Intelligent Systems

Propositional Logic

Dieter Fensel and Dumitru Roman

Where are we?

#	Title
1	Introduction
2	**Propositional Logic**
3	Predicate Logic
4	Theorem Proving, Description Logics and Logic Programming
5	Search Methods
6	CommonKADS
7	Problem Solving Methods
8	Planning
9	Agents
10	Rule Learning
11	Inductive Logic Programming
12	Formal Concept Analysis
13	Neural Networks
14	Semantic Web and Exam Preparation

Outline

STI · INNSBRUCK

- Motivation
- Technical Solution
  - Syntax
  - Semantics
  - Semantic Reasoning Methods
- Illustration by a Larger Example
- Extensions
- Summary
- References

MOTIVATION

Logic and Deduction

STI · INNSBRUCK

- Logic is used to formalize deduction
- **Deduction** = derivation of true statements (called conclusions) from statements that are assumed to be true (called premises)
- Natural language is **not** precise, so the careless use of logic can lead to claims that false statements are true, or to claims that a statement is true, even tough its truth does not necessarily follow from the premises
  - => Logic provides a way to talk about truth and correctness in a rigorous way, so that we can prove things, rather than make intelligent guesses and just hope they are correct

Why Propositional Logic?

STI · INNSBRUCK

- Propositional logic is a good vehicle to introduce basic properties of logic; used to:
  - Associate natural language expressions with semantic representations
  - Evaluate the truth or falsity of semantic representations relative to a knowledge base
  - Compute inferences over semantic representations
- One of the simplest and most common logic
  - The core of (almost) all other logics

What is Propositional Logic?

- An unambiguous formal language, akin to a programming language
  - Syntax: Vocabulary for expressing concepts without ambiguity
  - Semantics: Connection to what we're reasoning about
    - *Interpretation* - what the syntax means
  - Reasoning: How to prove things
    - What steps are allowed

Syntax

TECHNICAL SOLUTION

Syntax

STI · INNSBRUCK

- Logical constants: true, false
- Propositional symbols: P, Q, S, ... (atomic sentences)
- Wrapping parentheses: (…)
- Sentences are combined by connectives:

∧ ...and	[conjunction]
∨ ...or	[disjunction]
⟶...implies	[implication / conditional]
⟷..is equivalent	[biconditional]
¬ ...not	[negation]

- Literal: atomic sentence or negated atomic sentence

- User defines a set of propositional symbols, like P and Q
- User defines the semantics of each propositional symbol:
  - P means "It is hot"
  - Q means "It is humid"
  - R means "It is raining"
- A sentence (well formed formula) is defined as follows:
  - A symbol is a sentence
  - If S is a sentence, then $\neg S$ is a sentence
  - If S is a sentence, then (S) is a sentence
  - If S and T are sentences, then $(S \vee T)$, $(S \wedge T)$, $(S \rightarrow T)$, and $(S \leftrightarrow T)$ are sentences
  - A sentence results from a finite number of applications of the above rules

Syntax – BNF Grammar

STI · INNSBRUCK

$$
\begin{aligned}
\textit{Sentence} &\rightarrow \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} &\rightarrow \texttt{True} \mid \texttt{False} \mid \texttt{P} \mid \texttt{Q} \mid \texttt{R} \mid \texttt{...} \\
\textit{ComplexSentence} &\rightarrow (\textit{Sentence}\ ) \\
&\quad \mid \textit{Sentence Connective Sentence} \\
&\quad \mid \neg\ \textit{Sentence} \\
\textit{Connective} &\rightarrow\ \wedge \mid \vee \mid \rightarrow \mid \leftrightarrow
\end{aligned}
$$

Ambiguities are resolved through precedence $\neg\ \wedge\ \vee \rightarrow \leftrightarrow$ or parentheses

e.g. $\neg P \vee Q \wedge R \Rightarrow S$ is equivalent to $(\neg P) \vee (Q \wedge R)) \Rightarrow S$

Syntax – Examples

STI · INNSBRUCK

- P means "It is hot."
- Q means "It is humid."
- R means "It is raining."
- $(P \wedge Q) \rightarrow R$
  "If it is hot and humid, then it is raining"
- $Q \rightarrow P$
  "If it is humid, then it is hot"

- $\neg p \wedge \neg q$
- $\neg(p \wedge q)$
- $(p \wedge q) \vee r$
- $p \wedge q \wedge r$
- $(((\neg p) \wedge q) \rightarrow r) \leftrightarrow ((\neg r) \vee p)$
- $(\neg (p \vee q) \rightarrow q) \rightarrow r$
- $((\neg p) \vee (\neg q)) \leftrightarrow (\neg r)$
- Etc.

Semantics

TECHNICAL SOLUTION

Semantics

- Interpretations
- Equivalence
- Substitution
- Models and Satisfiability
- Validity
- Logical Consequence (Entailment)
- Theory

Semantics – Some Informal Definitions

- The meaning or **semantics** of a sentence determines its **interpretation**
- Given the truth values of all symbols in a sentence, it can be "evaluated" to determine its **truth value** (True or False)
- A **model** for a KB is a "possible world" (assignment of truth values to propositional symbols) in which each sentence in the KB is True
- A **valid sentence** or **tautology** is a sentence that is True under all interpretations, no matter what the world is actually like or how the semantics are defined (example: "It's raining or it's not raining")
- An **inconsistent sentence** or **contradiction** is a sentence that is False under all interpretations (the world is never like what it describes, as in "It's raining and it's not raining")
- **P entails Q**, written P |= Q, means that whenever P is True, so is Q; in other words, all models of P are also models of Q

Interpretations

- In propositional logic, truth values are assigned to the atoms of a formula in order to evaluate the truth value of the formula
- An assignment is a function

  $$v : P \rightarrow \{T,F\}$$

  v assigns a truth value to any atom in a given formula (*P* is the set of all propositional letters, i.e. atoms)
  Suppose *F* denotes the set of all propositional formulas. We can extend an assignment v to a function

  $$v : F \rightarrow \{T,F\}$$

  which assigns the truth value *v(A)* to any formula *A* in *F*.
  v is called an interpretation.

