Machine Learning

Learning Sets of Rules

Artificial Intelligence & Computer Vision Lab
School of Computer Science and Engineering
Seoul National University

Overview

- Introduction
- Sequential Covering Algorithms
- Learning Rule Sets: Summary
- Learning First-Order Rules
- Learning Sets of First-Order Rules: FOIL
- Induction as Inverted Deduction
- Inverting Resolution
- Summary

Introduction

- Set of “if-then” rules
  - The hypothesis is easy to interpret.
- Goal
  - Look at a new method to learn rules
- Rules
  - Propositional rules (rules without variables)
  - First-order predicate rules (with variables)

Introduction (cont.)

- So far . . .
  - Method 1: Learn decision tree → rules
  - Method 2: Genetic algorithm, encode rule set as a bit string
- From now . . . new method!
  - Learning first-order rule
  - Using sequential covering
- First-order rule
  - Difficult to represent using a decision tree or other propositional representation

If	*Parent(x,y)*	then	*Ancestor(x,y)*
If	*Parent(x,z) and Ancestor(z,y)*	then	*Ancestor(x,y)*

Sequential Covering Algorithms

- Algorithm
  1. Learn one rule that covers certain number of examples
  2. Remove those examples covered by the rule
  3. Repeat on the examples left until the learned rule has the performance greater than predefined threshold.

- Require that each rule has high accuracy but low coverage
  - High accuracy → the correct prediction
  - Accepting low coverage → the prediction NOT necessary for every training example

Sequential Covering Algorithms (cont.)

- SEQUENTIAL-COVERING(*Target-Attribute, Attributes, Examples, Threshold*)

```
Learned-Rules ← {}
Rule ← LEARN-ONE-RULE(Target-Attribute, Attributes, Examples)
While PERFORMANCE(Rule, Examples) > Threshold, do
    Learned_rules ← Learned_rules + Rule
    Examples ← Examples - {examples correctly classified by Rule}
    Rule ← LEARN-ONE-RULE(Target-Attribute, Attributes, Examples)
Learned-values ← sort Learned-values according to PERFORMANCE over
  Examples
return Learned-Rules
```

Sequential Covering Algorithms (cont.)

