CS 1571 Introduction to AI
Lecture 15

Inferences in first-order logic. Knowledge based systems.

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

Administration announcements

Midterm:

- **Thursday, October 25, 2012**
- **In-class**
- **Closed book**

What does it cover?

- **All material covered by the end of lecture today**

Homework 7:

- **out today to practice inferences in FOL**

Tic-tac-toe competion

The programs were played against each other 10 times (5 times as first and 5 times as second player)

Winners:

1. **Rishi Sadhir and Yuriy Koziy**
3. **Nathan Essel**

Reward: 20% credit for the homework

Sentences in Horn normal form

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

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

Implicative form: $(B \Rightarrow A) \wedge ((A \wedge C) \Rightarrow D)$

- 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**
- **Resolution rule (for clausal form) and modus ponens (for implicative form):**
  - Both are **complete inference rule** for unit inferences for KBs in the Horn normal form.
  - **Recall: Not all KBs are convertible to HNF !!!**

Horn normal form in FOL

- **First-order logic (FOL)**
  - adds variables and quantifiers, works with terms, predicates
- **HNF in FOL:** primitive sentences (propositions) are formed by predicates
- **Inference rules: generalized versions with substitutions**
  Example: modus ponens

  $\sigma =$ 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 resolution and generalized modus ponens:**
  - is **complete** for unit inferences for the KBs in HN;

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 α is entailed by the existing KB.

Both procedures are **complete for unit inferences in 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: $Faster(x, y) \wedge Faster(y, z) \Rightarrow Faster(x, z)$

