Algebraic and Discrete Methods in Biology

Propositional Resolution. Proving in Natural Style. First Order Predicate Logic

Nikolaj Popov and Tudor Jebelean

Research Institute for Symbolic Computation, Linz

popov@risc.uni-linz.ac.at

Outline

Propositional Resolution

Propositional resolution is a rule of inference. Applying iteratively the resolution rule in a suitable way allows us to decide whether a propositional formula is satisfiable.

It works only on expressions in clausal form. Before the rule can be applied, the premises and conclusions must be converted to this form. Fortunately, there is a simple procedure for making this conversion.

Propositional Resolution

Propositional resolution is a rule of inference. Applying iteratively the resolution rule in a suitable way allows us to decide whether a propositional formula is satisfiable.

It works only on expressions in clausal form. Before the rule can be applied, the premises and conclusions must be converted to this form. Fortunately, there is a simple procedure for making this conversion.

Propositional Resolution

Definition
A *literal* is an atom or the negation of an atom.
In the propositional calculus the atoms are the logical variables.

Definition
A clause expression is either a literal or a disjunction of literals.

If P and Q are logical variables, then the following are clause expressions:
P
$\neg P$
$\neg P \vee Q$

Propositional Resolution

Definition
A *literal* is an atom or the negation of an atom.
In the propositional calculus the atoms are the logical variables.

Definition
A clause expression is either a literal or a disjunction of literals.

If P and Q are logical variables, then the following are clause expressions:
P
$\neg P$
$\neg P \vee Q$

Propositional Resolution

Definition
A *literal* is an atom or the negation of an atom.
In the propositional calculus the atoms are the logical variables.

Definition
A clause expression is either a literal or a disjunction of literals.

If P and Q are logical variables, then the following are clause expressions:
P
$\neg P$
$\neg P \vee Q$

Propositional Resolution

Definition

A clause is the set of literals in a clause expression.

For example, the following sets are the clauses corresponding to the clause expressions above:

$\{P\}$

$\{\neg P\}$

$\{\neg P, Q\}$

Note that the empty set $\{\}$ is also a clause. It is equivalent to an empty disjunction and, therefore, is unsatisfiable.

Propositional Resolution

Definition
A clause is the set of literals in a clause expression.

For example, the following sets are the clauses corresponding to the clause expressions above:
$\{P\}$
$\{\neg P\}$
$\{\neg P, Q\}$

Note that the empty set $\{\}$ is also a clause. It is equivalent to an empty disjunction and, therefore, is unsatisfiable.

Propositional Resolution

Definition
A clause is the set of literals in a clause expression.

For example, the following sets are the clauses corresponding to the clause expressions above:
$\{P\}$
$\{\neg P\}$
$\{\neg P, Q\}$

Note that the empty set $\{\}$ is also a clause. It is equivalent to an empty disjunction and, therefore, is unsatisfiable.

Transformation into Normal Form
($\rightsquigarrow$ *CNF* $\rightsquigarrow$ *Clausal Form*)

Step 1 **Use the laws:**

$$\varphi \Leftrightarrow \psi \;=\; (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$$
$$\varphi \Rightarrow \psi \;=\; \neg\varphi \vee \psi$$

Step 2 **Use the laws:**

$$\neg(\neg\psi) \;=\; \psi$$
$$\neg(\varphi \vee \psi) \;=\; \neg\varphi \wedge \neg\psi$$
$$\neg(\varphi \wedge \psi) \;=\; \neg\varphi \vee \neg\psi$$

Step 3 **Use the law:**

$$(\varphi \wedge \psi) \vee \chi \;=\; (\varphi \vee \chi) \wedge (\psi \vee \chi)$$

Transformation into Normal Form
($\rightsquigarrow$ *CNF* $\rightsquigarrow$ *Clausal Form*)

Step 1 **Use the laws:**

$$\varphi \Leftrightarrow \psi \;=\; (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$$
$$\varphi \Rightarrow \psi \;=\; \neg\varphi \vee \psi$$

Step 2 **Use the laws:**

$$\neg(\neg\psi) \;=\; \psi$$
$$\neg(\varphi \vee \psi) \;=\; \neg\varphi \wedge \neg\psi$$
$$\neg(\varphi \wedge \psi) \;=\; \neg\varphi \vee \neg\psi$$

Step 3 **Use the law:**

$$(\varphi \wedge \psi) \vee \chi \;=\; (\varphi \vee \chi) \wedge (\psi \vee \chi)$$

Transformation into Normal Form
($\rightsquigarrow$ *CNF* $\rightsquigarrow$ *Clausal Form*)

Step 1 **Use the laws:**

$$\varphi \Leftrightarrow \psi \;=\; (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$$
$$\varphi \Rightarrow \psi \;=\; \neg\varphi \vee \psi$$

Step 2 **Use the laws:**

$$\neg(\neg\psi) \;=\; \psi$$
$$\neg(\varphi \vee \psi) \;=\; \neg\varphi \wedge \neg\psi$$
$$\neg(\varphi \wedge \psi) \;=\; \neg\varphi \vee \neg\psi$$

Step 3 **Use the law:**

$$(\varphi \wedge \psi) \vee \chi \;=\; (\varphi \vee \chi) \wedge (\psi \vee \chi)$$

Transformation into Normal Form
($\rightsquigarrow$ *CNF* $\rightsquigarrow$ *Clausal Form*)

Transform the formula from *CNF* into *Clausal Form*

Step 4 **Use the law:**

$$\varphi_1 \wedge \ldots \wedge \varphi_n = \{\varphi_1, \ldots, \varphi_n\}$$

Step 5 **Use the law:**

$$\{L_1 \vee \cdots \vee L_n, \ldots, L_m \vee \cdots \vee L_k\} = \{\{L_1, \ldots, L_n\}, \ldots, \{L_m, \ldots, L_k\}\}$$

Transformation into Normal Form
($\rightsquigarrow$ *CNF* $\rightsquigarrow$ *Clausal Form*)

Transform the formula from *CNF* into *Clausal Form*

***Step 4* Use the law:**

$$\varphi_1 \wedge \ldots \wedge \varphi_n = \{\varphi_1, \ldots, \varphi_n\}$$

***Step 5* Use the law:**

$$\{L_1 \vee \cdots \vee L_n, \ldots, L_m \vee \cdots \vee L_k\} = \{\{L_1, \ldots, L_n\}, \ldots, \{L_m, \ldots, L_k\}\}$$

Transformation into Normal Form ($\rightsquigarrow$ *CNF* $\rightsquigarrow$ *Clausal Form*)

Transform the formula from *CNF* into *Clausal Form*

***Step 4* Use the law:**

$$\varphi_1 \wedge \ldots \wedge \varphi_n = \{\varphi_1, \ldots, \varphi_n\}$$

***Step 5* Use the law:**

$$\{L_1 \vee \cdots \vee L_n, \ldots, L_m \vee \cdots \vee L_k\} = \{\{L_1, \ldots, L_n\}, \ldots, \{L_m, \ldots, L_k\}\}$$