Interpretations (cont')

STI · INNSBRUCK

- Evaluation of truth values of formulas (inductive definitions):

A	$v(A_1)$	$v(A_2)$	$v(A)$
$\neg A_1$	T		F
$\neg A_1$	F		T
$A_1 \vee A_2$	F	F	F
$A_1 \vee A_2$	otherwise		T
$A_1 \wedge A_2$	T	T	T
$A_1 \wedge A_2$	otherwise		F
$A_1 \rightarrow A_2$	T	F	F
$A_1 \rightarrow A_2$	otherwise		T
$A_1 \leftrightarrow A_2$	$v(A_1) = v(A_2)$		T
$A_1 \leftrightarrow A_2$	$v(A_1) \neq v(A_2)$		F

Interpretations (cont')

- Example:
  - Suppose v is an assignment for which
    $v(p) = F, \qquad v(q) = T.$
  - If $A = (\neg p \rightarrow q) \leftrightarrow (p \vee q)$, what is $v(A)$?

    Solution:

$$\begin{aligned}
v(A) &= v((\neg p \rightarrow q) \leftrightarrow (p \vee q)) \\
&= v(\neg p \rightarrow q) \leftrightarrow v(p \vee q) \\
&= (v(\neg p) \rightarrow v(q)) \leftrightarrow (v(p) \vee v(q)) \\
&= (\neg v(p) \rightarrow v(q)) \leftrightarrow (v(p) \vee v(q)) \\
&= (\neg F \rightarrow T) \leftrightarrow (F \vee T) \\
&= (T \rightarrow T) \leftrightarrow (F \vee T) \\
&= T \leftrightarrow T \\
&= T
\end{aligned}$$

Equivalence

- If A,B are formulas are such that

  $$v(A) = v(B)$$

  for **all** interpretations v, A is (logically) equivalent to B:

  $$A \equiv B$$

- Example: $\neg p \vee q \equiv p \rightarrow q$ since both formulas are true in all interpretations except when $v(p) = T$, $v(q) = F$ and are false for that particular interpretation
- Caution**:** $\equiv$ does **not** mean the same thing as $\leftrightarrow$:
  - $A \leftrightarrow B$ is a formula (syntax)
  - $A \equiv B$ is a relation between two formula (semantics)

  Theorem: *$A \equiv B$ if and only if $A \leftrightarrow B$ is true in every interpretation; i.e. $A \leftrightarrow B$ is a tautology.*

Substitution

STI · INNSBRUCK

- *A* is a subformula of *B* if it is a formula occurring within *B*; i.e. the formation tree for *A* is a subtree of the formation tree for *B*.
  - Example: The subformulas of $p \land (r \leftrightarrow p \lor \neg q)$ are $p \land (r \leftrightarrow p \lor \neg q)$, p, $r \leftrightarrow p \lor \neg q$, r , $p \lor \neg q$, $\neg q$, q
- Suppose *A* is a subformula of *B*, and *A'* is any formula. Then, we say that *B'* is a formula that results from substitution of *A'* for *A* in *B*, and we write it as

$$B' = B\{A \leftarrow A'\}$$

  if we obtain *B'* from *B* by replacing all occurrences of *A* in *B* with *A'*.
  - Example: Suppose

    $$B = (p \rightarrow q) \leftrightarrow (\neg q \rightarrow \neg p),\ A = p \rightarrow q,\ A' = \neg p \lor q$$

    then, $\quad B' = B\{A \leftarrow A'\} = (\neg p \lor q) \leftrightarrow (\neg q \rightarrow \neg p)$
- *__Theorem:__ Let A be a subformula of B, and let A' be a formula such that A ≡ A'. Then*

$$B \equiv B\{A \leftarrow A'\}$$

Equivalence and Substitution – Examples

STI · INNSBRUCK

- Examples of logically equivalent formulas

$$A \equiv \neg\neg A$$

$$A \vee B \equiv B \vee A \qquad A \wedge B \equiv B \wedge A$$

$$(A \vee B) \vee C \equiv A \vee (B \vee C) \qquad (A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$$

$$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$$

$$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$$

$$\neg(A \wedge B) \equiv \neg A \vee \neg B \qquad \neg(A \vee B) \equiv \neg A \wedge \neg B$$

$$A \wedge true \equiv A \qquad A \rightarrow false \equiv \neg A$$

- Example: Simplify $p \vee (\neg p \wedge q)$
  - Solution:

$$\begin{aligned} p \vee (\neg p \wedge q) &\equiv (p \vee \neg p) \wedge (p \vee q) \\ &\equiv \mathrm{T} \wedge (p \vee q) \\ &\equiv p \vee q \end{aligned}$$

