Version 2: First 10 pages have been revised

Version 3: Fixed an issue on page 12

4CCS1ELA & 5CCS2ELA Elementary Logic with Applications

1ST YEAR SEMESTER 1

Franch, Alex
KING'S COLLEGE LONDON | SEP-DEC

4CCS1ELA & 5CCS2ELA ELEMENTARY LOGIC WITH APPLICATIONS

Lecturer

Dr. Odinaldo Rodrigues

- Room S1.19
- Odinaldo.rodrigues@kcl.ac.uk
- +4402078482087
- Office H
    - Mon 1-2 pm S1.19
    - Tue 2-3 pm (different rooms, mini tutorials)

Material and Assessment

http://keats.kcl.ac.uk

Subscribe to News forum for this course

Two examinations Week 4 (17/10) and Week 9 (21/11) – coursework – each worth 7.5%

Establishing a formal Language

For logic to be logical (haha), we need to use a precise, formal language which can be automated.

Prepositional Logic

Prepositional Language

BB1: Propositions

Proposition: a declarative statement which is either **true** or **false**, but **not both**.

- A proposition has something called a **truth-vale**.
    - The truth-vale "**true**" can be written as T, true or associated with a 1.
    - The truth-value "**falsehood**" can be written as F, false or 0.

Examples of what is a proposition:

- Can I come as well?
    - This is **not** a proposition; it's a question.
- Bachelors are unmarried men.
    - This **is** a proposition; it can either be true or false.
- Tom, shut the door!
    - This is **not** a proposition; it's a command.

We can therefore conclude that **questions** and **commands** are not propositions.

BB2: Language connectives

Connective: is a symbol that is used to modify a statement or two combine two statements to make more complex statements.

A simple statement is a proposition. When we add a connective we have formed a complex statement.

There are two types of connectives:

- Unary (a connective that applies to one statement)
  - Not (¬)
- Binary (a connective that applies to two statements)
  - And (∧)
  - Or (∨)
  - If…, then… (→)
  - If and only if… (↔)

$$\boldsymbol{Connective + proposition = complex\ statement}$$

Complex statements can be either **well-formed** or **not**. A complex statement is not well formed when a unary connector is used to modify two propositions.

Nevertheless, we can combine binary and unary connectives to form a well-formed complex statement.

Tom shut the door ***not*** *the sky is blue* – Not a well-formed complex statement

Tom shut the door ***and not*** *the sky is blue* – Is a well-formed complex statement

Note that in English we would say Tom shut the door **and** the sky is **not** blue

Propositional Symbols

Not	¬	negation
And	∧	conjunction
Or	∨	disjunction
If X, then Y	$X \to Y$	implication
X if and only if Y	$X \leftrightarrow Y$	equivalence

Formulae

A **well-formed formulae** (**wff**) is defined as:

- All prepositional symbols are (atomic) formulae.
- If A is a formula, then ¬A is also a formula.
- If A and B are formulae, then so are: $(A \land B), (A \lor), (A \to B), (A \leftrightarrow B)$.
- Nothing else is a formula.

Note that we can omit outer parenthesis for simplicity.

Ambiguity in translation

Let's say we have the following propositions

- *P* represents "Bob likes Mary"
- *Q* represents "Bob likes Sue"

If we say:

Bob likes Mary and Bob likes Sue – we represent it as $P \wedge Q$

But what if we have:

"If Bob is rich (B) then Sue is happy (S) and Jim is happy (J).

Natural language can be ambiguous, but **formal** language is not, that is why we use parentheses.

Translation One: $B \rightarrow (S \wedge J)$

Translation Two: $(B \rightarrow S) \wedge J$

We can avoid parenthesis if we assume a precedence order (a bit like BODMASS).

From STRONGEST to WEAKEST:

$$\neg\,,\ \wedge\,,\ \vee,\ \rightarrow,\ \leftrightarrow$$

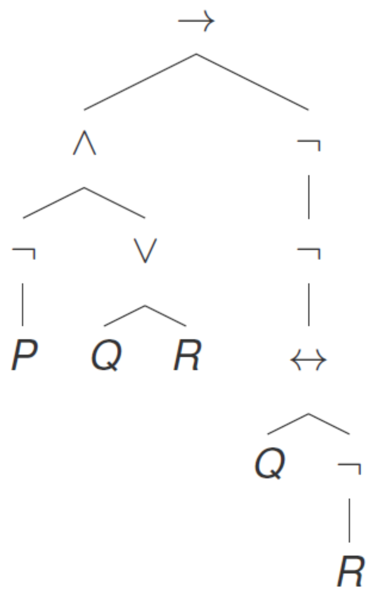
This means that translation one will now be: $B \rightarrow S \wedge J$ but translation two will remain the same.

Now that we have seen that connectives have a **fixed order of operations** we can create a **syntactical decomposition tree**.

Syntactical Trees

This is a very goof technique to see what is going on in your equation.

$$\big(\neg P \wedge (Q \vee R)\big) \rightarrow \big(\neg\neg(Q \leftrightarrow \neg R)\big)$$



Identifying falsehood or truth in a syntactic tree

It is easy to identify the number of outcomes because it is the number of prepositions.

$$Number\ of\ possibilities = 2^{n^{\underline{o}}\ of\ prepositions}$$

In the example above: $2^3 = 8$

Semantics

A WFF needs to be interpreted to give it a truth-value and thus a semantical meaning (1 or 0). To do this we can use **truth-tables** whilst using different interpretations for the formula.

$$v(P) = 1\ then\ P\ is\ true\ \boldsymbol{under\ interpretation\ v}$$
$$v(P) = 0\ then\ P\ is\ false\ \boldsymbol{under\ interpretation\ v}$$

Note that these are functions, hence for that value there cannot be two answers.

Truth-tables

Not (negation)

If we take the formula not of A:

$$v(\neg A) = \begin{Bmatrix} 0, if\ v(A) = 1 \\ 1, if\ v(A) = 0 \end{Bmatrix}$$

Truth-table:

A	$\neg A$
1	0
0	1

Remember that $\neg\neg A \equiv A$

And (conjunction)

The conjunction of $A \vee B$ is only true when both A and B are true. If we know $v(A)$ and $v(B)$ then:

$$v(A \wedge B) = \begin{Bmatrix} 1, & if\ v(A) = 1\ and\ v(B) = 1 \\ 0, & if\ v(A) = 0\ and\ v(B) = 0 \end{Bmatrix}$$

Truth-table:

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

Or (disjunction)

The disjunction of $A \vee B$ is true if, and only at least one of A, B or both are true.

$$v(A \vee B) = \begin{cases} 1, & if\ v(A) = 1\ or\ v(B) = 1 \\ 0, & if v(A) = 0\ and\ v(B) = 0 \end{cases}$$

Truth-table

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

TIP: It is only false if **both** negative. There is also something an exclusive or in which different rules apply.

If…,Then… (Implication)

An implication represents a conditional statement. In the statement $A \rightarrow B$, A is called the hypothesis (or antecedent or premise) and B is called the conclusion (or consequence).

$A \rightarrow B$ is often read as “if A, then B”.

TIP: The best way to think about it, is if it were like a promise.

“I promise, if you give me A, I will give you B.”

It is only false if I don’t give you B if you have given me A.

$$v(A \rightarrow B) = \begin{cases} 1, & if\ v(A) = 0\ or\ v(B) = 1 \\ 0, & if\ v(A) = 1\ and\ v(B) = 0 \end{cases}$$

A	B	$A \rightarrow B$
0	0	1
0	1	1
1	0	0
1	1	1

Equal (Equivalence, If and only if)

The equivalence $A \leftrightarrow B$ is true if and only if A and B have the same truth-values.

$$v(A \leftrightarrow B) = \begin{cases} 1, & if\ v(A) = v(B) \\ 0, & if\ v(A) \neq v(B) \end{cases}$$

Truth-table

A	B	$A \leftrightarrow B$
0	0	1
0	1	0
1	0	0
1	1	1

All interpretations for a formula

We use truth tables to see this.

interpretation	P	Q	R	$\neg Q$	$P \vee \neg Q$	$(P \vee \neg Q) \wedge R$
v_0						
v_1						
v_2						
v_3						
v_4						
v_5						
v_6						
v_7						

Tautologies and Contradictions

Tautology: is a formula which is true under all interpretations.

If we were to look at the formula $A \vee \neg A$, we will find it is always true, hence a tautology.

A	$\neg A$	$A \vee \neg A$
0	1	1
1	0	1

Contradiction: a formula which is false under all interpretations.

If we look at the formula $A \wedge \neg A$, we see that the statement will never be true hence a contradiction.

A	$\neg A$	$A \wedge \neg A$
0	1	0
1	0	0

Note that a **contingency** is a formula which is sometimes true and sometimes false.

Equivalences

Note that the symbol equals (=) is not within the logical language, in other words, whenever we use it within the logical language we are not creating a formula (it is not a formula).

Logical Equivalences

A and B are equivalent if **they are interchangeable with every possible interpretation.** Therefore, v(A) = v(B).

Meta language $A \equiv B$

Logical language: $A \leftrightarrow B$ (which is a tautology)

Fundamental Logical Equivalences

Idempotency: shows that whenever we pair two literals the logical equivalence is to the literal.

$$P \vee P \equiv P$$
$$P \wedge P \equiv P$$

Commutativity: the order in which you put the propositions does not count (for conjunction and disjunction).

$$P \vee Q \equiv Q \vee P$$

$$P \wedge Q \equiv Q \wedge P$$

Associativity: whenever you have three propositions with either conjunctions or disjunctions all together, the operational order (where the brackets go) is not important.

$$P \vee (Q \vee R) \equiv (P \vee Q) \vee R$$
$$P \wedge (Q \wedge R) \equiv (P \wedge Q) \wedge R$$

Distributivity: we can rearrange formulas which have a proposition by itself interacting with two others in brackets.

$$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$$
$$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$$

De Morgan's Law: shows that negation can be put into the brackets with the change of the symbol (conjunction or disjunction).

$$\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$$
$$\neg(P \vee Q) \equiv (\neg P \wedge \neg Q)$$

Excluded middle: note that **1** is meant to be a tautology and **0** is meant to be a contradiction.

$$\begin{aligned} P \vee \neg P &\equiv \mathbf{1} \\ P \wedge \neg P &\equiv \mathbf{0} \\ \neg\neg P &\equiv P \\ P \vee \mathbf{1} &\equiv \mathbf{1} \\ P \wedge \mathbf{1} &\equiv P \\ P \vee \mathbf{0} &\equiv P \\ P \wedge \mathbf{0} &\equiv \mathbf{0} \end{aligned}$$

Contraposition

$$\begin{aligned} P \to Q &\equiv \neg P \vee Q \\ P \to Q &\equiv \neg Q \to \neg P \\ \neg(P \to Q) &\equiv P \wedge \neg Q \end{aligned}$$

$$\begin{aligned} \mathbf{1} \leftrightarrow P &\equiv P \\ \mathbf{0} \leftrightarrow P &\equiv \neg P \\ \mathbf{1} \to P &\equiv P \\ P \to \mathbf{1} &\equiv \mathbf{1} \\ \mathbf{0} \to P &\equiv \mathbf{1} \\ P \to \mathbf{0} &\equiv \neg P \\ P \leftrightarrow P &\equiv \mathbf{1} \\ P \leftrightarrow \neg Q &\equiv \neg(P \leftrightarrow Q) \end{aligned}$$

Absorption

$$\begin{aligned} P \vee (P \wedge Q) &\equiv P \\ P \wedge (P \vee Q) &\equiv P \end{aligned}$$

The Equivalence Replacement Rule
Any sub-formula can be replaced by another equivalent sub-formula without changing the truth-value of its formula.