Transformation into Normal Form
($\rightsquigarrow$ *CNF* $\rightsquigarrow$ *Clausal Form*)

Example

Obtain *clausal form* for the formula $\varphi_1 \triangleq (P \vee \neg Q) \Rightarrow R$

$\{\{\neg P, R\}, \{Q, R\}\}$

Example

Obtain *clausal form* for the formula $\varphi_2 \triangleq (P \wedge (Q \Rightarrow R)) \Rightarrow S$

$\{\{\neg P, S\}, \{Q, S\}, \{\neg R, S\}\}$

Transformation into Normal Form
($\rightsquigarrow$ *CNF* $\rightsquigarrow$ *Clausal Form*)

Example

Obtain *clausal form* for the formula $\varphi_1 \triangleq (P \vee \neg Q) \Rightarrow R$

$\{\{\neg P, R\}, \{Q, R\}\}$

Example

Obtain *clausal form* for the formula $\varphi_2 \triangleq (P \wedge (Q \Rightarrow R)) \Rightarrow S$

$\{\{\neg P, S\}, \{Q, S\}, \{\neg R, S\}\}$

Transformation into Normal Form
($\rightsquigarrow$ *CNF* $\rightsquigarrow$ *Clausal Form*)

Example

Obtain *clausal form* for the formula $\varphi_1 \triangleq (P \vee \neg Q) \Rightarrow R$

$\{\{\neg P, R\}, \{Q, R\}\}$

Example

Obtain *clausal form* for the formula $\varphi_2 \triangleq (P \wedge (Q \Rightarrow R)) \Rightarrow S$

$\{\{\neg P, S\}, \{Q, S\}, \{\neg R, S\}\}$

Transformation into Normal Form
($\rightsquigarrow$ *CNF* $\rightsquigarrow$ *Clausal Form*)

Example

Obtain *clausal form* for the formula $\varphi_1 \triangleq (P \vee \neg Q) \Rightarrow R$

$\{\{\neg P, R\}, \{Q, R\}\}$

Example

Obtain *clausal form* for the formula $\varphi_2 \triangleq (P \wedge (Q \Rightarrow R)) \Rightarrow S$

$\{\{\neg P, S\}, \{Q, S\}, \{\neg R, S\}\}$

Propositional Resolution

Resolution

The idea: Suppose we know that P is true or Q is true, and suppose we also know that P is false or R is true.

$\{P, Q\}$
$\{\neg P, R\}$

One clause contains P, and the other contains $\neg P$.
If P is false, then by the first clause Q must be true.
If P is true, then, by the second clause, R must be true.

Since P must be either true or false, then it must be the case that Q is true or R is true.
In other words, we can cancel the P literals.

$$\frac{\{P, Q\} \quad \{\neg P, R\}}{\{Q, R\}}$$

Propositional Resolution

Resolution

The idea: Suppose we know that P is true or Q is true, and suppose we also know that P is false or R is true.

$\{P, Q\}$
$\{\neg P, R\}$

One clause contains P, and the other contains $\neg P$.
If P is false, then by the first clause Q must be true.
If P is true, then, by the second clause, R must be true.

Since P must be either true or false, then it must be the case that Q is true or R is true.
In other words, we can cancel the P literals.

$$\frac{\{P, Q\} \quad \{\neg P, R\}}{\{Q, R\}}$$

Propositional Resolution

Resolution

The idea: Suppose we know that P is true or Q is true, and suppose we also know that P is false or R is true.

$\{P, Q\}$
$\{\neg P, R\}$

One clause contains P, and the other contains $\neg P$.
If P is false, then by the first clause Q must be true.
If P is true, then, by the second clause, R must be true.

Since P must be either true or false, then it must be the case that Q is true or R is true.
In other words, we can cancel the P literals.

$$\frac{\{P, Q\} \quad \{\neg P, R\}}{\{Q, R\}}$$

Propositional Resolution

Resolution

The idea: Suppose we know that P is true or Q is true, and suppose we also know that P is false or R is true.

$\{P, Q\}$
$\{\neg P, R\}$

One clause contains P, and the other contains $\neg P$.
If P is false, then by the first clause Q must be true.
If P is true, then, by the second clause, R must be true.

Since P must be either true or false, then it must be the case that Q is true or R is true.
In other words, we can cancel the P literals.

$$\frac{\{P, Q\} \quad \{\neg P, R\}}{\{Q, R\}}$$

Propositional Resolution

More generally, given a clause containing a literal Q and another clause containing the literal $\neg Q$, we can infer the clause consisting of all the literals of both clauses without the complementary pair.
This rule of inference is called propositional resolution.

$$\frac{\{P_1, \ldots, P_n, Q\} \quad \{\neg Q, R_1, \ldots, R_m\}}{\{P_1, \ldots, P_n, , R_1, \ldots, R_m\}}$$

If either of the clauses is a singleton set, we see that the number of literals in the result is less than the number of literals in the other clause.

From the clause $\{\neg P, Q\}$ and the singleton clause $\{P\}$, we can derive the singleton clause $\{Q\}$.

$$\frac{\{\neg P, Q\} \quad \{P\}}{\{Q\}}$$

Note the similarity between this deduction and that of Modus Ponens.

Propositional Resolution

More generally, given a clause containing a literal Q and another clause containing the literal $\neg Q$, we can infer the clause consisting of all the literals of both clauses without the complementary pair.
This rule of inference is called propositional resolution.

$$\frac{\{P_1, \ldots, P_n, Q\} \quad \{\neg Q, R_1, \ldots, R_m\}}{\{P_1, \ldots, P_n, , R_1, \ldots, R_m\}}$$

If either of the clauses is a singleton set, we see that the number of literals in the result is less than the number of literals in the other clause.

From the clause $\{\neg P, Q\}$ and the singleton clause $\{P\}$, we can derive the singleton clause $\{Q\}$.

$$\frac{\{\neg P, Q\} \quad \{P\}}{\{Q\}}$$

Note the similarity between this deduction and that of Modus Ponens.

Propositional Resolution

More generally, given a clause containing a literal Q and another clause containing the literal $\neg Q$, we can infer the clause consisting of all the literals of both clauses without the complementary pair.
This rule of inference is called propositional resolution.

$$\frac{\{P_1, \ldots, P_n, Q\} \quad \{\neg Q, R_1, \ldots, R_m\}}{\{P_1, \ldots, P_n, , R_1, \ldots, R_m\}}$$

If either of the clauses is a singleton set, we see that the number of literals in the result is less than the number of literals in the other clause.

From the clause $\{\neg P, Q\}$ and the singleton clause $\{P\}$, we can derive the singleton clause $\{Q\}$.

$$\frac{\{\neg P, Q\} \quad \{P\}}{\{Q\}}$$

Note the similarity between this deduction and that of Modus Ponens.

Propositional Resolution

Resolving two singleton clauses leads to the empty clause; i.e. the clause consisting of no literals at all.

$$\frac{\{P\} \quad \{\neg P\}}{\{\}}$$

The derivation of the empty clause means that the database contains a contradiction.

Note that in resolving two clauses, only one pair of literals may be resolved at a time, even though there are multiple resolvable pairs.