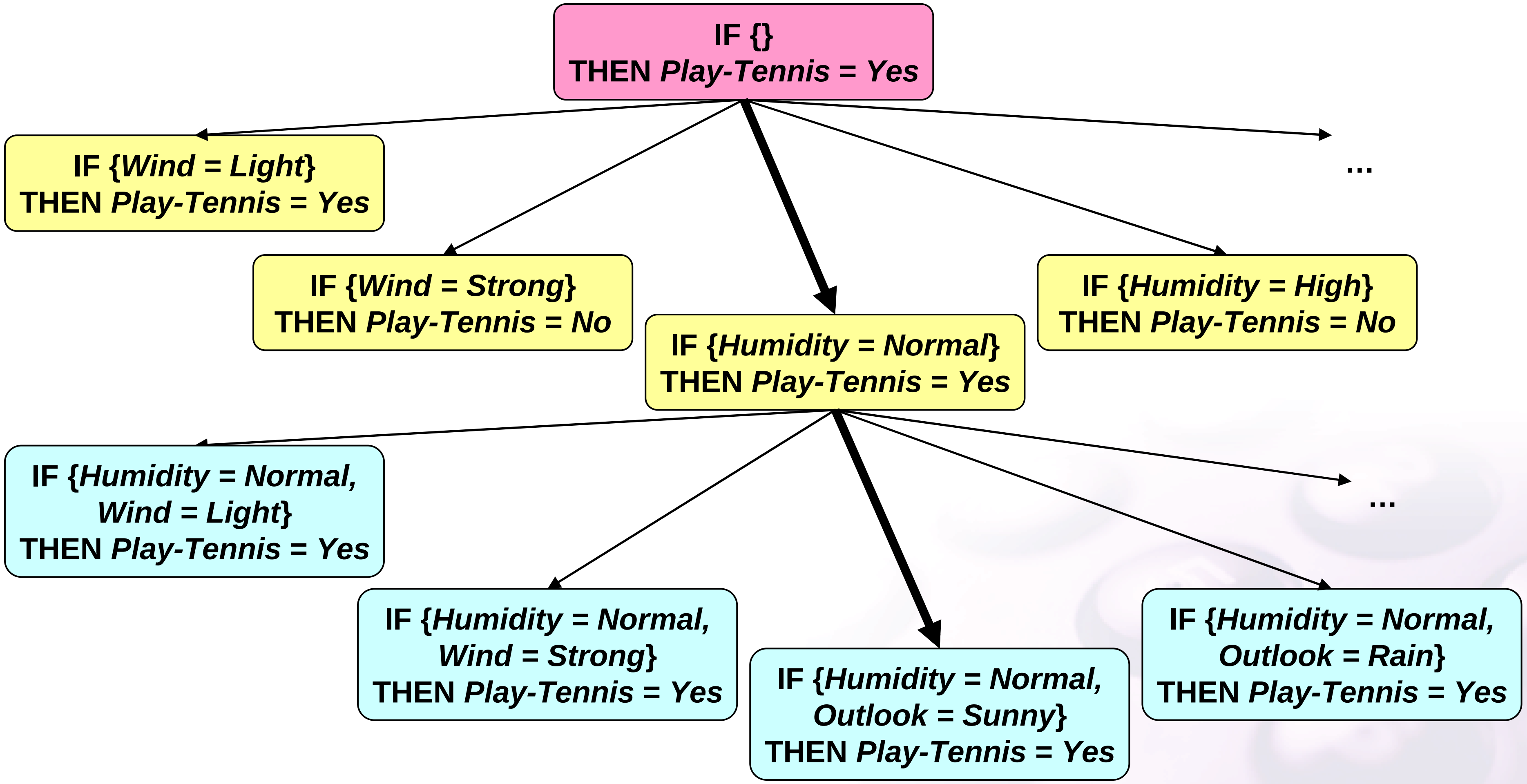
- One of the most widespread approaches to learning disjunctive sets of rules.
- Problem of learning disjunctive sets of rules reduced to a sequence of simpler problems, each requiring that a single of conjunctive rule be learned.
- It performs a greedy search, formulating a sequence of rules without backtracking. Not guarantee to find a smallest or best set of rules covering training examples.

Sequential Covering Algorithms (cont.)

- General to specific beam search
  - How do we learn each individual rule?
  - Requirements for LEARN-ONE-RULE
    - High accuracy, need not high coverage
  - One approach is . . .
    - To implement LEARN-ONE-RULE in similar way as in decision tree learning (ID3), but to follow only the most promising branch in the tree at each step.
    - As illustrated in the figure, the search begins by considering the most general rule precondition possible (the empty test that matches every instance), then greedily adding the attribute test that most improves rule performance over training examples.

Sequential Covering Algorithms (cont.)

- General to specific beam search


IF {}
THEN *Play-Tennis* = *Yes*
IF {*Wind* = *Light*}
THEN *Play-Tennis* = *Yes*
IF {*Wind* = *Strong*}
THEN *Play-Tennis* = *No*
IF {*Humidity* = *Normal*}
THEN *Play-Tennis* = *Yes*
IF {*Humidity* = *High*}
THEN *Play-Tennis* = *No*
…
IF {*Humidity* = *Normal*, *Wind* = *Light*}
THEN *Play-Tennis* = *Yes*
IF {*Humidity* = *Normal*, *Wind* = *Strong*}
THEN *Play-Tennis* = *Yes*
IF {*Humidity* = *Normal*, *Outlook* = *Sunny*}
THEN *Play-Tennis* = *Yes*
IF {*Humidity* = *Normal*, *Outlook* = *Rain*}
THEN *Play-Tennis* = *Yes*
…

Sequential Covering Algorithms (cont.)

- General to specific beam search
  - Greedy search without backtracking

    ➔ danger of suboptimal choice at any step

  - The algorithm can be extended using beam-search
    - Keep a list of the *k* best candidates at each step rather than a single best candidate
    - On each search step, descendants are generated for each of these *k* best candidates and the resulting set is again reduced to the *k* best candidates.

Sequential Covering Algorithms (cont.)

