $\zeta \models \Phi$ cannot hold.

•Or, show that it is impossible to find such an assignment; thus $\zeta \models \Phi$ must hold.

What is propositional logic?

$\zeta \models \Phi$, where $\zeta = \{f_1, f_2, \dots, f_n\}$ can be represented as:



Using the truth table method for formally establishing consequence/equivalence is « guaranteed to work » but there are some inconveniences!

QUESTION: Do you know what these are?

What is propositional logic?

Well-defined proof-theory

Logic is a form of knowledge representation:

- Sentences have to obey syntactic laws
- Truth is established with respect to a *model* of the world and an *interpretation* which maps symbols to world objects
- The truth table method is a *sound* and *complete* inference method, which checks truth in all possible models.
- But the truth table method is very inefficient!
 2^n models for n literals

What is propositional logic?

A *Well-defined proof-theory* may allow algebraic reasoning using inference rules, eg:

- *Resolution* (for CNF): complete for propositional logic

$$\frac{\alpha \vee \beta, \quad \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

- *Modus Ponens* (for Horn Form): complete for Horn KBs

$$\frac{\alpha_1, \dots, \alpha_n, \quad \alpha_1 \wedge \dots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

Can be used with *forward chaining* or *backward chaining*

What is propositional logic?

A *Well-defined proof-theory* may allow algebraic reasoning using inference rules, eg:

- *And-elimination*:

$$\frac{\alpha_1 \wedge \dots \wedge \alpha_n}{\alpha_i, \forall i = 1, \dots, n}$$

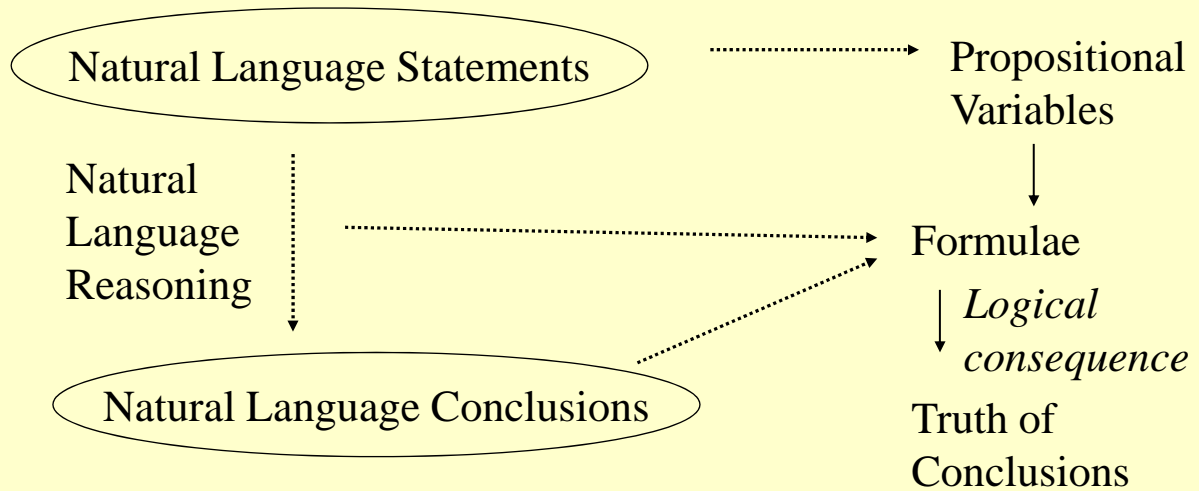
- *Implication elimination*:

$$\frac{\alpha \Rightarrow \beta}{\neg\alpha \vee \beta}$$

Before we look at the proof theory in any more detail, we will look at applying the logic

How to Use Propositional Logic?

1: We can use propositional logic to check the reasoning in natural language « documentation ».



