

Foundation of Logic

Foundation of Logic

Foundation of Logic

Foundation of Logic

What is logic?

“Logic is the beginning of wisdom, not the end”

The branch of philosophy concerned with analysing the patterns of reasoning by which a conclusion is drawn from a set of premises, without reference to meaning or context is known as logic.

(Collins English Dictionary)

Why is logic important?

- Logic is a **formal method for reasoning**.
- Logic is a formal language for **deducing** knowledge from a small number of explicitly stated **premises** (or hypotheses, axioms, facts).
- Logic provides a formal framework for **representing knowledge**.
- Logic differentiates between the **structure** and **content** of an argument.

What is an argument?

- An argument is just a **sequence of statements**.
- Some of these statements, the **premises**, are assumed to be true and serve as a basis for accepting another statement of the argument, called the **conclusion**.

Deduction and Inference

- If the conclusion is justified, based solely on the premises, the process of reasoning is called **deduction**.
- If the validity of the conclusion is based on *generalisation* from the premises, based on strong but inconclusive evidence, the process is called **inference** (sometimes called **induction**).

Two examples

- **Deductive** argument:

"Alexandria is a port or a holiday resort. Alexandria is not a port. Therefore, Alexandria is a holiday resort."

- **Inductive** argument

"Most students who did not do the tutorial questions will fail the exam. John did not do the tutorial questions. Therefore John will fail the exam."

Foundation of Logic

Mathematical Logic is a tool for working with compound statements. It includes:

- A formal language for expressing them.
- A methodology for objectively reasoning about their truth or falsity.
- It is the foundation for expressing formal proofs in all branches of mathematics.

We will talk about two logical systems:

- Propositional logic
- Predicate logic

Propositional Logic

Propositional Logic is the logic of compound statements built from simpler statements using so-called ***Boolean connectives***.

Some applications in computer science:

- Design of digital electronic circuits.
- Expressing conditions in programs.
- Queries to databases & search engines.

Definition: A *proposition* is simply:

- a *statement* (i.e., a declarative sentence)
 - with some definite meaning
- having a *truth value* that's either *true* (T) or *false* (F)
 - it is never both, neither, or somewhere “in between!”
 - however, you might not *know* the actual truth value.

Propositions in Propositional Logic

- Simple types of statements, called propositions, are treated as atomic building blocks for more complex statements.
- Atoms: p, q, r, \dots
(Corresponds with simple English sentences, e.g. **'I had salad for lunch'**)
- Complex propositions : built up from atoms using operators: $p \wedge q$
(Corresponds with compound English sentences, e.g., **"I had salad for lunch and I had steak for dinner."**)

Some Popular Boolean Operators

<u>Formal Name</u>	<u>Nickname</u>	<u>Arity</u>	<u>Symbol</u>
Negation operator	NOT	Unary	\neg
Conjunction operator	AND	Binary	\wedge
Disjunction operator	OR	Binary	\vee
Exclusive-OR operator	XOR	Binary	\oplus
Implication operator	IMPLIES	Binary	\rightarrow
Biconditional operator	IFF	Binary	\leftrightarrow

The Negation Operator

The unary *negation operator* “ \neg ” (*NOT*) transforms a prop. into its *negation*.

E.g. If $p =$ “I have brown hair.”

then $\neg p =$ “I do **not** have brown hair.”

The *truth table* for NOT:

T \equiv True; **F** \equiv False

p	$\neg p$
T	F
F	T

The Conjunction Operator

The binary *conjunction operator* “ \wedge ” (*AND*) combines two propositions to form their logical *conjunction*.

E.g. If p =“I will have salad for lunch.” and q =“I will have steak for dinner.”, then $p \wedge q$ =“I will have salad for lunch and I will have steak for dinner.”

- Note that a conjunction $p_1 \wedge p_2 \wedge \dots \wedge p_n$ of n propositions will have 2^n rows in its truth table.

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

The Disjunction Operator

The binary *disjunction operator* " \vee " (*OR*) combines two propositions to form their logical *disjunction*.

p = "My car has a bad engine."

q = "My car has a bad carburator."

$p \vee q$ = "Either my car has a bad engine, or my car has a bad carburetor."