- General to specific beam search

```
LEARN_ONE_RULE (target_attr,attributes,examples,k)
    Best-hypothesis ← Ø
    Candidate-hypotheses ← {Best_hypothesis}
    While Candidate-hypotheses is not empty, do
    1. Generate the next more specific candidate hypotheses
        – All_constraints ← the set of constraints (a=v) where a is attribute and v is its value in Examples.
        – New_candidate_hypotheses ← for each h in Candidate_hypotheses,
                                          for each c in All_constraints,
                                            create a specialization of h by adding the constant c
        – Remove from New_candidate_hyporheses any hypotheses that are duplicates, inconsistent, or
          not maximally specific.
    2. Update Best_hypothesis
        – For all h in New_candidates_hypotheses
            » if (PERFORMANCE(h, Examples, Target_attribute) >
              PERFORMANCE(Best_hypothesis, Examples, Target_attribute)) Then Best_hypothesis
              ← h
    3. Update Candidate_hypotheses
        – Candidate_hypotheses ← the k best members of New_candidates_hypotheses, according to the
          PERFORMANCE measure
    Return a rule of the form
       "IF Best-hypothesis THEN prediction"
       where predication is the most frequent value of target_attr among those examples that match Best-
      hypothesis
```

Sequential Covering Algorithms (cont.)

- General to specific beam search
  - PERFORMANCE*(h, examples, target_attribute)*

```
    $h\_examples \leftarrow$ the subset of $examples$ that match $h$
    Return - Entropy(h_examples), where entropy is with respect to
      Target_attribute
```

Sequential Covering Algorithms (cont.)

- Variations
  - Learn only rules that cover positive examples
    - In the case that the fraction of positive example is small
    - In this case, we can modify the algorithm to learn only from those rare example, and classify anything not covered by any rule as negative.
    - Instead of entropy, use a measure that evaluates the fraction of positive examples covered by the hypothesis
  - AQ-algorithm
    - Different covering algorithm
    - Searches rule sets for particular target value
    - Different single-rule algorithm
      - Guided by uncovered positive examples
      - Only attributes satisfied in positive examples are considered.

Learning Rule Sets: Summary

- *Sequential* or *Simultaneous*?
  - *Sequential* : Learning one rule at a time, removing the covered examples and repeating the process on the remaining examples
  - *Simultaneous* : Learning the entire set of disjucts simultaneously as part of the single search for an acceptable decision tree as in ID3
- *General-to-specific* or *Specific-to-general*?
  - $G \rightarrow S$: Learn-One-Rule
  - $S \rightarrow G$: Find-S
- *Generate-and-test* or *Example-driven*?
  - G&T : search thru syntactically legal hypotheses
  - E-D : Find-S, Candidate-Elimination
- Post-pruning of rules?
  - Similar method to the one discussed in decision tree learning

Learning Rule Sets: Summary (cont.)

- What statistical evaluation method?
  - *Relative frequency* :
    - n_c/n (n : matched by rule, n_c: classified by rule correctly)
  - *m-estimate of accuracy* :
    - $(n_c + mp) / (n + m)$
    - p : The prior probability that a randomly drawn example will have classification assigned by the rule (e.g. if 12 out of 100 examples have the value predicted by the rule, then p=0.12)
    - m : Weight (or # of examples for weighting this prior p)
  - *Entropy*

$$-Entropy(S) = \sum_{i=1}^{c} p_i \log_2 p_i$$

Learning First-Order Rules

- From now . . .
  - We consider learning rule that contain variables (first-order rules)
  - Inductive learning of first-order rules : inductive logic programming (ILP)
    - Can be viewed as automatically inferring Prolog programs
  - Two methods are considered
    - FOIL
    - Induction as inverted deduction

Learning First-Order Rules (cont.)

- First-order rule
  - Rules that contain variables
  - Example
    - $Ancestor\ (x, y) \leftarrow Parent\ (x, y).$
    - $Ancestor\ (x, y) \leftarrow Parent\ (x, z) \wedge Ancestor\ (z, y)$: recursive
  - More expressive than propositional rules
    - IF $(Father_1 = Bob) \wedge (Name_2 = Bob) \wedge (Female_1 = True)$, THEN $Daughter_{1,2} = True$
    - IF $Father(y,x) \wedge Female(y)$, THEN $Daughter(x,y)$

Learning First-Order Rules (cont.)