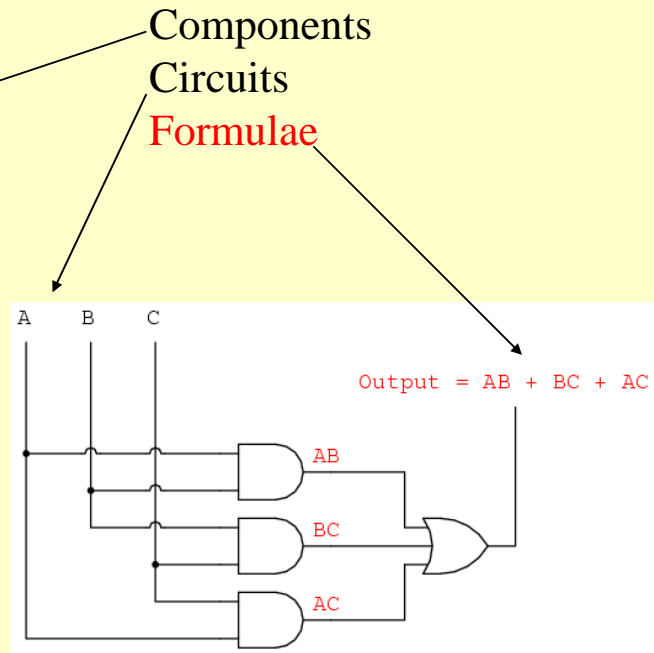
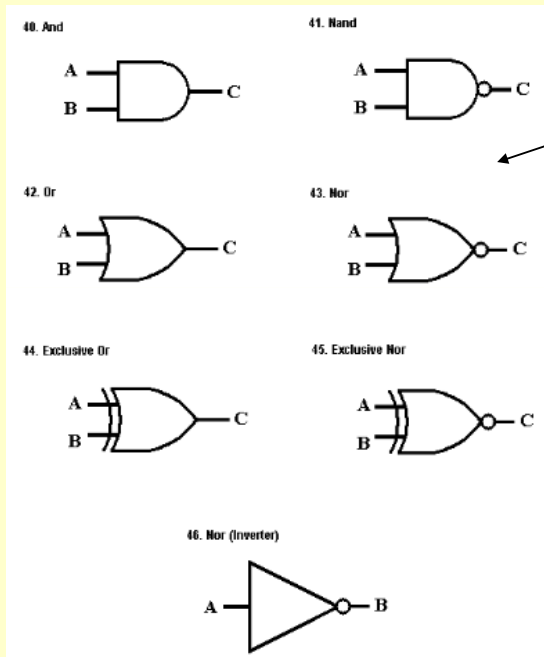
How to Use Propositional Logic?

Problem: Determine the validity of the following reasoning

« Paul can eat if he has cooked a meal in the last 30 minutes and he is hungry. He is hungry provided he has not eaten in the last 15 minutes and in the last 30 minutes he has not done any sport. If he is doing sport he cannot cook a meal at the same time. Therefore if Paul did not eat he had already eaten in the last 15 minutes or he did sport in the last 30 minutes »

How to Use Propositional Logic?

2: We can use propositional logic to model/design computer circuits



How to Use Propositional Logic?

2: We can use propositional logic to model/design computer circuits

Adequate set of connectives

DEFINITION:

A set of connectives, S , is *adequate* iff for every formula there is an *equivalent* formula with only connectives from that set.

QUESTION: What are the adequate sets of connectives for propositional logic?

How to Use Propositional Logic?

2: We can use propositional logic to model/design computer circuits

Normal forms:

- **Conjunctive Normal Form** (CNF—universal)
conjunction of disjunctions of literals
E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$
- **Disjunctive Normal Form** (DNF—universal)
disjunction of conjunctions of literals
E.g., $(A \wedge B) \vee (A \wedge \neg C) \vee (A \wedge \neg D) \vee (\neg B \wedge \neg C) \vee (\neg B \wedge \neg D)$
- **Horn Form** (restricted)
conjunction of *Horn clauses* (clauses with ≤ 1 positive literal)
E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$
Often written as set of implications: $B \Rightarrow A$ and $(C \wedge D) \Rightarrow B$

Definition: *normal* –

A form is *normal* iff for any formula, there is an equivalent formula written in that form.

QUESTION: How to prove that **CNF**, **DNF** and **HF** are normal

How to Use Propositional Logic?