- Note that $p \vee q$ means that p is true, or q is true, **or both** are true!
- So, this operation is also called *inclusive or*, because it **includes** the possibility that both p and q are true.

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

Nested Propositional Expressions

- Use parentheses to *group sub-expressions*:
“I just saw my old friend, and either he’s grown or I’ve shrunk.” = $f \wedge (g \vee s)$
 $(f \wedge g) \vee s$ would mean something different
 $f \wedge g \vee s$ would be ambiguous
- By convention, “ \neg ” takes *precedence* over both “ \wedge ” and “ \vee ”.

Tautologies/Contradictions/Contingencies

A *tautology* is a compound proposition that is **true** *no matter what* the truth values of its atomic propositions are!

Ex. $p \vee \neg p = 1$

A *contradiction* is a compound proposition that is **false** *no matter what* the truth values of its atomic propositions are!

Ex. $p \wedge \neg p = 0$

All other propositions are *contingencies*:

Some rows give T, others give F.

Logical Equivalence

Compound proposition p is *logically equivalent* to compound proposition q , written $p \Leftrightarrow q$, **IFF**
 p and q contain the same truth values
in all rows of their truth tables

We will also say: they express the same truth function (= the same function
from values for atoms to values for the whole formula).

Ex. Prove that $p \vee q \Leftrightarrow \neg(\neg p \wedge \neg q)$.

p	q	$p \vee q$	$\neg p$	$\neg q$	$\neg p \wedge \neg q$	$\neg(\neg p \wedge \neg q)$
F	F	F	T	T	T	F
F	T	T	T	F	F	T
T	F	T	F	T	F	T
T	T	T	F	F	F	T

The *Exclusive Or* Operator

The binary *exclusive-or operator* " \oplus " (*XOR*) combines two propositions to form their logical "exclusive or".

p = "I will earn an A in this course,"

q = "I will drop this course,"

$p \oplus q$ = "I will either earn an A in this course, or I will drop it (but not both!)"

- Note that $p \oplus q$ means that p is true, or q is true, but **not both**!
- This operation is called *exclusive or*, because it **excludes** the possibility that both p and q are true.

p	q	$p \oplus q$
F	F	F
F	T	T
T	F	T
T	T	F

The *Implication* Operator

The *implication* $p \rightarrow q$ states that p implies q .

i.e., if p is true, then q is true; but if p is not true, then q could be either true or false.

e.g., let p = "You study hard."

q = "You will get a good grade."

$p \rightarrow q$ = "If you study hard, then you will get a good grade."

- $p \rightarrow q$ is **false** only when (p is true but q is **not** true)
- $p \rightarrow q$ does **not** say that p causes q !
- $p \rightarrow q$ does **not** require that p or q are true!

p	q	$p \rightarrow q$
F	F	T
F	T	T
T	F	F
T	T	T

Implications between real sentences

- “If this lecture ever ends, then the sun has risen this morning.” *True or False?*
- “If Tuesday is a day of the week, then I am a penguin.” *True or False?*
- “If $1+1=6$, then Bush is president.” *True or False?*
- “If the moon is made of green cheese, then I am richer than Bill Gates.” *True or False?*

Biconditional Truth Table

- $p \leftrightarrow q$ means that p and q have the **same** truth value.
- Note this truth table is the exact **opposite** of \oplus 's!
Thus, $p \leftrightarrow q$ means $\neg(p \oplus q)$
- $p \leftrightarrow q$ does **not** imply that p and q are true, or that either of them causes the other.

p	q	$p \leftrightarrow q$
F	F	T
F	T	F
T	F	F
T	T	T

Converse/Contrapositive

Some terminology, for an implication $p \rightarrow q$:

- Its *converse* is: $q \rightarrow p$.
- Its *contrapositive*: $\neg q \rightarrow \neg p$.