Example. From $\neg(Q \vee R) \equiv \neg Q \wedge \neg R$ we obtain, by replacement,

$$(P \to \underbrace{\neg(Q \vee R)}_{G}) \wedge \neg Q \equiv (P \to \underbrace{(\neg Q \wedge \neg R)}_{H}) \wedge \neg Q$$

We can use replacement to simplify formulas, such that they are sometimes equivalent to one proposition.

The formula $(P \wedge Q) \vee \neg(\neg P \vee Q)$ can be simplified as follows.

$$\begin{aligned}(P \wedge Q) \vee \neg(\neg P \vee Q) &\equiv (P \wedge Q) \vee (\neg\neg P \wedge \neg Q)\\ &\equiv (P \wedge Q) \vee (P \wedge \neg Q)\\ &\equiv P \wedge (Q \vee \neg Q)\\ &\equiv P \wedge \mathbf{1}\\ &\equiv P\end{aligned}$$

Literals

A **literal** is a propositional symbol or the negation of a propositional symbol.

Examples.

P, $\neg P$, Q, $\neg Q$, etc

Some formulae that are **not** literals.

$\neg\neg P$, $P \wedge Q$, $Q \vee Q$, etc

They can have at most one connective and it must be a negation.

Disjunctive Normal Form (DNF)

A disjunction of one or more formulae, each of which is a conjunction of one or more literals.

$(P \wedge \neg Q) \vee (\neg P \wedge Q)$

P (special case, think of it as $P \vee P$)

Every propositional formula has an equivalent form in DNF.

Conjunctive Normal Form (CNF)

A conjunction of one or more formulae, each of which is a disjunction of one or more literals.

$(P \vee \neg Q) \wedge (\neg P \vee Q)$

P (special case, think of it as $P \wedge P$)

Finding DNF and CNF from Truth Tables

Say we have the formula $(P \to Q) \wedge (Q \to P)$ and we want to put it in CNF and DNF.

If we draw up the truth table:

P	Q	$P \to Q$	$Q \to P$	$(P \to Q) \land (Q \to P)$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	1	1	1

This means that the interpretations that make our formula true are:

When P and Q are zero **or** when P and Q are one, which can be written:

$$f(0,0), f(1,1)$$

Which means we can write this as a DNF like so:

$$(\neg P \land \neg Q) \lor (P \land Q)$$

If a zero is present you put "not Literal" if there's a one, you put the literal.

To find the **CNF**, we take the interpretations that are false and negate them.

$$f(0,1), (1,0)$$

Rewrite rules to obtain Normal Form

To ***DNF*** *or* ***CNF*** and vice-versa

$$\begin{aligned}
F \to G &\implies \neg F \lor G \\
F \leftrightarrow G &\implies (F \to G) \land (G \to F) \\
\neg(F \lor G) &\implies \neg F \land \neg G \\
\neg(F \land G) &\implies \neg F \lor \neg G \\
\neg\neg F &\implies F \\
F \land (G \lor H) &\implies (F \land G) \lor (F \land H) \\
(F \lor G) \land H &\implies (F \land H) \lor (G \land H)
\end{aligned}$$

Simplifying rewrite rules

$$\begin{aligned}
\mathbf{0} \vee G &\Longrightarrow G \\
\mathbf{0} \wedge G &\Longrightarrow \mathbf{0} \\
\neg \mathbf{1} &\Longrightarrow \mathbf{0} \\
\cdots \wedge G \wedge \cdots \wedge \neg G \wedge \cdots &\Longrightarrow \mathbf{0} \\
\cdots \wedge G \wedge \cdots \wedge G \wedge \cdots &\Longrightarrow \cdots \wedge G \wedge \cdots \wedge \ldots
\end{aligned}$$

NOTE: When you have **0** we can eliminate this in DNF.

Complete Sets of Connectives

A set is considered complete if a group of connectives can represent all the other connectives (by using rewrite rules).

Since every formula has a DNF the set {¬, ∧, ∨} *is a* ***complete set****.*

Truth-functions

A truth-function is a function that can only take a true or false form.

Any wff defines a **truth-function**, and vice-versa.

Example. Let f be the truth-function defined as follows:

$$f(P, Q, R) = 1 \text{ iff either } P = Q = 0 \text{ or } Q = R = 1$$

Then f is equal to 1 in exactly the following four cases:

$$f(0,0,1), f(0,0,0), f(1,1,1), f(0,1,1)$$

f can be represented by the formula

$$(\neg P \wedge \neg Q \wedge R) \vee (\neg P \wedge \neg Q \wedge \neg R) \vee (P \wedge Q \wedge R) \vee (\neg P \wedge Q \wedge R)$$

We can represent this by creating a truth table and looking for the values of the literals which make the formula true.

Example

We can also find the **CNF** by negating the formula that we found for the **DNF**.

Substitution Instances

We can substitute certain formulas with other propositional variables in order to input other values into our equations.

Let $W, H_1, \dots, H_n$ be formulae and $P_1, \dots, P_n$ be propositional variables.

Then the expression $W(P_1/H_1, \dots, P_n/H_n)$ denotes the formula obtained by replacing *simultaneously* all occurrences of P_1 in W by the formula H_1, …, and all occurrences of P_n by the formula H_n.

Example

Let W be $P \to (Q \to P)$, then $W(P/\neg P \vee R, Q/\neg P)$ is $\neg P \vee R \to (\neg P \to \neg P \vee R)$.

We say that the formula $\neg P \vee R \to (\neg P \to \neg P \vee R)$ is a *substitution instance* of $P \to (Q \to P)$.

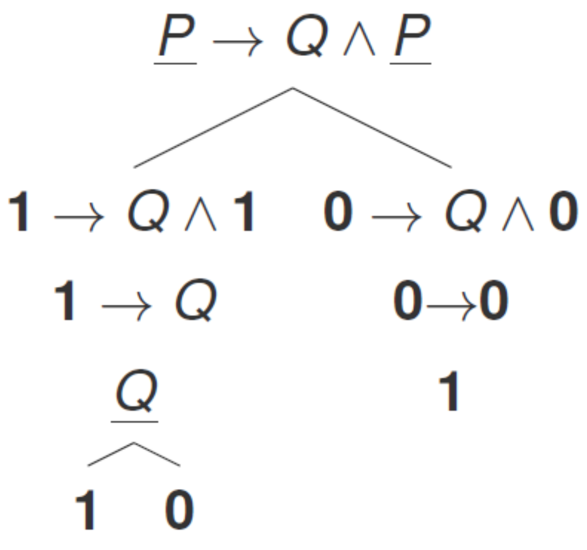
Quine's Method

Quine's method assumes certain parts of our formula are true or false and evaluates if this is possible.

On the left we assume that P is true whilst on the right we assume that P is false.

By using general simplification rules we can shrink our formulae.

Finally, we conclude that Q can be either a tautology or a contradiction if P is true, and if it's false we can see that Q will be true if P is false. Hence we know that our formula is a Contingency since it can sometimes be true or false (if all where to be 1's or all 0's it would be a tautology or contradiction respectively.)



Satisfiability

A formula F is satisfiable if there is an interpretation v that makes the formula F true. We say that v satisfies F.

Additionally, we can also check for satisfiability of a set of formulas say in a set S.

A set $\mathcal{S} = \{A_1, \ldots, A_n\}$ of propositional formulae is **satisfiable** (**consistent**) if there is an interpretation v satisfying *every* formula in $\mathcal{S}$.

	p_1	$\ldots$	p_m	A_1	$\ldots$	A_n
v	e_1	$\ldots$	e_m	1	$\ldots$	1

$p_1, \ldots, p_m$ are all the propositional symbols appearing in $\mathcal{S}$.

The set of formulae $\{A_1, \ldots, A_n\}$ is satisfiable if, and only if, the conjunction $A_1 \wedge \ldots \wedge A_n$ is satisfiable

Example

Let $A_1 = P \to Q$, $A_2 = Q \to R$ and $A_3 = R \to P$ and $\mathcal{S}$ be the set of formulae $\{A_1, A_2, A_3\}$. The combined truth-table for $\mathcal{S}$ is:

	p_1	p_2	p_3	A_1	A_2	A_3
	P	Q	R	$P \to Q$	$Q \to R$	$R \to P$
v_0	0	0	0	1	1	1
v_1	0	0	1	1	1	0
v_2	0	1	0	1	0	1
v_3	0	1	1	1	1	0
v_4	1	0	0	0	1	1
v_5	1	0	1	0	1	1
v_6	1	1	0	1	0	1
v_7	1	1	1	1	1	1

Thus, $\mathcal{S}$ is satisfiable, since v_0 and v_7 satisfy every formula in $\mathcal{S}$ (we can also say that $\mathcal{S}$ is *consistent*).

A **model** is an interpretation that makes a formula (or a set of formulae) true.

We denote the fact that v is a model of A by $v \Vdash A$.

The set of all models of a formula A is denoted by $mod(A)$.

Referring to our example:

$$mod(S) = \{v_0, v_7\}$$

Non-satisfiable example

Let $\mathcal{S}$ be the set of formulae $\{P \leftrightarrow \neg Q, Q \leftrightarrow R, R \leftrightarrow P\}$. The truth-table for its formulae is:

	P	Q	R	$P \leftrightarrow \neg Q$	$Q \leftrightarrow R$	$R \leftrightarrow P$
v_0	0	0	0	0	1	1
v_1	0	0	1	0	0	0
v_2	0	1	0	1	0	1
v_3	0	1	1	1	1	0
v_4	1	0	0	1	1	0
v_5	1	0	1	1	0	1
v_6	1	1	0	0	0	0
v_7	1	1	1	0	1	1

$\mathcal{S}$ is **not** satisfiable (i.e., $\mathcal{S}$ is inconsistent) and thus, $\text{mod}(\mathcal{S}) = \varnothing$.

Logical Consequence

In a **valid** argument, we can conclude B if it follows a set of premises $A_1, \dots, A_n$.

We can say that B is a logical consequence of $A_1, \dots, A_n$ if the implication hold for ***every interpretation v:***

$$\text{If } v(A_i) = 1, \text{ for all } 1 \le i \le n, \text{ then } v(B) = 1.$$

In other words, B is a logical consequence of the set of formulas **only if** in all instances in which the premises (set of formulae) are true the conclusion (B) is true.