- Terminology
  - Constants: e.g., *John*, *Kansas*, 42
  - Variables: e.g., *Name*, *State*, *x*
  - Predicates: e.g., *Father-Of*, *Greater-Than*
  - Functions: e.g., *age*, *cosine*
  - Term: constant, variable, or *function*(*term*)
  - Literals (atoms): *Predicate*(*term*) or negation (e.g., ¬*Greater-Than*(*age*(*John*), 42))
  - Clause: disjunction of literals with implicit universal quantification
  - Horn clause: at most one positive literal

    $(H \vee \neg L_1 \vee \neg L_2 \vee \ldots \vee \neg L_n)$

Learning First-Order Rules (cont.)

- First-order Horn clauses
  - Rules that have one or more preconditions and one single consequent. Predicates may have variables

- The following Horn clauses are equivalent:
  - $H \vee \neg L_1 \vee \ldots \vee \neg L_n$
  - $H \leftarrow (L_1 \wedge \ldots \wedge L_n)$
  - IF $(L_1 \wedge \ldots \wedge L_n)$, THEN H

Learning Sets of First-Order Rules: FOIL

- First-Order Inductive Learning (FOIL)
- Natural extension of “Sequential covering + Learn-one-rule”
- FOIL rule: Similar to Horn clause with two exceptions
  - Syntactic restriction: No function symbols
  - More expressive than Horn clauses :

    Negation allowed in rule bodies

Learning Sets of First-Order Rules: FOIL (cont.)

- FOIL (*Target_predicate*, *Predicates*, *Examples*)

```
Pos ← those Examples for which the Target_predicate is True
Neg ← those Examples for which the Target_predicate is False
Learned_rules ← { }
while Pos, do
        Learn a NewRule
        NewRule ← the rule that predicts Target_predicate with no preconditions
        NewRuleNeg ← Neg
        while NewRuleNeg, do
            Add a new literal to specialize NewRule
                Candidate_literals ← generate candidate new literals for NewRule,
                                     based on Predicates
                Best_literal ← $\underset{L \in Candidate\_literals}{\operatorname{argmax}} Foil\_Gain(L, NewRule)$
                Add Best_literal to preconditions of NewRule
                NewRuleNeg ← subset of NewRuleNeg (satisfying NewRule preconditions)
        Learned-rules ← Learned_rules + NewRule
        Pos ← Pos – {members of Pos covered by NewRule}
return Learned_rules
```

Learning Sets of First-Order Rules: FOIL (cont.)

- FOIL learns rules when the target literal is true.
  - Note that sequential covering learns both rules when the target literal is either of true and false
- Outer loop
  - Add a new rule to its disjunctive hypothesis
  - Specific-to-General search
- Inner loop
  - Find a conjunction
  - General-to-Specific search on each rule by starting with a NULL precondition and adding more literal (hill-climbing)
  - Note that sequential covering performs a beam search.

Learning Sets of First-Order Rules: FOIL (cont.)

- Generating Candidate Specializations in FOIL
  - Generate new literals, each of which may be added to the rule preconditions.

  - Current Rule : $P(x_1, x_2, \ldots , x_k) \leftarrow L_1 ,\ldots, L_n$
    - Add new literal L_{n+1} to get more specific Horn clause
    - Form of literal
      - $Q(v_1, v_2, \ldots , v_k)$: Q in predicates and the v_i are either new variable or variable already present in the rule where at least one of v_i must already exist as a variable in the rule
      - $Equal(x_j, x_k)$: x_j and x_k are variables already present in the rule
      - Negation of above

Learning Sets of First-Order Rules: FOIL (cont.)

- Guiding the Search in FOIL
  - Consider all possible bindings (substitution) : prefer rules that possess more positive bindings
  - *Foil_Gain*(*L*, *R*)

$$Foil\_Gain(L,R) \equiv t\left(\log_2\left(\frac{p_1}{p_1+n_1}\right)-\log_2\left(\frac{p_0}{p_0+n_0}\right)\right)$$

  - $L \equiv$ candidate predicate to add to rule R
  - $p_0 \equiv$ number of positive bindings of R
  - $n_0 \equiv$ number of negative bindings of R
  - $p_1 \equiv$ number of positive bindings of $R + L$
  - $n_1 \equiv$ number of negative bindings of $R + L$
  - $t \equiv$ number of positive bindings of R also covered by $R + L$

  - Based on the numbers of positive and negative bindings covered before and after adding the new literal

Learning Sets of First-Order Rules: FOIL (cont.)

