CS 1571 Introduction to AI
Lecture 12

Logical reasoning systems

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Logical inference in FOL

Logical inference problem:

- Given a knowledge base KB (a set of sentences) and a sentence α , does the KB semantically entail α?

$$KB \models \alpha \quad ?$$

In other words: In all interpretations in which sentences in the KB are true, is also α true?

Logical inference problem in the first-order logic is undecidable !!!. No procedure that can decide the entailment for all possible input sentences in finite number of steps.

Resolution inference rule

- **Recall:** Resolution inference rule is sound and complete (refutation-complete) for the propositional logic and CNF

$$\frac{A \vee B, \quad \neg A \vee C}{B \vee C}$$

- **Generalized resolution rule is sound and complete** (refutation-complete) for the first-order logic and CNF (w/o equalities)

$\sigma = UNIFY\ (\phi_i, \neg \psi_j) \neq fail$

$$\frac{\phi_1 \vee \phi_2 \ldots \vee \phi_k, \quad \psi_1 \vee \psi_2 \vee \ldots \psi_n}{SUBST(\sigma, \phi_1 \vee \ldots \vee \phi_{i-1} \vee \phi_{i+1} \ldots \vee \phi_k \vee \psi_1 \vee \ldots \vee \psi_{j-1} \vee \psi_{j+1} \ldots \psi_n)}$$

Example: $$\frac{P(x) \vee Q(x), \quad \neg Q(John) \vee S(y)}{P(John) \vee S(y)}$$

The rule can be also written in the **implicative form** (book)

Inference with resolution rule

- **Proof by refutation:**
  - Prove that $KB, \neg \alpha$ is **unsatisfiable**
  - resolution is **refutation-complete**

- **Main procedure (steps):**
  1. Convert $KB, \neg \alpha$ to CNF with ground terms and universal variables only
  2. Apply repeatedly the resolution rule while keeping track and consistency of substitutions
  3. Stop when empty set (contradiction) is derived or no more new resolvents (conclusions) follow

Dealing with equality

- Resolution works for first-order logic without equalities
- To incorporate equalities we need an additional inference rule
- **Demodulation rule**

$$\sigma = UNIFY\ (z_i, t_1) \neq fail \qquad z_i \text{ a term in } \phi_i$$

$$\frac{\phi_1 \vee \phi_2 \ldots \vee \phi_k, \quad t_1 = t_2}{SUBST(\{SUBST(\sigma, t_1) / SUBST(\sigma, t_2)\}, \phi_1 \vee \phi_2 \vee \ldots \vee \phi_k)}$$

- **Example:**

$$\frac{P(f(a)), f(x) = x}{P(a)}$$

- **Paramodulation rule: more powerful inference rule**
- **Resolution+paramodulation**
  – give refutation-complete proof theory for FOL

Sentences in Horn normal form

- **Horn normal form (HNF) in the propositional logic**
  – a special type of clause with at most one positive literal

$$(A \vee \neg B) \wedge (\neg A \vee \neg C \vee D)$$

Typically written as: $(B \Rightarrow A) \wedge ((A \wedge C) \Rightarrow D)$

- A clause with one literal, e.g. A, is also called **a fact**
- A clause representing an implication (with a conjunction of positive literals in antecedent and one positive literal in consequent), is also called **a rule**

- **Modus ponens:**

$$\frac{A \Rightarrow B, \quad A}{B}$$

  – is the **complete inference rule** for KBs in the Horn normal form. **Not all KBs are convertible to HNF !!!**

Horn normal form in FOL

First-order logic (FOL)

– **adds variables and quantifiers, works with terms**

Generalized modus ponens rule:

$$\sigma = \text{a substitution s.t.} \; \forall i \; SUBST(\sigma, \phi_i') = SUBST(\sigma, \phi_i)$$

$$\frac{\phi_1', \phi_2' \ldots, \phi_n', \quad \phi_1 \wedge \phi_2 \wedge \ldots \phi_n \Rightarrow \tau}{SUBST\ (\sigma, \tau)}$$

Generalized modus ponens:

- is **complete** for the KBs with sentences in Horn form;
- not all first-order logic sentences can be expressed in the Horn form

Forward and backward chaining

Two inference procedures based on modus ponens for **Horn KBs**:

- **Forward chaining**

  Idea: Whenever the premises of a rule are satisfied, infer the conclusion. Continue with rules that became satisfied.