For example, the following is not a legal application of propositional resolution.

$$\frac{\{\neg P, Q\} \quad \{P, \neg Q\}}{\{\}}$$ **Wrong!**

Propositional Resolution

Resolving two singleton clauses leads to the empty clause; i.e. the clause consisting of no literals at all.

$$\frac{\{P\} \quad \{\neg P\}}{\{\}}$$

The derivation of the empty clause means that the database contains a contradiction.

Note that in resolving two clauses, only one pair of literals may be resolved at a time, even though there are multiple resolvable pairs.

For example, the following is not a legal application of propositional resolution.

$$\frac{\{\neg P, Q\} \quad \{P, \neg Q\}}{\{\}}$$ **Wrong!**

Propositional Resolution

Resolving two singleton clauses leads to the empty clause; i.e. the clause consisting of no literals at all.

$$\frac{\{P\} \quad \{\neg P\}}{\{\}}$$

The derivation of the empty clause means that the database contains a contradiction.

Note that in resolving two clauses, only one pair of literals may be resolved at a time, even though there are multiple resolvable pairs.

For example, the following is not a legal application of propositional resolution.

$$\frac{\{\neg P, Q\} \quad \{P, \neg Q\}}{\{\}}$$ **Wrong!**

Propositional Resolution

Resolving two singleton clauses leads to the empty clause; i.e. the clause consisting of no literals at all.

$$\frac{\{P\} \quad \{\neg P\}}{\{\}}$$

The derivation of the empty clause means that the database contains a contradiction.

Note that in resolving two clauses, only one pair of literals may be resolved at a time, even though there are multiple resolvable pairs.

For example, the following is not a legal application of propositional resolution.

$$\frac{\{\neg P, Q\} \quad \{P, \neg Q\}}{\{\}}$$ **Wrong!**

Propositional Resolution

Resolving two singleton clauses leads to the empty clause; i.e. the clause consisting of no literals at all.

$$\frac{\{P\} \quad \{\neg P\}}{\{\}}$$

The derivation of the empty clause means that the database contains a contradiction.

Note that in resolving two clauses, only one pair of literals may be resolved at a time, even though there are multiple resolvable pairs.

For example, the following is not a legal application of propositional resolution.

$$\frac{\{\neg P, Q\} \quad \{P, \neg Q\}}{\{\neg P, P\}}$$ **Correct!**

Propositional Resolution

Resolving two singleton clauses leads to the empty clause; i.e. the clause consisting of no literals at all.

$$\frac{\{P\} \quad \{\neg P\}}{\{\}}$$

The derivation of the empty clause means that the database contains a contradiction.

Note that in resolving two clauses, only one pair of literals may be resolved at a time, even though there are multiple resolvable pairs.

For example, the following is not a legal application of propositional resolution.

$$\frac{\{\neg P, Q\} \quad \{P, \neg Q\}}{\{Q, \neg Q\}}$$ **Also correct!**

Propositional Resolution

To determine whether a formula ψ is a logical consequence of a set of formulae $\{\varphi_1, \ldots \varphi_n\}$

that is: $\varphi_1 \wedge \ldots \wedge \varphi_n \models \psi$

we add the formula $\neg\psi$ to the set $\{\varphi_1, \ldots \varphi_n\}$,

$\{\varphi_1, \ldots \varphi_n, \neg\psi\}$

transform in clausal form and try to derive the empty clause.

Propositional Resolution

To determine whether a formula ψ is a logical consequence of a set of formulae $\{\varphi_1, \ldots \varphi_n\}$

that is: $\varphi_1 \wedge \ldots \wedge \varphi_n \models \psi$

we add the formula $\neg\psi$ to the set $\{\varphi_1, \ldots \varphi_n\}$,

$\{\varphi_1, \ldots \varphi_n, \neg\psi\}$

transform in clausal form and try to derive the empty clause.

Propositional Resolution

To determine whether a formula ψ is a logical consequence of a set of formulae $\{\varphi_1, \ldots \varphi_n\}$

that is: $\varphi_1 \wedge \ldots \wedge \varphi_n \models \psi$

we add the formula $\neg\psi$ to the set $\{\varphi_1, \ldots \varphi_n\}$,

$\{\varphi_1, \ldots \varphi_n, \neg\psi\}$

transform in clausal form and try to derive the empty clause.

Propositional Resolution

To determine whether a formula ψ is a logical consequence of a set of formulae $\{\varphi_1, \ldots \varphi_n\}$

that is: $\varphi_1 \wedge \ldots \wedge \varphi_n \models \psi$

we add the formula $\neg\psi$ to the set $\{\varphi_1, \ldots \varphi_n\}$,

$\{\varphi_1, \ldots \varphi_n, \neg\psi\}$

transform in clausal form and try to derive the empty clause.

Propositional Resolution

If the empty clause is derived, the set of formulae $\{\varphi_1, \ldots \varphi_n, \neg\psi\}$ is unsatisfiable (or contradictory), and hence ψ is a logical consequence of $\{\varphi_1, \ldots \varphi_n\}$,

that is: $\varphi_1 \wedge \ldots \wedge \varphi_n \models \psi$.

If, on the other hand, the empty clause cannot be derived, and the resolution rule cannot be applied to derive any more new clauses, ψ is not a logical consequence of $\{\varphi_1, \ldots \varphi_n\}$,

that is: $\varphi_1 \wedge \ldots \wedge \varphi_n \not\models \psi$.

Propositional Resolution

If the empty clause is derived, the set of formulae $\{\varphi_1, \ldots \varphi_n, \neg\psi\}$ is unsatisfiable (or contradictory), and hence ψ is a logical consequence of $\{\varphi_1, \ldots \varphi_n\}$,

that is: $\varphi_1 \wedge \ldots \wedge \varphi_n \models \psi$.

If, on the other hand, the empty clause cannot be derived, and the resolution rule cannot be applied to derive any more new clauses, ψ is not a logical consequence of $\{\varphi_1, \ldots \varphi_n\}$,

that is: $\varphi_1 \wedge \ldots \wedge \varphi_n \not\models \psi$.

Propositional Resolution

If the empty clause is derived, the set of formulae $\{\varphi_1, \ldots \varphi_n, \neg\psi\}$ is unsatisfiable (or contradictory), and hence ψ is a logical consequence of $\{\varphi_1, \ldots \varphi_n\}$,

that is: $\varphi_1 \wedge \ldots \wedge \varphi_n \models \psi$.

If, on the other hand, the empty clause cannot be derived, and the resolution rule cannot be applied to derive any more new clauses, ψ is not a logical consequence of $\{\varphi_1, \ldots \varphi_n\}$,

that is: $\varphi_1 \wedge \ldots \wedge \varphi_n \not\models \psi$.

Propositional Resolution

If the empty clause is derived, the set of formulae $\{\varphi_1, \ldots \varphi_n, \neg\psi\}$ is unsatisfiable (or contradictory), and hence ψ is a logical consequence of $\{\varphi_1, \ldots \varphi_n\}$,

that is: $\varphi_1 \wedge \ldots \wedge \varphi_n \models \psi$.