- Examples
  - Target literal : *GrandDaughter(x, y)*
  - Training Examples :

GrandDaughter(Victor, Sharon), Father(Sharon,Bob), Father(Tom, Bob), Female(Sharon), Father(Bob, Victor)

  - Initial step : *GrandDaughter(x, y)* ←
  - Positive binding : {*x/Victor, y/Sharon*}
  - Negative binding : others

Learning Sets of First-Order Rules: FOIL (cont.)

- Candidate additions to the rule preconditions :
  Equal(x,y), Female(x), Female(y), Father(x,y),
  Father(y,x), Father(x,z), Father(z,x), Father(y,z),
  Father(z,y) and the negations
- For each candidate, calculate *FOIL_Gain*
- If *Father(y, z)* has the maximum value of *FOIL_Gain*, select *Father(y, z)* to add precondition of rule
  GrandDaughter(x, y) ← *Father(y,z)*
- Iteration…
- We add the best candidate literal and continue adding literals until we generate a rule like the following:
  GrandDaughter(x,y) ← *Father(y,z)* ∧ *Father(z,x)* ∧ *Female(y)*
- At this point we remove all negative examples covered by the rule and begin the search for a new rule.

Learning Sets of First-Order Rules: FOIL (cont.)

- Learning recursive rules sets
  - Predicate occurs in rule head.
  - Example
    - $\mathit{Ancestor}\ (x, y) \leftarrow \mathit{Parent}\ (x, z) \wedge \mathit{Ancestor}\ (z, y)$.
    - Rule: IF $\mathit{Parent}\ (x, z) \wedge \mathit{Ancestor}\ (z, y)$ THEN $\mathit{Ancestor}\ (x, y)$
  - Learning recursive rule from relation
    - Given: Appropriate set of training examples
    - Can learn using FOIL-based search
      - Requirement: $\mathit{Ancestor} \in \mathit{Predicates}$
      - Recursive rules still have to outscore competing candidates at FOIL-Gain
    - How to ensure termination? (i.e. no infinite recursion)
      - [Quinlan, 1990; Cameron-Jones and Quinlan, 1993]

Induction as Inverted Deduction

- Induction: Inference from specific to general
- Deduction: Inference from general to specific
- Induction can be cast as a deduction problem
- $(\forall < x_i, f(x_i) > \in D)\ (B \wedge h \wedge x_i) \vdash f(x_i)$

  D : a set of training data

  B : background knowledge

  x_i : ith training instance

  $f(x_i)$: target value

  $X \vdash Y$: "*Y* follows deductively from *X*", or "*X* entails *Y*"

➔ For every training instance x_i, the target value $f(x_i)$ must follow deductively from *B*, *h*, and x_i

Induction as Inverted Deduction (cont.)

- Learn target : *Child(u,v)* : child of *u* is *v*
- Positive example : *Child(Bob, Sharon)*
- Given instance: *Male(Bob), Female(Sharon), Father(Sharon,Bob)*
- Background knowledge :
  - *Parent(u,v)* ← *Father(u,v)*
- Hypothesis satisfying the $(B \wedge h \wedge x_i) \vdash f(x_i)$
  - h_1 : *Child(u, v)* ← *Father(v, u)* : no need of *B*
  - h_2 : *Child(u, v)* ← *Parent(v, u)* : need *B*
- The role of background knowledge
  - Expanding the set of hypotheses
  - New predicates (*Parent*) can be introduced into hypotheses(h_2)

Induction as Inverted Deduction (cont.)

- In view of induction as the inverse of deduction
- Inverse entailment operators is required

$O(B, D) = h$

such that $(\forall \langle x_i, f(x_i) \rangle \in D)\ (B \wedge h \wedge x_i) \vdash f(x_i)$

- Input: Training data $D = \{\langle x_i, f(x_i) \rangle\}$ and background knowledge B
- Output: A hypothesis h

Induction as Inverted Deduction (cont.)