2: We can use propositional logic to model/design computer circuits

We wish to reduce the following in constructing logic circuits –

- the number of different types of gates
- the total number of gates
- the depth of the circuit
- the width of the circuit

Execution Time?

NAND is adequate

Parallelisation/Synchronisation?

Why do we need predicate logic/calculus?

- Propositional logic: no variables or functions
- Predicate logic**: allows – through the use of *quantifiers* - *quantified variables* as parameters of predicates or functions.
- In **first order predicate logic** we must include a domain of discourse over which the quantifiers range.
- Higher order predicate logics**: allows predicates to be variables (might be needed to describe mathematical properties such as “every proposition implies itself” or “there are decidable propositions.”)

You use the logic which is appropriate for the properties/behaviour you wish to describe/reason about

How do we define predicate logic/calculus?

Just like with propositional logic, we need:

- A **language** which tells us how to build up sentences in the language (i.e., *syntax*), and what the sentences mean (i.e., *semantics*)
- An **inference procedure** which tells us which sentences are valid inferences from other sentences

But, we don't start from scratch – we can build predicate logic « on top of » the propositional logic that we have already defined.

The limitations of propositional logic – parameterised assertions

Propositional logic has no variables or functions, so how does this limit us in what we can express?

The predicate logic helps us to overcome these limitations

For example, the assertion "x is greater than 1", where x is a variable, is not a proposition because you can not tell whether it is true or false unless you know the value of x. Thus the propositional logic can not deal with such sentences. However, such assertions appear quite often in mathematical modelling and we want to do inferencing on those assertions

The limitations of propositional logic - equivalences

The **pattern** involved in the following *logical equivalences* can not be captured by the propositional logic:

- *Not **all** birds fly* is equivalent to ***Some** birds don't fly*.
- *Not **all** integers are even* is equivalent to ***Some** integers are not even*.
- *Not **all** cars are expensive* is equivalent to ***Some** cars are not expensive*,

We would like to be able to **generalise** to:

*NOT **ALL** Xs are U iff **SOME** Xs are not U*

The limitations of propositional logic – algorithmic reasoning

Propositional is not powerful enough to express statements such as:

- The function $f(x) = x^3$ is continuous.
- There is a prime between n and $2n$ (for $n > 1$).
- Deadlock cannot occur (in some protocol).
- Upon completion of this procedure, the stack is empty.
- The algorithm terminates on all inputs.

For example, the second assertion (known as the Bertrand-Chebyshev theorem) is of importance for algorithms that need to select primes of some suitable size

How could we state this assertion clearly, as a formal expression that might be computer-understandable? We cannot do it with propositional logic

Propositions, Sets and Predicates

A **proposition** is a sentence expressing something true or false.

A **set** can be defined by specifying a *property* that the elements of the set have in common

Predicates are a link between propositions and sets

For example, $\{x \mid x \text{ is a positive integer less than } 4\}$ is the set $\{1,2,3\}$.

Thus, an element of $\{x \mid P(x)\}$ is an object t for which the statement $P(t)$ is true. Such a sentence $P(x)$ is called a **Predicate**.

$P(x)$ is also called a **propositional function**, because each choice of x produces a proposition $P(x)$ that is either true or false

Propositions, Sets and Predicates – thinking in terms of relations

Formally there are different (equivalent) ways of giving semantics to predicates:

- an expression of the semantic type of sets.
- an indicator functions of sets, i.e. functions from an entity to a truth value.
- In first-order logic, a predicate can take the role as a *relation* between entities. (This is the point-of-view that will be most intuitive for us – for the moment)

Constructing Predicate Logic

So how do we construct a language suitable for (most of) mathematics and computer science and software engineering?

Let's start with a small fragment, say, arithmetic. A reasonable approach would be to use only:

- basic arithmetic concepts (addition, multiplication, order), and
- purely logical constructs such as “and”, “not” etc.

The logical constructs will include **all of propositional logic** so we can still combine assertions by connectives “and”, “or” and the like. But there will also be **quantifiers** that allow one to make statements about the existence of objects with certain properties and about properties of all objects.

This type of system is called **predicate logic**, but note that we should also allow for functions/relations.