Models and Satisfiability

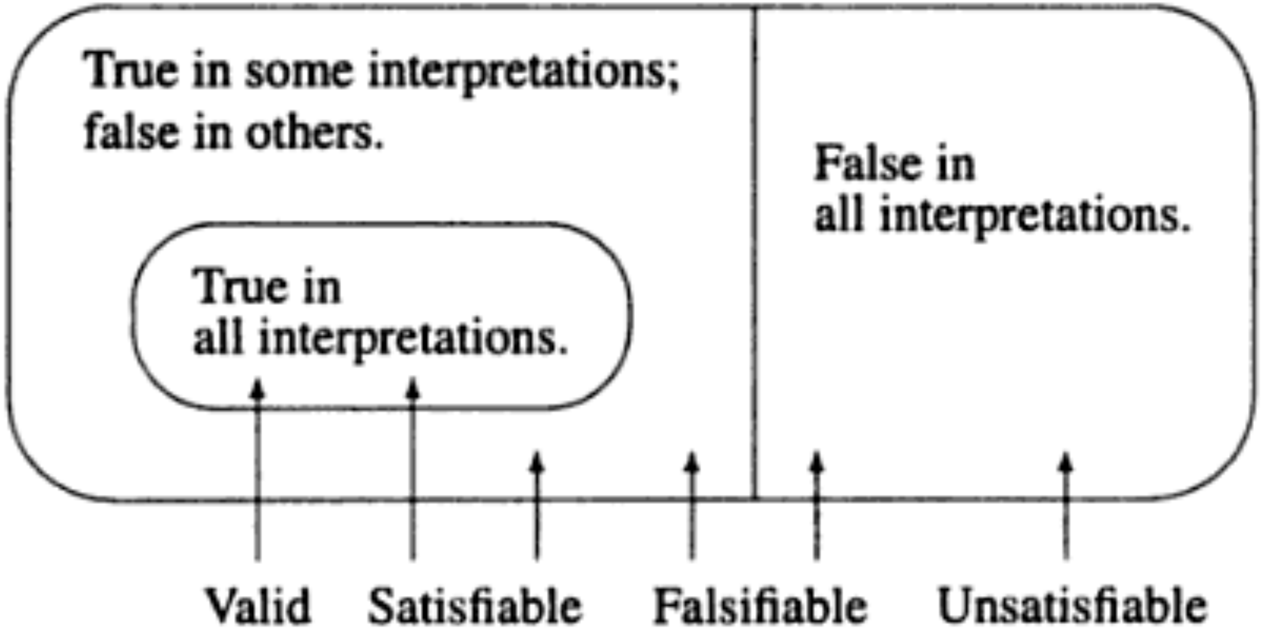
STI · INNSBRUCK

- A propositional **formula** A is satisfiable iff $v(A) = T$ in **some** interpretation v; such an interpretation is called a **model** for A.
  - A is unsatisfiable (or, contradictory) if it is false in every interpretation
- A **set of formulas** $U = \{A_1, A_2, \ldots, A_n\}$ is satisfiable iff there exists an interpretation v such that $v(A_1) = v(A_2) = \ldots = v(A_n) = T$; such an interpretation is called a **model** of U.
  - U is unsatisfiable if no such interpretation exists
- Relevant properties:
  - If U is satisfiable, then so is $U - \{A_i\}$ for any $i = 1, 2, \ldots, n$
  - If U is satisfiable and B is valid, then $U \cup \{B\}$ is also satisfiable
  - If U is unsatisfiable and B is **any** formula, $U \cup \{B\}$ is also unsatisfiable
  - If U is unsatisfiable and some A_i is valid, then $U - \{A_i\}$ is also unsatisfiable

Validity

STI · INNSBRUCK

- *A* is valid (or, a tautology), denoted |= *A*, iff *v*(*A*) = *T*, for **all** interpretations *v*
- *A* is not valid (or, falsifiable), denoted |≠ *A* if we can find **some** interpretation *v*, such that *v(A) = F*
- Relationship between validity, satisfiability, falsifiability, and unsatisfiability:



- Examples:
  - Valid (tautology): $(p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p)$
  - Not valid, but satisfiable: $q \rightarrow (q \rightarrow p)$
  - False (contradiction): $(p \wedge \neg p) \vee (q \wedge \neg q)$
- *Theorem:*

  (a) A is valid if and only if ¬A is unsatisfiable

  (b) A is satisfiable if and only if ¬A is falsifiable

Logical Consequence (i.e. Entailment)

STI · INNSBRUCK

- Let U be a set of formulas and A a formula. A is a **(logical) consequence** of U, if any interpretation v which is a model of U is also a model for A:

$$U \models A$$

- Example: $\{p \wedge r, \neg q \vee (p \wedge \neg p)\} \models (p \wedge \neg q) \rightarrow r$

  If some interpretation v is a model for the set
  $\{p \wedge r, \neg q \vee (p \wedge \neg p)\}$, it must satisfy

$$v(p) = v(r) = \mathrm{T}, \qquad v(q) = \mathrm{F}$$

  but in this interpretation, we also have

$$v((p \wedge \neg q) \rightarrow r) = \mathrm{T}$$

Theory

STI · INNSBRUCK

- A set of formulas T is a theory if it is closed under logical consequence. This means that, for every formula A, if $T \models A$, then A is in T
- Let U be a set of formulas. Then, the set of all consequences of U

  $$T(U) = \{A \mid U \models A\}$$

  is called the theory of U.
  The formulas in U are called the axioms for the theory $T(U)$.