We can say logical consequence in the following forms:

- B is a logical consequence of $A_1, \dots, A_n$.
- $A_1, \dots, A_n \models B$.
- The argument $A_1, \dots, A_n \models B$ is *valid*.
- B is *semantically entailed (or implied)* by $A_1, \dots, A_n$
- B is a *valid consequence* of $A_1, \dots, A_n$.

Example:

Show that $P, P \to Q \models P \wedge Q$.

Solution:

	P	Q	$P \to Q$	$P \wedge Q$
	0	0	1	0
	0	1	1	0
	1	0	0	0
$*$	1	1	1	1

The statement follows because in every row in which the columns for P and $P \to Q$ contain 1, so does the column for $P \wedge Q$.
In this case the only relevant row is the one with $*$.

Equivalent definitions of B is logically equivalent to the set of formulas.

$A_1, \ldots, A_n \models B$ if and only if

- $A_1 \wedge \ldots \wedge A_n \to B$ is a tautology (i.e., logically valid).
- $A_1 \wedge \ldots \wedge A_n \wedge \neg B$ is a contradiction (i.e., unsatisfiable)
- The set $\{A_1, \ldots, A_n, \neg B\}$ is inconsistent.

Let the symbol $\mathcal{I}$ denote the set of all interpretations.

Let $\mathcal{S} = \{A_1, \ldots, A_n\}$. We have that

$$\mathcal{S} \models B \text{ iff } \operatorname{mod}(\mathcal{S}) \subseteq \operatorname{mod}(B)$$

Notice that $\operatorname{mod}(\neg B) = \mathcal{I} - \operatorname{mod}(B)$ and hence $\operatorname{mod}(B) \cap \operatorname{mod}(\neg B) = \varnothing$.

Therefore, if $\mathcal{S} \models B$, then $\operatorname{mod}(\mathcal{S}) \cap \operatorname{mod}(\neg B) = \emptyset$ and hence $\mathcal{S} \cup \{\neg B\}$ is unsatisfiable.

Another way of expressing something like logical consequence is:

$$\frac{\begin{array}{l} J \to (H \land \neg C) \\ J \land (H \to C) \end{array}}{J}$$

Note that checking the validity of a statement is an exponential process. $2^k = \textit{number of solutions}$ in which k is the number or literals.

Inference Systems

Reasoning of an argument can be achieved not only by truth tables but also by an ***inference system (a deduction system)*** [a logical way of looking at it].

We can refer to it as a *computational device deriving formulae that derive from available premises.*

An **inference system** has to be:

Sound: all answers should be the same as when using a truth table (correct).

$$\text{if } A_1, \ldots, A_n \vdash B, \text{ then } A_1, \ldots, A_n \models B.$$

Complete: shows all the conclusions that can be made with a truth table (it can do all operations) or if B can be derived from the premises using its inference rules.

$$\text{if } A_1, \ldots, A_n \models B, \text{ then } A_1, \ldots, A_n \vdash B.$$

Correctness: the system follows its expected behavior.

Note that $\vdash \equiv | =$ just that one dash is for Natural Deduction whilst two dashes denotes proof via Truth Table.

It is described by a set of inference (or deviation) rules in the form:

$$\frac{A_1, \ldots, A_n}{B} \quad \begin{array}{l} \Rightarrow \quad \textit{Premises} \\ \Rightarrow \quad \textit{Conclusion} \end{array}$$

Formula B can be generated as long as $A_1, \ldots, A_n$ is true.

Natural Deduction

Natural Deduction is an inference system that manipulates assumptions towards conclusions. It is a *forward reasoning system.*

There are two ways of manipulating assumptions:

Introduction rule ($\ominus$ I): when we combine formulae with connectives to generate new formulae.

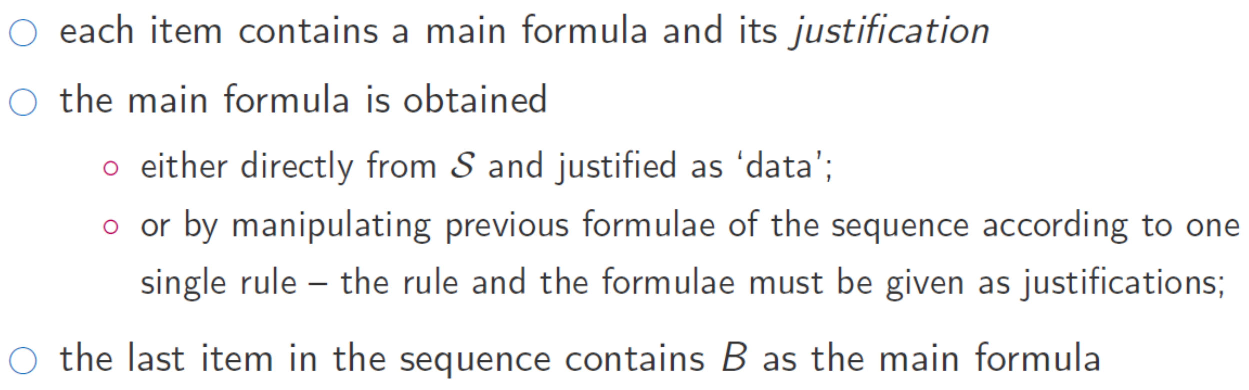
Elimination rule ($\ominus$ E): when we break down formulae by eliminating connectives to generate new formulae.

*$\ominus$ is a connective.

E.g. $\wedge E$ – Conjunction elimination.

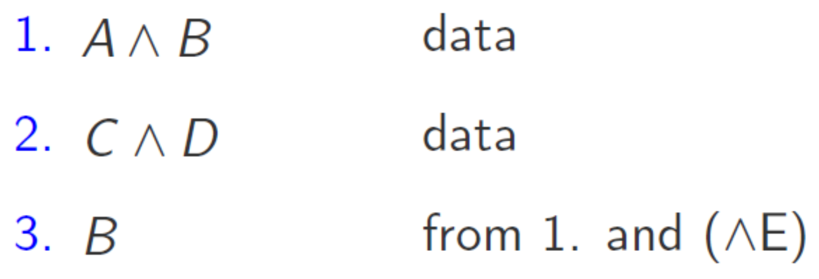
Natural Deduction Proof

Let S be a set of formulae. A natural deduction proof of $S \vdash B$ is a sequential number list such that:



- each item contains a main formula and its *justification*
- the main formula is obtained
  - either directly from $\mathcal{S}$ and justified as 'data';
  - or by manipulating previous formulae of the sequence according to one single rule – the rule and the formulae must be given as justifications;
- the last item in the sequence contains B as the main formula

For the specific example $A \wedge B,\ C \wedge D \vdash B$



1.	$A \wedge B$	data
2.	$C \wedge D$	data
3.	B	from 1. and ($\wedge$E)

- Before a rule can be applied, its premises must already appear in the proof.
- The results of the application of the rule is the new item in the proof.

Rules

Rules for Conjunction

$$\frac{A, B}{A \wedge B} \ (\wedge\mathrm{I}) \qquad \frac{A \wedge B}{A} \ \text{and} \ \frac{A \wedge B}{B} \ (\wedge\mathrm{E})$$

Rules for disjunction

$$\frac{A}{A \vee B} \ (\vee\mathrm{I}) \quad \text{and} \quad \frac{B}{A \vee B} \ (\vee\mathrm{I})$$

$$\frac{A \to C, B \to C, A \vee B}{C} \ (\vee\mathrm{E})$$

Rules for implication

$$\frac{A, A \to B}{B} \ (\to\mathrm{E})$$

$$\frac{\text{premises}}{A \to B} \ , \qquad \text{if} \ \frac{\text{premises}, \ A}{B} \qquad (\to \mathrm{I})$$

To do implication Introduction we must use a subcomputation box, in which we assume that the premises are true.

Subcomputation boxes
Defines a sub-proof that is dependent on an extra assumption.

Consider the following proof: $A \to B, B \to C \vdash A \to C$

1.	$A \to B$	data
2.	$B \to C$	data
3.	$A \to C$	($\to$I), from the subcomputation below

we write in our proof what we try to with the subcomputation box (3.).

we must write what we aim to prove --> $\underline{C}$

3.1	A	assumption
3.2	B	from (3.1), (1.) and ($\to$E)
3.3	C	from (3.2), (2.) and ($\to$E)

we can reference from inside and out the box

assumptions are only true inside the subcomputation box.

Rules for negation

$$\frac{A \to B, A \to \neg B}{\neg A} \quad (\neg \mathrm{I})$$

$$\frac{\neg A \to B, \neg A \to \neg B}{A} \quad (\neg \mathrm{E})$$

Variant Rules

There are other rules which are called "variants" which allow us to "skip" steps and simplify proofs. All of these can be derived from basic rules.

Disjunction $$\frac{A \vee B, \neg A}{B} \quad (\vee\text{E1}) \qquad \frac{A \vee B, \neg B}{A} \quad (\vee\text{E2})$$

Negation $$\frac{\neg\neg A}{A} \quad (\neg\text{E1}) \qquad \frac{\neg A \to B, A \to B}{B} \quad (\neg\text{E2})$$

Implication $$\frac{\neg A}{A \to B} \quad (\to\text{I1}) \qquad \frac{B}{A \to B} \quad (\to\text{I2})$$

$$\frac{A \to B}{\neg A \vee B} \quad (\to\text{E1})$$

Predicate Logic

The type of arguments representable in propositional logic is somewhat limited since we cannot refer to anything mentioned in such a statement.

Consider the statements:

- P: Blue cars are fast.
- Q: My car is blue.
- R: My car is fast.

We would like to conclude that R is a logical consequence of the first two, but with propositional logic alone, this cannot be done.

We need logic that can look deeper into the structure of statements and that is capable of representing relationships about individuals of a particular domain (e.g a collection of cars). Predicate logic allows us to do this.