Constructing Predicate Logic

Let's try a concrete example to see what an expression in predicate logic could look like:

There is a prime between n and $2n$ (for $n > 1$).

• for all x	- universal quantifier	The additional language constructs
• there exists a y	- existential quantifier	
• $x \leq y$	- binary relation	
• and	- logical connective	
• $y \leq 2x$	- binary relation	
• and	- logical connective	
• y is prime	- unary relation	

The assertion “is prime” in the last line still needs to be *formalised*.

Constructing Predicate Logic

Let's try a concrete example to see what an expression in predicate logic could look like:

There is a prime between n and $2n$ (for $n > 1$).

Clearly, we can express primality in terms of multiplication.

x is prime iff

• for all u	- universal quantifier
• for all v	- universal quantifier
• $x = u \cdot v$	- binary function, equation
• implies	- logical connective
• $u = 1$	- constant, equation
• or	- logical connective
• $v = 1$	- constant, equation

This is all we need: adding quantifiers increases the expressiveness of our language enormously.

Constructing Predicate Logic - The Range of Variables

Note that the definition of primality tacitly assumes that we interpret the variables as ranging over the integers.

If we were dealing with the rationals the definition would, of course, no longer make sense.

We will see in a moment that the range of quantifiers is pinned down precisely when we address the question of semantics: what exactly is the meaning of a formula in predicate logic?

Constructing Predicate Logic - The Syntax

We keep all logical connectives. But we add

- constants that denote individual objects,
- variables that range over individual objects,
- quantifiers that express “for all” and “there exists”,
- function symbols that denote functions,
- relation symbols that denote relations.

Constructing Predicate Logic - The Syntax

Notation:

- a, b, c, \dots for constants,
- x, y, z, \dots for variables,
- \forall for the *universal quantifier*,
- \exists for the *existential quantifier*,
- f, g, h, \dots for function symbols,
- R, P, Q, \dots for relation symbols.
- Always allow $=$ for equality.

We will need to treat this notation more formally at a later stage. First, let's consider some examples of possible usage.

Constructing Predicate Logic – Fully formal Notation

Fermat's last theorem: from informal to predicate logic

There do not exist four numbers, the last being larger than two, such that the sum of the first two, both raised to the power of the fourth, are equal to the third, also raised to the power of the fourth.

There are no positive integers x, y, z and n , where $n > 2$, such that $x^n + y^n = z^n$.

$$\neg \exists x, y, z, n \in \mathbb{N}^+ (x^n + y^n = z^n \wedge n > 2)$$

Predicate Logic – Expressiveness

Induction:

$$R(0) \wedge \forall x (R(x) \rightarrow R(x + 1)) \rightarrow \forall x R(x)$$

Here we have added another unary relation symbol R to our language. So $R(x)$ asserts that number x has some unspecified property.

The *Principle of Induction* then simply asserts that this formula is true over the natural numbers.

Note, though, that in many applications $R(x)$ would actually be replaced by a formula of arithmetic such as:

$$\sum_{i \leq x} i = x(x + 1)/2$$

Predicate Logic – Components of a formula

The components of a formula can be organized into a taxonomy like so:

- variables, constants and terms,
- equations,
- atomic formulae,
- propositional connectives, and
- quantifiers.

Only the quantification part is really new; propositional connectives will be the same as in *propositional logic* while terms and equations will be the same as in *equational logic*.

Predicate Logic – *Equational Logic (a quick review)*

The terms of equational logic are built up from variables and constants using function symbols (or operations). Identities (equalities) of the form

$$s=t,$$

where s and t are terms, constitute the formal language of equational logic. The syllogisms/axioms of equational logic are: reflexivity, symmetry, transitivity, “application” and substitution/unification.

NOTE: We will re-examine equational logic when we look at abstract algebra, for now you should have an intuitive understanding

Predicate Logic – *Equational Logic*

1. Reflexivity:

$$\frac{}{s = s}.$$

2. Symmetry:

$$\frac{s = t}{t = s}.$$

3. Transitivity:

$$\frac{s = t, t = v}{s = v}.$$

4. For f a function symbol and $n \geq 0$,

$$\frac{s_1 = t_1, \dots, s_n = t_n}{f(s_1, \dots, s_n) = f(t_1, \dots, t_n)}.$$

5. For θ a substitution (cf. **unification**),

$$\frac{s = t}{s\theta = t\theta}.$$

Equality plays a special role in our setup (and sometimes has a special symbol associated with it: ‘ \approx ’... although we will stick to ‘=’)

We will assume that the language has a special binary relation symbol that is used to denote equality:

$$s = t$$

where s and t are arbitrary terms.