Logical Equivalences

- *Identity:* $p \wedge \mathbf{T} \Leftrightarrow p$ $p \vee \mathbf{F} \Leftrightarrow p$
- *Domination:* $p \vee \mathbf{T} \Leftrightarrow \mathbf{T}$ $p \wedge \mathbf{F} \Leftrightarrow \mathbf{F}$
- *Idempotence:* $p \vee p \Leftrightarrow p$ $p \wedge p \Leftrightarrow p$
- *Double negation:* $\neg\neg p \Leftrightarrow p$
- *Commutativity:* $p \vee q \Leftrightarrow q \vee p$ $p \wedge q \Leftrightarrow q \wedge p$
- *Associativity:* $(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$
 $(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$

More Equivalence Laws

- *Distributive:* $p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$
 $p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$
- *De Morgan's Laws:*
 $\neg(p \wedge q) \Leftrightarrow \neg p \vee \neg q$
 $\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$
- *Trivial tautology/contradiction:*
 $p \vee \neg p \Leftrightarrow \mathbf{T}$ $p \wedge \neg p \Leftrightarrow \mathbf{F}$



Augustus
De Morgan
(1806-1871)

Predicate Logic

- *Predicate logic* is an extension of propositional logic that permits quantification over classes of entities.
- Propositional logic (recall) treats *simple propositions* (sentences) as atomic entities.
- A 'predicate' is just a property
- Predicates define relationships between any number of entities using qualifiers:
 - \forall "for all", "for every"
 - \exists "there exists"

Remember:

$\forall x$ 'for every x ', or 'for All x '

$\exists x$ 'there is an x ' or 'there Exists an x '

Applications of Predicate Logic

- It is one of the most-used formal notations for writing mathematical *definitions*, *axioms*, and *theorems*.
- For example, in *linear algebra*, a *partial order* is introduced saying that a relation R is *reflexive* and *transitive* – and these notions are defined using predicate logic.
- Basis for many Artificial Intelligence systems.
- Predicate-logic like statements are supported by some of the more sophisticated *database query engines*.

Propositional/Predicate logic

- In propositional logic, we could not simply say whether a formula is TRUE; what we could say is whether it is TRUE with respect to a given assignment of TRUE/FALSE to the Atoms in the formula
- e.g., $p \rightarrow q$ is TRUE with respect to the assignment $p = \text{TRUE}$, $q = \text{TRUE}$
- In predicate logic, we say that a formula is TRUE (FALSE) with respect to a **model**.

What is logic?

“Logic is the beginning of wisdom, not the end”

The branch of philosophy concerned with analysing the patterns of reasoning by which a conclusion is drawn from a set of premises, without reference to meaning or context is known as logic.

(Collins English Dictionary)

Why is logic important?

- Logic is a **formal method for reasoning**.
- Logic is a formal language for **deducing** knowledge from a small number of explicitly stated **premises** (or hypotheses, axioms, facts).
- Logic provides a formal framework for **representing knowledge**.
- Logic differentiates between the **structure** and **content** of an argument.

What is an argument?

- An argument is just a **sequence of statements**.
- Some of these statements, the **premises**, are assumed to be true and serve as a basis for accepting another statement of the argument, called the **conclusion**.

Deduction and Inference

- If the conclusion is justified, based solely on the premises, the process of reasoning is called **deduction**.
- If the validity of the conclusion is based on *generalisation* from the premises, based on strong but inconclusive evidence, the process is called **inference** (sometimes called **induction**).

Two examples

- **Deductive** argument:

"Alexandria is a port or a holiday resort. Alexandria is not a port. Therefore, Alexandria is a holiday resort."

- **Inductive** argument

"Most students who did not do the tutorial questions will fail the exam. John did not do the tutorial questions. Therefore John will fail the exam."

Foundation of Logic

Mathematical Logic is a tool for working with compound statements. It includes:

- A formal language for expressing them.
- A methodology for objectively reasoning about their truth or falsity.
- It is the foundation for expressing formal proofs in all branches of mathematics.

We will talk about two logical systems:

- Propositional logic
- Predicate logic

Propositional Logic

Propositional Logic is the logic of compound statements built from simpler statements using so-called ***Boolean connectives***.

Some applications in computer science:

- Design of digital electronic circuits.
- Expressing conditions in programs.
- Queries to databases & search engines.

Definition: A *proposition* is simply:

- a *statement* (i.e., a declarative sentence)
 - with some definite meaning
- having a *truth value* that's either *true* (T) or *false* (F)
 - it is never both, neither, or somewhere “in between!”
 - however, you might not *know* the actual truth value.

