CS 2740 Knowledge Representation
Lecture 10

First order logic inference.

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

Inference with generalized 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

Resolution example

KB $\qquad\qquad$ $\neg\alpha$

$$\overbrace{\neg P(w) \vee Q(w), \neg Q(y) \vee S(y),\ P(x) \vee R(x), \neg R(z) \vee S(z)}^{KB},\quad \neg S(A)$$

Resolution example

KB $\qquad\qquad$ $\neg\alpha$

$$\overbrace{\neg P(w) \vee Q(w), \neg Q(y) \vee S(y),\ P(x) \vee R(x), \neg R(z) \vee S(z)}^{KB},\quad \neg S(A)$$

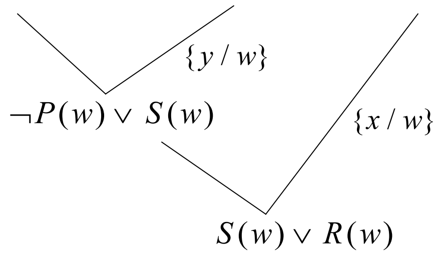
$\{y / w\}$

$\neg P(w) \vee S(w)$

Resolution example

KB $\neg\alpha$

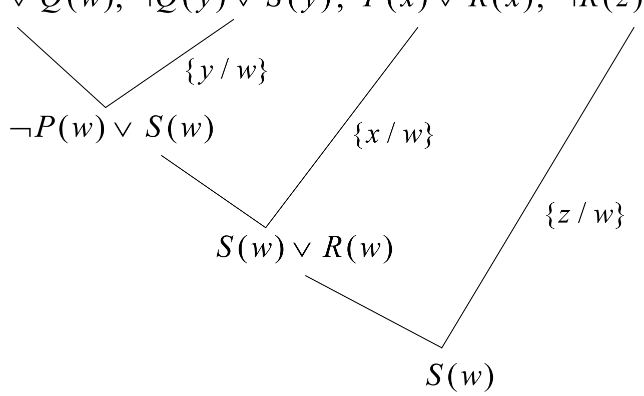
$$\neg P(w) \vee Q(w), \neg Q(y) \vee S(y), \; P(x) \vee R(x), \neg R(z) \vee S(z), \quad \neg S(A)$$



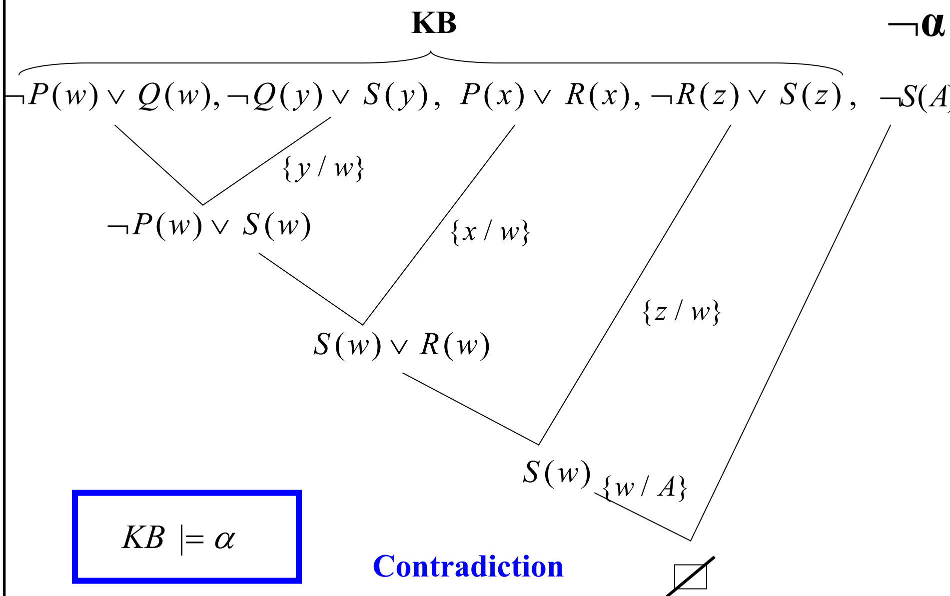
Resolution example

KB $\neg\alpha$

$$\neg P(w) \vee Q(w), \neg Q(y) \vee S(y), \; P(x) \vee R(x), \neg R(z) \vee S(z), \quad \neg S(A)$$