Unlike with all the other relation symbols, the meaning of ‘=’ is fixed once and for all: it is always interpreted as equality.

We will always assume that equality is part of the language, though predicate logic without equality is also of interest.

Predicate Logic – Adding quantifiers to formulae

Formulae in predicate logic can be built from:

- Propositional logic
- Equational logic
- Universal Quantifiers:

If φ is a formula and x a variable, then $(\exists x \varphi)$ and $(\forall x \varphi)$ are also formulae.

Predicate Logic – Free (unbounded) variables in sentences

DEFINITION

A variable that is not in the range of a quantifier is *free* or *unbound* (as opposed to bound).

A formula without free variables is closed, or a *sentence*.

One often indicates free variables like so:

$\varphi(x, y)$	x and y are free
$\exists x \varphi(x, y)$	only y is free
$\forall y \exists x \varphi(x, y)$	closed.

Predicate Logic – Substitution of (ground) terms for free variables

$\varphi(s, t)$	replace x by s , y by t
$\varphi[s/x, t/y]$	replace x by s , y by t

A priori, a formula $\Phi(x)$ with a free variable x has no truth value associated with it: we need to replace x by a ground term to be able to evaluate.

However, taking inspiration from equational logic, it is convenient to define the truth value a formula with free variables to be the same as its universal closure: put universal quantifiers in front, one for each free variable.

$$\forall x, y, z (x * (y * z) \approx (x * y) * z)$$

$$x * (y * z) \approx (x * y) * z$$

Predicate Logic – Clashing Variables

Note that according to our definition it is perfectly agreeable to quantify over an already bound variable. To make the formula legible one then has to rename variables. For practical reasons it is best to simply disallow clashes between free and bound variables.

$$\forall x (R(x) \wedge \exists x \forall y S(x, y))$$

Better: rename the x inside

$$\forall x (R(x) \wedge \exists z \forall y S(z, y))$$

These issues are very similar to problems that arise in programming languages (global and local variables, scoping issues). They need to be addressed but are not of central importance.

Predicate Logic – What is « truth »

How can we explain precisely what it means for an assertion to be true, to be valid? We would like some sweeping, global definition of truth that handles all areas of discourse, at least in mathematics and computer science. While this broad approach corresponds nicely to one's philosophical assumptions about the world it leads to some rather dicey problems: for example, what should we do with assertions like

“This sentence is false.”

QUESTION: Is there an equivalent problem with sets?

We certainly would want to have the ability to declare certain assertions such as

“100 is a prime number” as false.

But, how can we then avoid paradoxes as in the **self-referential** statement above?

Tarski's Theory of Truth – **the semantics of predicate logic**

In order to resolve the issue raised in the previous slide, one needs to distinguish carefully between an object language and a meta-language.

It is also helpful to restrict one's attention to truth in a particular domain such as, say, group theory or number theory. (We will see these later in the module)

In the 1930's Alfred Tarski was the first to seriously tackle to problem of explaining truth for a sentence in a formal language.

We already have a nice formal language for predicate logic

We can now use Tarski's ideas to attach meaning to a formula, to establish a relationship with the world of actual objects such as numbers, functions, hash tables, algorithms, and so on.

The key is to define the truth value of a formula relative to a given structure, a mini-universe that allows us to make sense out of the components of the formula.

The semantics of predicate logic – validity of formula

To formally define truth (or validity) in predicate logic one must formally define the relevant structures (mini universes of discourse):

- Natural numbers
 - Reals
 - Abstract data types
 - ...
- For now, we will deal with this intuitively (but as formally as possible)

Specifications in Predicate Logic

Which of the following formulae are valid (true), and why?