Propositions in Propositional Logic

- Simple types of statements, called propositions, are treated as atomic building blocks for more complex statements.
- Atoms: p, q, r, \dots
(Corresponds with simple English sentences, e.g. **'I had salad for lunch'**)
- Complex propositions : built up from atoms using operators: $p \wedge q$
(Corresponds with compound English sentences, e.g., **"I had salad for lunch and I had steak for dinner."**)

Some Popular Boolean Operators

<u>Formal Name</u>	<u>Nickname</u>	<u>Arity</u>	<u>Symbol</u>
Negation operator	NOT	Unary	\neg
Conjunction operator	AND	Binary	\wedge
Disjunction operator	OR	Binary	\vee
Exclusive-OR operator	XOR	Binary	\oplus
Implication operator	IMPLIES	Binary	\rightarrow
Biconditional operator	IFF	Binary	\leftrightarrow

The Negation Operator

The unary *negation operator* “ \neg ” (*NOT*) transforms a prop. into its *negation*.

E.g. If $p =$ “I have brown hair.”

then $\neg p =$ “I do **not** have brown hair.”

The *truth table* for NOT:

T \equiv True; **F** \equiv False

p	$\neg p$
T	F
F	T

The Conjunction Operator

The binary *conjunction operator* “ \wedge ” (*AND*) combines two propositions to form their logical *conjunction*.

E.g. If p =“I will have salad for lunch.” and q =“I will have steak for dinner.”, then $p \wedge q$ =“I will have salad for lunch and I will have steak for dinner.”

- Note that a conjunction $p_1 \wedge p_2 \wedge \dots \wedge p_n$ of n propositions will have 2^n rows in its truth table.

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

The Disjunction Operator

The binary *disjunction operator* " \vee " (*OR*) combines two propositions to form their logical *disjunction*.

p = "My car has a bad engine."

q = "My car has a bad carburetor."

$p \vee q$ = "Either my car has a bad engine, or my car has a bad carburetor."

- Note that $p \vee q$ means that p is true, or q is true, **or both** are true!
- So, this operation is also called *inclusive or*, because it **includes** the possibility that both p and q are true.

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

Nested Propositional Expressions

- Use parentheses to *group sub-expressions*:
“I just saw my old friend, and either he’s grown or I’ve shrunk.” = $f \wedge (g \vee s)$
 $(f \wedge g) \vee s$ would mean something different
 $f \wedge g \vee s$ would be ambiguous
- By convention, “ \neg ” takes *precedence* over both “ \wedge ” and “ \vee ”.

Tautologies/Contradictions/Contingencies

A *tautology* is a compound proposition that is **true** *no matter what* the truth values of its atomic propositions are!

Ex. $p \vee \neg p = 1$

A *contradiction* is a compound proposition that is **false** *no matter what* the truth values of its atomic propositions are!

Ex. $p \wedge \neg p = 0$

All other propositions are *contingencies*:

Some rows give T, others give F.

Logical Equivalence

Compound proposition p is *logically equivalent* to compound proposition q , written $p \Leftrightarrow q$, **IFF**
 p and q contain the same truth values
in all rows of their truth tables

We will also say: they express the same truth function (= the same function
from values for atoms to values for the whole formula).

Ex. Prove that $p \vee q \Leftrightarrow \neg(\neg p \wedge \neg q)$.

p	q	$p \vee q$	$\neg p$	$\neg q$	$\neg p \wedge \neg q$	$\neg(\neg p \wedge \neg q)$
F	F	F	T	T	T	F
F	T	T	T	F	F	T
T	F	T	F	T	F	T
T	T	T	F	F	F	T

The *Exclusive Or* Operator

The binary *exclusive-or operator* " \oplus " (*XOR*) combines two propositions to form their logical "exclusive or".

p = "I will earn an A in this course,"

q = "I will drop this course,"

$p \oplus q$ = "I will either earn an A in this course, or I will drop it (but not both!)"

- Note that $p \oplus q$ means that p is true, or q is true, but **not both**!
- This operation is called *exclusive or*, because it **excludes** the possibility that both p and q are true.

p	q	$p \oplus q$
F	F	F
F	T	T
T	F	T
T	T	F

The *Implication* Operator

The *implication* $p \rightarrow q$ states that p implies q .

i.e., if p is true, then q is true; but if p is not true, then q could be either true or false.

e.g., let p = "You study hard."

q = "You will get a good grade."

