Artificial Intelligence Chapter 17.

Knowledge-Based Systems

Outline

- Confronting the Real World
- Reasoning Using Horn Clauses
- Maintenance in Dynamic Knowledge Bases
- Rule-Based Expert Systems
- Rule Learning
- Additional Readings and Discussion

17.1 Confronting the Real World

- Knowledge-based systems
  - Programs that reason over extensive knowledge bases.
  - Do the methods scale up sufficiently well for practical applications?
- Three major theoretical properties of logical reasoning systems
  - Soundness
    - To be confident that an inferred conclusion is true
  - Completeness
    - To be confident that inference will eventually produce any true conclusion
  - Tractability
    - To be confident that inference is feasible

17.1 Confronting the Real World

- Predicate calculus
  - Resolution refutation is sound and complete, but semi-decidable.
    - Semi-decidability makes the predicate calculus inherently intractable.
    - Even on problems for which resolution refutation terminates, the procedure is NP-hard.
    - This fact has led many to despair of using formal, logical methods for large-scale reasoning problems.
17.1 Confronting the Real World

- To make reasoning more efficient
  - We could use procedures that might occasionally prove an untrue formula.
  - We could use procedures that might not be guaranteed to find proofs of true formulas.
  - We could use a language that is less expressive than the full predicate calculus.
    - Reasoning is typically more efficient with Horn clauses.

17.2 Reasoning Using Horn Clauses

- Horn Clauses
  - Clauses that have at most one positive literal.
  - Rule: \( \lambda_0 : -\lambda_1, \ldots, \lambda_n \)
    - There is at least one negative literal and a single positive literal.
    - The literal \( \lambda_0 \) is called the head of the clause.
    - The ordered list of literals, \( \lambda_1, \ldots, \lambda_n \), is called body.
  - Fact: \( \lambda_0 : - \)
    - There may be no negative literals in the clause.
  - Goal: \( -\lambda_1, \ldots, \lambda_n \)
    - There may be no positive literal in the clause.
  - Horn clauses form the basis of the programming language PROLOG.

Inference over PROLOG Clauses (1/2)

- Resolution Operation
  - A goal can resolve with a fact by unifying the fact with one of the literals in the goal.
    - The resolvent is a new goal consisting of a list of all of the substitution instances of the other literals in the original goal.
  - A goal can resolve with a rule by unifying the head of the rule with one of the literals in the goal.
    - The resolvent is a new goal formed by appending the list of substitution instances of all of the literals in the body of the rule to the front of the list of substitution instances of all of the other (nonresolved-upon) literals in the goal.

Inference over PROLOG Clauses (2/2)

- PROLOG programs
  - The clauses are usually ordered in the following way: goal, facts, rules.
  - Proof of a goal clause succeeds when the new goal produced by a resolution is empty.
  - Depth-first, backtracking search procedure
    - A goal clause fails if the interpreter has tried all resolutions for one of the goal literals and none results in new goals that can be proved.
    - The interpreter backtracks to the previous goal clause and tries other resolutions on it.
Example of an AND/OR Tree (1)

- Body nodes are called AND nodes because they must all be proved.
- If there had been alternative resolutions possible, the search for a proof tree could have generated additional nodes, called OR nodes.

1. :- MOVES
2. BAT_OK :-
3. LIFTABLE :-
4. MOVES :- BAT_OK, LIFTABLE

Example of an AND/OR Tree (2)

- Consistency requires that the same term be substituted for the variable throughout the proof tree.