- 1 lemma " $(\exists x. \forall y. P x y) \longrightarrow (\forall y. \exists x. P x y)$ "
- 2 lemma " $(\forall x. P x \longrightarrow Q) = ((\exists x. P x) \longrightarrow Q)$ "
- 3 lemma " $((\forall x. P x) \wedge (\forall x. Q x)) = (\forall x. (P x \wedge Q x))$ "
- 4 lemma " $((\forall x. P x) \vee (\forall x. Q x)) = (\forall x. (P x \vee Q x))$ "
- 5 lemma " $((\exists x. P x) \vee (\exists x. Q x)) = (\exists x. (P x \vee Q x))$ "
- 6 lemma " $(\forall x. \exists y. P x y) \longrightarrow (\exists y. \forall x. P x y)$ "
- 7 lemma " $(\neg (\forall x. P x)) = (\exists x. \neg P x)$ "

Specifications in Predicate Logic: translating from English

“Every student in this class has studied calculus.”

- 1 First, we rewrite the statement so that we can easily identify the appropriate quantifiers to use.
“Every student, if he is in this class, then he has studied calculus.”
- 2 Next, we introduce a variable x for the student:
“For every student x , if x is in this class, then x has studied calculus.”
- 3 We define the predicates
 $C(x)$ for the statement “ x is in this class.”
 $S(x)$ for the statement “ x has studied calculus.”
- 4 The initial statement becomes $\forall x. C(x) \rightarrow S(x)$.

Specifications in Predicate Logic: translating from English

- 1 There is no dog that can talk. = $\underbrace{\text{There is no}}_{\neg\exists} \boxed{\text{dog } d} \text{ and } \boxed{d \text{ can talk.}} = \neg\exists d. (\text{dog}(d) \wedge \text{talk}(d)).$
- 2 There is no one in this class who knows French and Russian =
 $\neg\exists x. (\text{inclass}(x) \wedge \text{knows}(x, \text{French}) \wedge \text{knows}(x, \text{Russian})).$

where

- $\text{inclass}(x)$ means; “ x is in this class.”
- $\text{knows}(x, y)$ means: “ x knows language y .”

Specifications in Predicate Logic: translating from English

The order of quantifiers in mathematical statements is important.

Example

Let $P(x, y)$ be the statement $x < y$.

- What is the truth value of the formula $\forall x \exists y. P(x, y)$?
- What is the truth value of the formula $\exists y \forall x. P(x, y)$?
- $\forall x \exists y. P(x, y)$ means: "For every real number x there exists a real number y such that $x < y$." This statement is **true** because there always exists the real number $y = x + 1$, and $x < x + 1 = y$.
- $\exists y \forall x. P(x, y)$ means: "There exists a real number y such that $x < y$ for any real number x ." This statement is **false** because

NOTE: This disproves lemma 6 (slide 56)

Specifications in Predicate Logic: translating from English

General rules 1 and 2 can be used to propagate negation inside formulas with nested quantifiers.

Example (limit)

A function f has limit L at a , notation $\lim_{x \rightarrow a} f(x) = L$, if
 $\forall \epsilon > 0 \exists \delta > 0. \forall x (0 < |x - a| < \delta \rightarrow |f(x) - L| < \epsilon)$.

Use quantifiers and predicates to express the fact that f does not have limit L at a .

" f does not have limit L at a " $\equiv \neg(\lim_{x \rightarrow a} f(x) = L) \equiv$

$$\begin{aligned} &\neg \forall \epsilon > 0 \exists \delta > 0 \forall x. (0 < |x - a| < \delta \rightarrow |f(x) - L| < \epsilon) \equiv \\ &\exists \epsilon > 0 \neg \exists \delta > 0 \forall x. (0 < |x - a| < \delta \rightarrow |f(x) - L| < \epsilon) \equiv \\ &\exists \epsilon > 0 \forall \delta > 0 \neg \forall x. (0 < |x - a| < \delta \rightarrow |f(x) - L| < \epsilon) \equiv \\ &\exists \epsilon > 0 \forall \delta > 0 \exists x. \neg(0 < |x - a| < \delta \rightarrow |f(x) - L| < \epsilon) \equiv \\ &\exists \epsilon > 0 \forall \delta > 0 \exists x. (0 < |x - a| < \delta \wedge |f(x) - L| \geq \epsilon) \end{aligned}$$

because, in general, $\neg(A \rightarrow B) \equiv (A \wedge \neg B)$.

