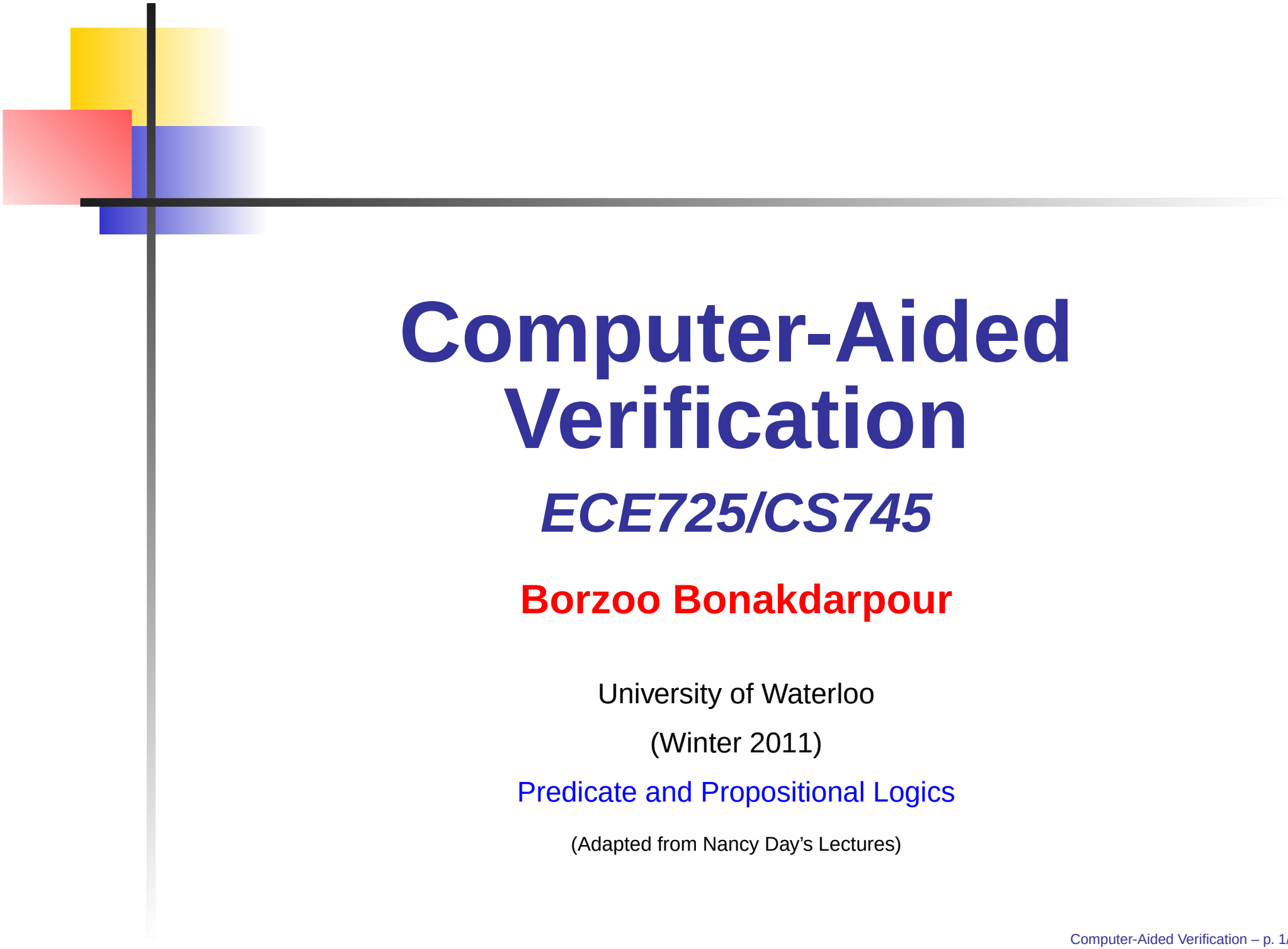


Computer-Aided Verification

ECE725/CS745

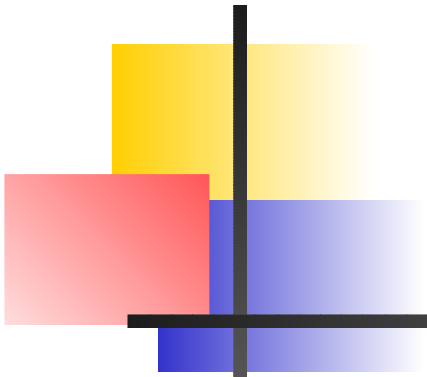
Borzoo Bonakdarpour

University of Waterloo

(Winter 2011)

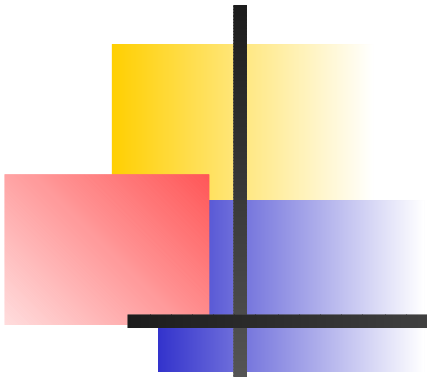
Predicate and Propositional Logics

(Adapted from Nancy Day's Lectures)



Agenda

- What is verification?
- What is logic?
- Propositional logic
- Predicate logic



Agenda

- *What is verification?*
- What is logic?
- Propositional logic
- Predicate logic

What is Verification?

Verification involves checking a *satisfaction* relation, usually in the form of a sequent:

$$\mathcal{M} \models \phi, \text{ where}$$

- $\mathcal{M}$ is a *model*,

What is Verification?

Verification involves checking a *satisfaction* relation, usually in the form of a sequent:

$$\mathcal{M} \models \phi, \text{ where}$$

- $\mathcal{M}$ is a *model*,
- ϕ is a *property* (or specification)

What is Verification?

Verification involves checking a *satisfaction* relation, usually in the form of a sequent:

$$\mathcal{M} \models \phi, \text{ where}$$

- $\mathcal{M}$ is a *model*,
- ϕ is a *property* (or specification)
- $\models$ is a relationship that should hold between $\mathcal{M}$ and ϕ; i.e., $(\mathcal{M}, \phi) \in \models$

What is Verification?

Verification involves checking a *satisfaction* relation, usually in the form of a sequent:

$$\mathcal{M} \models \phi, \text{ where}$$

- $\mathcal{M}$ is a *model*,
- ϕ is a *property* (or specification)
- $\models$ is a relationship that should hold between $\mathcal{M}$ and ϕ; i.e., $(\mathcal{M}, \phi) \in \models$

  We say that the model *satisfies* or "has" the property, or that we can conclude the property from the model.