1. :- Above (A, C)
2. On (A, B) :-
3. On (B, C) :-
4. Above (x, y) :- On (x, y)
5. Above (x, y) :- On (x, z), Above (z, y)

Forward Chaining

- Forward chaining system (e.g., OPS5)
  - Reasoning proceeds forward, beginning with facts, chaining through rules, and finally establishing the goal.
    - A rule is applicable if each of the literals in its antecedent can be unified with a corresponding fact.
    - When more than one rule is applicable, some sort of external conflict resolution scheme is used to decide which rule will be applied.
  - Model of aspects of human cognitive processing
    - Facts: working of short-term memory
    - Rules: long-term memory

17.3 Maintenance in Dynamic Knowledge Bases

- Knowledge base of propositional calculus atoms
  - The atoms are called premisses instead of facts.
  - The rules are not restricted to being Horn rules.
  - A spreadsheet reasoning system
    - Any cell that has a formula in it gets its value immediately calculated from the values of components.
    - If the values of any components of a formula happen to change, the value of that formula is automatically changed.

\[
\begin{array}{cccc}
P & Q & R & S \\
1 & 1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
P & Q & R & S \\
1 & 0 & 0 & 1 \\
\end{array}
\]
17.3 Maintenance in Dynamic Knowledge Bases

- Truth maintenance systems (TMSs)
  - *Dynamic knowledge base*: the spreadsheet is extended to contain any number of premises and rules.
  - The formulas entered in the values of rule cells are called justifications.
  - The values of certain premiss cells can be omitted.
  - When a cell has a value of 1 or 0, it is called IN; otherwise, OUT.
  - Allowing values of cells to be OUT permits an interesting generalization of rule types (e.g., U if R is True and S is Out).

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Premises of the KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>P ( \land \neg Q )</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

- Assumption-based truth maintenance systems (ATMSs)
  - The spreadsheet is used in a backward direction.
  - We start with a particular cell’s formula and ask which premiss values will make the formula in that cell *True*.
  - Other formulas called the *background theory* are considered together.
  - *Labels* of the cells
    - The values of the cells are taken to be formulas-expressed in DNF form in terms of the premiss atoms.
  - ATMSs can be used for a variety of purposes.
    - One is to perform diagnosis.

- Conversion of a TMS to an ATMS

<table>
<thead>
<tr>
<th>Premises</th>
</tr>
</thead>
<tbody>
<tr>
<td>P ( \lor Q )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \neg P \lor Q )</td>
</tr>
</tbody>
</table>

Background theory: \( P \land \neg Q \land R \land \neg S \land W \land U \)

Even if this book is used, students can get through it without advanced knowledge in logical reasoning. For example, a student can use a spreadsheet program to make a truth table and then use the spreadsheet program to perform dynamic knowledge maintenance. This is what is called a truth maintenance system (TMS).

- TMSS should be thought of as knowledge base maintenance systems that perform automatic updates when the truth values of certain concepts are changed.
  - The use of IN’s and OUT’s permits an elementary sort of “monotonic” or defeasible inference not sanctioned by ordinary reasoning systems.
  - Cycles among the rules present some difficulties.
    - One is mutual justification.

Stages in calculation when \( P \) becomes OUT:

Stage 1: calculation of new \( Q \), using its formula and previous value of \( R \):

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Stage 2: calculation of new \( R \), given previous values of \( P \) and \( Q \):

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
17.4 Rule-Based Expert Systems

- **Expert System**
  - One of the most successful applications of AI reasoning technique using facts and rules

- **Definition of Expert System (or knowledge-based system)**
  - "AI Programs that achieve expert-level competence in solving problems by bringing to bear a body of knowledge [Feigenbaum, McCorduck & Nii 1988]"

- **Base Structure of Expert System**
  - Knowledge Base
    - Consists of predicate-calculus facts and rules about subject at hand.
  - Inference Engine
    - Consists of all the processes that manipulate the knowledge base to deduce information requested by the user.

---

17.4 Rule-Based Expert Systems (2)

- **Explanation subsystem**
  - Analyzes the structure of the reasoning performed by the system and explains it to the user.

- **User interface**
  - Consists of some kind of natural language processing system or graphical user interfaces with menus.

- **Knowledge acquisition subsystem**
  - Checks the growing knowledge base for possible inconsistencies and incomplete information.