Quantifiers

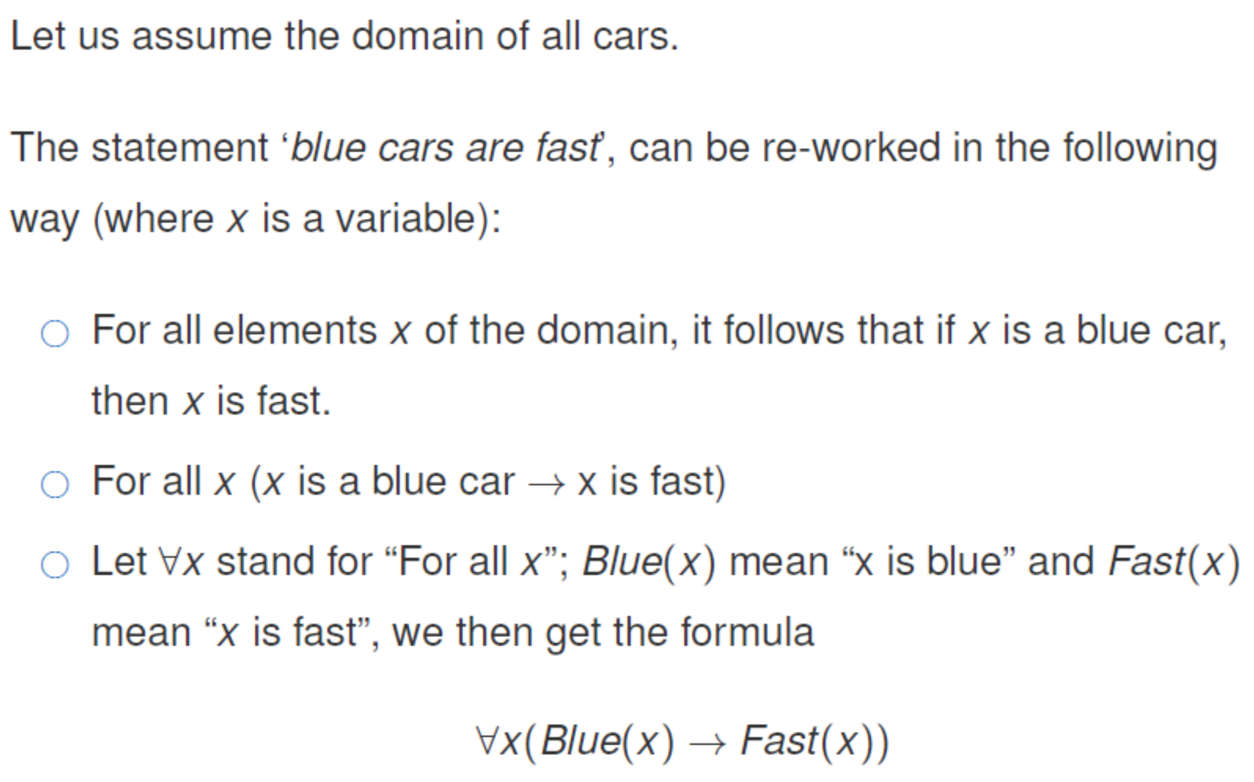
Universal quantifier

It is the quantifier that tells us "for all". For instance, the symbol $\forall x$ means for all x.

Existential quantifier

The **existential quantifier ($\exists y$)** talks about one particular element in the domain.

$\exists x \forall y(< (x, y) \vee = (x, y))$ which is read *"There is an element x, that for all elements y in which x < y or x = y."*



Let us assume the domain of all cars.

The statement ‘*blue cars are fast*’, can be re-worked in the following way (where x is a variable):

- For all elements x of the domain, it follows that if x is a blue car, then x is fast.
- For all x (x is a blue car $\rightarrow$ x is fast)
- Let $\forall x$ stand for “For all x”; $Blue(x)$ mean “x is blue” and $Fast(x)$ mean “x is fast”, we then get the formula

$$\forall x(Blue(x) \rightarrow Fast(x))$$

This can be read “All blue cars are fast cars.”

Using the predicate logic

If we had something like:

$$teaches(x, y)$$

$$teaches(Odinaldo, ELA)$$

We would read it as “*Odinaldo teaches ELA*”.

Unary predicate symbol – takes 1 value such as $max(x)$

Binary predicate symbol – takes 2 values such as $teaches(x, y)$

Domain

The domain of disclousure is the set of elements that our formula is for. We cannot have some formulae with a certain domain and other formulae with another domain as the formulas will be interpreted against one single domain. Also know that all quanitifiers apply to this domain.

This domain can never be empty.

Symbols

Predicate symbols

Unary or **Binary,** they will be true or false if the element has the designated property.

Blue(x) – true if x is blue.

Travels(x,y) – true if x travels to y.

Function symbols

Function symbols are used when an output is wanted from the domain.

+(x,y) or x + y, it gives out a number.

Quantifiers

They describe the extent to which certain properties occur.

$\forall x$ – which means for all x

This implies that all the values of the domain.

TIP: This tends to have an implication.

$\exists x$ – for some (at least one) x

This implies that at one value in the domain.

TIP: This tends to have a conjunction.

Example 1

Representing "Any integer greater than 5 is also greater than 3"

Assuming domain are numbers in general, we will need two predicate symbols:

- One to represent the integer numbers.
- One to represent that number is greater than another one.

NOTE is our domain were just integers we wouldn't need the first one.

The formula for our statement would be the following:

$$\forall x \left(\left(Int(x) \wedge >(x,5) \right) \rightarrow >(x,3) \right)$$

- Let the predicate *Int* be used to designate that a number is integer and the predicate $>(x, y)$ be used to indicate that the integer in the variable x is greater than the integer in the variable y
- We could 'name' the number **5** as the constant symbol 5 and the number **3** as the constant symbol 3
- *Int* is a unary predicate symbol denoting the property that the object referred to by it is integer (*Int* is a unary relation!)
- $>$ is a binary predicate symbol such that $>(x, y)$ denotes that 'x is greater than y' ($>$ is a binary relation!)

For simplicity we might also write our formula in *infix* form:

$$\forall x((Int(x) \wedge x > 5) \rightarrow x > 3)$$

This can be read as "For all x if x is an integer then x > 5 and therefore x > 3.

Example 2
Anybody(x) that likes John(a) likes at least one of John's brothers(y).

$$\forall x(Likes(x, a) \rightarrow \exists y(Brother(y, a) \wedge Likes(x, y))$$

For all x which like John, there is a y who is a brother of John and is liked by x.

Example 3
Someone likes someone.

- There exists (at least) a person x, such that x likes somebody
- There exists (at least) a person x and (at least) a person y such that x likes y.
- Let $\exists x$ stand for "there exists an element of the domain x" and $Likes(x, y)$ mean that x likes y, we get: $\exists x \, \exists y \, Likes(x, y)$.

Terms

A **term** is either a variable, a constant, or a function symbol applied to arguments that are terms.

- **Individual variables:** denote arbitrary objects of the domain.
- **Individual constants:** denote particular objects from the domain.
- **Function symbols:** denote particular functions over the domain.

A term always evaluates to **an element of the domain**, whereas the atomic formula always evaluates to **true or false**.

Well formulated formulas (WFF)

A well formulated formula has all of these as their properties:

1. Any atom is a wff (e.g. $P(x),\ Q(x,y),\ x>y,\ x=a$)
2. If $\mathcal{F}$ and $\mathcal{G}$ are wffs, then $\neg(\mathcal{F})$, $(\mathcal{F}\wedge\mathcal{G})$, $(\mathcal{F}\vee\mathcal{G})$, $(\mathcal{F}\rightarrow\mathcal{G})$ and $(\mathcal{F}\leftrightarrow\mathcal{G})$ are wffs
3. If $\mathcal{F}$ is a wff and x is a variable then $\forall x(\mathcal{F})$ and $\exists x(\mathcal{F})$ are wffs

A 'substring' of a wff $\mathcal{F}$ which is itself well-formed is a *subformula* of $\mathcal{F}$.

Connective strength (precedence)

The quantifiers have the same strength as the not and behave like so.

$$\forall x(\neg(\exists y(\forall z(P(x,y,z)))))\ \text{(parenthesised form)}$$

$$\forall x\neg\exists y\forall zP(x,y,z)\ \text{(unparenthesised form)}$$

Scope

A formula can said to be bound to a quantifier if it is within the scope of this quantifier.

$$\forall x(F),\qquad F\ is\ in\ the\ scope\ of\ \forall x, hence\ all\ x\ in\ F\ are\ bound\ to\ this\ quantifier$$

Note that a variable can have both free and bound occurrences in a wff.

Closed wff also called a ***sentence***, occurs when all variables are bound.

Open wff are the wff which can have some bound and some open variables.

Example

2. $\exists x \exists y(P(x,y) \rightarrow Q(x))$

The scope of $\exists x$ is $\exists y(P(x,y) \rightarrow Q(x))$. Therefore all three occurrences of x are bound.

The scope of $\exists y$ is $P(x,y) \rightarrow Q(x)$. Therefore, both occurrences of y are bound.

Note that this formula is a sentence.

Substituting variables

Given a domain D, we have to interpret every element of the domain as a constant. When we do so and we want to substitute a variable of our wff with a constant d in our domain such that $d \in D$, we can $F(x/d)$ but we can only substitute the **free occurrences** of x.

Example. Let $\mathcal{F}$ be $P(x) \wedge \exists x \exists y P(x,y)$ and $D = \mathbb{N}$. Then $\mathcal{F}(x/0)$ is $P(0) \wedge \exists x \exists y P(x,y)$.

Interpreting quantifiers

- $\mathcal{I}(\forall x(\mathcal{F})) = 1$ iff **for every** substitution for x of an element d taken from D, we have that $\mathcal{I}(\mathcal{F}(x/d)) = 1$. Otherwise $\mathcal{I}(\forall x(\mathcal{F})) = 0$.
- $\mathcal{I}(\exists x(\mathcal{F})) = 1$ iff **for some** substitution for x of an element d taken from D, we have that $\mathcal{I}(\mathcal{F}(x/d)) = 1$. Otherwise $\mathcal{I}(\exists x(\mathcal{F})) = 0$.

RULE OF THUMB:

1. Take the outermost quantifier out.
2. Substitute a value from the domain for the free instances of the new formula.
    a. If this is true, then the original is true and vice-versa.
3. This process might have to be repeated several times, if there are various quantifiers.

Categories of predicate logic sentences

- **Logically valid (tautology)**: if it is true under **every** interpretation.
- **Satisfiable (contringency):** if it is true under **some** interpretation.
- **Unsatisfiable (contradiction)**: if it is **false** under every interpretation.

Two sentences are **logically equivalent** if: $F \equiv G$ if $F \leftrightarrow G$.