Resolution example



Answer predicate

In full FOL, we have the possibility of deriving $\exists x P(x)$ without being able to derive $P(t)$ for any t.

e.g. the three-blocks problem

$\exists x \exists y [\mathrm{On}(x,y) \wedge \mathrm{Green}(x) \wedge \neg \mathrm{Green}(y)]$

but cannot derive which block is which

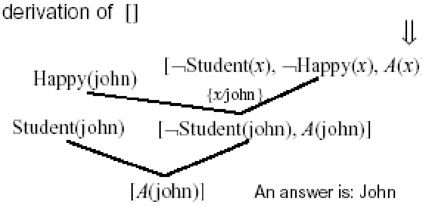
Solution: answer-extraction process

- replace query $\exists x P(x)$ by $\exists x [P(x) \wedge \neg A(x)]$

  where A is a new predicate symbol called the <u>answer predicate</u>
- instead of deriving [], derive any clause containing just the answer predicate
- can always convert to and from a derivation of []

KB: Student(john)
Student(jane)
Happy(john)

Q: $\exists x [\mathrm{Student}(x) \wedge \mathrm{Happy}(x)]$



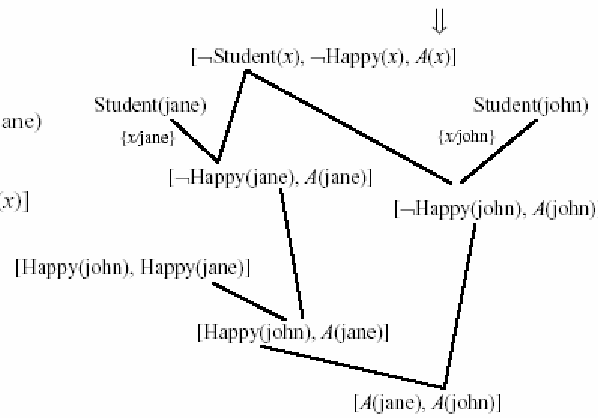
Disjunctive answers

KB:

Student(john)
Student(jane)
Happy(john) ∨ Happy(jane)

Query:

∃x[Student(x) ∧ Happy(x)]



An answer is: either Jane or John

Note:

- can have variables in answer
- need to watch for Skolem symbols... (next)

Undecidability of resolution-refutation

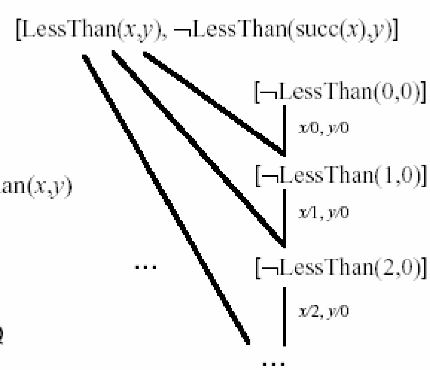
KB:

LessThan(succ(x),y) ⊃ LessThan(x,y)

Query:

LessThan(zero,zero)

Should fail since KB |≠ Q



Infinite branch of resolvents

cannot use a simple depth-first procedure to search for []

Efficiency of resolution

For the propositionalized KB

- worst case is exponential in the number literals

Speed ups of the resolution-refutation algorithm:

- **Clause elimination**. Assume a clause contains literal r such that ¬ r does not appear in any other clause. The clause cannot lead to the contradiction {} and hence can be eliminated.
- **Tautology. A** clause with a literal and its negation. Any path to {} can bypass tautology.
- **Subsumed clause.** A clause for which there exists another clause with only a subset of its literals. A path to {} need only to pass through the short clause.