---

17.4 Rule-Based Expert Systems (3)

- "Knowledge engineer"
  - Usually a computer scientist with AI training.
  - Works with an expert in the field of application in order to represent the relevant knowledge of the expert in a forms of that can be entered into the knowledge base.

- The process of building the system usually iterates through many cycles.

- Rule-based expert systems
  - Often based on reasoning with propositional logic Horn clauses.

- The knowledge base
  - Consists of rules gathered from experts.
  - Example: loan officer in a bank
  - "Decide whether or not to grant a personal loan to an individual."

---

17.4 Rule-Based Expert Systems (4)

- **Atoms**
  - OK (The loan should be approved.)
  - COLLAT (The collateral for the loan is satisfactory.)
  - PYMT (The applicant is able to the loan payments.)
  - REP (The applicant has a good financial reputation.)
  - APP (The appraisal on the collateral is sufficiently greater than the loan amount.)
  - RATING (The applicant has a good credit rating.)
  - INC (The applicant's income exceeds his/her expenses.)
  - BAL (The applicant has an excellent balance sheet.)

- **Rules**
  1. COLLAT ∧ PYMT ∧ REP ⊃ OK
  2. APP ⊃ COLLAT
  3. RATING ⊃ REP
  4. INC ⊃ PYMT
  5. BAL ∧ REP ⊃ OK
17.4 Rule-Based Expert Systems (5)

- To prove OK
  - The inference engine searches for AND/OR proof tree using either backward or forward chaining.
- The AND/OR proof tree
  - Root node: OK
  - Leaf node: facts
  - The root and leaves will be connected through the rules.
- Using the preceding rule in a backward-chaining
  - The user’s goal, to establish OK, can be done either by proving both BAL and REP or by proving each of COLLAT, PYMT, and REP.
  - Applying the other rules, as shown, results in other sets of nodes to be proved.

(C) 2000 SNU CSE Artificial Intelligence Lab (SCAI)

17.4 Rule-Based Expert Systems (6)

- By backward-chaining

![Diagram showing the backward-chaining process with rules and nodes labeled]

(C) 2000 SNU CSE Artificial Intelligence Lab (SCAI)

17.4 Rule-Based Expert Systems (7)

- Consulting system
  - Attempt to answer a user’s query by asking questions about the truth of propositions that they might know about.
  - Backward-chaining through the rules is used to get to askable questions.
  - If a user were to “volunteer” information, bottom-up, forward chaining through the rules could be used in an attempt to connect to the proof tree already built.
  - The ability to give explanations for a conclusion
    - Very important for acceptance of expert system advice.
  - Proof tree
    - Used to guide the explanation-generation process.

(C) 2000 SNU CSE Artificial Intelligence Lab (SCAI)

17.4 Rule-Based Expert Systems (8)

- In many applications, the system has access only to uncertain rules, and the user not be able to answer questions with certainty.
- MYCIN [Shortliffe 1976]: Diagnose bacterial infections.

Rule 300
If: 1) The infection which requires therapy is meningitis, and 2) The patient does have evidence of serious skin or soft tissue infection, and 3) Organisms were not seen on the stain of the culture, and 4) The type of the infection is bacterial
Then: There is evidence that the organism (other than those seen on cultures or smears) which might be causing the infection is staphylococcus - coag - pos (.75); streptococcus - group - a (.5).

(C) 2000 SNU CSE Artificial Intelligence Lab (SCAI)
17.4 Rule-Based Expert Systems (9)

- **PROSPECTOR** [Duda, Gaschnig & Hart 1979, Campbell, et al. 1982]
  - Reason about ore deposits.
  - If there is a pre-intrusive, thorough-going fault system, then there is (5, 0.7) a regional environment favorable for a porphyry copper deposit.
- The numbers (.75 and .5 in MYCIN, and 5, 0.7 in PROSPECTOR) are ways to represent the certainty or strength of a rule.
- The numbers are used by these systems in computing the certainty of conclusions.