Semantic Reasoning Methods

TECHNICAL SOLUTION

Semantic Reasoning Methods

- Several basic methods for determining whether a given set of premises propositionally entails a given conclusion
  - Truth Table Method
  - Deductive (Proof) Systems
  - Resolution

Truth Table Method

STI · INNSBRUCK

- One way of determining whether or not a set of premises logically entails a possible conclusion is to check the truth table for the logical constants of the language
- This is called the truth table method and can be formalized as follows:
  - Step 1: Starting with a complete truth table for the propositional constants, iterate through all the premises of the problem, for each premise eliminating any row that does not satisfy the premise
  - Step 2: Do the same for the conclusion
  - Step 3: Finally, compare the two tables; If every row that remains in the premise table, i.e. is not eliminated, also remains in the conclusion table, i.e. is not eliminated, then the premises logically entail the conclusion

Example

- Simple sentences:
  - *Amy loves Pat: lovesAmyPat*
  - *Amy loves Quincy: lovesAmyQuincy*
  - *It is Monday: ismonday*
- Premises:
  - *If Amy loves Pat, Amy loves Quincy: lovesAmyPat → lovesAmyQuincy*
  - *If it is Monday, Amy loves Pat or Quincy:*

    ismonday → lovesAmyPat ∨ lovesAmyQuincy
- Question:
  - *If it is Monday, does Amy love Quincy?*

    i.e. is *ismonday → lovesAmyQuincy* entailed by the premises?

Step 1: Truth table for the premises

lovesAmyPat	*lovesAmyQuincy*	*ismonday*	*lovesAmyPat* → *lovesAmyQuincy*	*ismonday* → *lovesAmyPat* ∨ *lovesAmyQuincy*
T	*T*	*T*	T	T
T	*T*	*F*	T	T
T	*F*	*T*	F	T
T	*F*	*F*	F	T
F	*T*	*T*	T	T
F	*T*	*F*	T	T
F	*F*	*T*	T	F
F	*F*	*F*	T	T

Step 1: Eliminate non-sat interpretations

lovesAmyPat	lovesAmyQuincy	ismonday	lovesAmyPat → lovesAmyQuincy	ismonday → lovesAmyPat ∨ lovesAmyQuincy
T	T	T	T	T
T	T	F	T	T
T	F	T	F	T
T	F	F	F	T
F	T	T	T	T
F	T	F	T	T
F	F	T	T	F
F	F	F	T	T

Step 2: Truth table for the conclusion

lovesAmyPat	*lovesAmyQuincy*	*ismonday*	$ismonday \rightarrow lovesAmyQuincy$
T	*T*	*T*	T
T	*T*	*F*	T
T	*F*	*T*	F
T	*F*	*F*	T
F	*T*	*T*	T
F	*T*	*F*	T
F	*F*	*T*	F
F	*F*	*F*	T

Step 2: Eliminate non-sat interpretations

lovesAmyPat	*lovesAmyQuincy*	*ismonday*	$ismonday \rightarrow lovesAmyQuincy$
T	*T*	*T*	T
T	*T*	*F*	T
T	*F*	*T*	F
T	*F*	*F*	T
F	*T*	*T*	T
F	*T*	*F*	T
F	*F*	*T*	F
F	*F*	*F*	T

Step 3: Comparing tables

- Finally, in order to make the determination of logical entailment, we compare the two rightmost tables and notice that every row remaining in the premise table also remains in the conclusion table.
  - In other words, the premises logically entail the conclusion.
- The truth table method has the merit that it is easy to understand
  - It is a direct implementation of the definition of logical entailment.
- In practice, it is awkward to manage two tables, especially since there are simpler approaches in which *only one table* needs to be manipulated
  - **Validity Checking**
  - **Unsatisfability Checking**

Validity checking

STI · INNSBRUCK

- Approach: To determine whether a set of sentences

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

  logically entails a sentence φ, form the sentence

  $$(\varphi_1 \wedge \ldots \wedge \varphi_n \rightarrow \varphi)$$

  and check that it is valid.
- To see how this method works, consider the previous example and write the tentative conclusion as shown below.

  (lovesAmyPat → lovesAmyQuincy) ∧ (ismonday → lovesAmyPat ∨ lovesAmyQuincy) → (ismonday → lovesAmyQuincy)
- Then, form a truth table for our language with an added column for this sentence and check its satisfaction under each of the possible interpretations for our logical constants

Unsatisfability Checking

STI · INNSBRUCK

- It is almost exactly the same as the validity checking method, except that it works negatively instead of positively.
- To determine whether a finite set of sentences $\{\varphi_1,\ldots,\varphi_n\}$ logically entails a sentence φ, we form the sentence

$$(\varphi_1 \wedge \ldots \wedge \varphi_n \wedge \neg\varphi)$$

  and check that it is *unsatisfiable*.

- Both the validity checking method and the satisfiability checking method require about the same amount of work as the truth table method, but they have the merit of manipulating **only one table**

Example – A truth table

p	q	r	$p \rightarrow q$	$p \rightarrow r$	$p \rightarrow r \vee q$	$(p \rightarrow q) \wedge (p \rightarrow r) \rightarrow (p \rightarrow r \vee q)$	$p \rightarrow r \wedge q$	$(p \rightarrow q) \wedge (p \rightarrow r) \rightarrow (p \rightarrow r \wedge q)$
T	*T*	*T*	T	T	T	T	T	T
T	*T*	*F*	T	F	T	T	F	T
T	*F*	*T*	F	T	T	T	F	T
T	*F*	*F*	F	F	F	T	F	T
F	*T*	*T*	T	T	T	T	T	T
F	*T*	*F*	T	T	T	T	T	T
F	*F*	*T*	T	T	T	T	T	T
F	*F*	*F*	T	T	T	T	T	T

Deductive (proof) systems

STI · INNSBRUCK

- Semantic methods for checking logical entailment have the merit of being conceptually simple; they directly manipulate interpretations of sentences
- Unfortunately, the number of interpretations of a language grows **exponentially** with the number of logical constants.
  - When the number of logical constants in a propositional language is large, the number of interpretations may be impossible to manipulate.
- Deductive (proof) systems provide an alternative way of checking and communicating logical entailment that addresses this problem
  - In many cases, it is possible to create a “proof” of a conclusion from a set of premises that is much smaller than the truth table for the language;
  - Moreover, it is often possible to find such proofs with less work than is necessary to check the entire truth table