  Typical usage: If we want to infer all sentences entailed by the existing KB.

- **Backward chaining (goal reduction)**

  Idea: To prove the fact that appears in the conclusion of a rule prove the premises of the rule. Continue recursively.

  Typical usage: If we want to prove that the target (goal) sentence is entailed by the existing KB.

Both procedures are **complete for KBs in Horn form !!!**

Forward chaining example

- **Forward chaining**

Idea: Whenever the premises of a rule are satisfied, infer the conclusion. Continue with rules that became satisfied

Assume the KB with the following rules:

KB:
R1: $Steamboat(x) \wedge Sailboat(y) \Rightarrow Faster(x,y)$
R2: $Sailboat(y) \wedge RowBoat(z) \Rightarrow Faster(y,z)$
R3: $Faster(x,y) \wedge Faster(y,z) \Rightarrow Faster(x,z)$

F1: $Steamboat(Titanic)$
F2: $Sailboat(Mistral)$
F3: $RowBoat(PondArrow)$

Theorem: $Faster(Titanic, PondArrow)$ **?**

Forward chaining example

KB:
R1: $Steamboat(x) \wedge Sailboat(y) \Rightarrow Faster(x,y)$
R2: $Sailboat(y) \wedge RowBoat(z) \Rightarrow Faster(y,z)$
R3: $Faster(x,y) \wedge Faster(y,z) \Rightarrow Faster(x,z)$

F1: $Steamboat(Titanic)$
F2: $Sailboat(Mistral)$
F3: $RowBoat(PondArrow)$

?

Forward chaining example

KB: R1: $Steamboat(x) \wedge Sailboat(y) \Rightarrow Faster(x, y)$

R2: $Sailboat(y) \wedge RowBoat(z) \Rightarrow Faster(y, z)$

R3: $Faster(x, y) \wedge Faster(y, z) \Rightarrow Faster(x, z)$

F1: $Steamboat(Titanic)$

F2: $Sailboat(Mistral)$

F3: $RowBoat(PondArrow)$

Rule R1 is satisfied:

F4: $Faster(Titanic, Mistral)$ ⬅

Forward chaining example

KB: R1: $Steamboat(x) \wedge Sailboat(y) \Rightarrow Faster(x, y)$

R2: $Sailboat(y) \wedge RowBoat(z) \Rightarrow Faster(y, z)$

R3: $Faster(x, y) \wedge Faster(y, z) \Rightarrow Faster(x, z)$

F1: $Steamboat(Titanic)$

F2: $Sailboat(Mistral)$

F3: $RowBoat(PondArrow)$

Rule R1 is satisfied:

F4: $Faster(Titanic, Mistral)$ ⬅

Rule R2 is satisfied:

F5: $Faster(Mistral, PondArrow)$ ⬅

Forward chaining example

KB: R1: $Steamboat(x) \wedge Sailboat(y) \Rightarrow Faster(x, y)$

R2: $Sailboat(y) \wedge RowBoat(z) \Rightarrow Faster(y, z)$

R3: $Faster(x, y) \wedge Faster(y, z) \Rightarrow Faster(x, z)$

F1: $Steamboat(Titanic)$

F2: $Sailboat(Mistral)$

F3: $RowBoat(PondArrow)$

Rule R1 is satisfied:

F4: $Faster(Titanic, Mistral)$ ⬅

Rule R2 is satisfied:

F5: $Faster(Mistral, PondArrow)$ ⬅

Rule R3 is satisfied:

F6: $Faster(Titanic, PondArrow)$ ⬅

Backward chaining example

- **Backward chaining (goal reduction)**
  Idea: To prove the fact that appears in the conclusion of a rule prove the antecedents (if part) of the rule repeat recursively.