17.5 Rule Learning

- **Inductive rule learning**
  - Creates new rules about a domain, not derivable from any previous rules.
  - Ex) Neural networks
- **Deductive rule learning**
  - Enhances the efficiency of a system’s performance by deducting additional rules from previously known domain rules and facts.
  - Ex) EBG (explanation-based generalization)

17.5.1 Learning Propositional Calculus Rules

- Train rules from given training set
  - Seek a set of rules that covers only positive instances
    - Positive instance: OK = 1
    - Negative instance: OK = 0
  - From training set, we desire to induce rules of the form
    \[ \alpha_1 \land \alpha_2 \land \cdots \land \alpha_n \Rightarrow \text{OK} \quad \text{where} \quad \alpha_i \in \{\text{APP, RATING, INC, BAL}\} \]
  - We can make some rule more specific by adding an atom to its antecedent to make it cover fewer instances.
    - Cover: If the antecedent of a rule has value True for an instance in the training set, we say that the rule covers that instance.
    - Adding a rule makes the system using these rules more general.
    - Searching for a set of rules can be computationally difficult.
  - Here, we use "greedy" method which is called separate and conquer.

17.5.1 Learning Propositional Calculus Rules (2)

- **Separate and conquer**
  - First attempt to find a single rule that covers only positive instances
    - Start with a rule that covers all instances
    - Gradually make it more specific by adding atoms to its antecedent.
  - Gradually add rules until the entire set of rules covers all and only the positive instances.
  - Trained rules can be simplified using pruning.
    - Operations and noise-tolerant modifications help minimize the risk of overfitting.
17.5.1 Learning Propositional Calculus Rules (3)

- Example: loan officer in a bank
  - Start with the provisional rule \( T \supset \text{OK} \).
  - Which cover all instances.
  - Add an atom it cover fewer negative instances-working toward covering only positive ones.
  - Decide, which item should we added?
  - From \{APP, RATING, INC, BAL\} by
    \[ r_a = \frac{n^A_a}{n_a} \]
    \( n_a \): the total number of instance covered by the antecedent of the rule after the addition of \( \alpha \)
    \( n^A_a \): the total number of positive instance covered by the antecedent of the rule after the addition of \( \alpha \)

(C) 2000 SNU CSE Artificial Intelligence Lab (SCAI) 29

17.5.1 Learning Propositional Calculus Rules (4)

- Select that \( \alpha \) yielding the largest value of \( r_a \).
  - \( r_{\text{app}} = 3/6 = 0.5 \)
  - \( r_{\text{RATING}} = 4/6 = 0.667 \) So, we select BAL, yielding the provisional rule.
  - \( r_{\text{INC}} = 3/6 = 0.5 \) \( \text{BAL} \supset \text{OK} \)
  - \( r_{\text{BAL}} = 3/4 = 0.75 \)

<table>
<thead>
<tr>
<th>Individual</th>
<th>APP</th>
<th>RATING</th>
<th>INC</th>
<th>BAL</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Individual</th>
<th>APP</th>
<th>RATING</th>
<th>INC</th>
<th>BAL</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 12.1

<table>
<thead>
<tr>
<th>Reduced Data</th>
</tr>
</thead>
</table>

17.5.1 Learning Propositional Calculus Rules (5)

- Rule \( \text{BAL} \supset \text{OK} \) covers the positive instances 3, 4, and 7, but also covers the negative instance 1.
  - So, select another atom to make this rule more specific.
  - We have already decided that the first component in the antecedent is BAL, so we have to consider it.
  - We select RATING because \( r_{\text{RATING}} \) is based on a larger sample.
  - \( r_{\text{app}} = 2/3 = 0.667 \)
  - \( r_{\text{RATING}} = 3/3 = 1.0 \)
  - \( r_{\text{INC}} = 2/2 = 1.0 \) \( \text{BAL} \wedge \text{RATING} \supset \text{OK} \)
  - We need more rule which cover positive instance 6.
  - To learn the next rule, eliminate from the table all of the positive instances already covered by the first rule.