If, on the other hand, the empty clause cannot be derived, and the resolution rule cannot be applied to derive any more new clauses, ψ is not a logical consequence of $\{\varphi_1, \ldots \varphi_n\}$,

that is: $\varphi_1 \wedge \ldots \wedge \varphi_n \not\models \psi$.

Propositional Resolution

Example

Show that $(P \Rightarrow Q) \wedge (Q \Rightarrow R) \vDash (P \Rightarrow R)$.

Example

Show that $\vDash (A \wedge (A \Rightarrow B)) \Rightarrow B$.

Example

Show that $C \vDash C \wedge ((A \wedge (A \Rightarrow B)) \Rightarrow B)$.

Propositional Resolution

Example

Show that $(P \Rightarrow Q) \ \wedge \ (Q \Rightarrow R) \ \vDash \ (P \Rightarrow R)$.

Example

Show that $\vDash \ (A \wedge (A \Rightarrow B)) \Rightarrow B$.

Example

Show that $C \ \vDash \ C \ \wedge \ ((A \wedge (A \Rightarrow B)) \Rightarrow B)$.

Propositional Resolution

Example

Show that $(P \Rightarrow Q) \ \wedge \ (Q \Rightarrow R) \ \vDash \ (P \Rightarrow R)$.

Example

Show that $\vDash \ (A \wedge (A \Rightarrow B)) \Rightarrow B$.

Example

Show that $C \ \vDash \ C \ \wedge \ ((A \wedge (A \Rightarrow B)) \Rightarrow B)$.

Outline

Proving in Natural Style

Proving in natural style means proving in a style which is similar to the style which scientists naturally use when they do proofs.

It is important to understand very well the principles of proving in natural style, because this allows us to construct proofs in an intuitive way, and also helps us in understanding the proofs done by other people or by computer programs.

We have seen several equivalences, which allow transformations of one formula into another. Here we look situations of logical consequences, which allow us to infer new formulae from one or more given formulae.

Proving in Natural Style

Proving in natural style means proving in a style which is similar to the style which scientists naturally use when they do proofs.

It is important to understand very well the principles of proving in natural style, because this allows us to construct proofs in an intuitive way, and also helps us in understanding the proofs done by other people or by computer programs.

We have seen several equivalences, which allow transformations of one formula into another. Here we look situations of logical consequences, which allow us to infer new formulae from one or more given formulae.

Proving in Natural Style

Proving in natural style means proving in a style which is similar to the style which scientists naturally use when they do proofs.

It is important to understand very well the principles of proving in natural style, because this allows us to construct proofs in an intuitive way, and also helps us in understanding the proofs done by other people or by computer programs.

We have seen several equivalences, which allow transformations of one formula into another. Here we look situations of logical consequences, which allow us to infer new formulae from one or more given formulae.

Proving in Natural Style

$\varphi \wedge \psi$	$\models$	φ	True conjunct
$\varphi,\ \psi$	$\models$	$\varphi \wedge \psi$	Conjunction
φ	$\models$	$\varphi \vee \psi$	Disjunction
$\varphi,\ \varphi \Rightarrow \psi$	$\models$	ψ	Modus ponens
$\neg\psi,\ \varphi \Rightarrow \psi$	$\models$	$\neg\varphi$	Modus tollens
$\neg\varphi,\ \varphi \vee \psi$	$\models$	ψ	Resolution

Proving in Natural Style

Example

Using the natural style proving, show that
$A \vee B$ is a logical consequence of
$P \wedge Q,\ Q \Rightarrow C,\ B \Rightarrow \neg P,\ (C \wedge \neg B) \Rightarrow A$.

Example

Using the natural style proving, show that
$((A \vee B) \Rightarrow C) \models ((A \Rightarrow C) \wedge (B \Rightarrow C))$.

Proving in Natural Style

Example

Using the natural style proving, show that
$A \vee B$ is a logical consequence of
$P \wedge Q,\ Q \Rightarrow C,\ B \Rightarrow \neg P,\ (C \wedge \neg B) \Rightarrow A$.

Example

Using the natural style proving, show that
$((A \vee B) \Rightarrow C) \quad \vDash \quad ((A \Rightarrow C) \wedge (B \Rightarrow C))$.

Useful proof rules in propositional logic

Goal oriented rules

When the goal is $\neg\varphi$:
Assume φ and try to obtain a contradiction.

When the goal is $\varphi \wedge \psi$:
Prove φ and prove ψ.

When the goal is $\varphi \vee \psi$:
Assume $\neg\varphi$ and prove ψ.

When the goal is $\varphi \Rightarrow \psi$:
Assume φ and prove ψ (*"the deduction rule"*).

When the goal is $\varphi \Leftrightarrow \psi$:
Assume φ and prove ψ, and then assume ψ and prove φ.

Useful proof rules in propositional logic

Assumption oriented rules

When an assumption is $\neg(\neg\varphi)$:
Transform it into φ.

When an assumption is $\neg(\varphi \wedge \psi)$:
Transform it into $(\neg\varphi) \vee (\neg\psi)$.

When an assumption is $\neg(\varphi \vee \psi)$:
Transform it into two assumptions $\neg\varphi$ and $\neg\psi$.

When an assumption is $\neg(\varphi \Rightarrow \psi)$:
Transform it into two assumptions φ and $\neg\psi$.

When an assumption is $\neg(\varphi \Leftrightarrow \psi)$:
Transform it into two assumptions $\varphi \vee \psi$ and $(\neg\varphi) \vee (\neg\psi)$.

Useful proof rules in propositional logic

Assumption oriented rules

When an assumption is $\varphi \wedge \psi$:
Transform it into two assumptions φ and ψ.

When an assumption is $\varphi \vee \psi$:
Split the proof into two branches, on one assume φ and on the other one assume ψ (*"proof by cases"*).

When an assumption is $\varphi \Rightarrow \psi$:
If φ is already known (or it can be proven), obtain ψ (*"modus ponens"*).

When an assumption is $\varphi \Leftrightarrow \psi$:
Use any one of the implications, depending on the other formulae which are known.

Outline

First-Order Predicate Logic. Syntax

Formulae in first order predicate logic are composed of variables, constants, function symbols, predicate symbols, logical connectives, and quantifiers.

Terms are composed of variables, constants, and function symbols. Intuitively, a term denotes an object. A term consists of a variable (v), a constant (c), or a function symbol applied to one or several other terms ($f[t_1, \ldots, t_n]$).

Examples:

- x is a term because it consists of one variable.
- a is a term because it consists of one constant.
- $f[x, g[a]]$ is a term because it consists of the function symbol f applied to two terms.

First-Order Predicate Logic. Syntax

Formulae in first order predicate logic are composed of variables, constants, function symbols, predicate symbols, logical connectives, and quantifiers.

Terms are composed of variables, constants, and function symbols. Intuitively, a term denotes an object. A term consists of a variable (v), a constant (c), or a function symbol applied to one or several other terms ($f[t_1, \ldots, t_n]$).