Quantifier equivalences

$\neg \forall x F \equiv \exists x \neg F$
$\neg \exists x F \equiv \forall \neg F$

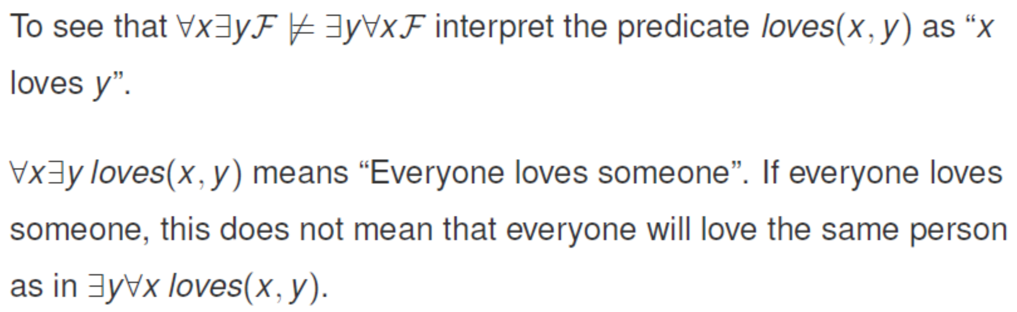
Consequently, if we were to negate both sides, it will also be equivalent:

$\forall xF \equiv \neg\neg\forall xF \equiv \neg\exists x\neg F$
$\exists xF \equiv \neg\neg\exists xF \equiv \neg\forall x\neg F$

Additionally, when you have more than one type of quantifier, the order of appearance is irrelevant.

$$\forall x\forall yF \equiv \forall y\forall xF$$

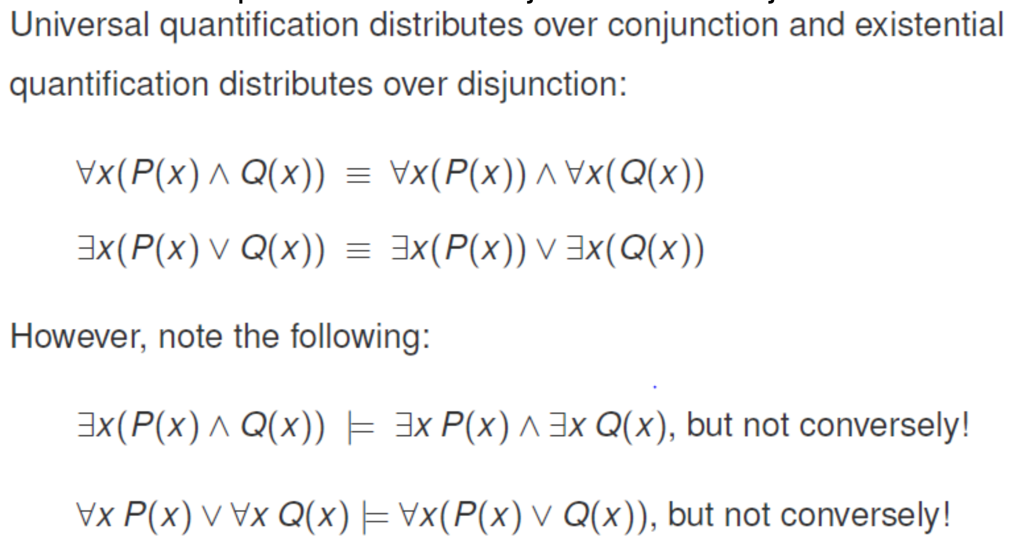
Although when different quantifiers appear **we have to be careful**:



To see that $\forall x\exists y\mathcal{F} \not\models \exists y\forall x\mathcal{F}$ interpret the predicate $loves(x, y)$ as "x loves y".

$\forall x\exists y\ loves(x, y)$ means "Everyone loves someone". If everyone loves someone, this does not mean that everyone will love the same person as in $\exists y\forall x\ loves(x, y)$.

Distribution of quantifiers over conjunction and disjunction



Universal quantification distributes over conjunction and existential quantification distributes over disjunction:

$$\forall x(P(x) \wedge Q(x)) \equiv \forall x(P(x)) \wedge \forall x(Q(x))$$

$$\exists x(P(x) \vee Q(x)) \equiv \exists x(P(x)) \vee \exists x(Q(x))$$

However, note the following:

$\exists x(P(x) \wedge Q(x)) \models \exists x\, P(x) \wedge \exists x\, Q(x)$, but not conversely!

$\forall x\, P(x) \vee \forall x\, Q(x) \models \forall x(P(x) \vee Q(x))$, but not conversely!

Transformations and simplifications

- When the variable x has no occurrence in a formula $\mathcal{F}$, then binding x with a quantifier in that formula has no effect.

  Example:

  $$\forall x \forall y(P(y) \to Q(y)) \equiv \forall y(P(y) \to Q(y)) \equiv \exists x \forall y(P(y) \to Q(y))$$

- When the variable x is already bound by another quantifier in the formula $\mathcal{F}$, then binding x again has no effect.

  Examples:

  $$\forall x(\forall x(P(x) \to Q(x)) \equiv \forall x(P(x) \to Q(x))$$
  $$\exists x(\forall x(P(x) \to Q(x)) \equiv \forall x(P(x) \to Q(x))$$

This is called vacuous quantification.

If y is a new variable that does not occur in $\mathcal{F}$, then the following equivalences hold:

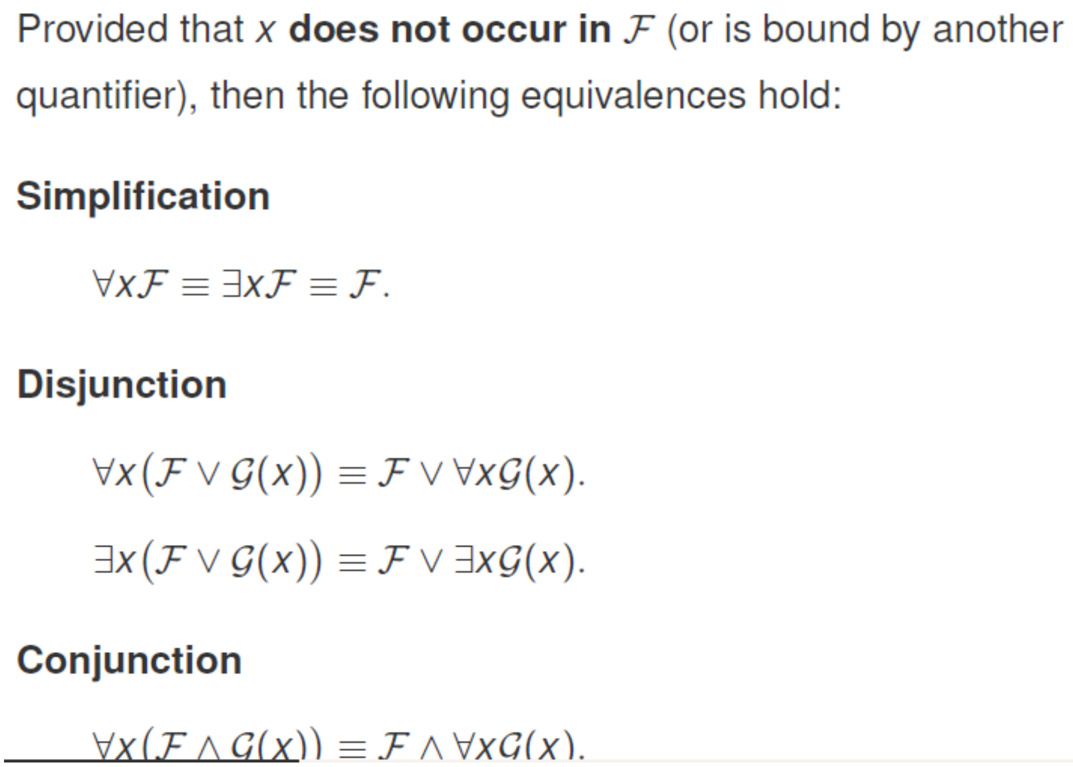
$$\exists x \mathcal{F} \equiv \exists y \mathcal{F}(x/y).$$
$$\forall x \mathcal{F} \equiv \forall y \mathcal{F}(x/y).$$

Recall that $\mathcal{F}(x/y)$ is obtained from $\mathcal{F}$ by replacing all **free** occurrences of x in it by y.

Example

$$\forall x P(x) \vee \forall x Q(x) \to \forall x(P(x) \vee Q(x)) \equiv \forall y P(y) \vee \forall z Q(z) \to \forall x(P(x) \vee Q(x))$$

On the second half of the equivalence above, all quantifiers use a different variable.



Provided that x **does not occur in** $\mathcal{F}$ (or is bound by another quantifier), then the following equivalences hold:

Simplification

$$\forall x \mathcal{F} \equiv \exists x \mathcal{F} \equiv \mathcal{F}.$$

Disjunction

$$\forall x (\mathcal{F} \lor \mathcal{G}(x)) \equiv \mathcal{F} \lor \forall x \mathcal{G}(x).$$

$$\exists x (\mathcal{F} \lor \mathcal{G}(x)) \equiv \mathcal{F} \lor \exists x \mathcal{G}(x).$$

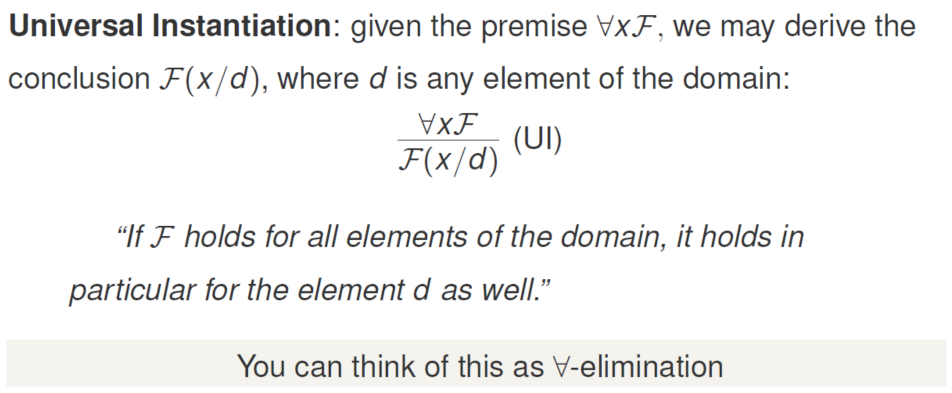
Conjunction

$$\forall x (\mathcal{F} \land \mathcal{G}(x)) \equiv \mathcal{F} \land \forall x \mathcal{G}(x).$$

Quantifier Rules

We can now add new rules to the previous propositional rules for natural deduction.

Universal Instantiation



Universal Instantiation: given the premise $\forall x \mathcal{F}$, we may derive the conclusion $\mathcal{F}(x/d)$, where d is any element of the domain:

$$\frac{\forall x \mathcal{F}}{\mathcal{F}(x/d)} \text{ (UI)}$$

"If $\mathcal{F}$ holds for all elements of the domain, it holds in particular for the element d as well."

You can think of this as $\forall$-elimination

Universal Generalization

Universal Generalisation: given the premise $\mathcal{F}(x/d)$ takes place for any element d in the domain, we may derive $\forall x\mathcal{F}$.

$$\frac{\mathcal{F}(x/d)}{\forall x\mathcal{F}}\ \text{(UG)}$$

Note. The element d in the premise of (UG) must not be specific but **arbitrary**, i.e., we cannot make any assumptions about d other that it comes from the domain.

You can think of this rule as $\forall$-introduction

Existential Instantiation

Existential Instantiation : given the premise $\exists x\mathcal{F}$, we may derive $\mathcal{F}(x/e)$ where e is a special element of the domain:

$$\frac{\exists x\mathcal{F}}{\mathcal{F}(x/e)}\ \text{(EI)}$$

Note. We cannot select an arbitrary element of the domain in (EI), but rather *it must be an element e for which $\mathcal{F}(x/e)$ is true.*

Usually, we have no knowledge of what the actual element is, only that it does exist. Because it exists, we can give it a *new* name, not used anywhere before, say, e and continue our derivation.