KB: R1: $Steamboat(x) \wedge Sailboat(y) \Rightarrow Faster(x, y)$

R2: $Sailboat(y) \wedge RowBoat(z) \Rightarrow Faster(y, z)$

R3: $Faster(x, y) \wedge Faster(y, z) \Rightarrow Faster(x, z)$

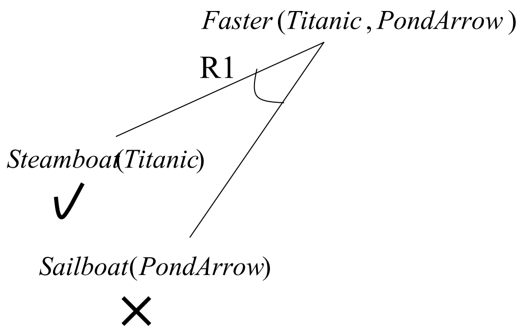
F1: $Steamboat(Titanic)$

F2: $Sailboat(Mistral)$

F3: $RowBoat(PondArrow)$

Theorem: $Faster(Titanic, PondArrow)$

Backward chaining example



F1: $Steamboat(Titanic)$

F2: $Sailboat(Mistral)$

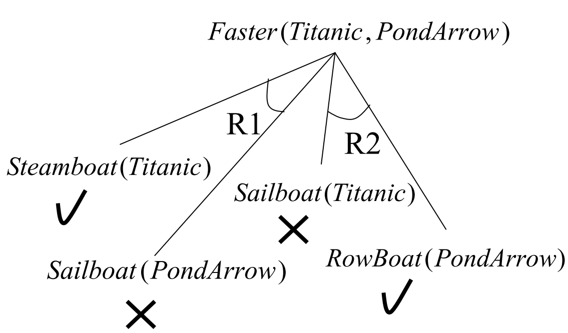
F3: $RowBoat(PondArrow)$

$$Steamboat(x) \wedge Sailboat(y) \Rightarrow Faster(x, y)$$

$$Faster(Titanic, PondArrow)$$

$$\{x / Titanic, y / PondArrow\}$$

Backward chaining example



F1: $Steamboat(Titanic)$

F2: $Sailboat(Mistral)$

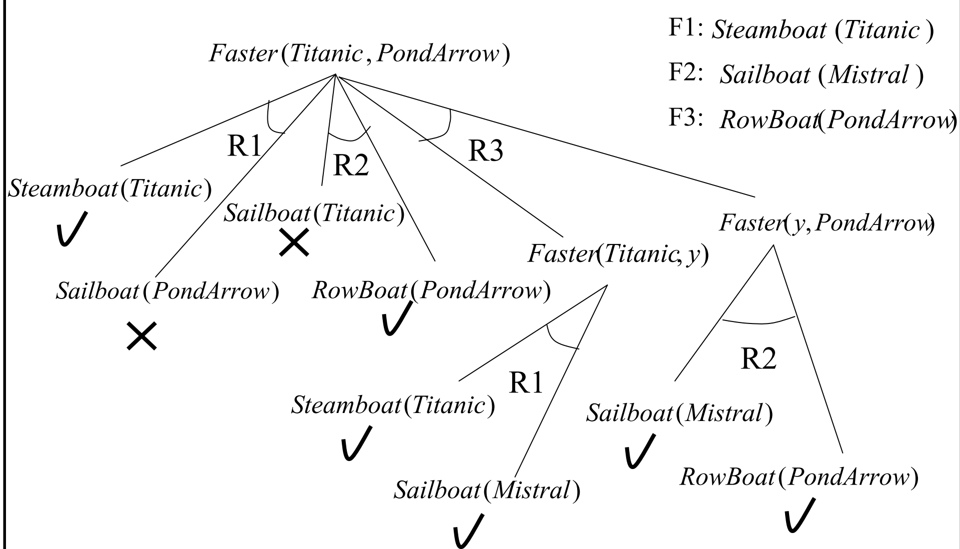
F3: $RowBoat(PondArrow)$

$$Sailboat(y) \wedge RowBoat(z) \Rightarrow Faster(y, z)$$

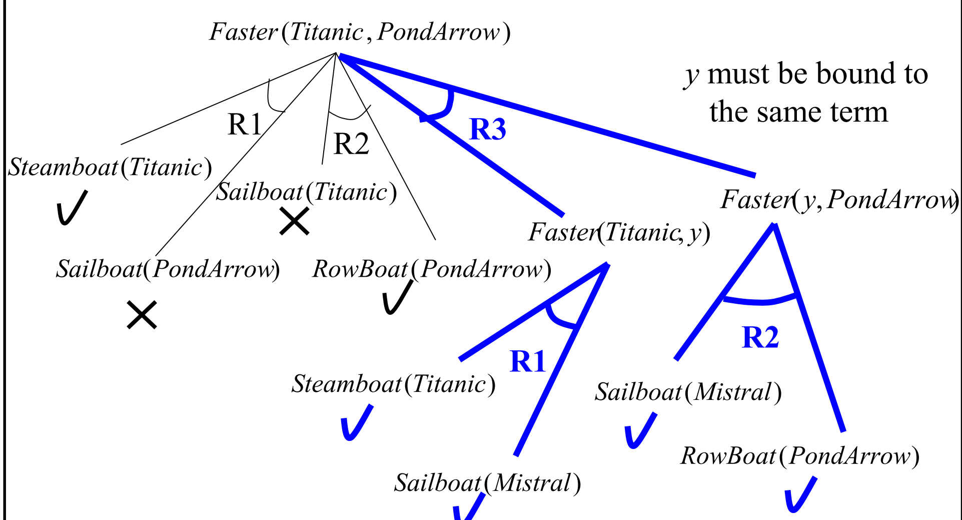
$$Faster(Titanic, PondArrow)$$

$$\{y / Titanic, z / PondArrow\}$$

Backward chaining example

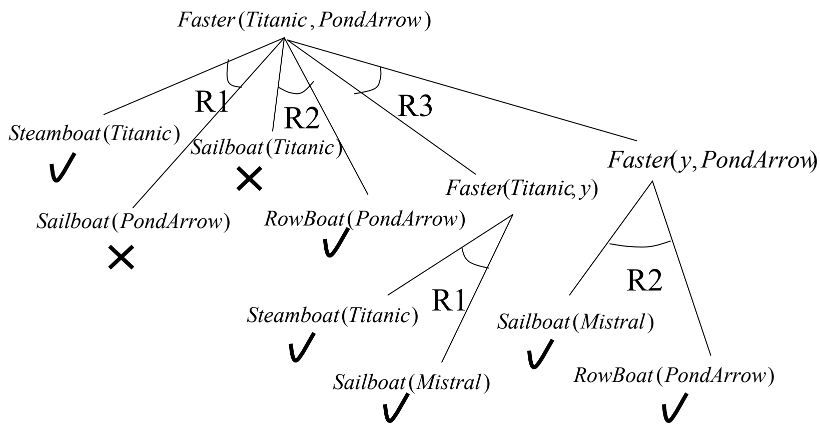


Backward chaining



Backward chaining

- The search tree: **AND/OR tree**
- Special search algorithms exits (including heuristics): AO, AO*



Knowledge-based system

Knowledge base
Inference engine

- **Knowledge base:**
  - A set of sentences that describe the world in some formal (representational) language (e.g. first-order logic)
  - Domain specific knowledge
- **Inference engine:**
  - A set of procedures that work upon the representational language and can infer new facts or answer KB queries (e.g. resolution algorithm, forward chaining)
  - Domain independent

Retrieval of KB information

- The reasoning algorithms operating upon the KB need to access and manipulate information stored there
  - Large KBs consist of thousands of sentences
- **Problem:** retrieval of sentences from the KB (e.g. for the purpose of unification)
  - Simple flat list of conjuncts can be very long and searching it exhaustively is inefficient
- **Solution:** indexing
  - Store and maintain the sentences in a table (hash table) according to predicate symbols they include

Table-based indexing of KBs

Assume the knowledge is expressed in the implicative form, with sentences corresponding to facts and rules

- For each predicate we can store its:
  - positive literals
  - negative literals,
  - rules in which it occurs in the premise,
  - rules in which it occurs in the conclusion.