Efficiency of resolution

Speed-ups:

- **Ordering strategies**
  - many possible ways to order search, but best and simplest is unit preference
  - prefer to resolve unit clauses first
  - Why? Given unit clause and another clause, the resolvent is a smaller one
- **Set of support**
  - KB is usually satisfiable, so not very useful to resolve among clauses with ancestors in KB
  - contradiction arises from interaction with the negated theorem
  - always resolve with at least one clause that has an ancestor in the negated theorem

Efficiency of resolution

- **Special treatment for equality**
  - instead of using axioms for equality
  - use new inference rule: **paramodulation**
- **Demodulation rule**

  $\sigma = UNIFY(z, t_1) \neq fail$ **where** $\phi_k[z]$ **includes term** z

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

- **Example:** $\dfrac{P(f(a)), f(x) = x}{P(a)}$
- **Paramodulation rule:** more powerful
- Resolution+paramodulation give a refutation-complete proof theory for FOL

Efficiency of resolution

Speed-ups:

- **Sorted logic**
  - terms get sorts:
  - x: Male mother:[Person → Female]
  - keep taxonomy of sorts
  - only unify $P(s)$ with $P(t)$ when sorts are compatible assumes only “meaningful” paths will lead to {}

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**
- **Inference for definite clauses:**
  - **Modus ponens inference rule**

Horn normal form in FOL

First-order logic (FOL)
- **adds variables, 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 **sound** and **complete** for **definite clauses** and **no functions**;
- In general it is semidecidable
- Not all first-order logic sentences can be expressed in the HNF 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 α is entailed by the existing KB.