Schemata

STI · INNSBRUCK

- An important component in the treatment of proofs is the notion of a schema
- A *schema* is an expression satisfying the grammatical rules of our language except for the occurrence of *metavariables* in place of various subparts of the expression.
  - For example, the following expression is a pattern with metavariables φ and ψ.

$$\varphi \rightarrow (\psi \rightarrow \varphi)$$

- An **instance** of a sentence schema is the expression obtained by substituting expressions for the metavariables.
  - For example, the following is an instance of the preceding schema.

$$p \rightarrow (q \rightarrow p)$$

Rules of Inference

STI · INNSBRUCK

- The basis for proof systems is the use of correct rules of inference that can be applied directly to sentences to derive conclusions that are guaranteed to be correct under all interpretations
  - Since the interpretations are not enumerated, time and space can often be saved
- A rule of inference is a pattern of reasoning consisting of:
  - One set of sentence schemata, called *premises*, and
  - A second set of sentence schemata, called *conclusions*
- A rule of inference is sound if and only if, for every instance, the premises logically entail the conclusions

E.g. Modus Ponens (MP)

STI · INNSBRUCK

$$\frac{\varphi \rightarrow \psi \qquad \varphi}{\psi}$$

$$\frac{p \rightarrow (q \rightarrow r) \qquad p}{q \rightarrow r}$$

$$\frac{\mathit{raining} \rightarrow \mathit{wet} \qquad \mathit{raining}}{\mathit{wet}}$$

$$\frac{(p \rightarrow q) \rightarrow r \qquad p \rightarrow q}{r}$$

$$\frac{\mathit{wet} \rightarrow \mathit{slippery} \qquad \mathit{wet}}{\mathit{slippery}}$$

- I.e. we can substitute for the metavariables complex sentences
- Note that, by stringing together applications of rules of inference, it is possible to derive conclusions that cannot be derived in a single step. This idea of stringing together rule applications leads to the notion of a proof.

Axiom schemata

STI · INNSBRUCK

- The *implication introduction schema* (**II**), together with Modus Ponens, allows us to infer implications

  $\varphi \rightarrow (\psi \rightarrow \varphi)$

- The *implication distribution schema* (**ID**) allows us to distribute one implication over another

  $(\varphi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \chi))$

- The *contradiction realization schemata* (**CR**) permit us to infer a sentence if the negation of that sentence implies some sentence and its negation

  $(\psi \rightarrow \neg\varphi) \rightarrow ((\psi \rightarrow \varphi) \rightarrow \neg\psi)$

  $(\neg\psi \rightarrow \neg\varphi) \rightarrow ((\neg\psi \rightarrow \varphi) \rightarrow \psi)$

Axiom schemata (cont')

- The *equivalence schemata* (**EQ**) captures the meaning of the $\leftrightarrow$ operator

  $(\varphi \leftrightarrow \psi) \rightarrow (\varphi \rightarrow \psi)$

  $(\varphi \leftrightarrow \psi) \rightarrow (\psi \rightarrow \varphi)$

  $(\varphi \rightarrow \psi) \rightarrow ((\psi \rightarrow \varphi) \rightarrow (\psi \leftrightarrow \varphi))$

- The meaning of the other operators in propositional logic is captured in the following axiom schemata

  $(\varphi \leftarrow \psi) \leftrightarrow (\psi \rightarrow \varphi)$

  $(\varphi \vee \psi) \leftrightarrow (\neg\varphi \rightarrow \psi)$

  $(\varphi \wedge \psi) \leftrightarrow \neg(\neg\varphi \vee \neg\psi)$

- The above axiom schemata are jointly called the *standard axiom schemata* for Propositional Logic
  - They all are **valid**

Proofs

- A *proof* of a conclusion from a set of premises is a sequence of sentences terminating in the conclusion in which each item is either
  - (1) a premise,
  - (2) an instance of an axiom schema, or
  - (3) the result of applying a rule of inference to earlier items in sequence
- Example:

1. $p \rightarrow q$	Premise
2. $q \rightarrow r$	Premise
3. $(q \rightarrow r) \rightarrow (p \rightarrow (q \rightarrow r))$	II
4. $p \rightarrow (q \rightarrow r)$	MP : 3,2
5. $(p \rightarrow (q \rightarrow r)) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r))$	ID
6. $(p \rightarrow q) \rightarrow (p \rightarrow r)$	MP : 5,4
7. $p \rightarrow r$	MP : 6,1

Proofs (cont')

STI · INNSBRUCK

- If there exists a proof of a sentence φ from a set Δ of premises and the standard axiom schemata using Modus Ponens, then φ is said to be *provable* from Δ, written as

$$\Delta \mid\!- \varphi$$

- There is a close connection between provability and logical entailment:

A set of sentences Δ logically entails a sentence φ
if and only if φ is provable from Δ

- *Soundness Theorem:*

If φ is provable from Δ, then Δ logically entails φ.

- *Completeness Theorem:*

If Δ logically entails φ, then φ is provable from Δ.

