

Lecture 6: Inductive Logic Programming

Cognitive Systems II - Machine Learning

Part II: Special Aspects of Concept Learning

FOIL, Inverted Resolution, Sequential Covering

last change November 27, 2008

Motivation

- it is useful to learn the target function as a set of if-then-rules
 - one of the most expressive and human readable representations
 - e.g. decision trees
 - **Inductive Logic Programming (ILP):**
 - rules are learned directly
 - designed to learn *first-order rules* (i.e. including variables)
 - *sequential covering* to incrementally grow the final set of rules
 - PROLOG programs are sets of first-order rules
- ⇒ a general-purpose method capable of learning such rule sets may be viewed as an algorithm for automatically inferring PROLOG programs

Examples

● Propositional Logic:

IF Humidity=normal AND Outlook=sunny

THEN PlayTennis=yes

IF Humidity=normal AND Temperature=mild AND Wind=weak

THEN PlayTennis=yes

```
playTennis :- humidity(normal), outlook(sunny).
```

```
playTennis :- humidity(normal), temperature(mild),  
              wind(weak).
```

● First Order Logic:

IF Parent(x,y) THEN Ancestor(x,y)

IF Parent(x,z) AND Ancestor(z,y) THEN Ancestor(x,y)

```
ancestor(X,Y) :- parent(X,Y).
```

```
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).
```

Sequential Covering

- **basic strategy:** learn one rule, remove the data it covers, then iterate this process
- one of the most widespread approaches to learn a disjunctive set of rules (each rule itself is conjunctive)
- subroutine **LEARN-ONE-RULE**
 - accepts a set of positive and negative examples as input and outputs a single rule that covers many of the positive and few of the negative examples
 - **high accuracy:** predictions should be correct
 - **low coverage:** not necessarily predictions for each example
- performs a greedy search without backtracking
 - ⇒ no guarantee to find the smallest or best set of rules

Sequential Covering Algorithm

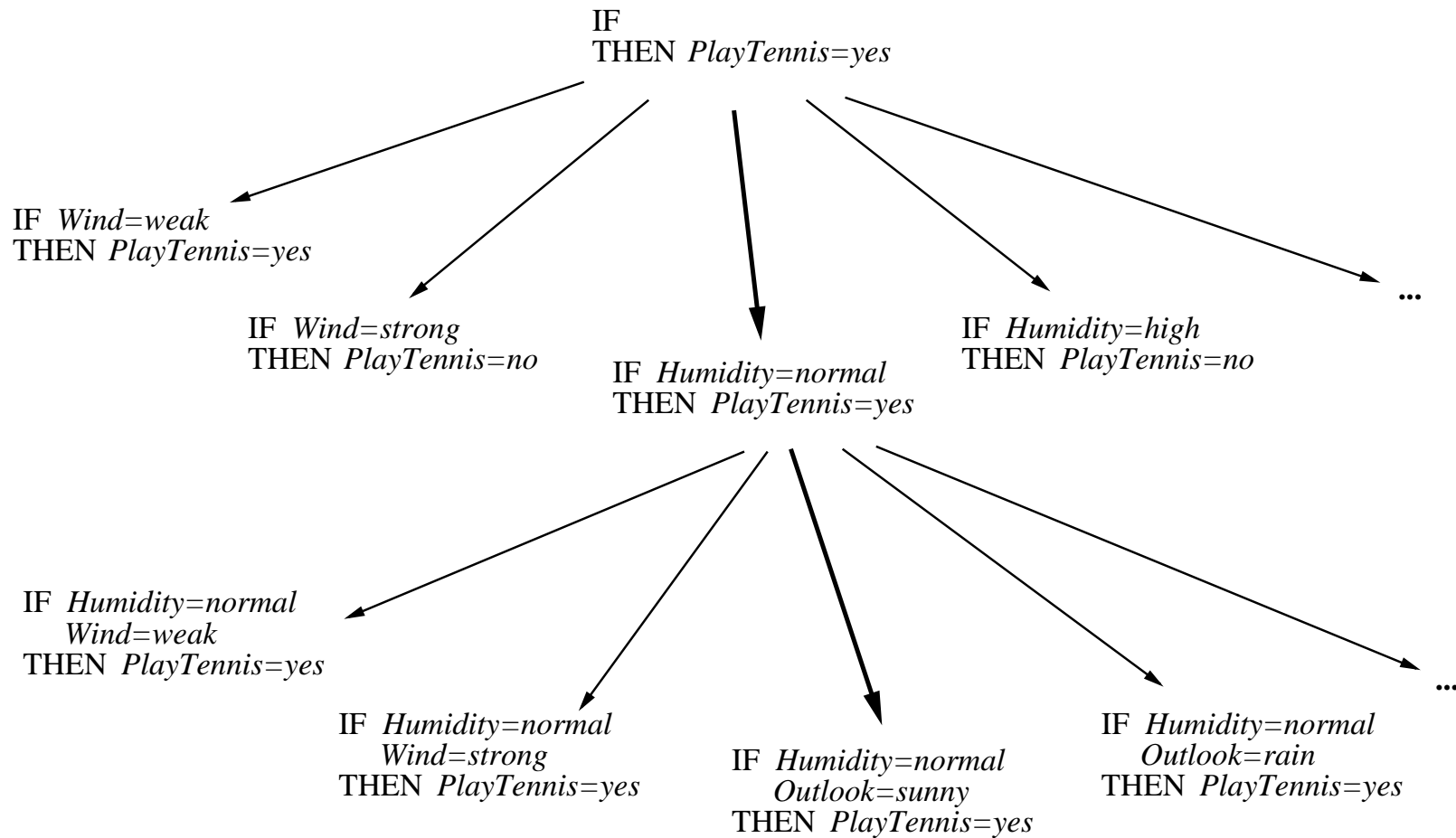
SEQUENTIAL-COVERING(*Target_attribute*, *Attributes*, *Examples*, *Threshold*)