$p \rightarrow q$ = "If you study hard, then you will get a good grade."

- $p \rightarrow q$ is **false** only when (p is true but q is **not** true)
- $p \rightarrow q$ does **not** say that p causes q !
- $p \rightarrow q$ does **not** require that p or q are true!

p	q	$p \rightarrow q$
F	F	T
F	T	T
T	F	F
T	T	T

Implications between real sentences

- “If this lecture ever ends, then the sun has risen this morning.” *True or False?*
- “If Tuesday is a day of the week, then I am a penguin.” *True or False?*
- “If $1+1=6$, then Bush is president.” *True or False?*
- “If the moon is made of green cheese, then I am richer than Bill Gates.” *True or False?*

Biconditional Truth Table

- $p \leftrightarrow q$ means that p and q have the **same** truth value.
- Note this truth table is the exact **opposite** of \oplus 's!
Thus, $p \leftrightarrow q$ means $\neg(p \oplus q)$
- $p \leftrightarrow q$ does **not** imply that p and q are true, or that either of them causes the other.

p	q	$p \leftrightarrow q$
F	F	T
F	T	F
T	F	F
T	T	T

Converse/Contrapositive

Some terminology, for an implication $p \rightarrow q$:

- Its *converse* is: $q \rightarrow p$.
- Its *contrapositive*: $\neg q \rightarrow \neg p$.

Logical Equivalences

- *Identity:* $p \wedge \mathbf{T} \Leftrightarrow p$ $p \vee \mathbf{F} \Leftrightarrow p$
- *Domination:* $p \vee \mathbf{T} \Leftrightarrow \mathbf{T}$ $p \wedge \mathbf{F} \Leftrightarrow \mathbf{F}$
- *Idempotence:* $p \vee p \Leftrightarrow p$ $p \wedge p \Leftrightarrow p$
- *Double negation:* $\neg \neg p \Leftrightarrow p$
- *Commutativity:* $p \vee q \Leftrightarrow q \vee p$ $p \wedge q \Leftrightarrow q \wedge p$
- *Associativity:* $(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$
 $(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$

More Equivalence Laws

- *Distributive:* $p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$
 $p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$
- *De Morgan's Laws:*
 $\neg(p \wedge q) \Leftrightarrow \neg p \vee \neg q$
 $\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$
- *Trivial tautology/contradiction:*
 $p \vee \neg p \Leftrightarrow \mathbf{T}$ $p \wedge \neg p \Leftrightarrow \mathbf{F}$



Augustus
De Morgan
(1806-1871)

Predicate Logic

- *Predicate logic* is an extension of propositional logic that permits quantification over classes of entities.
- Propositional logic (recall) treats *simple propositions* (sentences) as atomic entities.
- A 'predicate' is just a property
- Predicates define relationships between any number of entities using qualifiers:
 - \forall "for all", "for every"
 - \exists "there exists"

Remember:

$\forall x$ 'for every x ', or 'for All x '

$\exists x$ 'there is an x ' or 'there Exists an x '

Applications of Predicate Logic

- It is one of the most-used formal notations for writing mathematical *definitions*, *axioms*, and *theorems*.
- For example, in *linear algebra*, a *partial order* is introduced saying that a relation R is *reflexive* and *transitive* – and these notions are defined using predicate logic.
- Basis for many Artificial Intelligence systems.
- Predicate-logic like statements are supported by some of the more sophisticated *database query engines*.

Propositional/Predicate logic

- In propositional logic, we could not simply say whether a formula is TRUE; what we could say is whether it is TRUE with respect to a given assignment of TRUE/FALSE to the Atoms in the formula
- e.g., $p \rightarrow q$ is TRUE with respect to the assignment $p = \text{TRUE}$, $q = \text{TRUE}$
- In predicate logic, we say that a formula is TRUE (FALSE) with respect to a **model**.