Examples:

- x is a term because it consists of one variable.
- a is a term because it consists of one constant.
- $f[x, g[a]]$ is a term because it consists of the function symbol f applied to two terms.

First-Order Predicate Logic. Syntax

Formulae in first order predicate logic are composed of variables, constants, function symbols, predicate symbols, logical connectives, and quantifiers.

Terms are composed of variables, constants, and function symbols. Intuitively, a term denotes an object. A term consists of a variable (v), a constant (c), or a function symbol applied to one or several other terms ($f[t_1, \ldots, t_n]$).

Examples:

- x is a term because it consists of one variable.
- a is a term because it consists of one constant.
- $f[x, g[a]]$ is a term because it consists of the function symbol f applied to two terms.

First-Order Predicate Logic. Syntax

Formulae are composed of terms, predicate symbols, logical connectives, and quantifiers.
A formula can be *atomic* or *composite*.

An atomic formula consists of a predicate symbol applied to one or more terms:
$(P[t_1, \ldots, t_n])$.

A composite formula consists of several atomic formulae connected with logical connectives (like in propositional logic), or of a quantified formula of the form:

$$\forall v \varphi, \quad \exists v \varphi,$$

where v is a variable and φ is a formula.

First-Order Predicate Logic. Syntax

Formulae are composed of terms, predicate symbols, logical connectives, and quantifiers.
A formula can be *atomic* or *composite*.

An atomic formula consists of a predicate symbol applied to one or more terms:
$(P[t_1, \ldots, t_n])$.

A composite formula consists of several atomic formulae connected with logical connectives (like in propositional logic), or of a quantified formula of the form:

$$\forall v \varphi, \quad \exists v \varphi,$$

where v is a variable and φ is a formula.

First-Order Predicate Logic. Syntax

Formulae are composed of terms, predicate symbols, logical connectives, and quantifiers.
A formula can be *atomic* or *composite*.

An atomic formula consists of a predicate symbol applied to one or more terms:
$(P[t_1, \ldots, t_n])$.

A composite formula consists of several atomic formulae connected with logical connectives (like in propositional logic), or of a quantified formula of the form:

$$\forall v \varphi, \quad \exists v \varphi,$$

where v is a variable and φ is a formula.

First-Order Predicate Logic. Syntax

The quantifier $\forall x$ is called *universal* and a formula having the shape $\forall v \varphi$ is called *universally quantified*.

The quantifier $\exists x$ is called *existential* and a formula having the shape $\exists v \varphi$ is called *existentially quantified*.

The variable v is called *the quantified variable*, and φ is called *the scope* of the quantifier. Every occurrence of v in φ is called *bound* by the respective quantifier. When a variable occurs in a formula, but it is not bound by any quantifier, then we say that the respective variable is *free*.

First-Order Predicate Logic. Syntax

The quantifier $\forall x$ is called *universal* and a formula having the shape $\forall v \varphi$ is called *universally quantified*.

The quantifier $\exists x$ is called *existential* and a formula having the shape $\exists v \varphi$ is called *existentially quantified*.

The variable v is called *the quantified variable*, and φ is called *the scope* of the quantifier. Every occurrence of v in φ is called *bound* by the respective quantifier. When a variable occurs in a formula, but it is not bound by any quantifier, then we say that the respective variable is *free*.

First-Order Predicate Logic. Syntax

The quantifier $\forall x$ is called *universal* and a formula having the shape $\forall v \varphi$ is called *universally quantified*.

The quantifier $\exists x$ is called *existential* and a formula having the shape $\exists v \varphi$ is called *existentially quantified*.

The variable v is called *the quantified variable*, and φ is called *the scope* of the quantifier. Every occurrence of v in φ is called *bound* by the respective quantifier. When a variable occurs in a formula, but it is not bound by any quantifier, then we say that the respective variable is *free*.

First-Order Predicate Logic. Syntax

Example

The following formula expresses the statement “For every two points, there is one and only one line through the two points”

$$\forall x \forall y((P[x] \wedge P[y] \wedge \neg E[x,y]) \quad \Rightarrow \quad \exists z! \, (L[z] \wedge T[z,x,y]))$$

The formula uses the following predicate symbols:

$P[x]$: “ x is point”
$L[z]$: “ z is line”
$E[x,y]$: “x equals y”
$T[z,x,y]$: “z passes through x and y”

The quantifier $\exists x!$ (“there exists exactly one”) is only an abbreviation for a more complicated formula:

$$\exists x! P[x] \quad : \quad \exists x P[x] \wedge (\forall y P[y] \Rightarrow E[x,y])$$

First-Order Predicate Logic. Syntax

Example

The following formula expresses the statement “For every two points, there is one and only one line through the two points”

$$\forall x \forall y((P[x] \wedge P[y] \wedge \neg E[x,y]) \quad \Rightarrow \quad \exists z! \, (L[z] \wedge T[z,x,y]))$$

The formula uses the following predicate symbols:

$P[x]$: “ x is point”
$L[z]$: “ z is line”
$E[x,y]$: “x equals y”
$T[z,x,y]$: “z passes through x and y”

The quantifier $\exists x!$ (“there exists exactly one”) is only an abbreviation for a more complicated formula:

$$\exists x! P[x] \quad : \quad \exists x P[x] \wedge (\forall y P[y] \Rightarrow E[x,y])$$

First-Order Predicate Logic. Syntax

Example

The following formula expresses the statement “For every two points, there is one and only one line through the two points”

$$\forall x \forall y((P[x] \wedge P[y] \wedge \neg E[x,y]) \quad \Rightarrow \quad \exists z! \, (L[z] \wedge T[z,x,y]))$$

The formula uses the following predicate symbols:

$P[x]$: “ x is point”
$L[z]$: “ z is line”
$E[x,y]$: “x equals y”
$T[z,x,y]$: “z passes through x and y”

The quantifier $\exists x!$ (“there exists exactly one”) is only an abbreviation for a more complicated formula:

$$\exists x! P[x] \quad : \quad \exists x P[x] \wedge (\forall y P[y] \Rightarrow E[x,y])$$

First-Order Predicate Logic. Syntax

Example

The same formula can be written without using $\exists!$:

$$\forall x \forall y \Big(P[x] \wedge P[y] \wedge \neg E[x,y] \Rightarrow \exists z \Big((L[z] \wedge T[z,x,y]] \wedge \forall t \Big(\underbrace{(L[t] \wedge T[t,x}_{\text{scope of the}} \Big)}_{\text{scope of the } \exists z \text{ quantifi}} \Big)$$

variable z is bound in this

The formula becomes very long and may not fit into the slide.

First-Order Predicate Logic. Syntax

Example

The same formula can be written without using $\exists!$:

$$\forall x \forall y \Big(P[x] \wedge P[y] \wedge \neg E[x,y] \Rightarrow \exists z \Big(\underbrace{(L[z] \wedge T[z,x,y]] \wedge \forall t \Big(\underbrace{(L[t] \wedge T[t,x}_{\text{scope of the}}}_{\text{scope of the } \exists z \text{ quantifi}} \Big)$$

variable z is bound in this

The formula becomes very long and may not fit into the slide.