What is Verification?

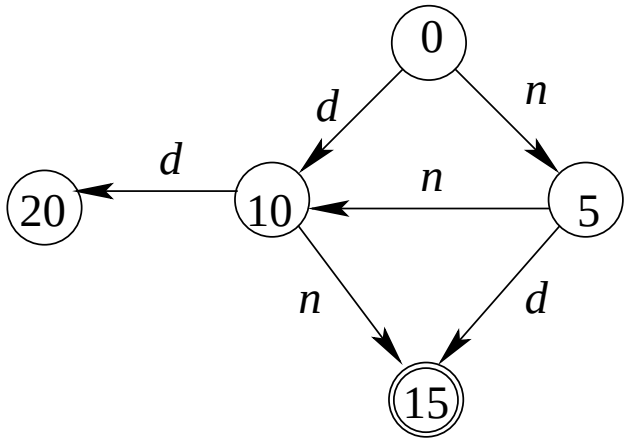
Verification involves:

1. specifying the model/system/implementation
2. specifying the property/specification
3. choosing the satisfaction relation
4. checking the satisfaction relation

These 4 steps are *NOT* independent.

Example

Consider the operation of a soft drink vending machine which charges 15 cents for a can. The following figure is a *model* $\mathcal{M}$ of such machine.

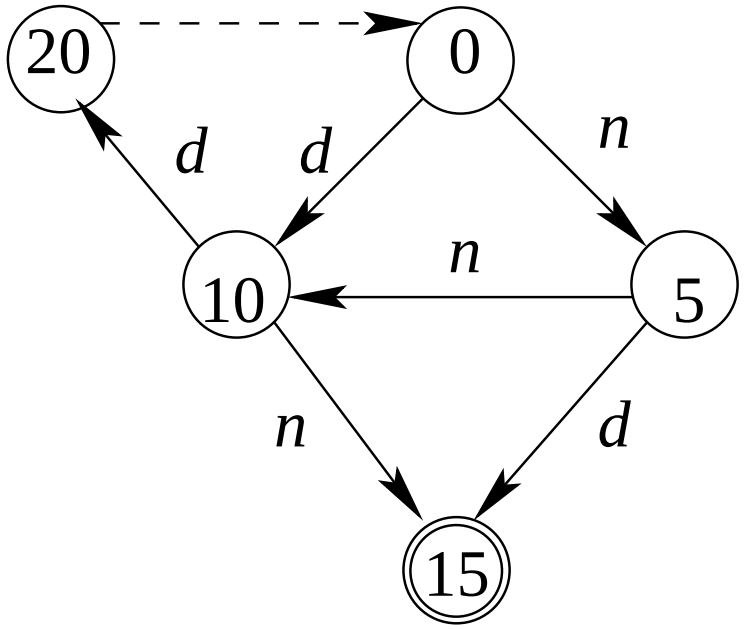


The following regular expression *specifies* the acceptable behavior of the machine: $\phi = n(d + nn)$

Question. $\mathcal{M} \models \phi$?

Example (cont'd)

What about this model?



Models and Properties

- The term *model* is used loosely here. It might not be executable, and it might not be a complete description of the system's behaviour. The terms *implementation* and *specification* are relative.

Models and Properties

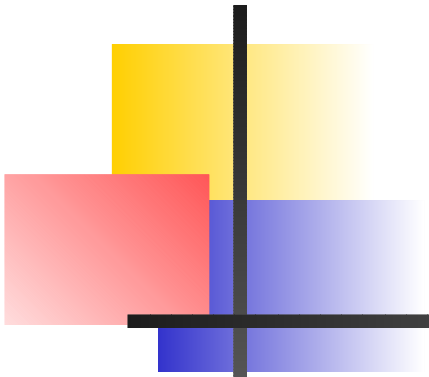
- The term *model* is used loosely here. It might not be executable, and it might not be a complete description of the system's behaviour. The terms *implementation* and *specification* are relative.
- An implementation generally contains more details than a specification. The specification for one level of verification might be the implementation at a higher level of verification.

Models and Properties

- The term *model* is used loosely here. It might not be executable, and it might not be a complete description of the system's behaviour. The terms *implementation* and *specification* are relative.
- An implementation generally contains more details than a specification. The specification for one level of verification might be the implementation at a higher level of verification.
- In hardware, often the model is a description of the circuit in a hardware description language such as VHDL or Verilog. The real thing is the physical realization of the chip.

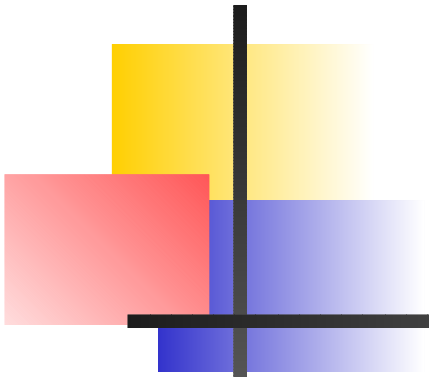
Models and Properties

- The term *model* is used loosely here. It might not be executable, and it might not be a complete description of the system's behaviour. The terms *implementation* and *specification* are relative.
- An implementation generally contains more details than a specification. The specification for one level of verification might be the implementation at a higher level of verification.
- In hardware, often the model is a description of the circuit in a hardware description language such as VHDL or Verilog. The real thing is the physical realization of the chip.
- Sometimes the model is actually a specification and the property is an attribute such as completeness or consistency.



Logic and Verification

- Different modelling languages and logics give us different ways of expressing $\mathcal{M}$ and ϕ and defining membership of the pair $(\mathcal{M}, \phi)$ to $\models$.



Logic and Verification

- Different modelling languages and logics give us different ways of expressing $\mathcal{M}$ and ϕ and defining membership of the pair $(\mathcal{M}, \phi)$ to $\models$.

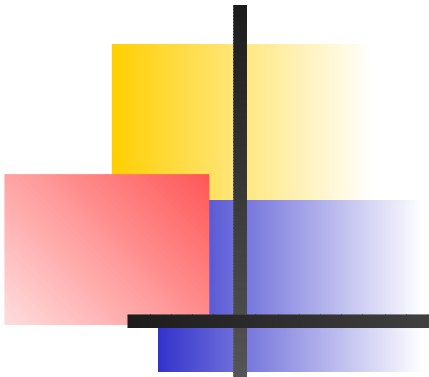
- Hopefully, the calculation of the satisfaction relation is *compositional* in either the property or the model. This decomposes the verification task.