- Attractive features to formulating the learning task
  1. This formulation subsumes the common definition of learning (which has no background knowledge *B*)
  2. By incorporating the notion of *B*, this formulation allows a more rich definition of when a hypothesis is said to fit the data
  3. By incorporating *B*, this formulation invites learning methods that use this *B* to guide search for *h*

Induction as Inverted Deduction (cont.)

- Practical difficulties in this formulation
  1. The requirement of the formulation does not naturally accommodate noisy training data.
  2. The language of first-order logic is so expressive, and the number of hypotheses that satisfy the formulation is so large.
  3. In most ILP system, the complexity of the hypothesis space search increases as *B* is increased.

Inverting Resolution

- Resolution rule

$$\frac{\begin{array}{c} P \vee L \\ \neg L \vee R \end{array}}{P \vee R}$$ (L: literal P,R : clause)

- Resolution Operator (propositional form)
  - Given initial clauses C_1 and C_2, find a literal L from clause C_1 such that $\neg L$ occurs in clause C_2.
  - Form the resolvent C by including all literal from C_1 and C_2, except for L and $\neg L$. More precisely, the set of literals occurring in the conclusion C is

$$C = (C_1 - \{L\}) \cup (C_2 - \{\neg L\})$$

Inverting Resolution (cont.)

- Example 1

$$\begin{tikzcd}
 & & C_2\text{: } \mathit{KnowMaterial} \lor \neg \mathit{Study} \arrow[dl] \\
C_1\text{: } \mathit{PassExam} \lor \neg \mathit{KnowMaterial} \arrow[r] & \boxed{C\text{: } \mathit{PassExam} \lor \neg \mathit{Study}} &
\end{tikzcd}$$

- Example 2
  - C_1: $A \lor B \lor C \lor \neg D$
  - C_2: $\neg B \lor E \lor F$

  $\Rightarrow C : A \lor C \lor \neg D \lor E \lor F$

Inverting Resolution (cont.)

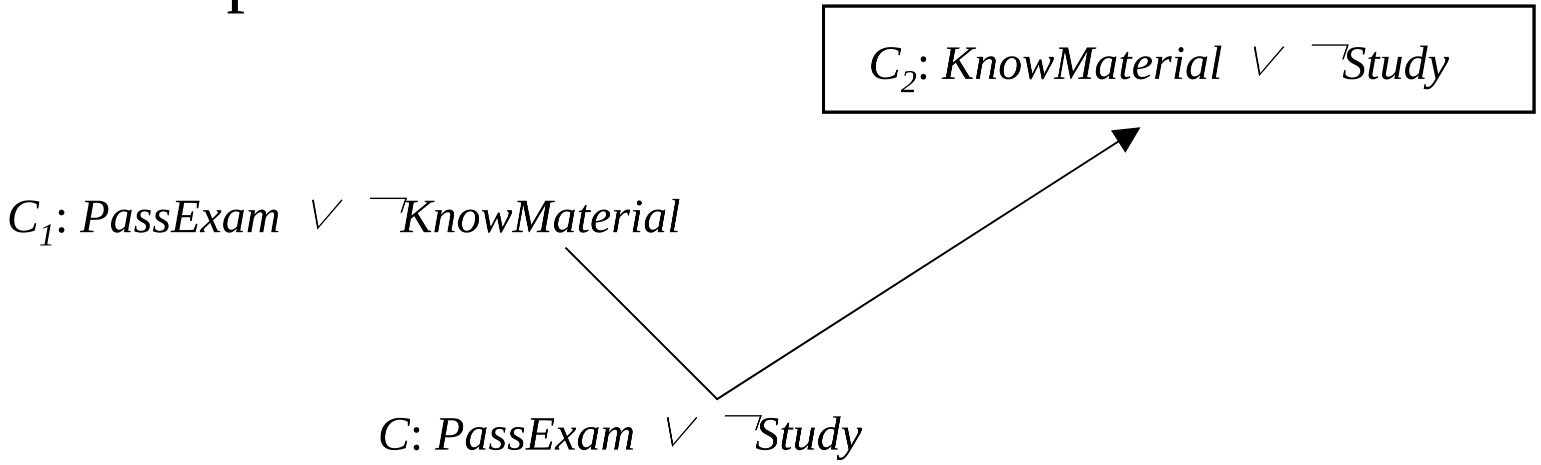
- $O(C, C_1)$
  - Perform inductive inference

- Inverse Resolution Operator (propositional form)
  - Given initial clauses C_1 and C, find a literal L that occurs in clause C_1, but not in Clause C.
  - From the second clause C_2 by including the following literals

$$C_2 = (C - (C_1 - \{L\})) \cup \{\neg L\}$$

Inverting Resolution (cont.)