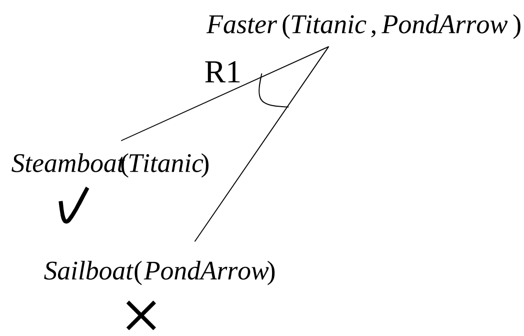
F1: $Steamboat(Titanic)$

F2: $Sailboat(Mistral)$

F3: $RowBoat(PondArrow)$

Theorem: $Faster(Titanic, PondArrow)$

Backward chaining example



F1: $Steamboat(Titanic)$

F2: $Sailboat(Mistral)$

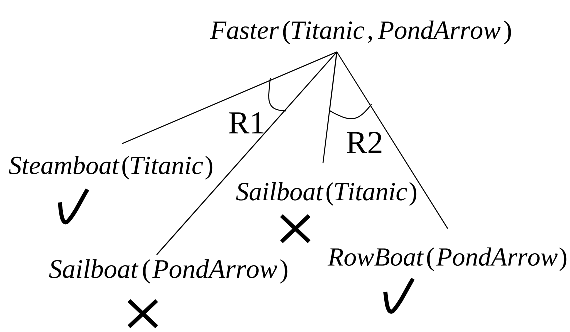
F3: $RowBoat(PondArrow)$

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

$$Faster(Titanic, PondArrow)$$

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

Backward chaining example



F1: $Steamboat(Titanic)$

F2: $Sailboat(Mistral)$

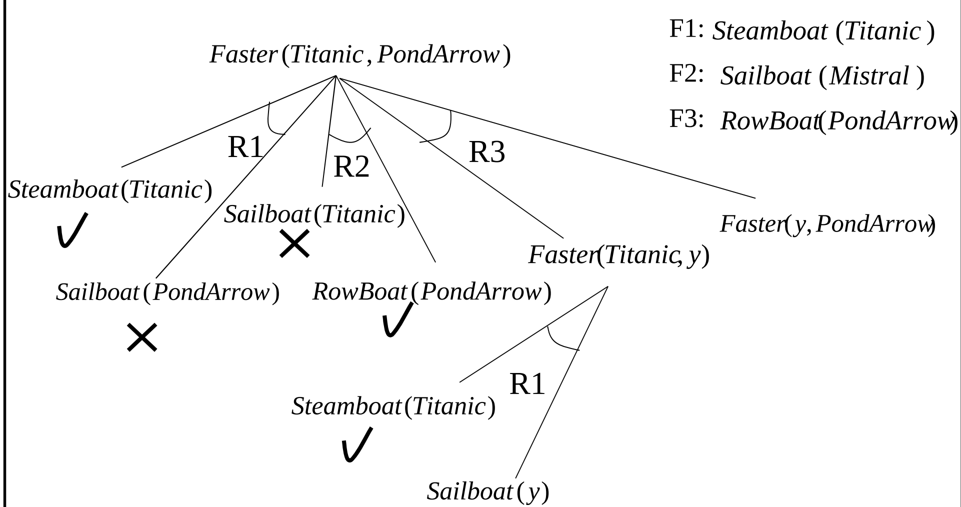
F3: $RowBoat(PondArrow)$

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

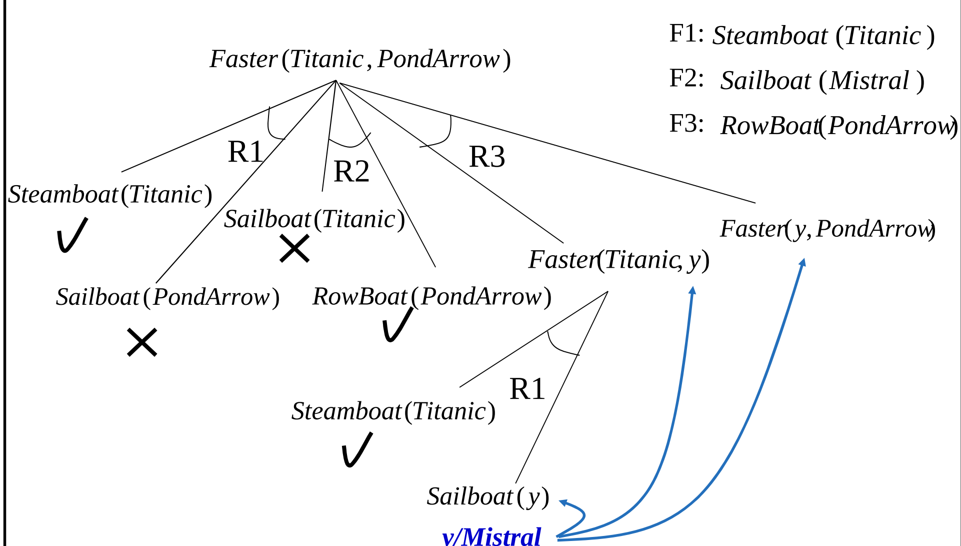
$$Faster(Titanic, PondArrow)$$

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

Backward chaining example

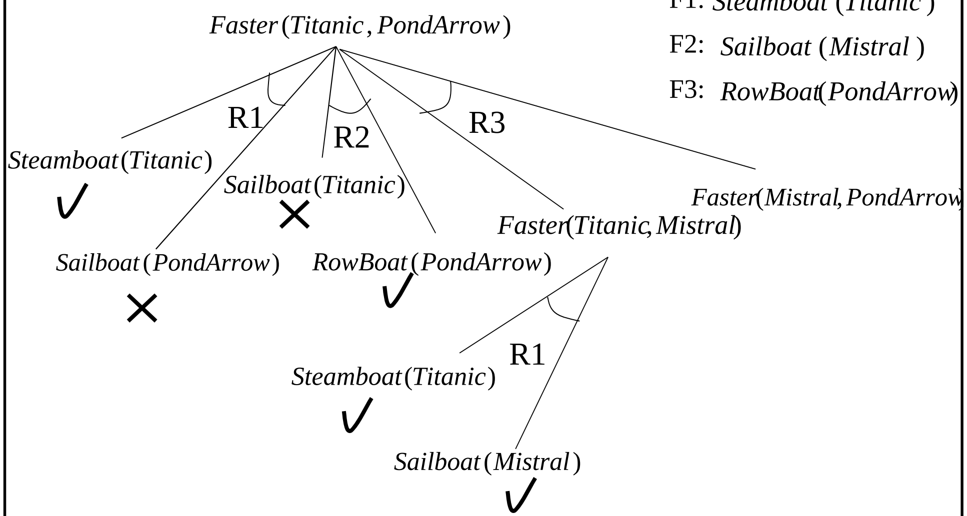


Backward chaining example



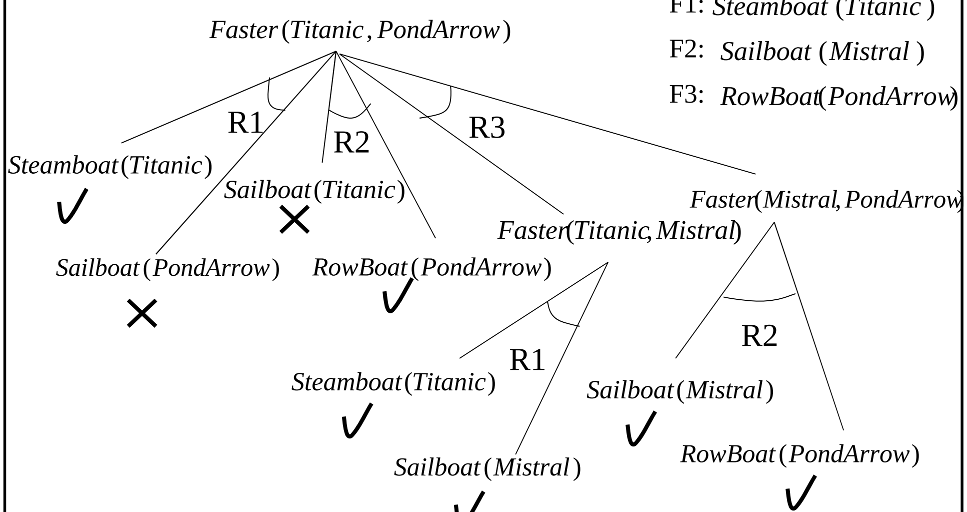