(C) 2000 SNU CSE Artificial Intelligence Lab (SCAI) 31

17.5.1 Learning Propositional Calculus Rules (5)

- Begin the process all over again with reduced table
  - Start with the rule \( T \supset \text{OK} \).
  - \( r_{\text{app}} = 1/4 = 0.25 \)
  - RATING=APP=0, arbitrarily select APP.
  - \( r_{\text{RATING}} = 0/3 = 0.0 \)
  - \( \text{APP} \supset \text{OK} \)
  - \( r_{\text{INC}} = 1/4 = 0.25 \) This rule covers negative instances 1, 8, and 9
  - \( r_{\text{BAL}} = 0/1 = 0.0 \) we need another atom to the antecedent.
  - \( r_{\text{RATING}} = 1/2 = 0.5 \) Select RATING, and the we get
  - \( r_{\text{INC}} = 1/2 = 0.5 \)
  - \( \text{APP} \wedge \text{RATING} \supset \text{OK} \)
  - \( r_{\text{BAL}} = 0/1 = 0.0 \) This rule covers negative example 9.
  - Finally we get \( \text{APP} \wedge \text{RATING} \wedge \text{INC} \supset \text{OK} \) which covers only positive instances with first rule, so we are finished.

(C) 2000 SNU CSE Artificial Intelligence Lab (SCAI) 32
17.5.1 Learning Propositional Calculus Rules (6)

- Pseudocode of this rule learning process.
  - Generic Separate-and-conquer algorithm (GSCA)
    - $\mathcal{E}$ is the initial training set of instances of binary-valued features each labeled by the value of an atom, $\gamma$
    - $\pi$ is a set of rules to be learned
    - $\rho$ is one of the rules; it has $\gamma$ as its consequent and (the conjunction of atoms) $\Gamma$ as its antecedent
    - $\alpha$ is an atom drawn from one of the features in $\mathcal{E}$

(C) 2000 SNU CSE Artificial Intelligence Lab (SAIL)

17.5.1 Learning Propositional Calculus Rules (7)

- \textbf{GSCA}
  1. Initialize $\mathcal{S}_0 \leftarrow \mathbb{E}$.
  2. Initialize $\pi \leftarrow$ empty set of rules.
  3. repeat The outer loop adds rules until $\pi$ covers all (or most) of the positive instances.
  4. Initialize $\Gamma \leftarrow \emptyset$.
  5. Initialize $\rho \leftarrow \emptyset \cup \gamma$.
  6. repeat The inner loop adds atoms to $\Gamma$ until $\rho$ covers only (or mainly) positive instances.
  7. Select an atom $\alpha$ to add to $\Gamma$. This is a nondeterministic choice point that can be used for backtracking.
  8. $\Gamma \leftarrow \Gamma \land \alpha$.
  9. until $\rho$ covers only (or mainly) positive instances in $\mathcal{S}_{out}$.
  10. $\pi \leftarrow \pi, \rho$. We add the rule $\rho$ to the set of rules.
  11. $\mathcal{S}_{out} \leftarrow \mathcal{S}_{out} \cup \{\text{the positive instances in } \mathcal{S}_{pos} \text{ covered by } \pi\}$.
  12. until $\pi$ covers all (or most) of the positive instances in $\mathcal{E}$.