- $Learned_Rules \leftarrow \{\}$
- $Rule \leftarrow \mathbf{LEARN-ONE-RULE}(Target_attribute, Attributes, Examples)$
- While **PERFORMANCE**(*Rule*, *Examples*) > *Threshold*, Do
 - $Learned_rules \leftarrow Learned_rules + Rule$
 - $Examples \leftarrow Examples - \{ \text{examples correctly classified by } Rule \}$
 - $Rule \leftarrow \mathbf{LEARN-ONE-RULE}(Target_attribute, Attributes, Examples)$
- $Learned_rules \leftarrow \text{sort } Learned_rules \text{ accord to } \mathbf{PERFORMANCE} \text{ over } Examples$
- return $Learned_rules$

General to Specific Beam Search

- **question:** How shall LEARN-ONE-RULE be designed to meet the needs of the sequential covering algorithm?
- organize the search through H analogous to ID3
 - **but** follow only the most promising branch in the tree at each step
 - begin by considering the most general rule precondition (i.e. empty test)
 - then greedily add the attribute test that most improves rule performance over the training examples
 - unlike to ID3, this implementation follows only a single descendant at each search step rather than growing a subtree that covers all possible values of the selected attribute

General to Specific Beam Search



General to Specific Beam Search

- so far a local greedy search (analoguous to hill-climbing) is employed
 - danger of suboptimal results
 - susceptible to the typical hill-climbing problems
- ⇒ extension to **beam search**
 - algorithm maintains a list of the k best candidates at each step
 - at each step, descendants are generated for each of the k candidates and the resulting set is again reduced to the k most promising candidates

LEARN-ONE-RULE

LEARN-ONE-RULE(*Target_attribute*, *Attributes*, *Examples*, *k*)

Returns a single rule that covers some of the *Examples*. Conducts a general to specific greedy beam search for the best rule, guided by the *PERFORMANCE* metric.