Backward chaining example

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

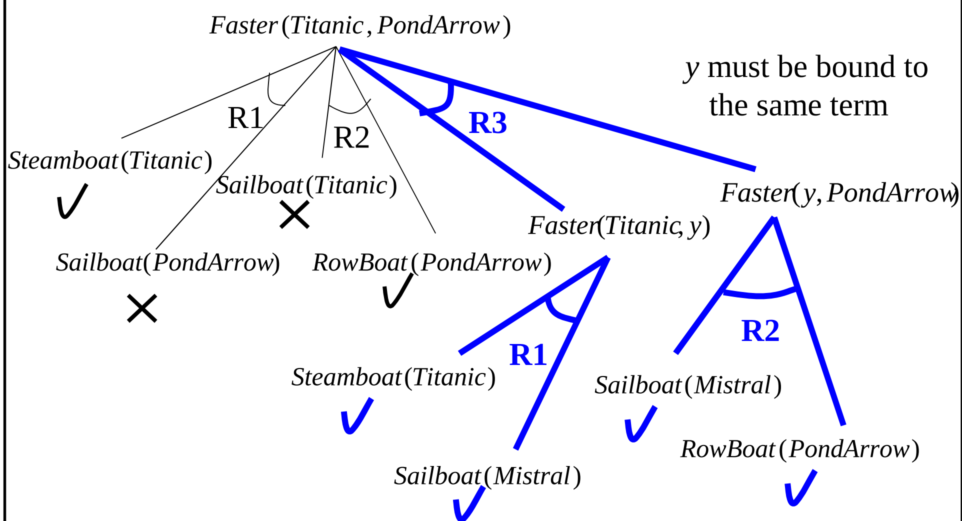


Backward chaining example

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

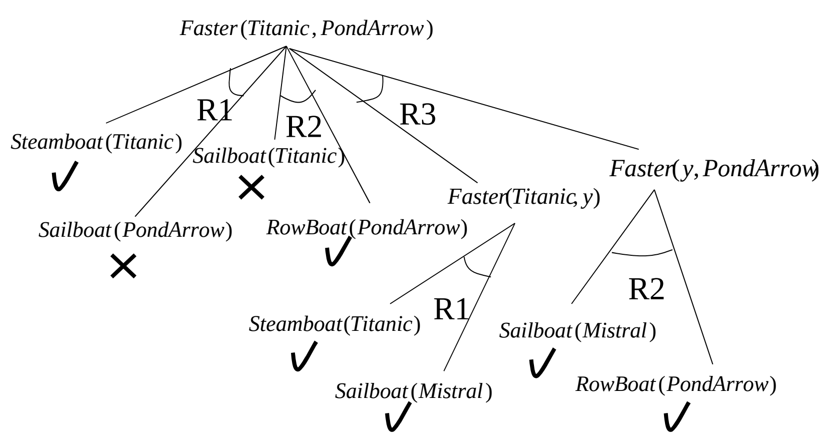


Backward chaining



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

Automated reasoning systems

Examples and main differences:

- **Theorem provers**
  - Prove sentences in the first-order logic. Use inference rules, resolution rule and resolution refutation.
- **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:

- **A Rule base (includes rules)**
- **A 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 literals
  - 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 (consistent with logic ?)
  - **QUERY** the user, etc …

Production systems

Based on rules, but different from KBs in the Horn form
Knowledge base is divided into:

- **A Rule base (includes rules)**
- **A Working memory (includes facts)**

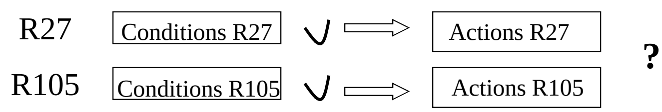
A special type of if – then rule

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

- **Antecedent:** a conjunction of literals
  - 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 ← !!! Different from logic
  - **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**

Production systems

- Why is conflict resolution important? Or, why do we care about the order?
- Assume that we have two rules and the preconditions of both are satisfied:

R1: $A(x) \wedge B(x) \wedge C(y) \Rightarrow add\ \ D(x)$

R2: $A(x) \wedge B(x) \wedge E(z) \Rightarrow delete\ \ A(x)$

- What can happen if rules are triggered in different order?

Production systems

- Why is conflict resolution important? Or, Why do we care about the order?
- Assume that we have two rules and the preconditions of both are satisfied:

R1: $A(x) \wedge B(x) \wedge C(y) \Rightarrow add\ \ D(x)$

R2: $A(x) \wedge B(x) \wedge E(z) \Rightarrow delete\ \ A(x)$

- What can happen if rules are triggered in different order?
  - If R1 goes first, R2 condition is still satisfied and we infer D(x)
  - If R2 goes first we may never infer D(x)

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)

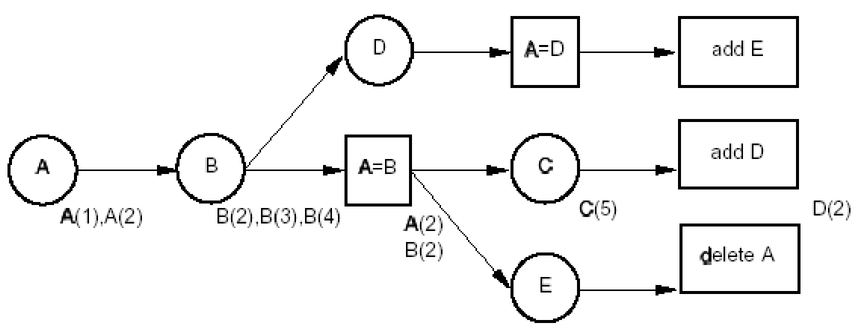
Rete algorithm

- 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

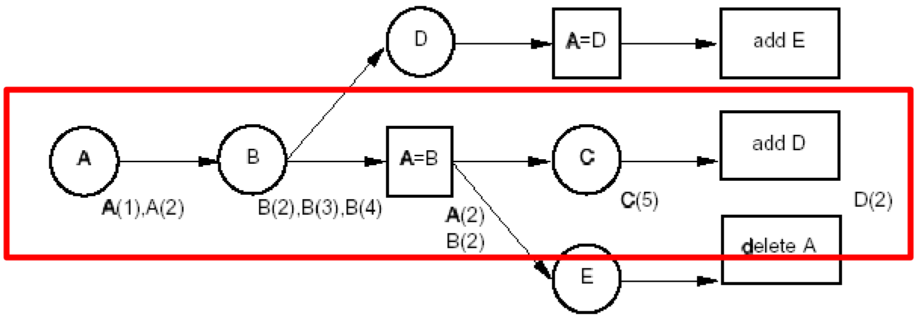
Rete algorithm. Network.



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

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

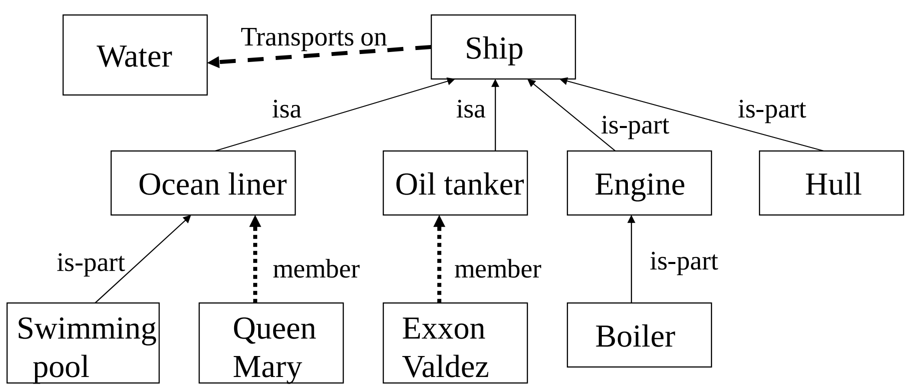
- Knowledge about the world described in terms of graphs. Nodes correspond to:
  - **Concepts or objects** in the domain.

  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