Proofs (cont')

- The concept of provability is important because it suggests how we can automate the determination of logical entailment
  - Starting from a set of premises Δ, we enumerate conclusions from this set
  - If a sentence φ appears, then it is provable from Δ and is, therefore, a logical consequence
  - If the negation of φ appears, then $\neg\varphi$ is a logical consequence of Δ and φ is not logically entailed (unless Δ is inconsistent)
  - Note that it is possible that neither φ nor $\neg\varphi$ will appear

Resolution

- **Propositional resolution** is an extremely powerful rule of inference for Propositional Logic
- Using propositional resolution (without axiom schemata or other rules of inference), it is possible to build a theorem prover that is sound and complete for all of Propositional Logic
- The search space using propositional resolution is much smaller than for standard propositional logic
- Propositional resolution works only on expressions in *clausal form*
  - Before the rule can be applied, the premises and conclusions must be converted to this form

Clausal Forms

STI · INNSBRUCK

- A clause is a set of literals which is assumed (implicitly) to be a disjunction of those literals
  - Example: $\neg p \vee q \vee \neg r \quad \leadsto \quad \{\neg p, q, \neg r\}$
- Unit clause: clause with only one literal; e.g. $\{\neg q\}$
- Clausal form of a formula: Implicit conjunction of clauses
- Example: $p \wedge (\neg p \vee q \vee \neg r) \wedge (\neg q \vee q \vee \neg r) \wedge (\neg q \vee p)$

$$\Updownarrow$$

$$\{\{p\}, \{\neg p, q, \neg r\}, \{\neg q, q, \neg r\}, \{\neg q, p\}\}$$

  Abbreviated notation: $\{p, \bar{p}q\bar{r}, \bar{q}q\bar{r}, \bar{q}p\}$
- Notation:
  - l-literal, l^c-complement of l
  - C-clause (a set of literals)
  - S-a clausal form (a set of clauses)

(1) If *l* appears in some clause of *S*, but l^c does not appear in any clause, then, if we delete all clauses in *S* containing *l*, the new clausal form *S'* is satisfiable if and only if *S* is satisfiable

Example: Satisfiability of $S = \{pq\bar{r}, p\bar{q}, \bar{p}q\}$ is equivalent to satisfiability of $S' = \{p\bar{q}, \bar{p}q\}$

(2) Suppose *C* = {*l*} is a unit clause and we obtain *S'* from *S* by deleting *C* and l^c from all clauses that contain it; then, *S* is satisfiable if and only if *S'* is satisfiable

Example: $S = \{p, \bar{p}q\bar{r}, \bar{q}q\bar{r}, q\bar{p}\}$ is satisfiable iff $S' = \{q\bar{r}, \bar{q}q\bar{r}, q\}$ is satisfiable

STI · INNSBRUCK

(3) If S contains two clauses C and C', such that C is a subset of C', we can delete C' without affecting the (un) satisfiability of S

Example: $S = \{p, \bar{p}q\bar{r}, \bar{q}q\bar{r}, q\bar{p}\}$ is satisfiable iff
$S' = \{p, \bar{q}q\bar{r}, q\bar{p}\}$ is satisfiable

(4) If a clause C in S contains a pair of complementary literals l, l^c, then C can be deleted from S without affecting its (un)satisfiability

Example: $S = \{p, \bar{p}q\bar{r}, \bar{q}q\bar{r}, q\bar{p}\}$ is satisfiable iff
$S' = \{p, \bar{p}q\bar{r}, q\bar{p}\}$ is such

Converting to clausal form

STI · INNSBRUCK

Theorem: *Every propositional formula can be transformed into an equivalent formula in CNF*

1. Implications:

$\varphi_1 \rightarrow \varphi_2$	$\rightarrow$	$\neg\varphi_1 \vee \varphi_2$
$\varphi_1 \leftarrow \varphi_2$	$\rightarrow$	$\varphi_1 \vee \neg\varphi_2$
$\varphi_1 \leftrightarrow \varphi_2$	$\rightarrow$	$(\neg\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \neg\varphi_2)$

2. Negations:

$\neg\neg\varphi$	$\rightarrow$	φ
$\neg(\varphi_1 \wedge \varphi_2)$	$\rightarrow$	$\neg\varphi_1 \vee \neg\varphi_2$
$\neg(\varphi_1 \vee \varphi_2)$	$\rightarrow$	$\neg\varphi_1 \wedge \neg\varphi_2$

3. Distribution:

$\varphi_1 \vee (\varphi_2 \wedge \varphi_3)$	$\rightarrow$	$(\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3)$
$(\varphi_1 \wedge \varphi_2) \vee \varphi_3$	$\rightarrow$	$(\varphi_1 \vee \varphi_3) \wedge (\varphi_2 \vee \varphi_3)$
$(\varphi_1 \vee \varphi_2) \vee \varphi_3$	$\rightarrow$	$\varphi_1 \vee (\varphi_2 \vee \varphi_3)$
$(\varphi_1 \wedge \varphi_2) \wedge \varphi_3$	$\rightarrow$	$\varphi_1 \wedge (\varphi_2 \wedge \varphi_3)$

4. Operators:

$\varphi_1 \vee \ldots \vee \varphi_n$	$\rightarrow$	$\{\varphi_1,\ldots,\varphi_n\}$
$\varphi_1 \wedge \ldots \wedge \varphi_n$	$\rightarrow$	$\{\varphi_1\}\ldots\{\varphi_n\}$

Example

- Transform the formula

$$(p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p)$$

into an equivalent formula in CNF

Solution:

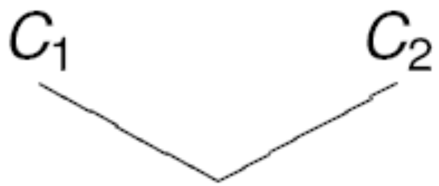
$$
\begin{aligned}
& (p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p) \\
\equiv\ & (\neg p \vee q) \rightarrow (\neg\neg q \vee \neg p) \\
\equiv\ & \neg(\neg p \vee q) \vee (\neg\neg q \vee \neg p) \\
\equiv\ & (\neg\neg p \wedge \neg q) \vee (\neg\neg q \vee \neg p) \\
\equiv\ & (p \wedge \neg q) \vee (q \vee \neg p) \\
\equiv\ & (p \vee q \vee \neg p) \wedge (\neg q \vee q \vee \neg p)
\end{aligned}
$$