- Example 1


C_2: *KnowMaterial* $\vee$ $\neg$*Study*
C_1: *PassExam* $\vee$ $\neg$*KnowMaterial*
C: *PassExam* $\vee$ $\neg$*Study*

- Example 2
  - $C_1: B \vee D$, $C: A \vee B$

  $\Rightarrow C_2: A \vee \neg D$ (if $C_2: A \vee \neg D \vee B$??)
  - Inverse resolution is nondeterministic
    - One heuristic for choosing among alternatives: Shorter clauses over longer clauses are preferred.

Inverting Resolution (cont.)

- First-order resolution
  - Substitution
    - Mapping of variables to terms
    - Ex) $\theta = \{x/Bob, z/y\}$
  - Unifying Substitution
    - For two literal L_1 and L_2, provided $L_1\theta = L_2\theta$
    - Ex) $\theta = \{x/Bill, z/y\}$

      $L_1 = Father(x, y), L_2 = Father(Bill, z)$

      $L_1\theta = L_2\theta = Father(Bill, y)$

Inverting Resolution (cont.)

- First-order resolution
  - Resolution Operator (first-order form)
    - Find a literal L_1 from clause C_1, literal L_2 from clause C_2, and substitution θ such that $L_1\theta = \neg L_2\theta$.
    - From the resolvent C by including all literals from $C_1\theta$ and $C_2\theta$, except for $L_1\theta$ and $\neg L_2\theta$. More precisely, the set of literals occurring in the conclusion C is

$$C = (C_1 - \{L_1\})\theta \cup (C_2 - \{L_2\})\theta$$

Inverting Resolution (cont.)

- Example
  - $C_1 = White(x) \leftarrow Swan(x)$, $C_2 = Swan(Fred)$
  - $C_1 = White(x) \vee \neg Swan(x)$,
  - $L_1 = \neg Swan(x)$, $L_2 = Swan(Fred)$

  unifying substitution $\theta = \{x/Fred\}$

  then $L_1\theta = \neg L_2\theta = \neg Swan(Fred)$

  $(C_1 - \{L_1\})\theta = White(Fred)$

  $(C_2 - \{L_2\})\theta = \emptyset$

  $\therefore \; C = White(Fred)$

Inverting Resolution (cont.)

- Inverse resolution : First-order case

$C=(C_1-\{L_1\})\theta_1 \cup (C_2-\{L_2\})\theta_2$

$\quad$ (where, $\theta = \theta_1\theta_2$ (factorization))