Outline

- First-order inference rules: Modus Ponens, Modus Tollens
- Resolution

Modus Ponens

- Assume you are given the following two statements:
 - “you are in this class”
 - “if you are in this class, you will get a grade”
- Let p = “you are in this class”
- Let q = “you will get a grade”
- By Modus Ponens, you can conclude that you will get a grade

$$p$$
$$\underline{p \rightarrow q}$$
$$\therefore q$$

Modus Ponens

- Consider $(p \wedge (p \rightarrow q)) \rightarrow q$

p	q	$p \rightarrow q$	$p \wedge (p \rightarrow q)$	$(p \wedge (p \rightarrow q)) \rightarrow q$
T	T	T	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	T

p
<u>$p \rightarrow q$</u>
$\therefore q$

Chain Rule

From $p \rightarrow q$, and $q \rightarrow r$, we can infer $p \rightarrow r$

$$p \rightarrow q$$

$$\underline{q \rightarrow r}$$

$$\therefore p \rightarrow r$$

Modus Tollens

- Assume that we know: $\neg q$ and $p \rightarrow q$
 - Recall that $p \rightarrow q = \neg q \rightarrow \neg p$
- Thus, we know $\neg q$ and $\neg q \rightarrow \neg p$
- We can conclude $\neg p$

$\neg q$

$p \rightarrow q$

$\therefore \neg p$

Modus Tollens

- Assume you are given the following two statements:
 - “you will not get a grade”
 - “if you are in this class, you will get a grade”
- Let p = “you are in this class”
- Let q = “you will get a grade”
- By Modus Tollens, you can conclude that you are not in this class

Addition and Simplification

- Addition: If you know that p is true, then $p \vee q$ will ALWAYS be true

$$\underline{p}$$

$$\therefore p \vee q$$

- Simplification: If $p \wedge q$ is true, then p will ALWAYS be true

$$\underline{p \wedge q}$$

$$\therefore p$$

Rules of inference for the universal quantifier

- Assume that we know that $\forall x P(x)$ is true
 - Then we can conclude that $P(c)$ is true
 - Here c stands for some specific constant
 - This is called “universal instantiation”
- Assume that we know that $P(c)$ is true for any value of c
 - Then we can conclude that $\forall x P(x)$ is true
 - This is called “universal generalization”

Rules of inference for the existential quantifier

- Assume that we know that $\exists x P(x)$ is true
 - Then we can conclude that $P(c)$ is true for some value of c
 - This is called “existential instantiation”
- Assume that we know that $P(c)$ is true for some value of c
 - Then we can conclude that $\exists x P(x)$ is true
 - This is called “existential generalization”

Anatomy of a propositional function

$$P(x) = x + 5 > x$$

variable

predicate

Universal instantiation (UI)

- A predicate that has no variables is called a ground atom.
- Every instantiation of a universally quantified sentence is entailed by it:

$$\forall v \alpha$$

$$\text{Subst}(\{v/g\}, \alpha)$$

for any variable v and ground term g (*Subst(x,y) = substitution of y by x*)

- E.g., $\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$ yields:

$$\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$$

$$\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$$

$$\text{King}(\text{Father}(\text{John})) \wedge \text{Greedy}(\text{Father}(\text{John})) \Rightarrow \text{Evil}(\text{Father}(\text{John}))$$

.

.

.

Existential instantiation (EI)

- For any sentence α , variable v , and constant symbol k that does *not* appear elsewhere in the knowledge base:

$$\frac{\exists v \alpha}{\text{Subst}(\{v/k\}, \alpha)}$$

- E.g., $\exists x \text{Crown}(x) \wedge \text{OnHead}(x, \text{John})$ yields:

$$\text{Crown}(C_1) \wedge \text{OnHead}(C_1, \text{John})$$

provided C_1 is a new constant symbol, called a **Skolem constant**

EI versus UI

- UI can be applied several times to *add* new sentences; the new KB is logically equivalent to the old.
- EI can be applied once to replace the existential sentence; the new KB is not equivalent to the old but is satisfiable if the old KB was satisfiable.