First-Order Predicate Logic. Syntax

Example

The same formula can be written without using $\exists!$:

$$\forall x \forall y \Big(P[x] \wedge P[y] \wedge \neg E[x,y] \Rightarrow \exists z \Big(\underbrace{(L[z] \wedge T[z,x,y]] \wedge \forall t \Big(\underbrace{(L[t] \wedge T[t,x}_{\text{scope of the}}}_{\text{scope of the } \exists z \text{ quantifi}} \\ \text{variable } z \text{ is bound in this}$$

The formula becomes very long and may not fit into the slide.

First-Order Predicate Logic. Semantics

An interpretation for a formula contains all the elements which are necessary for evaluating the truth value of the formula: a domain for the variables, a concrete constant from this domain for each constant of the formula, a concrete function for each function symbol, and a concrete predicate for each predicate symbol.

$$I \quad : \quad \begin{cases} \text{"domain"} \ldots & D \neq \emptyset \\ \text{constant symbol} \ldots & c_I \in D \\ \text{functional symbol} \ldots \\ \quad \text{(arity } n\text{)} & f_I : D^n \to D \\ \text{predicate symbol} \quad \ldots \\ \quad \text{(arity } m\text{)} & P_I : D^m \to \{\mathbb{T}, \mathbb{F}\} \end{cases}$$

First-Order Predicate Logic. Semantics

Like in propositional logic, for each interpretation, a formula φ has a certain *truth value*, denoted as

$$\langle \varphi \rangle_I,$$

and therefore to each formula φ it corresponds a function σ_φ, which is considered the semantics of this formula.

In order to evaluate quantified formulae, since the particular elements of the domain cannot occur in formulae, one uses the notion of *"truth value under the interpretation and a certain assignment to the free variables"*. Not only that the formulae have truth values, but the terms also have *object values* (under the interpretation and a certain assignment).

First-Order Predicate Logic. Semantics

Like in propositional logic, for each interpretation, a formula φ has a certain *truth value*, denoted as

$$\langle \varphi \rangle_I,$$

and therefore to each formula φ it corresponds a function σ_φ, which is considered the semantics of this formula.

In order to evaluate quantified formulae, since the particular elements of the domain cannot occur in formulae, one uses the notion of *"truth value under the interpretation and a certain assignment to the free variables"*. Not only that the formulae have truth values, but the terms also have *object values* (under the interpretation and a certain assignment).

First-Order Predicate Logic. Semantics

The notions of semantical equivalence and semantical logical consequence are defined like in propositional logic.

Two formulae φ and ψ are called *semantically equivalent* if they have the same semantic function:

$$\varphi \equiv \psi \text{ iff } \sigma_\varphi = \sigma_\psi,$$

that is, if they have the same truth value for every interpretation.

The formula ψ is a semantical logical consequence of the set of formulae Φ (notation: $\Phi \models \psi$) iff, for any interpretation I: if all formulae in the set Φ evaluate to true, then also the formula ψ evaluates to true.

First-Order Predicate Logic. Semantics

The notions of semantical equivalence and semantical logical consequence are defined like in propositional logic.

Two formulae φ and ψ are called *semantically equivalent* if they have the same semantic function:

$$\varphi \equiv \psi \text{ iff } \sigma_\varphi = \sigma_\psi,$$

that is, if they have the same truth value for every interpretation.

The formula ψ is a semantical logical consequence of the set of formulae Φ (notation: $\Phi \models \psi$) iff, for any interpretation I: if all formulae in the set Φ evaluate to true, then also the formula ψ evaluates to true.

First-Order Predicate Logic. Semantics

The notions of semantical equivalence and semantical logical consequence are defined like in propositional logic.

Two formulae φ and ψ are called *semantically equivalent* if they have the same semantic function:

$$\varphi \equiv \psi \text{ iff } \sigma_\varphi = \sigma_\psi,$$

that is, if they have the same truth value for every interpretation.

The formula ψ is a semantical logical consequence of the set of formulae Φ (notation: $\Phi \models \psi$) iff, for any interpretation *I*: if all formulae in the set Φ evaluate to true, then also the formula ψ evaluates to true.

First-Order Predicate Logic. Semantics

Using the definition one can check the following elementary equivalences:

$$\begin{aligned} \neg(\forall x \varphi) &\equiv \exists x (\neg \varphi) \\ \neg(\exists x \varphi) &\equiv \forall x (\neg \varphi) \end{aligned}$$

$$\begin{aligned} \forall x (\varphi \wedge \psi) &\equiv (\forall x \varphi) \wedge (\forall x \psi) \\ \exists x (\varphi \vee \psi) &\equiv (\exists x \varphi) \vee (\exists x \psi) \end{aligned}$$

$$\left.\begin{aligned} \forall x (\varphi \vee \psi) &\equiv (\forall x \varphi) \vee \psi \\ \exists x (\varphi \wedge \psi) &\equiv (\exists x \varphi) \wedge \psi \end{aligned}\right\} \text{ if the variable } x \text{ does not occur in } \psi$$

Note also that by changing the name of a quantified variable, the formula remains equivalent.

First-Order Predicate Logic. Semantics

Using the definition one can check the following elementary equivalences:

$$\begin{aligned} \neg(\forall x \varphi) &\equiv \exists x (\neg \varphi) \\ \neg(\exists x \varphi) &\equiv \forall x (\neg \varphi) \end{aligned}$$

$$\begin{aligned} \forall x (\varphi \wedge \psi) &\equiv (\forall x \varphi) \wedge (\forall x \psi) \\ \exists x (\varphi \vee \psi) &\equiv (\exists x \varphi) \vee (\exists x \psi) \end{aligned}$$

$$\left.\begin{aligned} \forall x (\varphi \vee \psi) &\equiv (\forall x \varphi) \vee \psi \\ \exists x (\varphi \wedge \psi) &\equiv (\exists x \varphi) \wedge \psi \end{aligned}\right\} \text{ if the variable } x \text{ does not occur in } \psi$$

Note also that by changing the name of a quantified variable, the formula remains equivalent.

First-Order Predicate Logic. Proving

Using the semantics of the formulae, we identify inference rules which can be used for proving in predicate logic, in addition to the rules used in propositional logic.

When the goal is $\forall x \varphi[x]$:
Consider x_0 arbitrary but fixed (a new constant) and prove $\varphi[x_0]$.

When the goal is $\exists x \varphi[x]$:
Find an appropriate term t and prove $\varphi[t]$.

When an assumption is $\forall x \varphi[x]$:
Find an appropriate term t and use $\varphi[t]$ as an additional assumption.

When an assumption is $\exists x \varphi[x]$:
Take an x_0 (a new constant) and assume that it satisfies $\varphi[x_0]$.

First-Order Predicate Logic. Proving

Using the semantics of the formulae, we identify inference rules which can be used for proving in predicate logic, in addition to the rules used in propositional logic.

When the goal is $\forall x \varphi[x]$:
Consider x_0 arbitrary but fixed (a new constant) and prove $\varphi[x_0]$.