Logic and Verification

- Different modelling languages and logics give us different ways of expressing $\mathcal{M}$ and ϕ and defining membership of the pair $(\mathcal{M}, \phi)$ to $\models$.
- Hopefully, the calculation of the satisfaction relation is *compositional* in either the property or the model. This decomposes the verification task.
- The model and property both describes sets of *behaviours*.

Logic and Verification

- Different modelling languages and logics give us different ways of expressing $\mathcal{M}$ and ϕ and defining membership of the pair $(\mathcal{M}, \phi)$ to $\models$.
- Hopefully, the calculation of the satisfaction relation is *compositional* in either the property or the model. This decomposes the verification task.
- The model and property both describes sets of *behaviours*.
- The satisfaction relation is a relation between the set of behaviours of the model and the set of behaviours of the property.



Agenda

- What is verification?
- *What is logic?*
- Propositional logic
- Predicate logic

What is Logic?

According to Kelly:

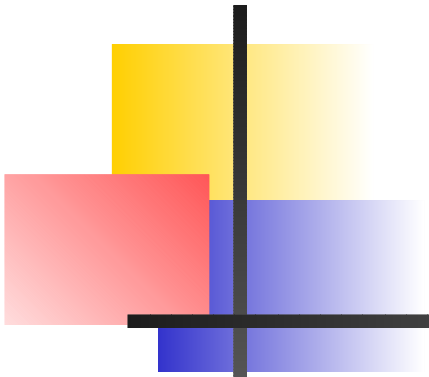
In general, *logic* is about reasoning. It is about the *validity* of arguments, *consistency* among statements (. . .) and matters of *truth* and *falsehood*.

In a formal sense logic is concerned only with the *form of arguments* and the principles of *valid inferencing* .

What is Logic?

According to Webster's, logic is:

the science of *correct reasoning*, valid induction or deduction. Symbolic logic is a modern type of formal logic using special mathematical symbols for propositions, quantifiers, and relationships among propositions and concerned with the elucidation of permissible operations upon such symbols.



Elements of a Logic

A logic consists of:

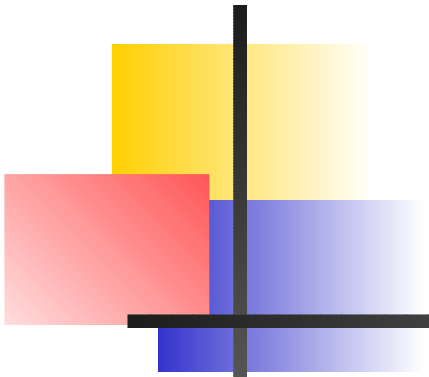
1. syntax
2. semantics
3. proof procedure(s) (also called proof theory)

Syntax and Semantics

- syntax:
  - define “well-formed formula”
- semantics:
  - define “ $\models$ ” (“satisfies”)
    $\mathcal{M} \models \phi$ (satisfaction relation)
  - define $\phi_1, \phi_2, \phi_3 \models \psi$ (“entails”, or semantic entailment) means:
    from the *premises* ϕ_1, ϕ_2, ϕ_3, we may conclude ψ, where ϕ_1, ϕ_2, ϕ_3, and are all well-formed formulae in the logic.

Proof Procedure

- proof procedure(s):
  - define " $\vdash$ " (pronounced "proves")
  - a proof procedure is a way to calculate $\phi_1, \phi_2, \phi_3, \cdots \vdash \psi$ (also called a sequent). By "calculation", we mean that there is a procedure for determining if $((\phi_1, \phi_2, \phi_3, \cdots), \psi) \in \vdash$
  - there may be multiple proof procedures, which we will indicate by subscripting $\vdash$, e.g., the *sequent calculus* proof procedure for propositional logic will be $\vdash_{\mathsf{SQ}}$.
  - for some logics, there is not a proof procedure that always terminates for any sequent.



Proof Procedures

Proof procedures are algorithms that perform mechanical manipulations on strings of symbols. A proof procedure does not make use of the meanings of sentences, it only manipulates them as formal strings of symbols.

There may be multiple ways to prove a sequent in a particular proof procedure.

Soundness and Completeness

The semantics and the proof procedures ($\models$ and $\vdash$) are related in the concepts of *soundness* and *completeness*.

Definition. A proof procedure is *sound* if $\phi_1, \phi_2, \phi_3 \vdash \psi$ then $\phi_1, \phi_2, \phi_3 \models \psi$.
A proof procedure is sound if it proves only tautologies.

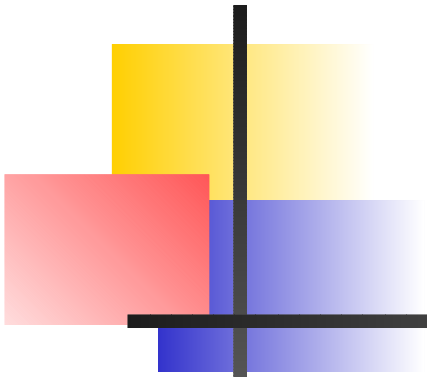
Definition. A proof procedure is *complete* if $\phi_1, \phi_2, \phi_3 \models \psi$ then $\phi_1, \phi_2, \phi_3 \vdash \psi$.
A proof procedure is complete if it proves every tautology.

Note that in the literature, there is not consistent use of the symbols $\models$ and $\vdash$.

Consistency

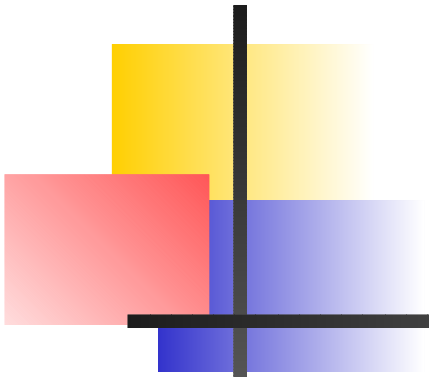
Definition. A proof procedure is *consistent* if it is not possible to prove both A and $\neg A$, i.e.,

not both $\vdash A$ and $\vdash \neg A$.