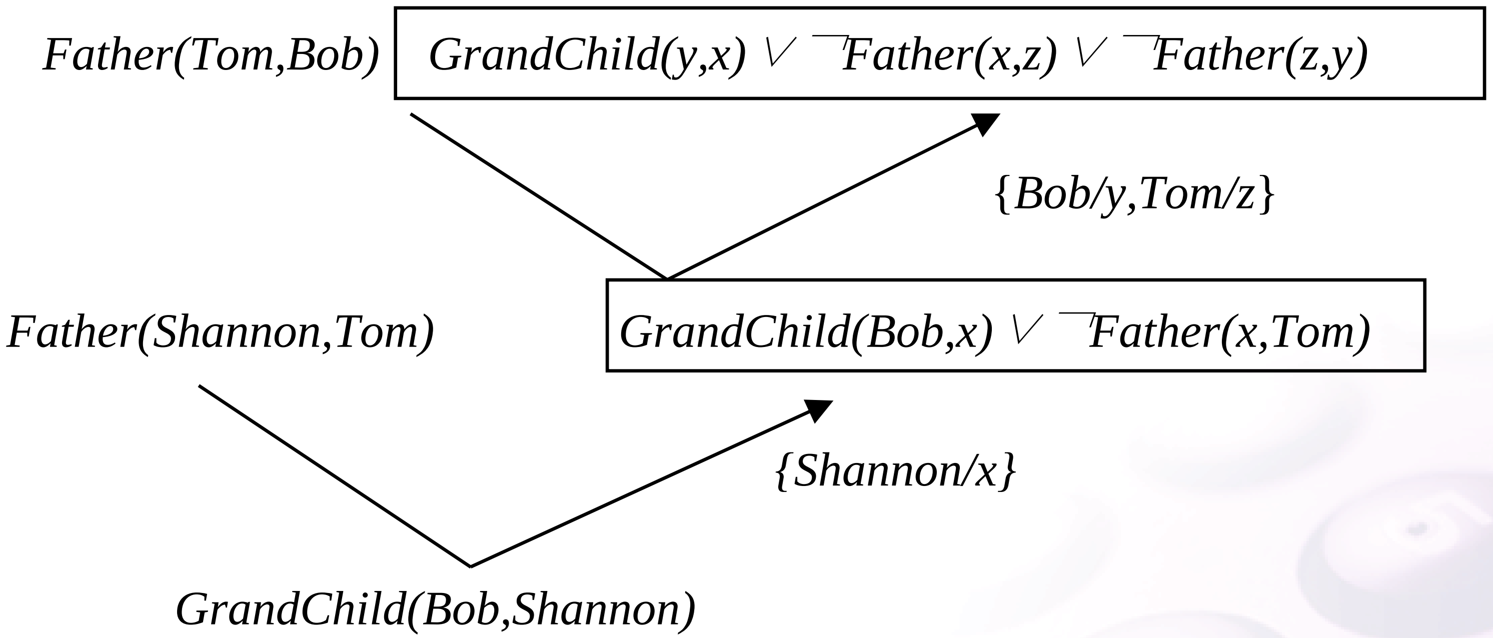
$C - (C_1-\{L_1\})\theta_1 = (C_2-\{L_2\})\theta_2$

$\quad$ (where, $L_2 = \neg L_1\theta_1\theta_2^{-1}$)

$\therefore C_2=(C-(C_1-\{L_1\})\theta_1)\theta_2^{-1} \cup \{\neg L_1\theta_1\theta_2^{-1}\}$

Inverting Resolution (cont.)

- Inverse resolution : First-order case
  - Multistep inverse resolution



Inverting Resolution (cont.)

- Inverse resolution : First-order case
  - $C = GrandChild(Bob,Shannon)$

    $C_1 = Father(Shannon,Tom)$

    $L_1 = Father(Shannon,Tom)$

    Suppose we choose inverse substitution

    $\theta_1^{-1} = \{\}, \ \theta_2^{-1} = \{Shannon/x)$

    $(C-(C_1-\{L_1\})\theta_1)\theta_2^{-1} = (C\theta_1)\theta_2^{-1} = GrandChild(Bob,x)$

    $\{\neg L_1\theta_1\theta_2^{-1}\} = \neg Father(x,Tom)$

  $\therefore C_2 = GrandChild(Bob,x) \vee \neg Father(x,Tom)$

  or equivalently $GrandChild(Bob,x) \leftarrow \neg Father(x,Tom)$

Summary

- Learning rules from data
- Sequential covering algorithms
  - Learning single rules by search
    - Beam search
    - Alternative covering methods
  - Learning rule sets
- First-order rules
  - Learning single first-order rules
    - Representation: First-order Horn clauses
    - Extending Sequential-Covering and Learn-One-Rule: Variables in rule preconditions

Summary (cont.)

- FOIL: Learning first-order rule sets
  - Idea: Inducing logical rules from observed relations
  - Guiding search in FOIL
  - Learning recursive rule sets
- Induction as inverted deduction
  - Idea: Inducing logical rule as inverted deduction
  - $O(B, D) = h$
    - such that $(\forall \langle x_i, f(x_i) \rangle \in D)\ (B \wedge h \wedge x_i) \vdash f(x_i)$
  - Generate only hypotheses satisfying the constraint, $(B \wedge h \wedge x_i) \vdash f(x_i)$
    - Cf. FOIL : Generates many hypotheses at each search step based on syntax, including those not satisfying this constraint
  - Inverse resolution operator can consider only a small fraction of the available data
    - Cf. FOIL : Consider all available data