- Suppose C_1, C_2 are clauses such that l in C_1, l^c in C_2. The clauses C_1 and C_2 are said to be **clashing clauses** and they clash on the complementary literals l, l^c

  C, the resolvent of C_1, C_2 is the clause

  $$Res(C_1, C_2) = (C_1 - \{l\}) \cup (C_2 - \{l^c\})$$

  C_1 and C_2 are called the **parent clauses** of C.



$$C = (C_1 - \{l\}) \cup (C_2 - \{l^c\})$$

- Example:

  The clauses

$$C_1 = \bar{p}q\bar{r}, \qquad C_2 = \bar{q}p$$

clash on $p, \bar{p}$.

$$Res(C_1, C_2) = q\bar{r} \cup \bar{q} = q\bar{r}\bar{q}$$

C_1, C_2 also clash on $q, \bar{q}$, so, another way to find a resolvent for these two clauses is

$$Res(C_1, C_2) = \bar{p}\bar{r} \cup p = \bar{p}\bar{r}p$$

Resolution (cont')

- *Theorem: Resolvent C is satisfiable if and only if the parent clauses C_1,C_2 are simultaneously satisfiable*
- Resolution Algorithm:

  Input: S – a set of clauses

  Output: "S is satisfiable" or "S is not satisfiable"

  1. Set $S_0 := S$
  2. Suppose S_i has already been constructed
  3. To construct S_{i+1}, choose a pair of clashing literals and clauses C_1,C_2 in S (if there are any) and derive

     $C := Res(C_1,C_2)$

     $S_{i+1} := S_i \cup \{C\}$

  1. If C is the empty clause, output "S is not satisfiable"; if $S_{i+1} = S_i$, output "S is satisfiable"
  2. Otherwise, set $i := i + 1$ and go back to Step 2

Resolution (cont')

- Example: Show that $(p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p)$ is a valid formula
  Solution: We will show that

  $$\neg[(p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p)]$$

  is not satisfiable.
  (1) Transform the formula into CNF:

  $$\begin{aligned} \neg[(p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p)] &\equiv (p \rightarrow q) \wedge \neg(\neg q \rightarrow \neg p) \\ &\equiv (\neg p \vee q) \wedge \neg q \wedge \neg\neg p \\ &\equiv (\neg p \vee q) \wedge \neg q \wedge p \end{aligned}$$

  (2) Show, using resolution, that $S = \{\bar{p}q, \bar{q}, p\}$
  1. $S_0 = \{\bar{p}q, \bar{q}, p\}$
  2. $C_1 = \bar{p}q, \quad C_2 = \bar{q}, \quad C = \bar{p}, \qquad S_1 = \{\bar{p}q, \bar{q}, p, \bar{p}\}$
  3. $C_1 = p, \quad C_2 = \bar{p}, \quad C$ is the empty clause
- A derivation of the empty clause from S is called a refutation of S

Resolution (cont')

- *Theorem: If the set of a clauses labeling the leaves of a resolution tree is satisfiable, then the clause at the root is satisfiable*
- *Theorem (Soundness): If the empty clause is derived from a set of clauses, then the set of clauses is unsatisfiable*
- *Theorem (Completeness) If a set of clauses is unsatisfiable, then the empty clause can be derived from it using resolution algorithm*

ILLUSTRATION BY A LARGER EXAMPLE

Problem Example

STI · INNSBRUCK

- For each of these sets of premises, what relevant conclusion or conclusions can be drawn? Explain the rules of inference used to obtain each conclusion from the premises.

  (a) “If I eat spicy foods, then I have strange dreams.” “I have strange dreams if there is thunder while I sleep.” “I did not have strange dreams.”

  (b) “I am dreaming or hallucinating.” “I am not dreaming.” “If I am hallucinating, I see elephants running down the road.”

  (c) “If I work, it is either sunny or partly sunny.” “I worked last Monday or I worked last Friday.” “It was not sunny on Tuesday.” “It was not partly sunny on Friday.”

Solution (a)

(a) “If I eat spicy foods, then I have strange dreams.” “I have strange dreams if there is thunder while I sleep.” “I did not have strange dreams.”

- The relevant conclusions are: “I did not eat spicy food” and “There is no thunder while I sleep”.
- Let the primitive statements be:
  - *s, ‘I eat spicy foods’*
  - *d, ‘I have strange dreams’*
  - *t, ‘There is thunder while I sleep’*
- Then the premises are translated as: $s \rightarrow d$, $t \rightarrow d$, and $\neg d$.
- And the conclusions: $\neg s$, $\neg t$.
- Steps Reason

1. $s \rightarrow d$ *premise*
2. $\neg d$ *premise*
3. $\neg s$ *Modus Tollens to Steps 1 and 2*
4. $t \rightarrow d$ *premise*
5. $\neg t$ *Modus Tollens to Steps 4 and 2.*

Solution (b)

(b) “I am dreaming or hallucinating.” “I am not dreaming.” “If I am hallucinating, I see elephants running down the road.”