Agenda

- What is verification?
- What is logic?
- *Propositional logic*
- Predicate logic



Propositional Logic: Syntax

Its syntax consists of:

- Two constant symbols: **true** and **false**
- Proposition letters
- Propositional connectives
- Brackets

Propositional Connectives

Definition. The propositional (logical) connectives are:

Symbol	Informal Meaning
$\neg$	negation (not)
$\wedge$	conjunction (and, both)
$\vee$	disjunction (or, at least one of)
$\Rightarrow$	implication (implies, logical consequence, conditional, if . . . then)
$\Leftrightarrow$	equivalent (biconditional, if and only if)

Others may use different symbols for these operations.

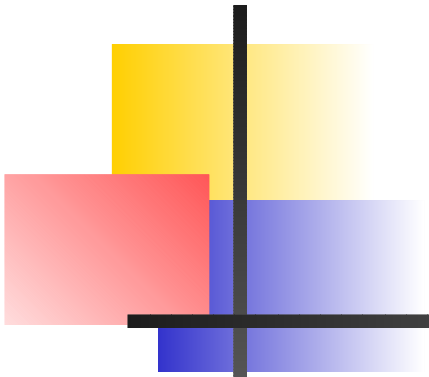
Terminology

For an implication $p \Rightarrow q$:

- p is the *premise* or *antecedent* or *hypothesis*
- q is the *consequent* or *conclusion*

$\neg b \Rightarrow \neg a$ is called the *contrapositive* of $a \Rightarrow b$.

The set of connectives $\{\wedge, \neg\}$ are complete in the sense that all the other connectives can be defined using them, e.g., $a \vee b = \neg(\neg a \wedge \neg b)$. Other subsets of the binary connectives are also complete in the same sense.



Propositions

Definition. *Proposition letters* represent declarative sentences, i.e., sentences that are true or false. Sentences matching proposition letters are atomic (non-decomposable), meaning they don't contain any of the propositional connectives.

Here are some examples:

- It is raining outside.
- The sum of 2 and 5 equals 3.
- The value of program variable a is 42.

Sentences that are interrogative (questions), or imperative (commands) are not propositions.

Using Symbols

Because in logic, we are only concerned with the structure of the argument and which structures of arguments are valid, we "encode" the sentences in symbols to create a more compact and clearer representation of the argument. We call these propositional symbols or proposition letters.

DO NOT use T, F, t, or f in any font as symbols representing sentences!

Well-formed formulas

Definition. The *well-formed formulae* of propositional logic are those obtained by the following construction rules:

- **true**, **false**, and the proposition letters are atomic formulas.
- If a is an atomic formula, then a is a formula.
- If p and q are formulas, then each of the following are formulas:

$$(\neg p) \quad (p \wedge q) \quad (p \vee q) \quad (p \Rightarrow q) \quad (p \Leftrightarrow q)$$

No other expressions are formulas.

Note that this is an inductive definition, meaning the set is defined by basis elements, and rules to construct elements from elements in the set.

Well-formed formulas

Brackets around the outermost formula are usually omitted. Brackets can be omitted using the following rules of precedence of operators: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$.

Associativity: $\Rightarrow$ is right associative meaning $p \Rightarrow q \Rightarrow r$ is $p \Rightarrow (q \Rightarrow r)$.

Note: Some texts do not use exactly these rules of precedence, they rank $\wedge$ and $\vee$ at the same level of precedence, and $\Rightarrow$ and $\Leftrightarrow$ at the same level of precedence.

Semantics

Semantics means “meaning”. Semantics relate two worlds. Semantics provide an interpretation (mapping) of expressions in one world in terms of values in another world. Semantics are often a *function* from expressions in one world to expressions in another world.

The semantics (i.e., the mapping) is often called a *model* or an *interpretation*. We write $\mathcal{M} \models \phi$ to mean the model satisfies the formula. In propositional logic, models are called Boolean valuations.

Proof procedures transform the syntax of a logic in ways that respect the semantics.

Semantics of Propositional Logic

We have described the syntax for propositional logic, which is the domain of the semantic function.

Classical logic is two-valued. The two possible truth values are **T**, and **F**, which are two distinct values.

The range of the semantic function for propositional logic is the set of truth values:

$$\mathbf{Tr} = \{\mathbf{T}, \mathbf{F}\}$$

Note that these truth values are distinct from the syntax elements **true**, and **false**.

Semantics of Propositional Logic

Truth Values: **Tr** = {**T**, **F**}

There are functions on these truth values that correspond to the meaning of the propositional connectives. We overload the operators "$\wedge$", "$\vee$", etc. to be both part of the syntax of propositional logic, and operations on the sets of truth values in our model for propositional logic.

$\neg : \mathbf{Tr} \rightarrow \mathbf{Tr}$
$\wedge : (\mathbf{Tr} \times \mathbf{Tr}) \rightarrow \mathbf{Tr}$
etc.

Truth tables are used to describe the functions of operations on these truth values.

Truth Tables

p	$\neg p$
T	**F**
F	**T**

p	q	$p \wedge q$	$p \vee q$	$p \Rightarrow q$	$p \Leftrightarrow q$
T	**T**	**T**	**T**	**T**	**T**
T	**F**	**F**	**T**	**F**	**F**
F	**T**	**F**	**T**	**T**	**F**
F	**F**	**F**	**F**	**T**	**T**

Boolean Valuations

Definition. A *Boolean valuation* is a mapping v from the set of propositional formulas to the set $\mathbf{Tr}$ meeting the conditions:

- $v(\mathbf{true}) = \mathbf{T}, v(\mathbf{false}) = \mathbf{F}$
- $v(\neg p) = \neg(v(p))$
- for all the connectives: $v(p \circ q) = v(p) \circ v(q)$

Note that $\neg(v(p))$ and $v(p) \circ v(q)$ are given by the truth tables on the previous slide.

Satisfiability

Definition. A formula a is *satisfiable* if there is a Boolean valuation v such that $v(a) = \mathbf{T}$.

We sometimes say that the formula “has a satisfying assignment” to mean that it is satisfiable.

We are mostly interested in the propositional formulas that map to T in all the possible Boolean valuations (i.e., in all model).

Tautologies

Definition. A propositional formula a is a *tautology* (also called valid or a theorem) if $v(a) = \mathbf{T}$ for every Boolean valuation v.

i.e., , a tautology is a formula that is true for all possible truth values of the propositional letters used in the formula. The last column of the truth table for a tautology contains all $\mathbf{T}$.

Note that a formula a is a tautology iff $\neg a$ is not satisfiable.