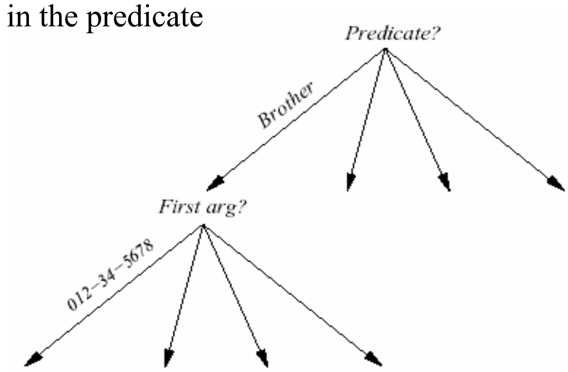
Key	Positive	Negative	Conclusion	Premise
Brother	$Brother(Richard, John)$ $Brother(Ted, Jack)$ $Brother(Jack, Bobbie)$	$\neg Brother(Ann, Sam)$	$Brother(x,y) \wedge Male(y) \Rightarrow Brother(y,x)$	$Brother(x,y) \wedge Male(y) \Rightarrow Brother(y,x)$ $Brother(x,y) \Rightarrow Male(x)$
Male	$Male(Jack)$ $Male(Ted)$ ...	$\neg Male(Ann)$ ...	$Brother(x,y) \Rightarrow Male(x)$	$Brother(x,y) \wedge Male(y) \Rightarrow Brother(y,x)$

Indexing and retrieval of KB information

Problem: the number of elements (clauses) with the same predicate can be huge

Solution: tree-based indexing

- structure the KB further, create tables for different symbols that occur in the predicate



Indexing of information in KBs

Problem: matching of sentences with variables

- Too many entries need to be searched and this even if the resulting set is small

  Assume: *Taxpayer(SSN, zipCode, net_income,dependents)*

  We want to match e.g.: *Taxpayer(x, 15260, y,5)*

- **Partial solution: cross-indexing**
- Create more special tables combining predicates and arguments

  e.g. have a table for: *Taxpayer+zip_code+num_dependents*

- Choose and search the most promising table for retrieval
- No universal solution for all possible matchings, since all the number of all tables would go up exponentially

Automated reasoning systems

Examples and main differences:

- **Theorem provers**
  – Prove sentences in the first-order logic
- **Deductive retrieval systems**
  – Systems based on rules (KBs in Horn form)
  – Prove theorems or infer new assertions (forward, backward chaining)
- **Production systems**
  – Systems based on rules with actions in antecedents
  – Forward chaining mode of operation
- **Semantic networks**
  – Graphical representation of the world, objects are nodes in the graphs, relations are various links

Production systems

Based on rules, but different from KBs in the Horn form

Knowledge base is divided into:

- **rule base (includes rules)**
- **working memory (includes facts)**

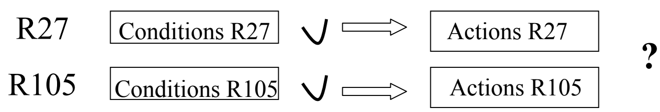
A special type of if – then rule

$$p_1 \wedge p_2 \wedge \ldots p_n \Rightarrow a_1, a_2, \ldots, a_k$$

- **Antecedent:** a conjunction of literal
  – facts, statements in predicate logic
- **Consequent:** a conjunction of actions. An action can:
  – **ADD** the fact to the KB (working memory)
  – **REMOVE** the fact from the KB
  – **QUERY** the user, etc …

Production systems

- Use **forward chaining to do reasoning**:
  - If the antecedent of the rule is satisfied (rule is said to be "active") then its consequent can be executed (it is "fired")
- **Problem:** Two or more rules are active at the same time. Which one to execute next?



- Strategy for selecting the rule to be fired from among possible candidates is called **conflict resolution**
- Why do we care about the order?
  - action of R27 can delete one of the preconditions of R105 and deactivate the R105
  - **Note:** this is not a problem in Horn KB (no deletions)

Production systems

- **Problems with production systems:**
  - Additions and Deletions can change a set of active rules;
  - If a rule contains variables testing all instances in which the rule is active may require a large number of unifications.
  - Conditions of many rules may overlap, thus requiring to repeat the same unifications multiple times.
- **Solution: Rete algorithm**
  - gives more efficient solution for managing a set of active rules and performing unifications
  - Implemented in the system **OPS-5** (used to implement XCON – an expert system for configuration of DEC computers)