When the goal is $\exists x \varphi[x]$:
Find an appropriate term t and prove $\varphi[t]$.

When an assumption is $\forall x \varphi[x]$:
Find an appropriate term t and use $\varphi[t]$ as an additional assumption.

When an assumption is $\exists x \varphi[x]$:
Take an x_0 (a new constant) and assume that it satisfies $\varphi[x_0]$.

First-Order Predicate Logic. Proving

Using the semantics of the formulae, we identify inference rules which can be used for proving in predicate logic, in addition to the rules used in propositional logic.

When the goal is $\forall x \varphi[x]$:
Consider x_0 arbitrary but fixed (a new constant) and prove $\varphi[x_0]$.

When the goal is $\exists x \varphi[x]$:
Find an appropriate term t and prove $\varphi[t]$.

When an assumption is $\forall x \varphi[x]$:
Find an appropriate term t and use $\varphi[t]$ as an additional assumption.

When an assumption is $\exists x \varphi[x]$:
Take an x_0 (a new constant) and assume that it satisfies $\varphi[x_0]$.

First-Order Predicate Logic. Proving

Using the semantics of the formulae, we identify inference rules which can be used for proving in predicate logic, in addition to the rules used in propositional logic.

When the goal is $\forall x \varphi[x]$:
Consider x_0 arbitrary but fixed (a new constant) and prove $\varphi[x_0]$.

When the goal is $\exists x \varphi[x]$:
Find an appropriate term t and prove $\varphi[t]$.

When an assumption is $\forall x \varphi[x]$:
Find an appropriate term t and use $\varphi[t]$ as an additional assumption.

When an assumption is $\exists x \varphi[x]$:
Take an x_0 (a new constant) and assume that it satisfies $\varphi[x_0]$.

First-Order Predicate Logic. Proving

Using the semantics of the formulae, we identify inference rules which can be used for proving in predicate logic, in addition to the rules used in propositional logic.

When the goal is $\forall x \varphi[x]$:
Consider x_0 arbitrary but fixed (a new constant) and prove $\varphi[x_0]$.

When the goal is $\exists x \varphi[x]$:
Find an appropriate term t and prove $\varphi[t]$.

When an assumption is $\forall x \varphi[x]$:
Find an appropriate term t and use $\varphi[t]$ as an additional assumption.

When an assumption is $\exists x \varphi[x]$:
Take an x_0 (a new constant) and assume that it satisfies $\varphi[x_0]$.

First-Order Predicate Logic. Proving

Example

We want to express the following statement:
"The protein *a* activates the protein *b* in the cytoplasm."

We denote by constants *a*, *b* the respective proteins and by constant *c* the cytoplasm. We also denote by the predicates "IsPresent" and "Activates" the respective atomic relations, and by the function "active" the activated version of a protein.

In order to encode it in predicate logic, the previous statement has to be expressed in more detail:
"If the protein *a* is present in *c* and the protein *b* is present in *c* and *a* activates *b*,
then the activated version of *b* is present in *c*."

(1)
$$(\text{IsPresent}[a, c] \wedge \text{IsPresent}[b, c] \wedge \text{Activates}[a, b]) \Rightarrow \text{IsPresent}[\text{active}[b], c]$$

First-Order Predicate Logic. Proving

Example

We want to express the following statement:
"The protein *a* activates the protein *b* in the cytoplasm."

We denote by constants a, b the respective proteins and by constant c the cytoplasm. We also denote by the predicates "IsPresent" and "Activates" the respective atomic relations, and by the function "active" the activated version of a protein.

In order to encode it in predicate logic, the previous statement has to be expressed in more detail:
"If the protein *a* is present in *c* and the protein *b* is present in *c* and *a* activates *b*,
then the activated version of *b* is present in *c*."

(1)
$$(\text{IsPresent}[a, c] \wedge \text{IsPresent}[b, c] \wedge \text{Activates}[a, b]) \Rightarrow \text{IsPresent}[\text{active}[b], c]$$

First-Order Predicate Logic. Proving

Example

We want to express the following statement:
"The protein *a* activates the protein *b* in the cytoplasm."

We denote by constants a, b the respective proteins and by constant c the cytoplasm. We also denote by the predicates "IsPresent" and "Activates" the respective atomic relations, and by the function "active" the activated version of a protein.

In order to encode it in predicate logic, the previous statement has to be expressed in more detail:
"If the protein *a* is present in *c* and the protein *b* is present in *c* and *a* activates *b*,
then the activated version of *b* is present in *c*."

(1)
$$(\text{IsPresent}[a, c] \wedge \text{IsPresent}[b, c] \wedge \text{Activates}[a, b]) \Rightarrow \text{IsPresent}[\text{active}[b], c]$$

First-Order Predicate Logic. Proving

Example

We want to express the following statement:
"The protein *a* activates the protein *b* in the cytoplasm."

We denote by constants a, b the respective proteins and by constant c the cytoplasm. We also denote by the predicates "IsPresent" and "Activates" the respective atomic relations, and by the function "active" the activated version of a protein.

In order to encode it in predicate logic, the previous statement has to be expressed in more detail:
"If the protein *a* is present in *c* and the protein *b* is present in *c* and *a* activates *b*,
then the activated version of *b* is present in *c*."

(1)
$$(\text{IsPresent}[a, c] \wedge \text{IsPresent}[b, c] \wedge \text{Activates}[a, b]) \Rightarrow \text{IsPresent}[\text{active}[b], c]$$

First-Order Predicate Logic. Proving

Lets us consider the statement:
“We say that a protein activates another protein, if and only if whenever the first protein is present in some container, then the later becomes active.”

Let the variables P, Q stand for proteins, and C stand for a container (like e. g. cytoplasm). We detail this statement as:
“For any P and for any Q, P activates Q iff for any C, if P is present in C, then Q becomes active.”

(2) $\forall_P \forall_Q (\text{Activates}[P, Q] \Leftrightarrow \forall_C (\text{IsPresent}[P, C] \Rightarrow \text{BecomesActive}[Q]))$

First-Order Predicate Logic. Proving

Lets us consider the statement:
“We say that a protein activates another protein, if and only if whenever the first protein is present in some container, then the later becomes active.”

Let the variables P, Q stand for proteins, and C stand for a container (like e. g. cytoplasm). We detail this statement as:
“For any P and for any Q, P activates Q iff for any C, if P is present in C, then Q becomes active.”

$$(2)\quad \forall_P \forall_Q (\mathrm{Activates}[P, Q] \Leftrightarrow \forall_C (\mathrm{IsPresent}[P, C] \Rightarrow \mathrm{BecomesActive}[Q]))$$

First-Order Predicate Logic. Proving

Lets us consider the statement:
“We say that a protein activates another protein, if and only if whenever the first protein is present in some container, then the later becomes active.”

Let the variables P, Q stand for proteins, and C stand for a container (like e. g. cytoplasm). We detail this statement as:
“For any P and for any Q, P activates Q iff for any C, if P is present in C, then Q becomes active.”