- Initialize *Best_hypothesis* to the most general hypothesis \emptyset
- Initialize *Candidate_hypotheses* to the set $\{Best_hypothesis\}$
- While *Candidate_hypotheses* is not empty, Do
 - 1. Generate the next more specific *candidate_hypotheses*
 - *New_Candidate_hypotheses* \leftarrow new generated and specialized candidates
 - 2. Update *Best_hypotheses*
 - Select hypothesis *h* from *New_candidate_hypotheses* with best *PERFORMANCE* over *Examples*
 - IF *PERFORMANCE* of *h* > *PERFORMANCE* of *Best_hypothesis* THEN set *h* as new *Best_hypothesis*
 - 3. Update *Candidate_hypotheses*
 - *Candidate_hypotheses* \leftarrow the *k* best members of *New_Candidate_hypotheses*

LEARN-ONE-RULE cont.

- Return a rule of the form
“IF *Best_hypothesis* THEN *prediction*”
where *prediction* is the most frequent value of *Target_attribute* among those *Examples* that match *Best_hypothesis*.

Example: Learn One Rule

- Learn *one* rule that covers a certain amount of positive examples
- with high accuracy
- remove the covered positive examples

Example:

(s1) Sky = sunny

(s2) Sky = rainy

(a1) AirTemp = warm

(a2) AirTemp = cold

(h1) Humidity = normal

(h2) Humidity = high

(w1) Water = warm

(w2) Water = cool

Example cont.

- Assume $k = 4$
- Current most specific hypotheses: $s1, s2, a1, a2, h1, h2, w1, w2$
- Assume best possible hypotheses wrt performance P : $s1, a2, h1, w1$
- Generate new candidate hypotheses, e.g. by specializing $s1$:

$s1$ & $s1$ (duplicate)

$s1$ & $s2$ (inconsistent)

$s1$ & $a1$

$s1$ & $a2$

$s1$ & $h1$

Performance Measures

- Relative Frequency (numbers of correctly classified examples by all examples)

$$\frac{n_c}{n}$$

- Entropy (S as set of examples that match precondition, p_i proportion of examples from S for which the target function takes the i -th value)

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

Sequential vs. Simultaneous Covering

● sequential covering:

- learn just one rule at a time, remove the covered examples and repeat the process on the remaining examples
- many search steps, making independent decisions to select each precondition for each rule

● simultaneous covering:

- ID3 learns the entire set of disjunctive rules simultaneously as part of a single search for a decision tree
- fewer search steps, because each choice influences the preconditions of all rules

⇒ Choice depends of how much data is available

- plentiful → sequential covering (more steps supported)
- scarce → simultaneous covering (decision sharing effective)

Differences in Search

● **generate-then-test:**

- search through all syntactically legal hypotheses
 - generation of the successor hypotheses is only based on the syntax of the hypothesis representation
 - training data is considered after generation to choose among the candidate hypotheses
 - each training example is considered several times
- ⇒ impact of noisy data is minimized

● **example driven:**

- individual training examples constrain the generation of hypotheses
 - e.g. FIND-S, CANDIDATE ELIMINATION
- ⇒ search can easily be misled

Learning First-Order Rules

- propositional expressions do not contain variables and are therefore less expressive than first-order expressions
- no general way to describe essential relations among the values of attributes
- now we consider learning first-order rules (Horn Theories)
 - a Horn clause is a clause containing at most one positive literal
 - expression of the form:
$$H \vee \neg L_1 \vee \neg L_2 \vee \dots \vee \neg L_n$$
$$\iff H \leftarrow (L_1 \wedge L_2 \wedge \dots \wedge L_n)$$
$$\iff \text{IF } (L_1 \wedge L_2 \wedge \dots \wedge L_n) \text{ THEN } H$$
 - FOL terminology see CogSysI

FOIL (Quinlan, 1990)

- natural extension of SEQUENTIAL-COVERING and LEARN-ONE-RULE
- outputs sets of first-order rules similar to Horn Clauses with two exceptions
 1. **more restricted**, because literals are not permitted to contain function symbols
 2. **more expressive**, because literals in the body can be negated
- differences between FOIL and earlier algorithms:
 - seeks only rules that predict when the target literal is *True*
 - conducts a simple hill-climbing search instead of beam search