Semantic Entailment

$$\phi_1, \phi_2, \phi_3 \models \psi$$

means that for all v, if $v(\phi_1) = \mathbf{T}$ and $v(\phi_2) = \mathbf{T}$ and $v(\phi_3) = \mathbf{T}$, then $v(\psi) = \mathbf{T}$, which is equivalent to saying

$$(\phi_1 \wedge \phi_2 \wedge \phi_3) \Rightarrow \psi$$

is a tautology, i.e.,

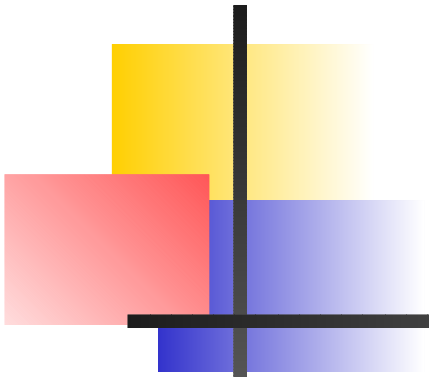
$$\boxed{(\phi_1, \phi_2, \phi_3 \models \psi) \quad \equiv \quad (\phi_1 \wedge \phi_2 \wedge \phi_3) \Rightarrow \psi}$$

Models and Entailment

In propositional (and predicate) logic, $\models$ is overloaded and has two meanings:

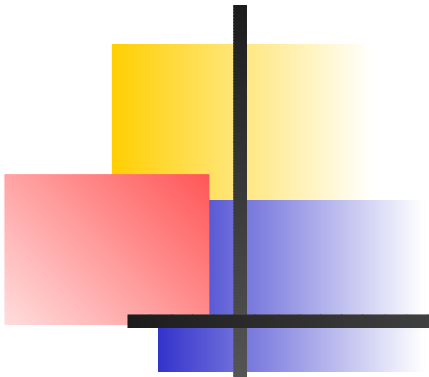
- $\mathcal{M} \models \phi$ relates a model to a formula, saying that $\mathcal{M}$ satisfies the formula. This is called a *satisfaction relation*.
- $\psi \models \phi$ relates two formulas, saying that forall v (i.e., for all possible models), if $v(\psi) = \mathbf{T}$, then $v(\phi) = \mathbf{T}$. This is called *semantic entailment*.

These two uses can be distinguished by their context.



Falsehood

Definition. A *falsehood* (contradiction) is a formula that is false for all possible truth values of the propositional symbols used in the formula. The last column of the truth table for a tautology contains all $\mathbf{F}$.



Decidability

A logic is *decidable* if there is an algorithm to determine if any formula of the logic is a tautology (is a theorem, is valid).

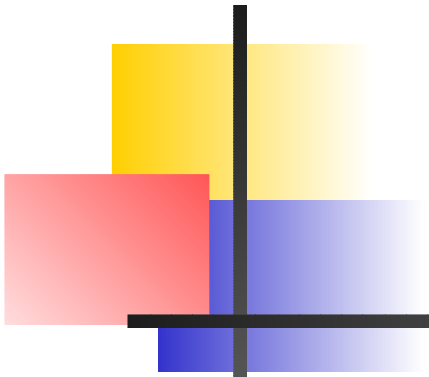
Propositional logic is decidable because we can always construct the truth table for the formula.

Question: What is the *complexity* of deciding propositional logic? Discuss!

Proof Procedures

We can always determine if a formula is a tautology by using truth tables to determine the value of the formula for every possible combination of values for its proposition letters, but this would be very tedious since the size of the truth table grows *exponentially*.

Proof procedures for propositional logic are alternate means to determine tautologies. As long as the proof procedure is sound, we can use the proof procedure in place of truth tables to determine tautologies.



Proof Styles

A *proof procedure* is a set of rules we use to transform premises and conclusions into new premises and conclusions.

A *goal* is a formula that we want to prove is a tautology. It has premises and conclusions.

A *proof* is a sequence of proof rules that when chained together relate the premise of the goal to the conclusion of the goal.

Sequent Calculus

Definition. A sequent is a pair (Γ, Δ) of finite sets of formulae.

We will write a sequent as $\Gamma \hookrightarrow \Delta$ and drop the set brackets around the sets of formulae. X will represent a single formula, where as Γ is a set of formulae.

(More commonly, $\hookrightarrow$ is written as $\rightarrow$.)

The $\hookrightarrow$ is like implication. A sequent asserts: if all the formulae on the left of the arrow are true, then at least one of the formulae on the right are true.

Meaning of a Sequent

We can extend Boolean valuations to describe the meaning of a sequent.

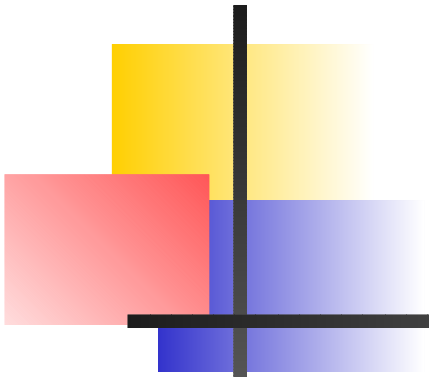
$v(\Gamma \hookrightarrow \Delta) = \mathbf{T}$

iff $v(X) = \mathbf{F}$ for some X in Γ

or $v(X) = \mathbf{T}$ for some X in Δ.

When there is nothing on the LHS or RHS of the arrow, we assume it is the empty set of formulae.

This means $v(\hookrightarrow) = \mathbf{F}$, and $v(\hookrightarrow X) = v(X)$.



Axioms

$$X \hookrightarrow X \text{ (Id)}$$

$$\mathbf{false} \hookrightarrow$$

$$\hookrightarrow \mathbf{true}$$

Sequent Schemata

The rules of the sequent calculus are written in the form:

$$\frac{S_1}{S_2}$$

If a formula matches the schema S_1, then it can be replaced by one matching S_2.

Stated another way: if you know S_1 is true, then S_2 is also true.

Sequent Calculus Rules

Structural Rule (Thinning)

If $\Gamma_1 \subseteq \Gamma_2$ and $\Delta_1 \subseteq \Delta_2$ then:

$$\frac{\Gamma_1 \hookrightarrow \Delta_1}{\Gamma_2 \hookrightarrow \Delta_2}$$

Thinning is like *precondition strengthening* and *postcondition weakening* for those familiar with that terminology.

Adding formulae on the RHS of the sequent is adding them to a disjunction, so this is weakening the RHS formulae.

Adding formulae on the LHS of the sequent is adding them to a conjunction, so this is strengthening the LHS formulae.