(C) 2000 SNU CSE Artificial Intelligence Lab (SAIL)

17.5.2 Learning First-Order Logic Rules

- Inductive logic programming (ILP)
  - Concentrate on methods for inductive learning of Horn clauses in first order predicate calculus.
- The Objective of ILP
  - To learn a, $\pi$, consisting of Horn clauses, $\rho$, each of which is of the form $\rho : \neg a_1, a_2, \ldots, a_n$ where, the $a_i$ are atomic formulas that unify with ground atomic facts.
  - $\mathcal{E} : \pi$ should evaluate to \textit{True} when its variables are bound to some set of values known to be in the relation we are trying to learn (positive instance; \textit{training set}).
  - $\mathcal{E} : \pi$ should evaluate to \textit{False} when its variables are bound to some set of values known not to be in the relation (negative instance).

(C) 2000 SNU CSE Artificial Intelligence Lab (SAIL)

17.5.2 Learning First-Order Logic Rules (2)

- We want $\pi$ to cover the positives instances and not cover negative ones.
- Background knowledge
  - The ground atomic facts with which the $\alpha$ are to unify.
- We assume
  - They are given-as either subsidiary PROLOG programs, which can be run and evaluated, or explicitly in the form of a list of facts.
- Example
  - Delivery robot navigating around in a building finds...
17.5.2 Learning First-Order Logic Rules (3)

- Example (continued)
  - Through experience, that it is easy to go between certain pairs of locations and not so easy to go between certain other pairs.
  - Location A, B, and C arejunctions, and all of the other locations are shop.
  - Junction (x)
    - Whether junction or not.
  - Shop (x,y)
    - Whether shop or not which is connected to junction x.

(C) 2000 SNU CSE Artificial Intelligence Lab (SCAI)

17.5.2 Learning First-Order Logic Rules (4)

- Example (continued)
  - We want a learning program to learn a program, Easy (x,y) that covers the positive instances in \( \Sigma \) but not the negative ones.
  - Easy (x,y) can use the background subexpression Junction(x) and Shop(x,y).
  - Training set
    - Easy(\( \Sigma^+ \))
    - Easy(\( \Sigma^- \))

(C) 2000 SNU CSE Artificial Intelligence Lab (SCAI)

17.5.2 Learning First-Order Logic Rules (5)

- We assume that
  - The robot can evaluate Junction(x) and Shop(x, y) for any values of the variables.
  - For all of the locations named in \( \Sigma \), only the following pairs give a value True for Shop:
    \[ \{< A1,A >, < A2,A >, < B1,B >, < B2,B >, < C1,C >, < C2,C > \} \]
  - Following PROLOG program covers all of the positive instances of the training set and none of the negative ones
    Easy(x, y) :-Junction(x), Junction(y)
    :-Shop(x, y)
    :-Shop(y, x)

(C) 2000 SNU CSE Artificial Intelligence Lab (SCAI)

17.5.2 Learning First-Order Logic Rules (6)

- Learning process
  - Use generalized separate and conquer algorithm (GSCA)
    - Start with a program having a single rule with no body
    - Add literal to the body until the rule covers only(or mainly) positive instances
    - Add rules in the same way until the program covers all(or most) and only(with few exceptions) positive instances.
17.5.2 Learning First-Order Logic Rules (7)

- Practical ILP systems restrict the literals in various ways.
- Typical allowed additions:
  - Literals used in the background knowledge
  - Literals whose arguments are a subset of those in the head of the clause.
  -Literal that introduces a new distinct variable different from those in the head of the clause.
  - A literal that equates a variable in the head of the clause with another such variable or with a term mentioned in the background knowledge.
  - A literal that is the same (except for its arguments) as that in the head of the clause.