(2) $\forall_P \forall_Q (\text{Activates}[P, Q] \Leftrightarrow \forall_C (\text{IsPresent}[P, C] \Rightarrow \text{BecomesActive}[Q]))$

First-Order Predicate Logic. Proving

Let us now consider the statement:
“If a protein is present in some container and if it becomes active, then the active version of the protein is also present in that container.”

In more detail:
“For any C, for any P, if P is present in C and if P becomes active, then the active version of P is present in C.”

(3)
$\forall_C \forall_P ((\text{IsPresent}[P, C] \wedge \text{BecomesActive}[P]) \Rightarrow \text{IsPresent}[\text{active}[P], C])$

First-Order Predicate Logic. Proving

Let us now consider the statement:
“If a protein is present in some container and if it becomes active, then the active version of the protein is also present in that container.”

In more detail:
“For any C, for any P, if P is present in C and if P becomes active, then the active version of P is present in C.”

(3)
$$\forall_C \forall_P ((\text{IsPresent}[P, C] \wedge \text{BecomesActive}[P]) \Rightarrow \text{IsPresent}[\text{active}[P], C])$$

First-Order Predicate Logic. Proving

Let us now consider the statement:
“If a protein is present in some container and if it becomes active, then the active version of the protein is also present in that container.”

In more detail:
“For any C, for any P, if P is present in C and if P becomes active, then the active version of P is present in C.”

(3)
$$\forall_C \forall_P ((\text{IsPresent}[P, C] \wedge \text{BecomesActive}[P]) \Rightarrow \text{IsPresent}[\text{active}[P], C])$$

First-Order Predicate Logic. Proving

We can now formally prove that (1) is a logical consequence of (2) and (3), by using the natural inference rules given above for propositional logic and for predicate logic.

In order to prove (1),
$(\text{IsPresent}[a, c] \wedge \text{IsPresent}[b, c] \wedge \text{Activates}[a, b]) \Rightarrow \text{IsPresent}[\text{active}[b], c]$

we assume:
(4) $\text{IsPresent}[a, c] \wedge \text{IsPresent}[b, c] \wedge \text{Activates}[a, b]$

and we prove:
(5) $\text{IsPresent}[\text{active}[b], c]$.

First-Order Predicate Logic. Proving

We can now formally prove that (1) is a logical consequence of (2) and (3), by using the natural inference rules given above for propositional logic and for predicate logic.

In order to prove (1),
$(\text{IsPresent}[a, c] \wedge \text{IsPresent}[b, c] \wedge \text{Activates}[a, b]) \Rightarrow \text{IsPresent}[\text{active}[b], c]$

we assume:
(4) $\text{IsPresent}[a, c] \wedge \text{IsPresent}[b, c] \wedge \text{Activates}[a, b]$

and we prove:
(5) $\text{IsPresent}[\text{active}[b], c]$.

First-Order Predicate Logic. Proving

From (4)
$\text{IsPresent}[a, c] \wedge \text{IsPresent}[b, c] \wedge \text{Activates}[a, b]$

we obtain:
(6) $\text{IsPresent}[a, c]$

(7) $\text{IsPresent}[b, c]$

(8) $\text{Activates}[a, b]$

First-Order Predicate Logic. Proving

We instantiate formula (2)
$\forall_P \forall_Q (\text{Activates}[P, Q] \Leftrightarrow \forall_C (\text{IsPresent}[P, C] \Rightarrow \text{BecomesActive}[Q]))$

with $P : a$ and $Q : b$ (because Activates$[P, Q]$ matches (8))
$\text{Activates}[a, b]$

and we obtain:
(9) $\text{Activates}[a, b] \Longleftrightarrow \forall_C (\text{IsPresent}[a, C] \Longrightarrow \text{BecomesActive}[b])$

First-Order Predicate Logic. Proving

Using modus ponens, from (8) by (9) we obtain:
(10) $\forall_C(\text{IsPresent}[a, C] \Rightarrow \text{BecomesActive}[b]))$

We instantiate formula (10) with $C : c$
(because IsPresent$[a, C]$ matches (6)) and we obtain:
(11) $\text{IsPresent}[a, c] \Rightarrow \text{BecomesActive}[b]$

From (6)
$\text{IsPresent}[a, c]$

by (11) we obtain:
(12) $\text{BecomesActive}[b]$

First-Order Predicate Logic. Proving

Using modus ponens, from (8) by (9) we obtain:
(10) $\forall_C(\text{IsPresent}[a, C] \Rightarrow \text{BecomesActive}[b]))$

We instantiate formula (10) with $C : c$
(because IsPresent[a, C] matches (6)) and we obtain:
(11) $\text{IsPresent}[a, c] \Rightarrow \text{BecomesActive}[b]$

From (6)
$\text{IsPresent}[a, c]$

by (11) we obtain:
(12) $\text{BecomesActive}[b]$

First-Order Predicate Logic. Proving

Using modus ponens, from (8) by (9) we obtain:
(10) $\forall_C(\text{IsPresent}[a, C] \Rightarrow \text{BecomesActive}[b]))$

We instantiate formula (10) with $C : c$
(because IsPresent$[a, C]$ matches (6)) and we obtain:
(11) $\text{IsPresent}[a, c] \Rightarrow \text{BecomesActive}[b]$

From (6)
$\text{IsPresent}[a, c]$

by (11) we obtain:
(12) $\text{BecomesActive}[b]$

First-Order Predicate Logic. Proving

We instantiate formula (3)
$\forall_C \forall_P((\text{IsPresent}[P, C] \wedge \text{BecomesActive}[P]) \Rightarrow \text{IsPresent}[\text{active}[P], C])$

with $P : b$ and $C : c$ (because IsPresent$[b, c]$ matches (7))
(7) $\text{IsPresent}[b, c]$

and we obtain:
(13) $(\text{IsPresent}[b, c] \wedge \text{BecomesActive}[b]) \Rightarrow \text{IsPresent}[\text{active}[b], c]$

From (7) and (12)
(12) $\text{BecomesActive}[b]$

by (13) we obtain the formula (5)
$\text{IsPresent}[\text{active}[b], c]$

which had to be proven.

First-Order Predicate Logic. Proving

We instantiate formula (3)
$\forall_C \forall_P((\text{IsPresent}[P, C] \wedge \text{BecomesActive}[P]) \Rightarrow \text{IsPresent}[\text{active}[P], C])$

with $P : b$ and $C : c$ (because IsPresent$[b, c]$ matches (7))
(7) $\text{IsPresent}[b, c]$

and we obtain:
(13) $(\text{IsPresent}[b, c] \wedge \text{BecomesActive}[b]) \Rightarrow \text{IsPresent}[\text{active}[b], c]$

From (7) and (12)
(12) $\text{BecomesActive}[b]$

by (13) we obtain the formula (5)
$\text{IsPresent}[\text{active}[b], c]$

which had to be proven.

Conclusions

Logic plays a crucial role in reasoning and also automated reasoning

Having a subject formalized, one may apply automated reasoning techniques in order to derive its properties in an automated manner

Conclusions

Logic plays a crucial role in reasoning and also automated reasoning

Having a subject formalized, one may apply automated reasoning techniques in order to derive its properties in an automated manner