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

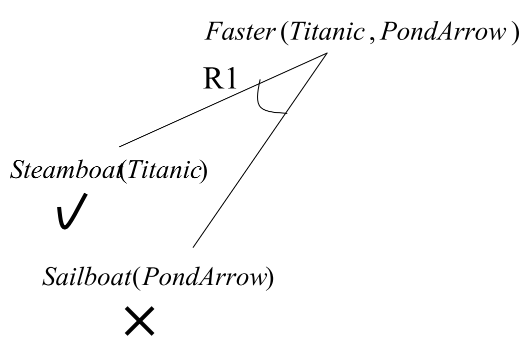
F1: $Steamboat(Titanic)$

F2: $Sailboat(Mistral)$

F3: $RowBoat(PondArrow)$

Theorem: $Faster(Titanic, PondArrow)$

Backward chaining example



F1: $Steamboat(Titanic)$

F2: $Sailboat(Mistral)$

F3: $RowBoat(PondArrow)$

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

$Faster(Titanic, PondArrow)$

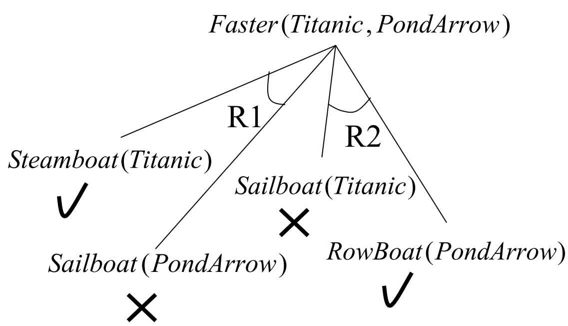
$\{x / Titanic, y / PondArrow\}$

Backward chaining example

F1: $Steamboat(Titanic)$

F2: $Sailboat(Mistral)$

F3: $RowBoat(PondArrow)$



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

$$Faster(Titanic, PondArrow)$$

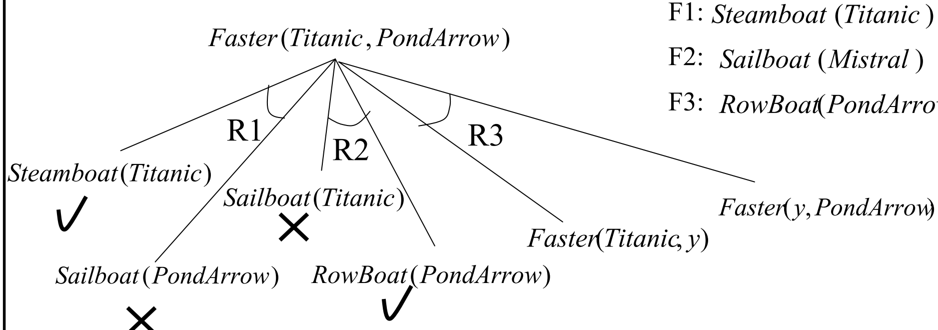
$$\{y / Titanic, z / PondArrow\}$$

Backward chaining example

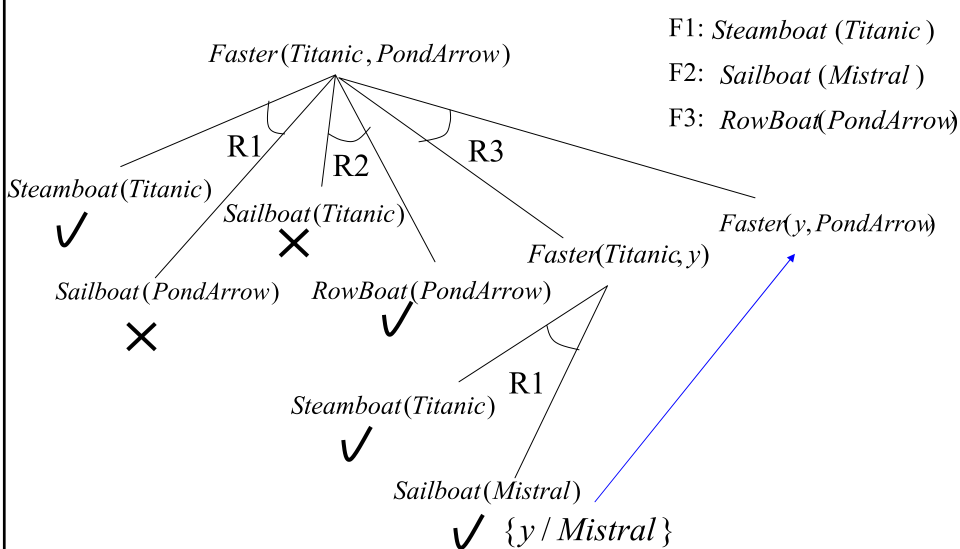
F1: $Steamboat(Titanic)$

F2: $Sailboat(Mistral)$

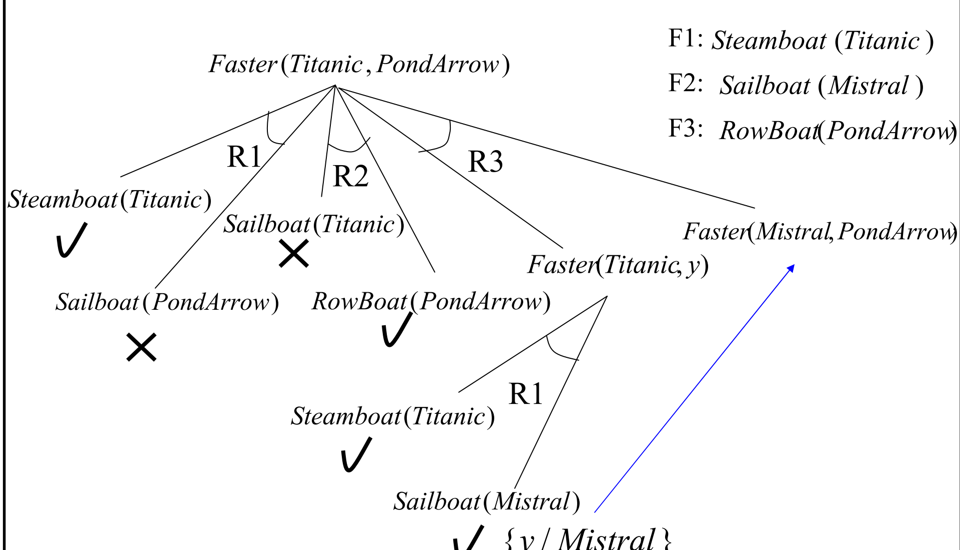
F3: $RowBoat(PondArrow)$



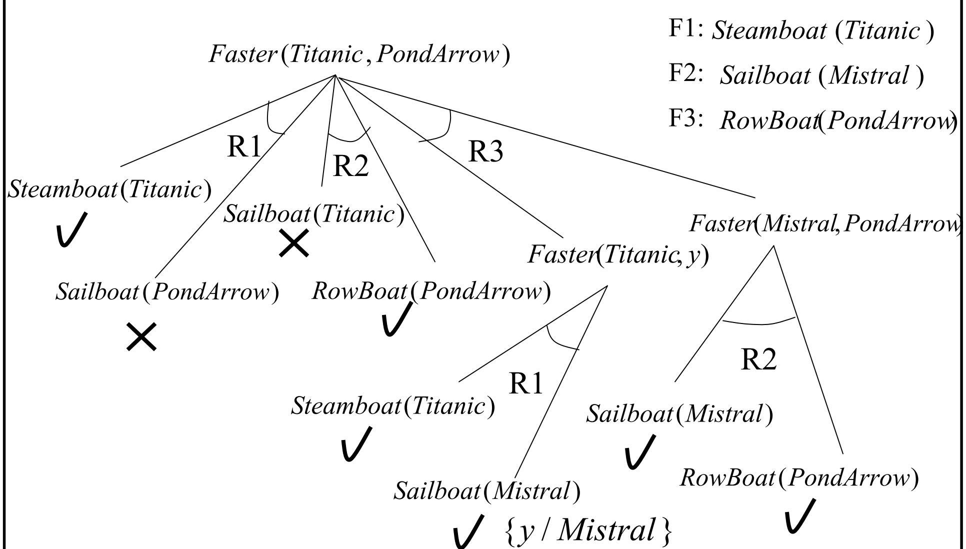
Backward chaining example



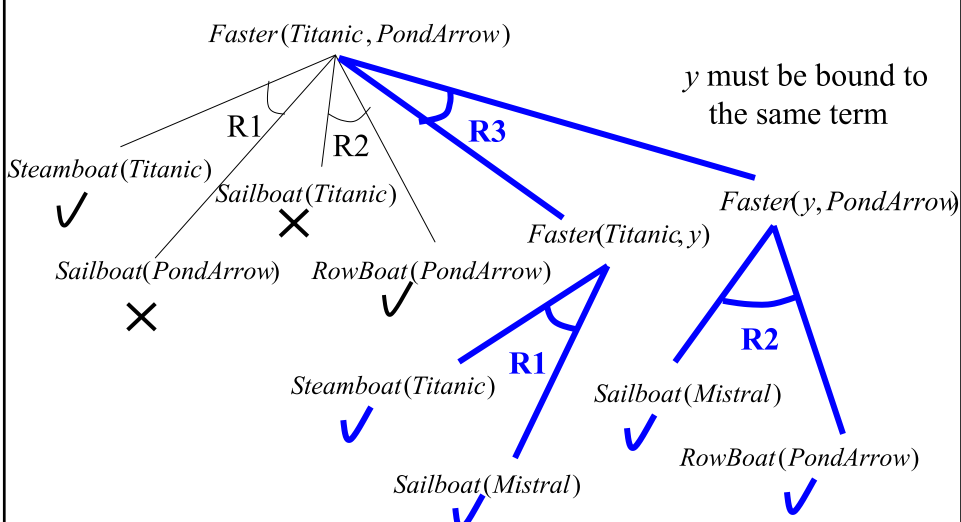
Backward chaining example



Backward chaining example



Backward chaining



Properties of backward chaining

- Depth-first recursive proof search:
  - space is linear in size of proof
- Incomplete due to possible infinite loops
  - fix by checking current goal against every goal on stack
- Inefficient due to repeated subgoals (both success and failure)
  - fix using caching of previous results (extra space)
- Widely used for **logic programming**

Logic programming: Prolog

- Algorithm = Logic + Control
- Basis:
  - backward chaining with Horn clauses + bells & whistles
- Widely used in Europe, Japan (basis of 5th Generation project)
- Program = set of clauses
  - `head :- literal`$_1$`, … literal`$_n$.

Example:

```
criminal(X) :- american(X), weapon(Y),
      sells(X,Y,Z), hostile(Z).
```

Logic programming: Prolog

Example:

```
criminal(X) :- american(X), weapon(Y),
        sells(X,Y,Z), hostile(Z).
```

- Depth-first, left-to-right backward chaining
- Built-in predicates for arithmetic etc., e.g., `X is Y*Z+3`
- Built-in predicates that have side effects (e.g., input and output predicates, assert/retract predicates)

- Closed-world assumption ("negation as failure")
  - e.g., given `alive(X) :- not dead(X).`
  - `alive(joe)` succeeds if `dead(joe)` fails