Reduction to propositional inference

- Suppose the KB contains just the following:
 $\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$
 $\text{King}(\text{John})$
 $\text{Greedy}(\text{John})$
 $\text{Brother}(\text{Richard}, \text{John})$
- Instantiating the universal sentence in **all possible** ways, we have:
 $\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$
 $\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$
 $\text{King}(\text{John})$
 $\text{Greedy}(\text{John})$
 $\text{Brother}(\text{Richard}, \text{John})$
- The new KB is **propositionalized**: proposition symbols are
 John, Richard and also $\text{King}(\text{John})$, $\text{Greedy}(\text{John})$, $\text{Evil}(\text{John})$,
 $\text{King}(\text{Richard})$, etc.

Making modus ponens complete

- Modus ponens is incomplete in a general KB
 - Need other inference rules
 - E.g. $KB = \{A \wedge B\}$, we cannot infer anything with MP
- There is a (finite) set of inference rules such that repeatedly applying them forms a complete inference procedure
 - All valid sentences can be inferred in this way
- For each inference rule, we can associate the infinite set of logical axioms (implications) that corresponds to each possible instantiation of the rule
 - E.g. and-elimination corresponds to all axioms of the form $A \wedge B \Rightarrow A$
 - Note that the axioms are in our FOL language, whereas inference rules are in a “meta-language”
- Modus ponens becomes complete if we add to our knowledge bases all the logical axioms corresponding to the other inference rules

Inference in FOL

- **Entailment:** $KB \models \alpha$ whenever every model of KB is also a model of α .

Inference procedures:

- o **propositionalization** (Universal and Existential elimination; convert to Propositional Logic; apply prop logic inference)
- o **lifted inference rules**, and in particular **refutation/resolution** proof for FOL
- o **forward/backward chaining** for definite clauses

Propositionalization

- Problem: with function symbols, there are infinitely many ground terms,
 - e.g., *Father(Father(Father(John)))*
- Theorem: Herbrand (1930). If a sentence α is entailed by an FOL KB, it is entailed by a finite subset of the propositionalized KB
- Idea: For $n = 0$ to ∞ do
 - create a propositional KB by instantiating with depth- n terms
 - see if α is entailed in this KB (e.g. using refutation)
- Problem: works if α is entailed, loops if α is not entailed
- Theorem: Turing (1936), Church (1936) Entailment for FOL is semi-decidable (algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every non-entailed sentence.)

Propositionalization: Example

$\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$

King(John)

Hassan

Giro

Lukasz

Jacomo

John

$\square \forall y \text{ Greedy}(y)$

Would like to conclude Evil(John).

By Propositionalization, write out:

$\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$

$\text{King}(\text{Hassan}) \wedge \text{Greedy}(\text{Hassan}) \Rightarrow \text{Evil}(\text{Hassan})$

$\text{King}(\text{Giro}) \wedge \text{Greedy}(\text{Giro}) \Rightarrow \text{Evil}(\text{Giro})$

$\text{King}(\text{Jacomo}) \wedge \text{Greedy}(\text{Jacomo}) \Rightarrow \text{Evil}(\text{Jacomo})$

$\text{King}(\text{Lukasz}) \wedge \text{Greedy}(\text{Lukasz}) \Rightarrow \text{Evil}(\text{Lukasz})$

Greedy(John)

Greedy(Hassan)



finally, conclude
Evil(John)

Problems with propositionalization

- Propositionalization seems to generate lots of irrelevant sentences
- E.g., from:
 $\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$
 $\text{King}(\text{John})$
 $\forall y \text{ Greedy}(y)$
 $\text{Brother}(\text{Richard}, \text{John})$
- It seems obvious that *Evil(John)*, but propositionalization produces lots of facts such as *Greedy(Richard)* that are irrelevant
- With p k -ary predicates and n constants, there are $p \cdot n^k$ instantiations

Generalized Modus Ponens

- Lifted version of the propositional Modus Ponens
- For atomic sentences p_i , p_i' , and q , where there is a substitution θ that satisfies $\text{Subst}(\theta, p_i') = \text{Subst}(\theta, p_i)$ for all i

$$\frac{p'_1, p'_2, \dots, p'_n, (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q)}{\text{Subst}(\theta, q)}$$

•Example:

p_1' is King(John)

p_1 is King(x)

p_2' is Greedy(y)

p_2 is Greedy(x)

θ is {x/ John, y/ John}

q is Evil(x)

$\text{Subst}(\theta, q)$ is Evil(John)

we say θ “unifies” p_1' and p_1 ; and p_2' and p_2 .