(C) 2000 SNU CSE Artificial Intelligence Lab (SCAI) 41

17.5.2 Learning First-Order Logic Rules (8)

- The literal that we might consider adding to a clause are
  - Junction(x), Junction(y), Junction(z)
  - Shop(x,y), Shop(y,x), Shop(x,z)
  - Shop(y,x), (x=y)
- ILP version of GSCA
  - First, initialize first clause as Easy(x,y):-
  - Add Junction(x), so Easy(x, y) covers all following instances
      <C,B1>,<C,B2>,<B,A1>,<B,A2>,<B,B1>,<B,B2>,<C,C1>,<C,C2>\}
  - Negative instances:
    - Include more literal Junction(y) → Easy(x, y) :- Junction(x), Junction(y)

(C) 2000 SNU CSE Artificial Intelligence Lab (SCAI) 42

17.5.2 Learning First-Order Logic Rules (9)

- But program \(\Pi\) does not cover the following positive instances.
  - Junction(x), Junction(y) from \(\Xi\) to form the \(\Xi_{\text{ex}}\) to be used in next pass through inner loop.
    - \(\Xi_{\text{ex}}\): all negative instance in \(\Xi\) + the positive instance that are not covered yet.
    - Inner loop create initial clause “Easy(x,y) :-”
      - Add literal Shop(x,y) : Easy(x,y) :- Shop(x,y) → cover no negative instances, so we are finished with another pass through inner loop.
      - Covered positive instance by this rule (remove this from \(\Xi_{\text{ex}}\))
        - \{<A1,A>,<A2,A>,<B1,B>,<B2,B>,<C1,C>,<C2,C>\}

(C) 2000 SNU CSE Artificial Intelligence Lab (SCAI) 43

17.5.2 Learning First-Order Logic Rules (10)

- Now we have Easy(x,y) :- Junction(x), Junction(y)
  - Shop(x, y)
- To cover following instance
  - \{<A,B1>,<A,A2>,<B,B1>,<B,B2>,<C,C1>,<C,C2>\}
- Add Shop(y, x)
- Then we have Easy(x,y) :- Junction(x), Junction(y)
  - Shop(x, y)
  - Shop(y, x)
- This cover only positive instances.

(C) 2000 SNU CSE Artificial Intelligence Lab (SCAI) 44
17.5.3 Explanation-Based Generalization (1)

- Example: “Block world”
  - General knowledge of the “Block world”.
    - Rules
    - Fact
  - We want to prove “ ”
  - Proof is very simple ➔

17.5.3 Explanation-Based Generalization (2)

- Explanation: Set of facts used in the proof
  - Ex) explanation for “ ” is “ ”
  - From this explanation, we can make “ ”
    - Replacing constant ‘A’ by variable ‘x’, then, we have “Green(x)”
    - Then we can prove “ ”, as like the case of “ ”
  - ➔ Explanation Based Generalization

- Explanation-based generalization (EBG): Generalizing the explanation by replacing constant by variable
  - More rules might slow down the reasoning process, so EBG must be used with care-possibly by keeping information about the utility of the learned rules.

Additional Readings

- [Levesque & Brachman 1987]
  - Balance between logical expression and logical inference
- [Ullman 1989]
  - Datalog
- [Selman & Kautz 1991]
  - Approximate theory: Horn greatest-lower-bound, Horn least-upper-bound
- [Kautz, Kearns, & Selman 1993]
  - Characteristic model

Additional Readings

- [Roussel 1975, Colmerauer 1973]
  - PROLOG interpreter
- [Warren, Pereira, & Pereira 1977]
  - Development of efficient interpreter
- [Davis 1980]
  - A* algorithm searching AND/OR graphs
- [Selman & Levesque 1990]
  - Determination of minimum ATMS label: NP-complete problem
Additional Readings

- [Kautz, Kearns & Selman 1993]
  - TMS calculation based on characteristic model
  - Other results for TMS
- [Bobrow, Mittal & Stefik 1986], [Stefik 1995]
  - Construction of expert system

Additional Readings

- [McDermott 19982]
  - Examples of expert systems
- [Leonard-Barton 1987]
  - History and usage of DEC’s expert system
- [Kautz & Selman 1992], [Muggleton & Buntine 1988]
  - Predicate finding
- [Muggleton, King & Sternberg 1992]
  - Protein secondary structure prediction by GOLEM