You can think of this rule as $\exists$-elimination

Existential Generalisation

Existential Generalisation: given that $\mathcal{F}(x/d)$ is known to hold for a particular element d of the domain, we may derive $\exists x\mathcal{F}$.

$$\frac{\mathcal{F}(x/d)}{\exists x\mathcal{F}}\ \text{(EG)}$$

You can think of this rule as $\exists$-introduction

Applications

Propositional Horn and Definite clauses

Clause: a disjunction **of one or more** literals.

So now CNF is a conjunction of clauses.

$(\neg P \vee Q) \wedge (\neg P \vee R) - \textit{has two clauses}$
$\neg S \vee \neg R \vee T - \textit{has one clause}$

Horn clause: a clause with **no more than one** positive literal (can have zero).

Definite clause: a clause with **exactly one** positive literal (must have one).

1.Take any propositional well formed formula (wff) α
(from now on greek letters α (alpha), β (beta), γ (gamma) will denote any wff i.e. α stands for $\neg P, P \wedge Q, P \rightarrow Q$ …)

2.Transform α to CNF. Let's write '*cnf*(α)' to denote the transformation of α to CNF

3.*cnf*(α) is a formula of the form:
$\boldsymbol{\beta_1 \wedge \ldots \wedge \beta_n}$
where $n \geq 1$, and each $\boldsymbol{\beta_i}$ is a clause
(a disjunction of literals: positive or negative propositional variables)

4. If β_i is of the form
$\boldsymbol{\neg X_1 \vee \ldots \vee \neg X_m \vee X,\ m \geq 0,}$
(exactly one positive and 0 or more negative literals) then $\boldsymbol{\beta_i}$ is
definite clause

Changing definite clauses into definite rules
Using associativity, DeMorgane's laws and equivalence for implication:

- A definite clause of the form

$\neg \mathbf{X_1} \vee \ldots \vee \neg \mathbf{X_m} \vee \mathbf{X}$
can be represented as the equivalent:
- $\mathbf{X_1} \wedge \ldots \wedge \mathbf{X_m} \rightarrow \mathbf{X}$

$\neg X_1 \vee \ldots \vee \neg X_m \vee X$
$\equiv (\neg X_1 \vee \ldots \vee \neg X_m) \vee X$
$\equiv \neg(X_1 \wedge \ldots \wedge \neg X_m) \vee X$
$\equiv \mathbf{X_1} \wedge \ldots \wedge \mathbf{X_m} \rightarrow \mathbf{X}$

Once we have created definite rules, we can program with them.

Example

$\neg S \vee \neg R \vee T \equiv \neg(S \wedge T) \vee T \equiv (S \wedge T) \rightarrow T$

Now that we have this, we say that the formula to the right is the **definite rule**.

Logic Programming

Logical programming is a declarative style of programming. The programmer tells the program about certain things in the world. Then it asks a question. It is up to the compiler to deduce if what the programmer asks is a logical consequence of what he already knows.

Some languages like these are SQL and Prolog.

- A set of formulas that describe the world are given to the program (this is called the knowledge base).
- The user supplies a logical formula which might hold or not called a **query**.
- The system determines if this is a logical consequence of the properties in the program.

We normally use a truth table or natural deducation to determine this. But both are inefficient methods because they are difficult to code and truth tables' complexity are exponential meaning that the more variables we have the more it slows down our program. This means that we need an **algorithm** a set of logical steps that can be coded and carried out everytime to get a successful answer.

Definite Clause Prepositional Programming

First we list all the **definite clauses** of the formulas in CNF. Then we represent these as rules.

$$\boxed{\begin{array}{l} P \\ (\neg P \lor Q) \\ (\neg P \lor R) \\ \neg Q \lor S \\ \neg S \lor \neg R \lor T \end{array}} \longrightarrow \boxed{\begin{array}{l} \to P \\ P \to Q \\ P \to R \\ Q \to S \\ (S \land R) \to T \end{array}}$$

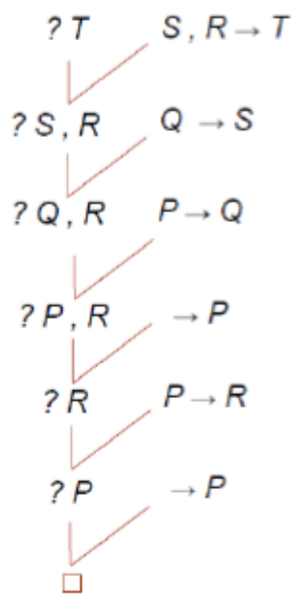
Then we create a query, note that we no longer use the conjunction sign, instead we use a comma and we always take one side of the conjunction or now coma, either left or right and stick to it.

When we process a query, we look for a head of the implication to match our query and we exchange it from the body. We can do this in a derivation free as follows.

Query $\mathcal{P}1$

$\mathcal{P}1$

$\rightarrow P$
$P \rightarrow Q$
$P \rightarrow R$
$Q \rightarrow S$
$S, R \rightarrow T$



SUCCEEDS

leftmost query selection always applied

If the program encounter's BLANK it succeeds!

Formal procedure:

- Let $\boldsymbol{P}$ be a definite clause program containing definite rules of the form $X_1, \dots, X_n \rightarrow Y$
- Let $?\,Q_1, \dots, Q_m$ be a query on $\boldsymbol{P}$

1. Choose a literal Q_i
2. *If* there is **no rule** $X_1, \dots, X_n \rightarrow Q_i$ in *P* then **exit** and **fail**,
   else **choose** a rule $X_1, \dots, X_n \rightarrow Q_i$ in $\boldsymbol{P}$
3. In the query term
   $?\,Q_1, \dots, Q_{i-1}, Q_i, Q_{i+1}, \dots, Q_m$
   replace Q_i by $X_1, \dots, X_n$:
   $?\,Q_1, \dots, Q_{i-1}, X_1, \dots, X_n, Q_{i+1}, \dots, Q_m$
4. *If* the query term is empty then **exit** and **success**,
   else go to step 1 and repeat 1 - 4.

Predicate Definite Clause Programming

Prenex Normal Form

Prenex normal form: a formula which starts with 0 or more quantifiers followed by a formula with no quantifiers.

$$Q_1x_1 \ldots Q_nx_nF$$

Where F is the formula with no quantifiers and the Q is either the existential or universal. The quantifiers are called the prefix and the formula the matrix.

All **wff's** can be transformed into PNF.

Algorithm to transform into PNF

1. Remove $\rightarrow$ *and* $\leftrightarrow$
2. Move negations inward (negations are only in front of atoms)
3. Rename variables, so that the variables for each quantifier are unique (standardization)
4. Move the quantifiers to the front.

Predicate Horn and Definite Clauses

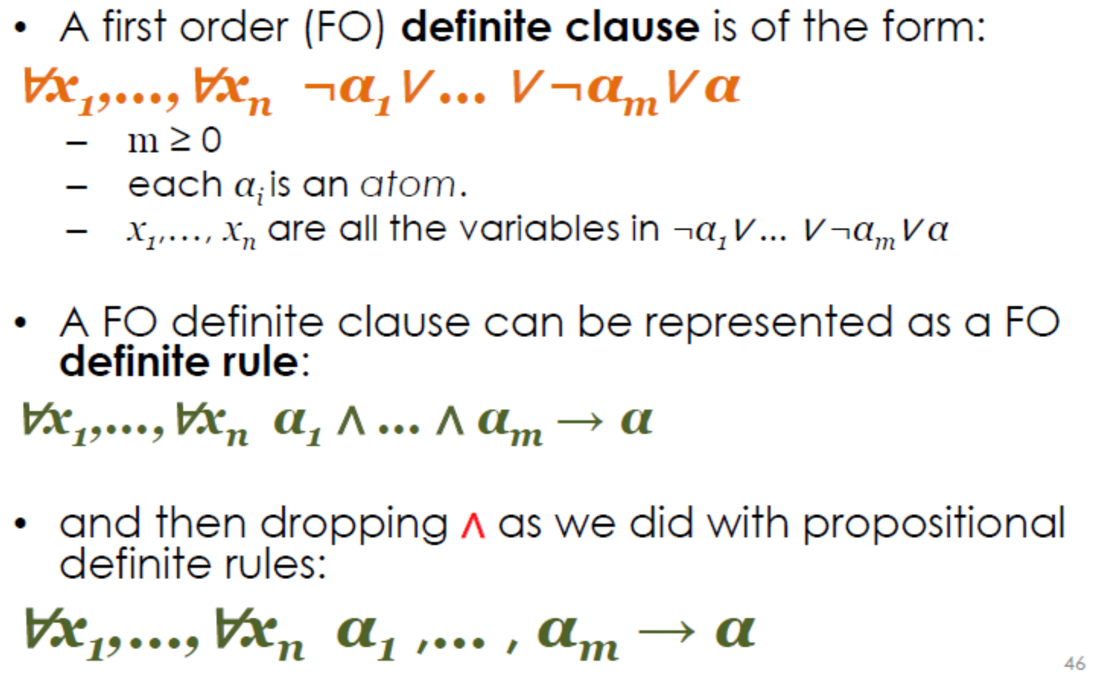
Horn clause: if a formula is in PNF, contains no more than one positive atom and **all quantifiers** are **universal**.

$\forall x \, \forall y \, \forall z \, \forall w$ *(1) is met)*

$\neg R(x, y) \lor S(x, z) \lor \neg P(x, w)$ *(2) is met)*

Definite clause: a horn clause with **exactly one positive** atom and 0 or more negative atoms.