Specifications in Predicate Logic

If a student₁ is good-at-logic, he₁ does-well-in-semantics.

QUESTION: Consider and judge the following possibilities:

- i. $[\exists x(\text{ST}(x) \wedge \text{LOG}(x))] \rightarrow \text{SEM}(x)$
- ii. $\exists x [(\text{ST}(x) \wedge \text{LOG}(x)) \rightarrow \text{SEM}(x)]$
- iii. $\forall x [(\text{ST}(x) \wedge \text{LOG}(x)) \rightarrow \text{SEM}(x)]$
- iv. $\exists x [\text{ST}(x) \wedge (\text{LOG}(x) \rightarrow \text{SEM}(x))]$

Specifications in Predicate Logic

Express the following using predicate logic

1. *Susan introduced Mary to a student that nobody liked.*
2. *Only-if John has-talked-to every witness will Mary be-satisfied.*
3. *John introduced only MARY to Kate.*
4. *John introduces Mary only to KATE.*

Specifications in Predicate Logic

Express the following using predicate logic

1. *All students in the class can program in Java or C++*
2. *Some of the students can program in Java or C++*
3. *Every student who could not program in Java or C++ was paired with a student who could*
4. *Some students program in Java some of the time, others never program in Java*

Proofs in Predicate Logic: additional rules for quantification

rule of inference	name
$\frac{\forall x P(x)}{P(c)}$	Universal instantiation
$\frac{P(c) \text{ for an arbitrary } c}{\forall x P(x)}$	Universal generalization
$\frac{\exists x P(x)}{P(c) \text{ for some element } c}$	Existential instantiation
$\frac{P(c) \text{ for some element } c}{\exists x P(x)}$	Existential generalization

Proofs in Predicate Logic: example

Show that the premises

- ① "A student in this class has not read the book."
- ② "Everyone in this class passed the exam."

imply "Someone who passed the exam has not read the book."

$C(x)$ = "x is in this class."

$B(x)$ = "x has read the book."

$P(x)$ = "x passed the exam."

We must show that $\exists x (P(x) \wedge \neg B(x))$ holds under the hypotheses $\exists x (C(x) \wedge \neg B(x))$ and $\forall x (C(x) \rightarrow P(x))$.

Proofs in Predicate Logic: example, continued

Step

- | | |
|--|-------------------------------------|
| 1. $\exists x (C(x) \wedge \neg B(x))$ | 1. Hypothesis |
| 2. $\forall x (C(x) \rightarrow P(x))$ | 2. Hypothesis |
| 3. $C(a) \wedge \neg B(a)$ | 3. Existential Instantiation from 1 |
| 4. $C(a)$ | 4. Simplification from 3 |
| 5. $C(a) \rightarrow P(a)$ | 5. Universal Instantiation from 2 |
| 6. $P(a)$ | 6. Modus Ponens from 4 and 5 |
| 7. $\neg B(a)$ | 7. Simplification from 3 |
| 8. $P(a) \wedge \neg B(a)$ | 8. Conj from 6 and 7 |
| 9. $\exists x (P(x) \wedge \neg B(x))$ | 9. Existential General. from 8 |

Thus, 9 can be inferred from the hypotheses (1 and 2)

Proofs in Predicate Logic: example, continued

QUESTION: choose **one** of these lemmas - which you believe to be true - and prove it formally (using the stepped approach, as seen in the previous example)

- 1 lemma " $(\exists x. \forall y. P x y) \longrightarrow (\forall y. \exists x. P x y)$ "
- 2 lemma " $(\forall x. P x \longrightarrow Q) = ((\exists x. P x) \longrightarrow Q)$ "
- 3 lemma " $((\forall x. P x) \wedge (\forall x. Q x)) = (\forall x. (P x \wedge Q x))$ "
- 4 lemma " $((\forall x. P x) \vee (\forall x. Q x)) = (\forall x. (P x \vee Q x))$ "
- 5 lemma " $((\exists x. P x) \vee (\exists x. Q x)) = (\exists x. (P x \vee Q x))$ "
- 6 lemma " $(\forall x. \exists y. P x y) \longrightarrow (\exists y. \forall x. P x y)$ "
- 7 lemma " $(\neg (\forall x. P x)) = (\exists x. \neg P x)$ "

Hint: If you have problems choosing then we all agree that lemma 7 is true (intuitively), ...