Sequent Calculus Rules

Negation Rules

$$\frac{\Gamma \hookrightarrow \Delta, X}{\Gamma, \neg X \hookrightarrow \Delta} \qquad \frac{\Gamma, X \hookrightarrow \Delta}{\Gamma \hookrightarrow \Delta, \neg X}$$

Conjunction Rules

$$\frac{\Gamma, X, Y \hookrightarrow \Delta}{\Gamma, X \wedge Y \hookrightarrow \Delta} \qquad \frac{\Gamma_1 \hookrightarrow \Delta_1, X \quad \Gamma_2 \hookrightarrow \Delta_2, Y}{\Gamma_1, \Gamma_2 \hookrightarrow \Delta_1, \Delta_2, X \wedge Y}$$

Disjunction Rules

$$\frac{\Gamma \hookrightarrow \Delta, X, Y}{\Gamma \hookrightarrow \Delta, X \vee Y} \qquad \frac{\Gamma_1, X \hookrightarrow \Delta_1 \quad \Gamma_2, Y \hookrightarrow \Delta_2}{\Gamma_1, \Gamma_2, X \vee Y \hookrightarrow \Delta_1, \Delta_2}$$

Sequent Calculus Rules

Implication Rules

#1

$$\frac{\Gamma, X \hookrightarrow \Delta, Y}{\Gamma \hookrightarrow \Delta, X \Rightarrow Y}$$

#2

$$\frac{\Gamma_1 \hookrightarrow \Delta_1, X \qquad \Gamma_2, Y \hookrightarrow \Delta_2}{\Gamma_1, \Gamma_2, X \Rightarrow Y \hookrightarrow \Delta_1, \Delta_2}$$

The right-hand rule is similar to *modus ponens*. The idea is that if X can be derived, then from $X \Rightarrow Y$, Y can be derived, and therefore whatever can be derived from Y can be derived. If Γ_2, and Δ_1 are empty:

$$\frac{\Gamma_1 \hookrightarrow X \qquad Y \hookrightarrow \Delta_2}{\Gamma_1, X \Rightarrow Y \hookrightarrow \Delta_2}$$

Cut Rule

This a derived rule:

$$\frac{\Gamma_1 \hookrightarrow \Delta_1, X \qquad \Gamma_2, X \hookrightarrow \Delta_2}{\Gamma_1, \Gamma_2 \hookrightarrow \Delta_1, \Delta_2}$$

Question: How is it derived?

Proofs in the Sequent Calculus

Definition. A proof is a *tree* labelled with sequents (generally written with the root at the bottom), such that: if node N is labelled with $\Gamma \hookrightarrow \Delta$, then if N is a leaf node, $\Gamma \hookrightarrow \Delta$ must be an axiom; if N has children, their labels must be the premises from which $\Gamma \hookrightarrow \Delta$ follows by one of the rules. The label on the root node is the sequent that is proved.

Definition. A formula X is a theorem of the sequent calculus if the sequent $\hookrightarrow X$ has a proof, i.e.,

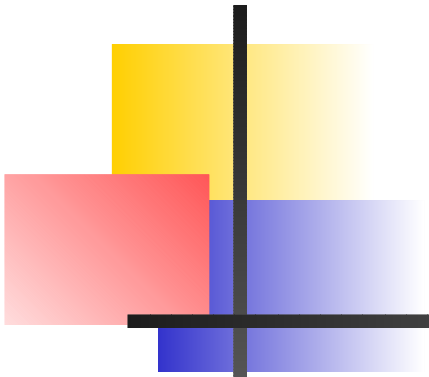
$$\vdash_{\mathsf{SQ}} X$$

The sequent calculus for propositional logic is both sound and complete.

Example

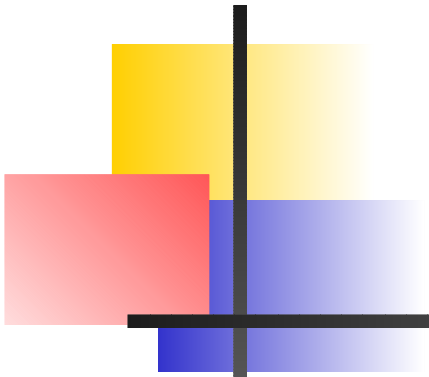
Show $\hookrightarrow A \Rightarrow (B \Rightarrow A)$

1.	$A \hookrightarrow A$	(id)
2.	$A, B \hookrightarrow A$	(Thinning)
3.	$A \hookrightarrow B \Rightarrow A$	(Implication #1)
4.	$\hookrightarrow A \Rightarrow (B \Rightarrow A)$	(Implication #1)



Agenda

- What is verification?
- What is logic?
- Propositional logic
- *Predicate logic*

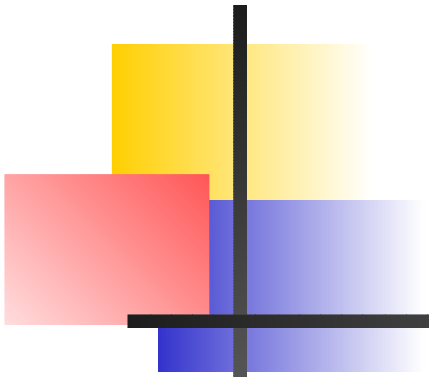


Motivation

There are some kinds of human reasoning that we cannot do in propositional logic. For example:

Every person likes ice cream.
Billy is a person.
Therefore, Billy likes ice cream.

In propositional logic, the best we can do is $(A \wedge B) \Rightarrow C$, which is not a tautology.



Motivation

We need to be able to refer to *objects* . We want to symbolize both a *claim* and the object about which the claim is made. We also need to refer to *relations* between objects, as in “Waterloo is west of Toronto”. If we can refer to objects, we also want to be able to capture the meaning of *every* and *some* of .

The *predicates* and *quantifiers* of predicate logic allow us to capture these concepts.

Examples

All grass is green.

$\forall y \bullet Grass(y) \Rightarrow Green(y)$

Something is rotten in the state of Denmark.

$\exists x \bullet Rotten(x) \wedge Denmark(x)$

We can also have n-ary predicates. Example:

Every even number is divisible by two.

$\forall x \bullet Even(x) \Rightarrow Div(x, 2)$

The "$\bullet$" is pronounced "such that". For the moment, we are not dealing with types.

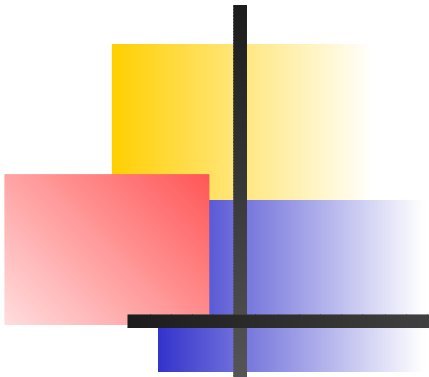
Quantifiers

Universal quantification $\forall$ corresponds to finite or infinite conjunction of the application of the predicate to all elements of the domain.