Definite Clauses to Definite Rules



- A first order (FO) **definite clause** is of the form:

$$\forall x_1,\ldots,\forall x_n \ \neg\alpha_1 \vee \ldots \vee \neg\alpha_m \vee \alpha$$

  - $m \geq 0$
  - each α_i is an *atom*.
  - $x_1,\ldots,x_n$ are all the variables in $\neg\alpha_1 \vee \ldots \vee \neg\alpha_m \vee \alpha$

- A FO definite clause can be represented as a FO **definite rule**:

$$\forall x_1,\ldots,\forall x_n \ \alpha_1 \wedge \ldots \wedge \alpha_m \rightarrow \alpha$$

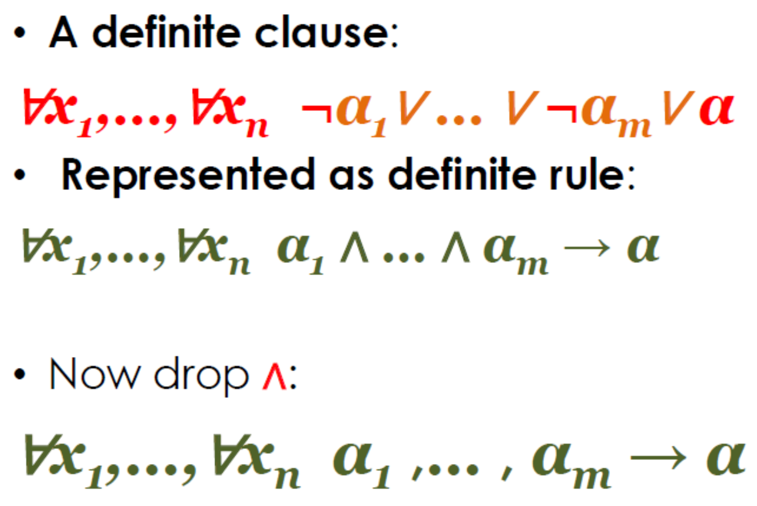
- and then dropping $\wedge$ as we did with propositional definite rules:

$$\forall x_1,\ldots,\forall x_n \ \alpha_1 ,\ldots, \alpha_m \rightarrow \alpha$$

46

Transforming predicate logic formulas into definite rules

1. Transform the predicate formula into PNF.
2. Transform the matrix of the formula into CNF.
3. If every quantifier is universal and every clause in the CNF form matrix contains exactly **one positive atom** (and all variables are in the scope of the universal quantifier in the prefix **definite clause**).
4. Represent each clause as a definite rule (REMEMBER COMMAS)



- **A definite clause:**

$$\forall x_1,\ldots,\forall x_n \ \neg\alpha_1 \vee \ldots \vee \neg\alpha_m \vee \alpha$$

- **Represented as definite rule:**

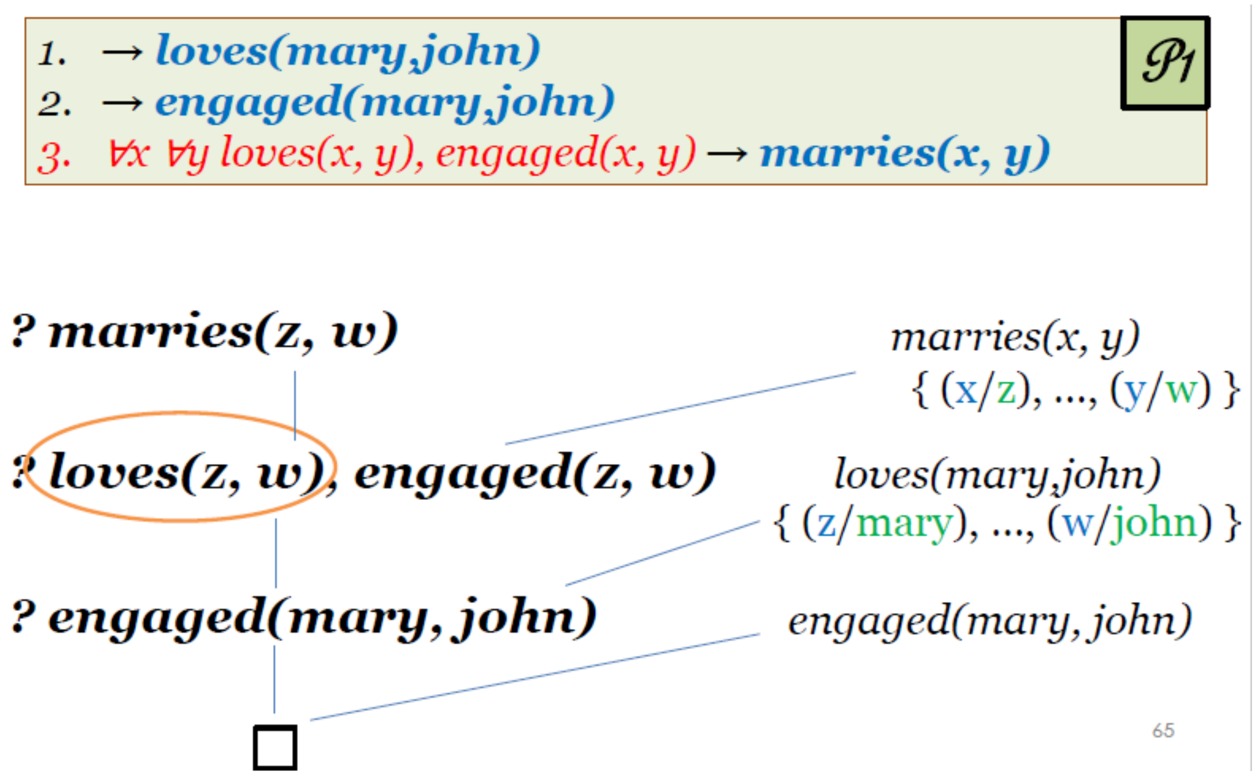
$$\forall x_1,\ldots,\forall x_n \ \alpha_1 \wedge \ldots \wedge \alpha_m \rightarrow \alpha$$

- Now drop $\wedge$:

$$\forall x_1,\ldots,\forall x_n \ \alpha_1 ,\ldots, \alpha_m \rightarrow \alpha$$

Querying a definite predicate logic program

It works the same way as a predicate one with the exception that querys have existential quantifiers (although we omit them) and we have **constants** and **variables**. Variables can be substituted by constants **but not vice versa**. All instances of a variable must be substituted.

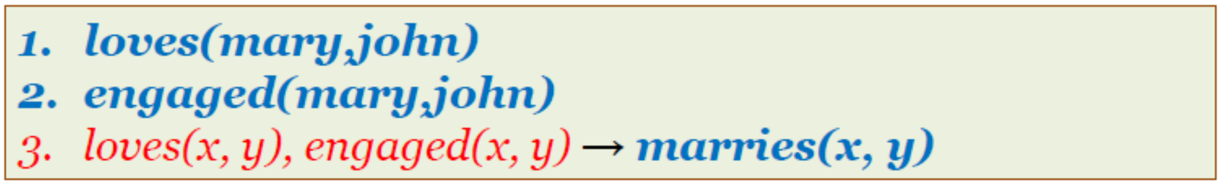


1. $\rightarrow$ *loves(mary,john)*
2. $\rightarrow$ *engaged(mary,john)*
3. $\forall x\ \forall y$ *loves(x, y), engaged(x, y)* $\rightarrow$ ***marries(x, y)***

Note that **unification** can also occur, which means that variables can be changed for other variables to match the correct state.

Notation conventions

From now on, our programs will look different:



1. ***loves(mary,john)***
2. ***engaged(mary,john)***
3. *loves(x, y), engaged(x, y)* $\rightarrow$ ***marries(x, y)***

Our first two are called **facts** and the third one is called a **rule**.

As a general rule we can use a small shortcut.

- P if A and B and ….and Q

A, B,…, Q → P

- P if A or B or ….or Q

A → P

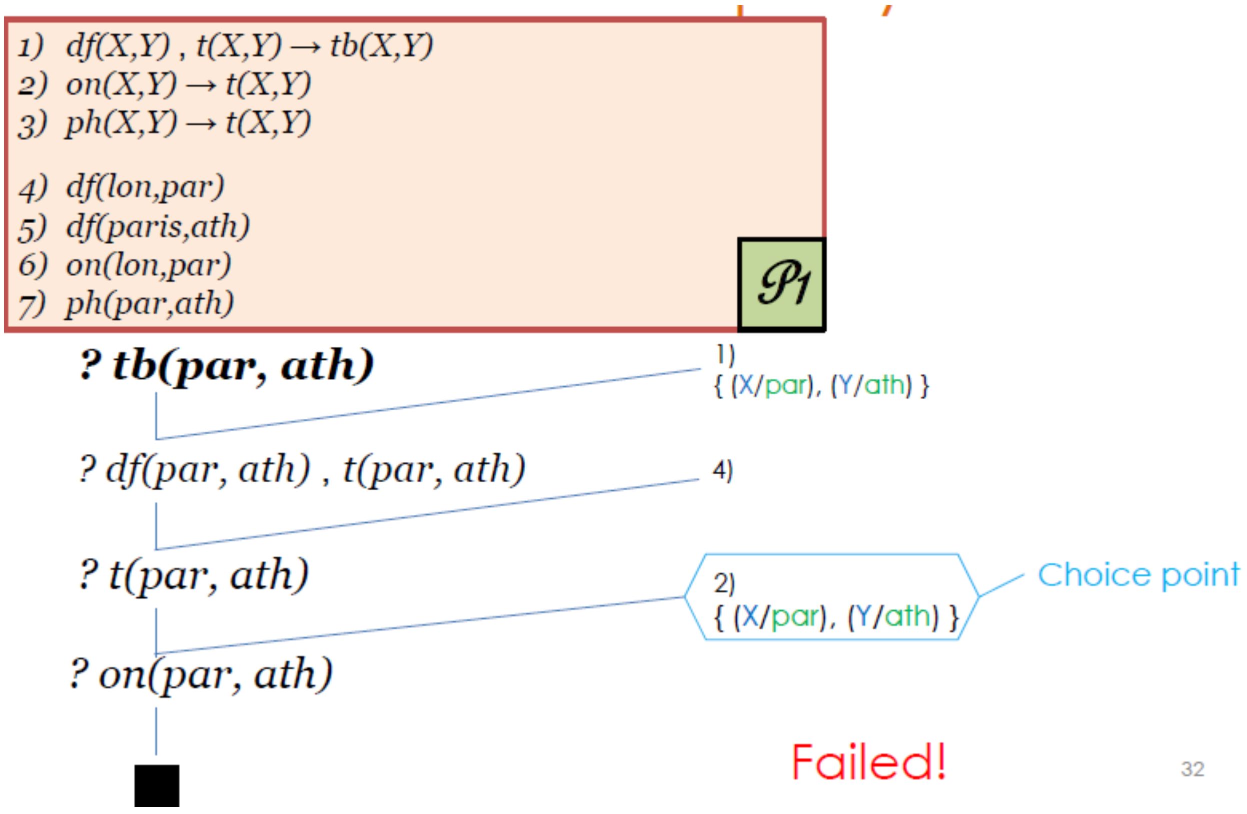
B → P

:

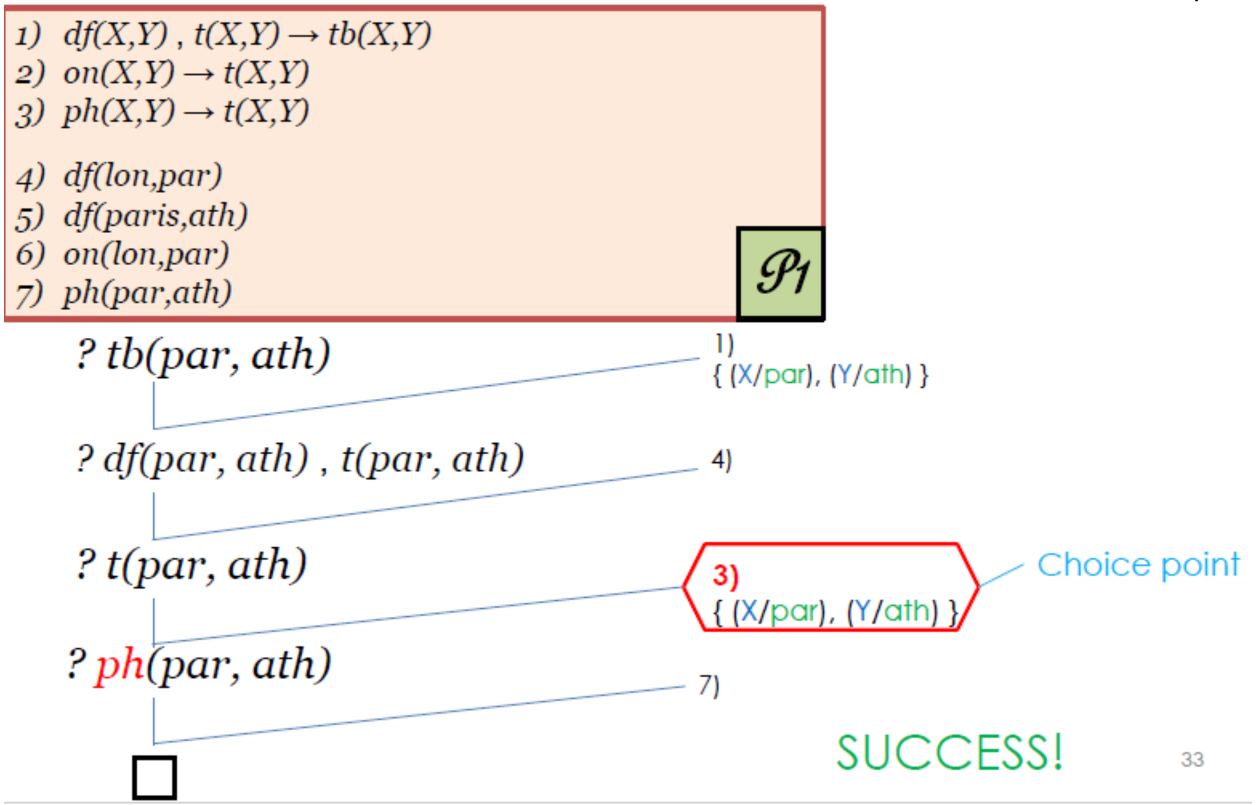
Q → P

Backtracking

At some points, our program can get stuck, at a choice point. In this case, the arrow-head we're looking for appears twice which means that as a silly little program we have to choose the first one and test if it works.


1) $df(X,Y), t(X,Y) \rightarrow tb(X,Y)$
2) $on(X,Y) \rightarrow t(X,Y)$
3) $ph(X,Y) \rightarrow t(X,Y)$
4) $df(lon,par)$
5) $df(paris,ath)$
6) $on(lon,par)$
7) $ph(par,ath)$

As seen, there I no on(par,ath) but if we quickly check, we can see that there will be a ph(par,ath) hence our program will backtrack and succeed.



1) $df(X,Y)\ ,\ t(X,Y) \rightarrow tb(X,Y)$
2) $on(X,Y) \rightarrow t(X,Y)$
3) $ph(X,Y) \rightarrow t(X,Y)$

4) $df(lon,par)$
5) $df(paris,ath)$
6) $on(lon,par)$
7) $ph(par,ath)$

$\mathcal{P}_1$

? $tb(par, ath)$ — 1) { (X/par), (Y/ath) }

? $df(par, ath)\ ,\ t(par, ath)$ — 4)

? $t(par, ath)$ — 3) { (X/par), (Y/ath) } — Choice point

? $ph(par, ath)$ — 7)

$\square$

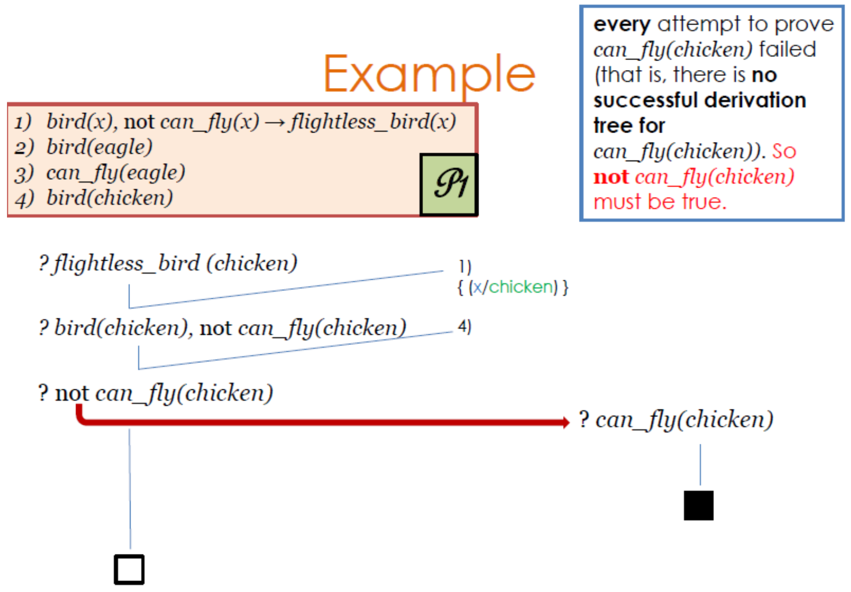
SUCCESS!

33

Negation as failure

Someone is innocent if they are not proved guilty. So if you fail to prove that he is guilty, he is innocent.

Sometimes in our program we night encounter a negated clause or part of a clause. It can be seen as not p (which is not the same as ¬P), and it relies on the basis that if we cannot prove p, then not p must be true.

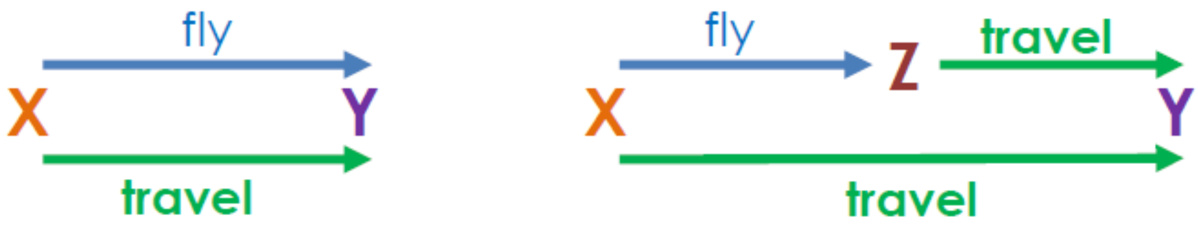

Example
1) bird(x), not can_fly(x) → flightless_bird(x)
2) bird(eagle)
3) can_fly(eagle)
4) bird(chicken)
𝒫1
every attempt to prove can_fly(chicken) failed (that is, there is no successful derivation tree for can_fly(chicken)). So not can_fly(chicken) must be true.
? flightless_bird (chicken)
1)
{ (x/chicken) }
? bird(chicken), not can_fly(chicken)
4)
? not can_fly(chicken)
? can_fly(chicken)

Recursion

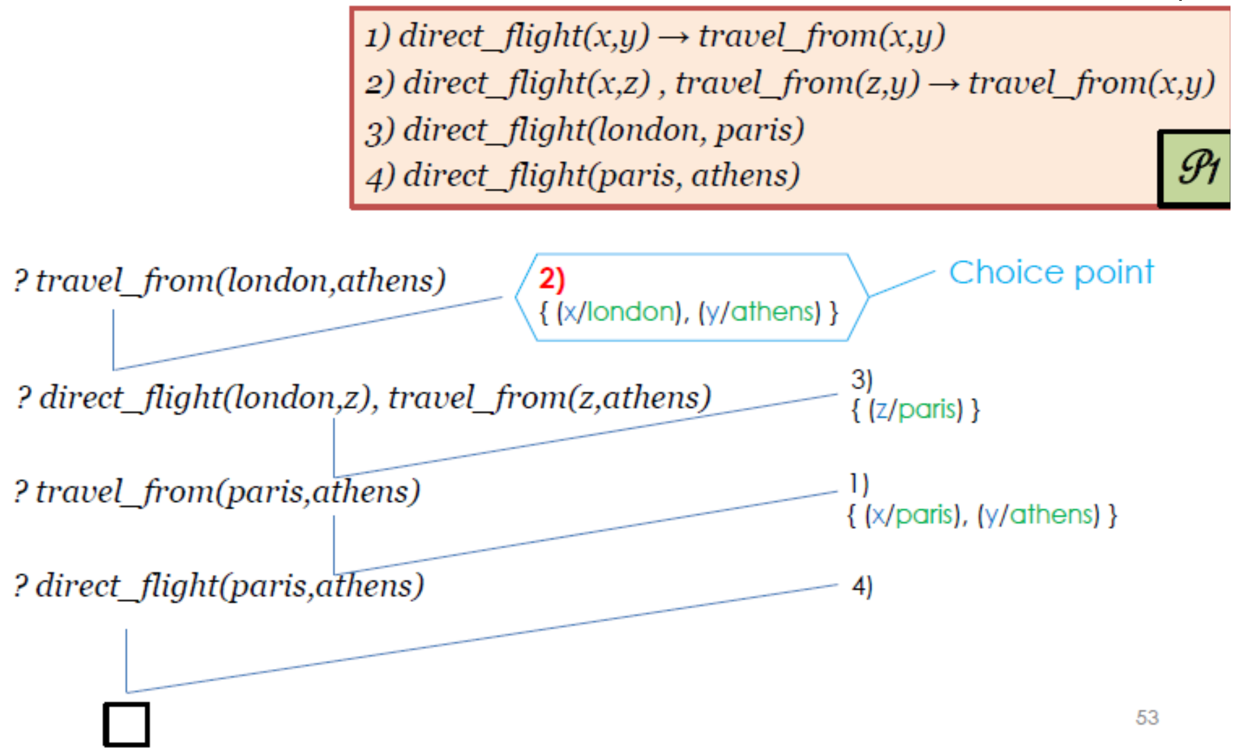
Recursion occurs when something is defined in terms of smaller versions of itself.

A recursive case consists of:

- A **base case**: a rule which cannot be expressed in terms of smaller versions of itself.
- A **recursive case:** the rule that can be expressed in smaller versions of itself.



$$direct\_flight(x,y) \rightarrow travel\_from(x,y)$$
$$direct\_flight(x,z)\,,\ travel\_from(z,y) \rightarrow travel\_from(x,y)$$



1) $direct\_flight(x,y) \rightarrow travel\_from(x,y)$
2) $direct\_flight(x,z)\ ,\ travel\_from(z,y) \rightarrow travel\_from(x,y)$
3) $direct\_flight(london, paris)$
4) $direct\_flight(paris, athens)$

$\mathcal{P}_1$
? travel_from(london,athens)
2) { (x/london), (y/athens) }
Choice point
? direct_flight(london,z), travel_from(z,athens)
3) { (z/paris) }
? travel_from(paris,athens)
1) { (x/paris), (y/athens) }
? direct_flight(paris,athens)
4)
$\square$
53

To do this example correctly you will have to backtrack (note that all derivation trees should be noted).