- The relevant conclusion is: “I see elephants running down the road.”.
- Let the primitive statements be:
  - *d, ‘I am dreaming’*
  - *h, ‘I am hallucinating’*
  - *e, ‘I see elephants running down the road’*
- Then the premises are translated as: $d \vee h$, $\neg d$, and $h \rightarrow e$.
- And the conclusion: e.
- Steps Reason

Steps	Reason
1. $d \vee h$	*premise*
2. $\neg d$	*premise*
3. h	*rule of disjunctive syllogism to Steps 1 and 2*
4. $h \rightarrow e$	*premise*
5. e	*Modus Ponens to Steps 4 and 3*

Solution (c)

(c) “If I work, it is either sunny or partly sunny.” “I worked last Monday or I worked last Friday.” “It was not sunny on Tuesday.” “It was not partly sunny on Friday.”

- There is no single relevant conclusion in this problem, its main difficulty is to to represent the premises so that one is able infer anything at all. One possible relevant conclusion is: “It was sunny or partly sunny last Monday or it was sunny last Friday.”.
- Let the primitive statements be:
  - *wm, ‘I worked last Monday’*
  - *wf , ‘I worked last Friday’*
  - *sm, ‘It was sunny last Monday’*
  - *st, ‘It was sunny last Tuesday’*
  - *sf , ‘It was sunny last Friday’*
  - *pm, ‘It was partly sunny last Monday’*
  - *pf , ‘It was partly sunny last Friday’*
- Then the premises are translated as: $wm \vee wf$, $wm \rightarrow (sm \vee pm)$, $wf \rightarrow (sf \vee pf)$, $\neg st$, and $\neg pf$.
- And the conclusion: $sf \vee sm \vee pm$.

Solution (c) – Method 1

STI · INNSBRUCK

Steps	Reason
1. $wf \rightarrow (sf \vee pf)$	*premise*
2. $\neg wf \vee sf \vee pf$	*expression for implication*
3. $\neg pf \rightarrow (\neg wf \vee sf)$	*expression for implication*
4. $\neg pf$	*premise*
5. $\neg wf \vee sf$	*modus ponens to Steps 3 and 4*
6. $wf \rightarrow sf$	*expression for implication*
7. $wm \vee wf$	*premise*
8. $\neg wm \rightarrow wf$	*expression for implication*
9. $\neg wm \rightarrow sf$	*rule of syllogism to Steps 8 and 6*
10. $wm \vee sf$	*expression for implication*
11. $\neg sf \rightarrow wm$	*expression for implication*
12. $wm \rightarrow (sm \vee pm)$	*premise*
13. $\neg sf \rightarrow (sm \vee pm)$	*rule of syllogism to Steps 11 and 12*
14. $sf \vee sm \vee pm$	*expression for implication.*

Solution (c) – Method 2 (Use the rule of resolution)

STI · INNSBRUCK

Steps	Reason
1. $wf \rightarrow (sf \vee pf)$	*premise*
2. $\neg wf \vee sf \vee pf$	*expression for implication*
3. $\neg pf$	*premise*
4. $\neg wf \vee sf$	*rule of resolution to Steps 2 and 3*
5. $wm \vee wf$	*premise*
6. $wm \vee sf$	*rule of resolution to Steps 4 and 5*
7. $wm \rightarrow (sm \vee pm)$	*premise*
8. $\neg wm \vee sm \vee pm$	*expression for implication*
9. $sf \vee sm \vee pm$	*rule of resolution to Steps 7 and 8*

EXTENSIONS

Extensions

- The most immediate way to develop a more complex logical calculus is to introduce rules that are sensitive to more fine-grained details of the sentences being used
  - When the atomic sentences of propositional logic are broken up into terms, variables, predicates, and quantifiers, they yield **first-order logic**, which keeps all the rules of propositional logic and adds some new ones
- Modal logic also offers a variety of inferences that cannot be captured in propositional calculus
  - For example, from "Necessarily *p*" we may infer that *p*. From *p* we may infer "It is possible that *p*".
- Many-valued logics are those allowing sentences to have values other than *true* and *false*
  - For example, *neither* and *both* are standard "extra values“; "continuum logic" allows each sentence to have any of an infinite number of "degrees of truth" between *true* and *false*
  - These logics often require calculational devices quite distinct from propositional calculus

SUMMARY

Summary

STI · INNSBRUCK

- Propositional logic is one of the simplest and most common logic and is the core of (almost) all other logics
  - Propositional logic commits only to the existence of facts that may or may not be the case in the world being represented
  - Propositional logic quickly becomes impractical, even for very small worlds
- This lecture focused on three core aspects of the propositional logic:
  - Syntax: Vocabulary for expressing concepts without ambiguity
  - Semantics: Connection to what we're reasoning about
    - *Interpretation* - what the syntax means
  - Reasoning: How to prove things
    - What steps are allowed

REFERENCES

References

- Mathematical Logic for Computer Science (2nd edition) by Mordechai Ben-Ari
  - http://www.springer.com/computer/foundations/book/978-1-85233-319-5
- Propositional Logic at The Internet Encyclopedia of Philosophy
  - http://www.iep.utm.edu/p/prop-log.htm
- Propositional Logic on Wikipedia
  - http://en.wikipedia.org/wiki/Propositional_calculus
- Many online resources
  - http://www.google.com/search?hl=en&q=propositional+logic
  - http://www.google.com/search?hl=en&q=propositional+calculus

Next Lecture

#	Title
1	Introduction
2	Propositional Logic
3	**Predicate Logic**
4	Theorem Proving, Description Logics and Logic Programming
5	Search Methods
6	CommonKADS
7	Problem Solving Methods
8	Planning
9	Agents
10	Rule Learning
11	Inductive Logic Programming
12	Formal Concept Analysis
13	Neural Networks
14	Semantic Web and Exam Preparation

Questions?