Existential quantification $\exists$ corresponds to finite or infinite disjunction of the application of the predicate to all elements of the domain.

Relationship between $\forall$ and $\exists$:

$\forall x \bullet P(x)$ is the same as $\neg \exists x \bullet \neg P(x)$
$\exists x \bullet P(x)$ is the same as $\neg \forall x \bullet \neg P(x)$



Functions

Consider how to formalize:

Mary's father likes music

One possible way: $\exists x \bullet Father(x, Mary) \wedge Likes(x, Music)$

which means: Mary has at least one father and he likes music. We'd like to capture the idea that Mary only has one father. We use functions to capture a single object that can be in relation to another object.

Example: $Likes(father(Mary), Music)$

We can also have n-ary functions.

Predicate Logic: Syntax

The *syntax* of predicate logic consists of:

1. constants
2. variables $x, y, \cdots$
3. functions
4. predicates
5. logical connectives
6. quantifiers $(\forall, \exists)$
7. punctuation: $\bullet$, ( )

Predicate Logic: Syntax

Definition. *Terms* are defined as follows:

1. Every constant is a term.
2. Every variable is a term.
3. If $t_1, t_2, t_3, \cdots t_n$ are terms then $f(t_1, t_2, t_3, \cdots t_n)$ is a term, where f is an n-ary function.
4. Nothing else is a term.

Predicate Logic: Syntax

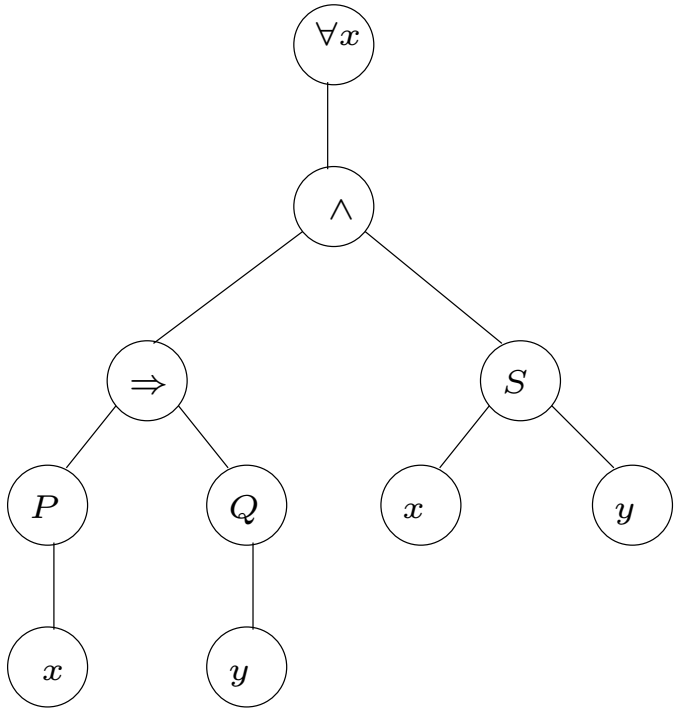
Definition. Well-formed formulas are defined as follows:

1. $P(t_1, t_2, t_3, \cdots t_n)$ is a wff, where t_i is a term, and P is an n-ary predicate. These are called *atomic formulas*.
2. If A and B are wffs, then so are $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \Rightarrow B)$, and $(A \Leftrightarrow B)$.
3. If A is a wff, so is $\forall x \bullet A$.
4. If A is a wff, so is $\exists x \bullet A$.
5. Nothing else is a wff.

We often omit the brackets using the same precedence rules as propositional logic for the logical connectives.

Scope and Binding of Variables

Consider a parse tree for $\forall x \bullet (P(x) \Rightarrow Q(x)) \wedge S(x, y)$:



Scope and Binding of Variables

Variables occur both in nodes next to quantifiers and as leaf nodes in the parse tree.

A variable x is *bound* if starting at the leaf of x, we walk up the tree and run into a node with a quantifier and x.

A variable x is *free* if starting at the leaf of x, we walk up the tree and do not run into a node with a quantifier and x.

The *scope* of a variable x is the subtree starting at the node with the variable and its quantifier (where it is bound) minus any subtrees with $\forall x$ or $\exists x$ at their root.

Example: $\forall x \bullet (\forall x \bullet (P(x) \wedge Q(x))) \Rightarrow (\neg P(x) \vee Q(y))$

A wff is *closed* if it contains no free occurrences of any variable.

Substitution

Variables are place holders. Given a variable x, a term t and a formula P, we define $P[t/x]$ to be the formula obtained by replacing all *free* occurrences of variable x in P with t.

We have to watch out for *variable capture* in substitution.

Given a term t, a variable x and a formula A, we say that "t is free for x in A" if no free x leaf in A occurs in the scope of $\forall y$ or $\exists y$ for any free variable y occurring in t.

Example:

A is $\forall y \bullet P(x) \wedge Q(y)$
t is $f(y)$
t is NOT free for x in A.

Whenever we use $P[t/x]$, t and x must both be free for x in P.

Predicate Logic: Semantics

Recall that a semantics is a mapping between two worlds. A *model* for predicate logic consists of:

1. a non-empty *domain* of objects: D
2. a mapping I, called an *interpretation* that associates the terms of the syntax with objects in a domain

It's important that D be non-empty, otherwise some tautologies would not hold such as $(\forall x.A(x)) \Rightarrow (\exists x.A(x))$.

Interpretations

An *interpretation* assigns:

1. a fixed element $c' \in D$ to each constant c of the syntax
2. an n-ary function $f' : D^n \to D$ to each n-ary function, f, of the syntax
3. an n-ary relation $R' \subseteq D^n$ to each n-ary predicate, R, of the syntax

Example of a Model

Let's say our syntax has the constant c, the function f (unary), and two predicates P, and Q (both binary). In our model, choose the domain to be the natural numbers.

- $I(c)$ is 0
- $I(f)$ is suc, the successor function
- $I(P)$ is $<$
- $I(Q)$ is $=$

Example of a Model

What is the meaning of $P(c, f(c))$ in this model?

$$\begin{aligned} I(P(c, f(c))) &= I(c) < I(f(c)) \\ &= 0 < \mathsf{suc}(I(c)) \\ &= 0 < \mathsf{suc}(0) \\ &= 0 < 1 \end{aligned}$$

which is true.