Soundness of GMP

- For any sentence p with universally quantified variables and for any substitution θ , $p \models \text{Subst}(\theta, p)$

- So, from (p_1', \dots, p_n') we can infer:

$$\text{Subst}(\theta, p_1') \wedge \dots \wedge \text{Subst}(\theta, p_n') \quad (1)$$

- From the implication $p_1 \wedge \dots \wedge p_n \Rightarrow q$ we can infer:

$$\text{Subst}(\theta, p_1) \wedge \dots \wedge \text{Subst}(\theta, p_n) \Rightarrow \text{Subst}(\theta, q) \quad (2)$$

By the GMP rule, when the first sentence (1) matches the premise of (2) exactly we can infer $\text{Subst}(\theta, q)$.

This follows immediately from Modus Ponens.

Unification

- To apply GMP we need a substitution that *unifies* a sentences in the KB with the premises, i.e. to find $\text{Subst}(\theta, p'_i) = \text{Subst}(\theta, p_i)$ for all i

$$\frac{p'_1, p'_2, \dots, p'_n, (p_1 \wedge p_2 \wedge \dots \wedge p_n) \ q}{\text{Subst}(\theta, q)}$$

• **Unification:** find a substitution of variables for terms that makes two sentences equivalent.

Example:

$\text{Unify}(\text{Knows}(\text{John}, x); \text{Knows}(\text{John}, \text{Jane})) = \{x/\text{Jane}\}$

$\text{Unify}(\text{Knows}(\text{John}, x); \text{Knows}(y, \text{Bill})) = \{x/\text{Bill}, y/\text{John}\}$

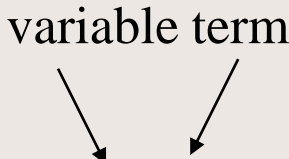
Write: $\text{unify}(\alpha, \beta) = \theta$, to denote the **unifier** θ , e.g.

$\theta = \text{unify}(\alpha, \beta) = \{x/\text{Jane}\}$

- Unifiers $\theta = \text{unify}(\alpha, \beta)$, can:
 - replace a variable by a constant term, e.g. $\{x/\text{John}\}$
 - replace a variable with a variable, e.g. $\{x/y\}$
 - replace a variable by a function expression,
e.g. $\{x / \text{Mother}(y)\}$
- CAREFUL: need to check variable (e.g. x) does not appear inside the complex term
- “Occur check”
- makes complexity quadratic

Examples

Find a **unifier** for the following sentences:

- 1 – **unify**(Knows(John,x), Knows(John,Jane)) $\theta = \{x/\text{Jane}\}$
- 2 – **unify**(Knows(John,x), Knows(y,Bill)) $\theta = \{x/\text{Bill}, y/\text{John}\}$
- 

Generalized Resolution

- Lifted version of resolution

$$p_1 \vee \dots \vee p_m, \neg q_1 \vee \dots \vee q_n, \text{ Unify}(p_1, \neg q_1) = \theta$$

$$\text{Subst}(\theta, p_2 \vee \dots \vee p_m \vee q_2 \vee \dots \vee q_n)$$

Example:

$\text{Animal}(F(x)) \vee \text{Loves}(G(x),x)$ and $\neg \text{Loves}(u,v) \vee \neg \text{Kills}(u,v)$

take unifier $\theta = \{u/G(x), v/x\}$, and get **resolvent clause**:

$\text{Animal}(F(x)) \vee \neg \text{Kills}(G(x),x)$

Resolution Algorithm

- start with KB, add $\neg \alpha$
- while $\text{False} \notin \text{KB}$ {
 - find two sentences $\alpha, \beta \in \text{KB}$ that unify
 - add $\text{resolvent}(\alpha, \beta)$ to KB}

Resolution is “**refutation-complete**:” will report “yes” (find the empty clause) if sentence is entailed.

Resolution Strategies

- Complete
 - Breadth-first (slow)
 - Set of Support
 - Linear resolution
 - Subsumption: remove all sentences in your KB that are subsumed. (e.g. remove $A \vee B(k)$ if $B(x)$ is in KB)
- Incomplete
 - Unit resolution
 - Input resolution