FOIL algorithm

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

- $Pos \leftarrow$ those *Examples* for which the *Target_predicate* is *True*
- $Neg \leftarrow$ those *Examples* for which the *Target_predicate* is *False*
- $Learned_rules \leftarrow \{\}$
- while *Pos*, Do
 - $NewRule \leftarrow$ the rule that predicts *Target_predicate* with no precondition
 - $NewRuleNeg \leftarrow Neg$
 - while *NewRuleNeg*, Do
 - $Candidate_literals \leftarrow$ generate new literals for *NewRule*, based on *Predicates*
 - $Best_literal \leftarrow \max_{L \in Candidate_literals} FoilGain(L, NewRule)$
 - add *Best_literal* to preconditions of *NewRule*
 - $NewRuleNeg \leftarrow$ subset of *NewRuleNeg* that satisfies *NewRule* preconditions
 - $Learned_rules \leftarrow Learned_rules + NewRule$
 - $Pos \leftarrow Pos - \{ \text{members of } Pos \text{ covered by } NewRule \}$
- Return *Learned_rules*

FOIL Hypothesis Space

● **outer loop (set of rules):**

- specific-to-general search
- initially, there are no rules, so that each example will be classified negative (most specific)
- each new rule raises the number of examples classified as positive (more general)
- disjunctive connection of rules

● **inner loop (preconditions for one rule):**

- general-to-specific search
- initially, there are no preconditions, so that each example satisfies the rule (most general)
- each new precondition raises the number of examples classified as negative (more specific)
- conjunctive connection of preconditions

Generating Candidate Specializations

- current rule:

$P(x_1, x_2, \dots, x_k) \leftarrow L_1 \dots L_n$ where

$L_1 \dots L_n$ are the preconditions and

$P(x_1, x_2, \dots, x_k)$ is the head of the rule

- FOIL generates candidate specializations by considering new literals L_{n+1} that fit one of the following forms:
 - $Q(v_1, \dots, v_r)$ where $Q \in \text{Predicates}$ and the v_i are new or already present variables (at least one v_i must already be present)
 - $Equal(x_j, x_k)$ where x_j and x_k are already present in the rule
 - the negation of either of the above forms

Training Data Example

Examples

GrandDaughter(Victor, Sharon)

→ GrandDaughter(Tom, Bob)

→ GrandDaughter(Victor, Victor)

Background Knowledge

Female(Sharon)

Father(Sharon, Bob)

Father(Tom, Bob)

Father(Bob, Victor)

FOIL Example

GrandDaughter(x,y) ←

Candidate additions to precondition:

*Equal(x,y), Female(x), Female(y), Father(x,y), Father(y,x),
Father(x,z), Father(z,x), Father(z,y), and the negations to
these literals*

Assume greedy selection of **Father(y,z)**:

GrandDaughter(x,y) ← Father(y,z)

Candidate additions:

*the ones from above and Female(z), Equal(z,y),
Father(z,w), Father(w,z), and their negations*

Learned Rule:

GrandDaughter(x,y) ← Father(y,z) ∧ Father(z,x) ∧ Female(y)

FOIL Gain

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

with

- L as new literal introduced in rule R to gain new rule R'
- t as number of positive bindings of rule R which are still covered by R'
- p_1 as number of positive bindings of rule R' and n_1 as number of negative bindings
- p_0 as number of positive bindings of rule R and n_0 as number of negative bindings

Remark: Bindings are the number of instantiations of the variables by constants. A binding is positive if the instantiated rule covers a positive example.

Learning Recursive Rule Sets

- Extend FOIL such that the target predicate can also be included in the preconditions with the same restrictions to variables as before.
- Problem: rule sets that produce infinite recursions
- FOIL uses a generate-and-test strategy alternatively recursive rule sets can be learned by analytical methods (see lecture inductive programming)

```
ancestor(X,Y) :- parent(X,Y).
```

```
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).
```


Induction as Inverted Deduction

- **observation:** induction is just the inverse of deduction
- in general, machine learning involves building theories that explain the observed data
- Given some data D and some background knowledge B , learning can be described as generating a hypothesis h that, together with B , explains D .