Valuations

Definition. A valuation v, in an interpretation I, is a function from the terms to the domain D such that:

1. $v(c) = I(c)$
2. $v(f(t1, \cdots t_n)) = f'(v(t_1, \cdots t_n))$
3. $v(x) \in D$, i.e., each variable is mapped onto some element in D

Example of a Valuation

- D is the set of natural numbers
- g is the function +
- h is the function suc
- c (constant) is 3
- y (variable) is 1

$$\begin{aligned} v(g(h(c), y)) &= v(h(c)) + v(y) \\ &= \mathsf{suc}(v(c)) + 1 \\ &= \mathsf{suc}(3) + 1 \\ &= 5 \end{aligned}$$

Predicate Logic: Satisfiability

Given a model m, with domain D and interpretation I, and a valuation v,

1. If A is an atomic wff, $P(t_1 \cdots t_n)$, m and v satisfy A iff $P'(v(t_1) \cdots v(t_n))$
2. If A has form $\neg B$, then m and v satisfy A iff m and v do not satisfy B.
3. If A has the form $B \wedge C$, then m and v satisfy A iff m and v satisfy B and m and v satisfy C , etc. for the other connectives.

Predicate Logic: Satisfiability

5. If A has the form $\forall x \bullet B$, then m and v satisfy A iff v satisfies B for all elements of D, i.e., for all $v(x) \in D$.
6. If A has the form $\exists x \bullet B$, then v satisfies A iff v satisfies B for some element of D, i.e., there is some $v(x) \in D$ for which B is satisfied.

Notice that if the formula is closed, then the valuation depends only on the model.

Notational convenience: while we have defined valuations for terms only, we will extend the use of v to be for wff also, mapping relations to their counterparts on the domain, and the logical connectives as we did in Boolean valuations.

Validity (Tautologies)

Definition. A predicate logic formula is *satisfiable* if *there exists* a model and *there exists* a valuation that satisfies the formula (i.e., in which the formula returns **T**).

Definition. A predicate logic formula is logically *valid* (tautology) if it is true in every model. It must be satisfied by every valuation in every model.

Definition. A wff, A, of predicate logic is a *contradiction* if it is false in every model. It must be false in every valuation in every model.

Semantic Entailment

Semantic entailment has the same meaning as it did for propositional logic.

$$\phi_1, \phi_2, \phi_3 \models \psi$$

means that for all v if $v(\phi_1) = \mathbf{T}$ and $v(\phi_2) = \mathbf{T}$, and $v(\phi_3) = \mathbf{T}$, then $v(\psi) = \mathbf{T}$, which is equivalent to saying

$$(\phi_1 \wedge \phi_2 \wedge \phi_3) \Rightarrow \psi$$

is a tautology, i.e.,

$$\boxed{\phi_1, \phi_2, \phi_3 \models \psi \;\equiv\; (\phi_1 \wedge \phi_2 \wedge \phi_3) \Rightarrow \psi}$$

Closed Formulas

Recall: A wff is *closed* if it contains no free occurrences of any variable.

We will mostly restrict ourselves to closed formulas. For formulas with free variables, close the formula by universally quantifying over all its free variables.

Sat., Tautologies, Contradictions

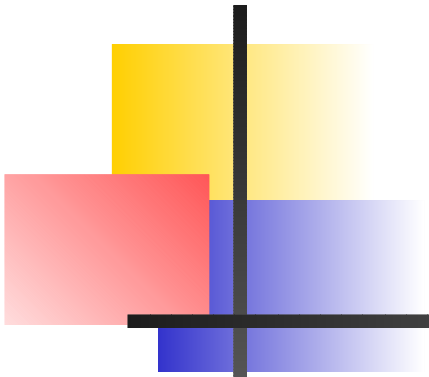
A closed predicate logic formula, is satisfiable if *there is* a model I in which the formula returns **T**.

A closed predicate logic formula, A, is a *tautology* if it is **T** in every model.

$$\models A$$

A closed predicate logic formula is a *contradiction* if it is **F** in every model.

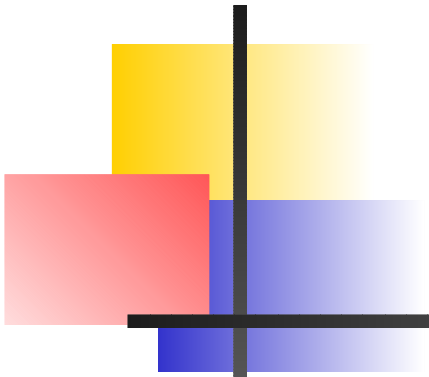
Question. What is the complexity of checking the satisfiability of a predicate logic formula?.



Proof by Refutation

A closed formula is a tautology (valid) iff its negation is a contradiction.

In other words, a closed formula is a valid iff its negation is not satisfiable.



Counterexamples

How can we show a formula is not a tautology?

Provide a counterexample. A *counterexample* for a closed formula is a model in which the formula does not have the truth value **T**.

What does first-order mean?

We can only quantify over variables.

In higher-order logics, we can quantify over functions, and predicates. For example, in second-order logic, we can express the induction principle:

$$\forall P \bullet (P(0) \wedge (\forall n \bullet P(n) \Rightarrow P(n+1))) \Rightarrow (\forall n \bullet P(n))$$

Propositional logic can also be thought of as zero-order.

An Axiomatic System for Predicate Logic

An extension of the axiomatic system for propositional logic.

Called FO_AL. Use only: $\Rightarrow, \neg, \forall$

Five axiom (schemes):

1. $A \Rightarrow (B \Rightarrow A)$
2. $(A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))$
3. $(\neg A \Rightarrow \neg B) \Rightarrow (B \Rightarrow A)$
4. $(\forall x \bullet A(x)) \Rightarrow A(t)$, where t is free for x in A
5. $\forall x \bullet (A \Rightarrow B) \Rightarrow (A \Rightarrow (\forall x \bullet B))$, where A contains no free occurrences of x

FO_AL Rules of Inference

Two rules of inference:

1. (modus ponens - MP) From A and $A \Rightarrow B$, B can be derived, where A and B are any well-formed formulas.
2. (generalization) From $A, \forall x \bullet A$ can be derived, where A is any well-formed formula and x is any variable.

Deduction Theorem

Theorem. If $H \cup \{A\} \vdash_{ph} B$ by a deduction containing no application of generalization to a variable that occurs free in A, then $H \vdash_{ph} (A \Rightarrow B)$.