Validity and Satisfiability in Predicate Logic

We begin by defining assignments in the context of predicate logic. We note that we need to use the notion of a structure (in the universe of discourse)

Definition: An **assignment** or **valuation** (over a structure A) associates variables of the language with elements in the ground set A .

Given an assignment $\sigma : \text{Var} \rightarrow A$, we can associate an element $\sigma(t)$ in A with each term t .

- $t = x$: then $\sigma(t) = \sigma(x)$
- $t = f(r_1, \dots, r_n)$: then $\sigma(t) = f^A(\sigma(r_1), \dots, \sigma(r_n))$

Example 2. Over \mathcal{N} let $\sigma(x) = 3$. Then $\sigma(x \cdot (1 + 1)) = 6$ whereas $\sigma(x) = 0$ produces $\sigma(x \cdot (1 + 1)) = 0$.

Validity and Satisfiability in Predicate Logic

Once we have an assignment for all the free variables in an atomic formula we can determine a truth value for it.

Definition: Let σ be an assignment over a structure A and $\Phi = R(t_1, \dots, t_n)$ be an atomic formula.

Define the **truth value Φ (under σ over A)** to be -

$$\mathcal{A}_\sigma(\varphi) = \begin{cases} \text{tt} & \text{if } R^A(\sigma(t_1), \dots, \sigma(t_n)) \text{ holds,} \\ \text{ff} & \text{otherwise.} \end{cases}$$

Example 3. Over the natural numbers \mathcal{N} suppose $\sigma(x) = 0$ and $\sigma(y) = 1$. Then

$$\mathcal{N}_\sigma(x + y < 1 + 1) = \mathcal{N}_\sigma(0 + 1 < 1 + 1) = \text{tt}$$

but for $\sigma(x) = \sigma(y) = 1$ we get

$$\mathcal{N}_\sigma(x + y < 1 + 1) = \mathcal{N}_\sigma(1 + 1 < 1 + 1) = \text{ff}$$

This truth value is extended to logical connectives and quantifiers in the natural way

Validity and Satisfiability in Predicate Logic

Definition: Validity

A formula φ is **valid in \mathcal{A} under assignment σ** if $\mathcal{A}_\sigma(\varphi) = 1$.

A formula φ is **valid in \mathcal{A}** if it is valid in \mathcal{A} for all assignments σ . The structure \mathcal{A} is then said to be a **model** for φ or to **satisfy** φ .

A sentence is **valid (or true)** if it is valid over any structure (of the appropriate signature).

Notation:

$$\mathcal{A} \models_\sigma \varphi, \quad \mathcal{A} \models \varphi, \quad \models \varphi$$

One uses the same notation for sets of formulae Γ . So $\mathcal{A} \models \Gamma$ means that $\mathcal{A} \models \varphi$ for all $\varphi \in \Gamma$.

Note the condition for validity: the formula has to hold in all structures.

Validity and Satisfiability in Predicate Logic

Definition: Satisfiability

A formula is **satisfiable** if there is some structure \mathcal{A} and some assignment σ for all the free variables in φ such that $\mathcal{A} \models_{\sigma} \varphi$.

Computing truth

More precisely, suppose we wanted to construct an algorithm

$\text{ValidQ}(\mathcal{A}, \varphi)$

that checks if formula φ is valid over structure \mathcal{A} .

What would be the appropriate input for such an algorithm?

The easy part is the formula: any standard representation will be fine.

The real problem is the structure \mathcal{A} .

For standard structures such as the natural numbers or reals we understand (more or less) how to interpret the operations. But in the general case we need some representation.

We will look at *algebraic structures* later in this module