Rete algorithm

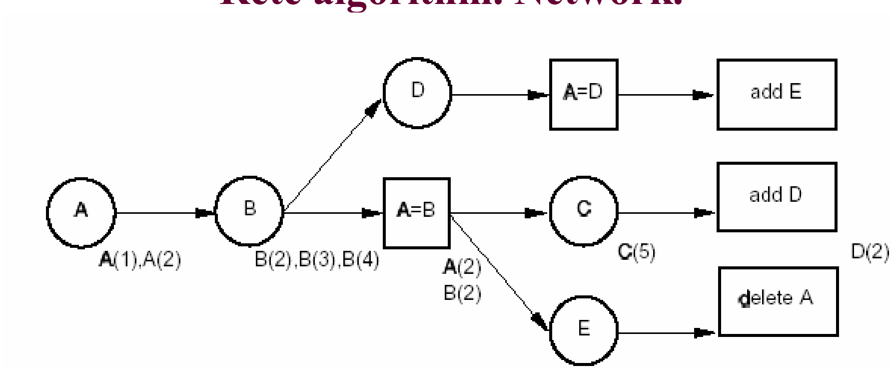
- Assume a set of rules:

$$A(x) \wedge B(x) \wedge C(y) \Rightarrow add \;\; D(x)$$
$$A(x) \wedge B(y) \wedge D(x) \Rightarrow add \;\; E(x)$$
$$A(x) \wedge B(x) \wedge E(z) \Rightarrow delete \;\; A(x)$$

- And facts: $A(1), A(2), B(2), B(3), B(4), C(5)$

- **Rete:**
  - Compiles the rules to a network that merges conditions of multiple rules together (avoid repeats)
  - Propagates valid unifications
  - Reevaluates only changed conditions

Rete algorithm. Network.



Rules: $A(x) \wedge B(x) \wedge C(y) \Rightarrow add \;\; D(x)$

$A(x) \wedge B(y) \wedge D(x) \Rightarrow add \;\; E(x)$

$A(x) \wedge B(x) \wedge E(z) \Rightarrow delete \;\; A(x)$

Facts: $A(1), A(2), B(2), B(3), B(4), C(5)$

Conflict resolution strategies

- **Problem:** Two or more rules are active at the same time. Which one to execute next?
- **Solutions:**
  - **No duplication** (do not execute the same rule twice)
  - **Recency.** Rules referring to facts newly added to the working memory take precedence
  - **Specificity.** Rules that are more specific are preferred.
  - **Priority levels.** Define priority of rules, actions based on expert opinion. Have multiple priority levels such that the higher priority rules fire first.

Semantic network systems

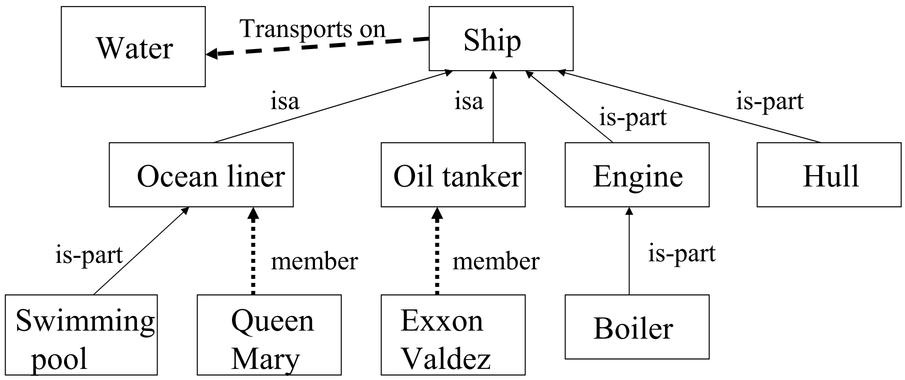
- Knowledge about the world described in terms of graphs. Nodes correspond to:
  - **Concepts or objects** in the domain.

  Links to relations. Three kinds:
  - **Subset links** (isa, part-of links)
  - **Member links** (instance links)
  - **Function links**.

  (Subset links and member links: Inheritance relation links)

- Can be transformed to the first-order logic language
- Graphical representation is often easier to work with
  - better overall view on individual concepts and relations

Semantic network. Example.



Inferred properties: *Queen Mary is a ship*
Queen Mary has a boiler