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

- the above equation casts the learning problem in the framework of deductive inference and formal logic

Induction as Inverted Deduction

● features of inverted deduction:

- subsumes the common definition of learning as finding some general concept
- background knowledge allows a more rich definition of when a hypothesis h is said to “fit” the data

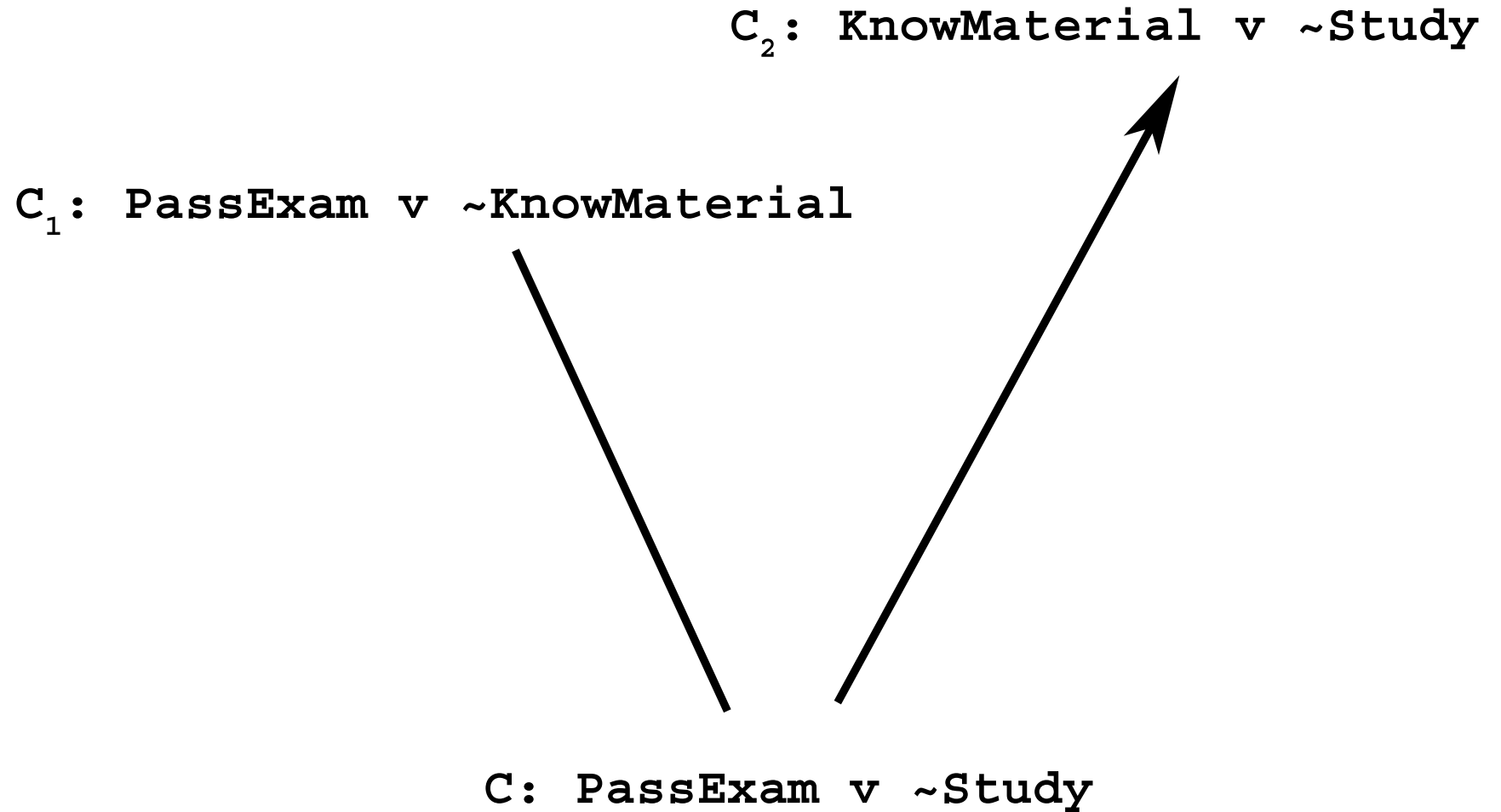
● practical difficulties:

- noisy data makes the logical framework completely lose the ability to distinguish between truth and falsehood
- search is intractable
- background knowledge often increases the complexity of H

Inverting Resolution

- **resolution** is a general method for automated deduction
- complete and sound method for deductive inference
- see CogSys1
- **Inverse Resolution Operator (propositional form):**
 1. Given initial clause C_1 and C , find a literal L that occurs in C_1 but not in clause C .
 2. Form the second clause C_2 by including the following literals
$$C_2 = (C - (C_1 - \{L\})) \cup \{\neg L\}$$
- inverse resolution is not deterministic

Inverting Resolution



Inverting Resolution

● Inverse Resolution Operator (first-order form):

● resolution rule:

1. Find a literal L_1 from clause C_1 , literal L_2 from clause C_2 , and substitution θ such that $L_1\theta = \neg L_2\theta$
2. Form the resolvent C by including all literals from $C_1\theta$ and $C_2\theta$, except for $L_1\theta$ and $\neg L_2\theta$. That is,

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

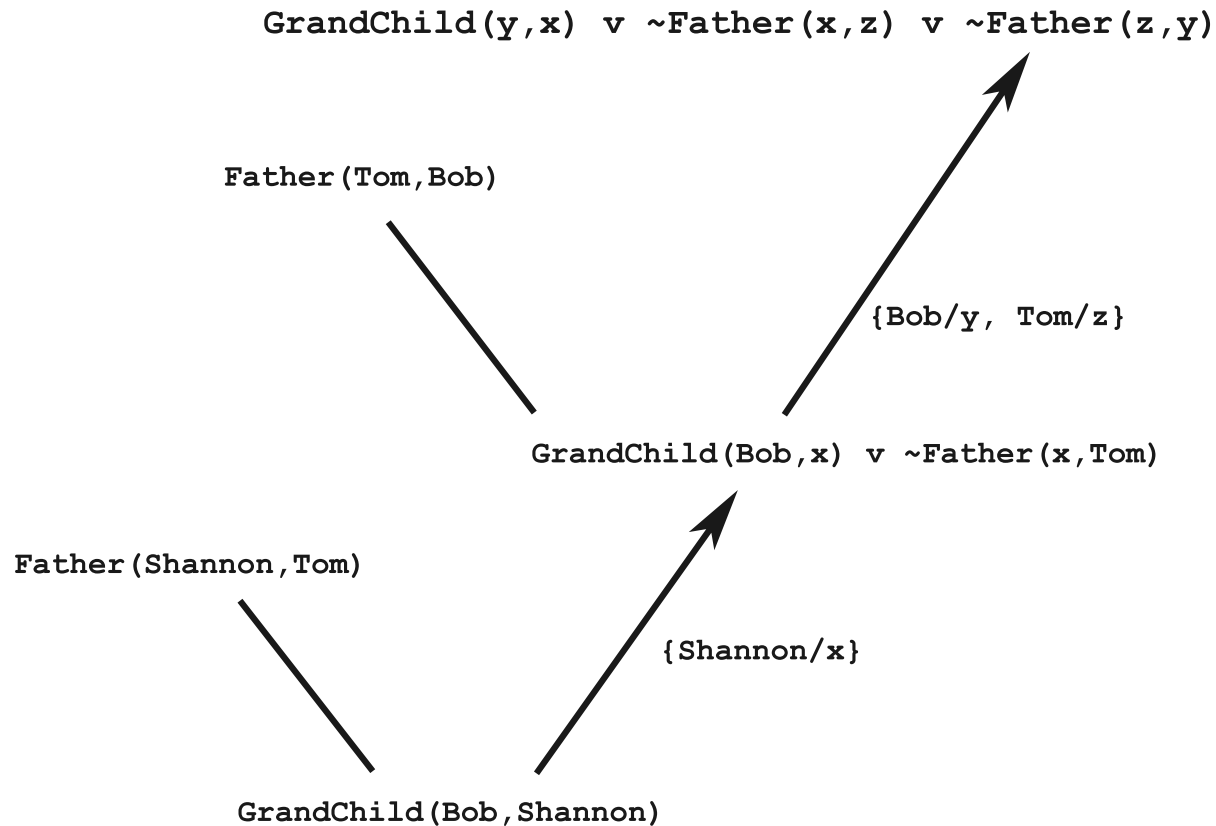
● analytical derivation of the inverse resolution rule:

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

$$C - (C_1 - \{L_1\})\theta_1 = (C_2 - \{L_2\})\theta_2 \text{ where } L_2 = \neg L_1\theta_1\theta_2^{-1}$$

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

Inverting Resolution



$$D = \{GrandChild(Bob, Shannon)\}$$

$$B = \{Father(Shannon, Tom), Father(Tom, Bob)\}$$

PROGOL

- Inverse resolution leads to combinatorial explosion of candidate hypotheses
- Alternative: use inverse entailment to generate just a single more specific hypothesis that entails the observed data

Generalization, θ -Subsumption, Entailment

- interesting to consider the relationship between the *more_general_than* relation and inverse entailment
- *more_general_than*: $h_j \geq_g h_k$ iff $(\forall x \in X)[h_k(x) \rightarrow h_j(x)]$. A hypothesis can also be expressed as $c(x) \leftarrow h(x)$.
- θ – *subsumption*: Consider two clauses C_j and C_k , both of the form $H \vee L_1 \vee \dots \vee L_n$, where H is a positive literal and the L_i are arbitrary literals. Clause C_j is said to θ – *subsume* clause C_k iff $(\exists \theta)[C_j\theta \subseteq C_k]$.
- *Entailment*: Consider two clauses C_j and C_k . Clause C_j is said to *entail* clause C_k (written $C_j \vdash C_k$) iff C_k follows deductively from C_j .

Generalization, θ -Subsumption, Entailment

- if $h_1 \geq_g h_2$ then $C_1 : c(x) \leftarrow h_1(x)$ θ -subsumes $C_2 : c(x) \leftarrow h_2(x)$
- furthermore, θ -subsumption can hold even when the clauses have different heads

$A : Mother(x, y) \leftarrow Father(x, z) \wedge Spouse(z, y)$

$B : Mother(x, L) \leftarrow Father(x, B) \wedge Spouse(B, y) \wedge Female(x)$

$A \theta \subseteq B$ if we choose $\theta = \{y/L, z/B\}$

- θ -subsumption is a special case of entailment

$A : Elephant(father_of(x)) \leftarrow Elephant(x)$

$B : Elephant(father_of(father_of(y))) \leftarrow Elephant(y)$

$A \vdash B$, but $\neg \exists \theta [A \theta \subseteq B]$

Generalization, θ -Subsumption, Entailment

- Generalization is a special case of θ -Subsumption
- θ -Subsumption is a special case of entailment
- In its most general form, inverse entailment produces intractable searches
- θ -Subsumption provides a convenient notion that lies midway between generalization and entailment!

Summary

- learns sets of first-order rules directly
- sequential covering algorithms learn just one rule at a time and perform many search steps
- hence, applicable if data is plentiful
- **FOIL** is a sequential covering algorithm
- a specific-to-general search is performed to form the result set
- a general-to-specific search is performed to form each new rule
- Induction can be viewed as the inverse of deduction
- hence, an inverse resolution operator can be found

Summary cont.

- Hypothesis language of Horn Clauses is more expressive than feature vectors (allowing variables, representing relations)
- Suitable for structured data (meshes, chemical structures, graph-representations in general)
- Not only for learning classifiers but also for learning general (recursive) programs (inductive programming)