Expert Systems

Introduction to Intelligent Systems
State-Space Representation of a Logic System

• Many problems states (e.g., tic-tac-toe) can be naturally described by data structures such as arrays

• The power and generality of logic allows for a different type of representation – that of propositions, predicates, and rules of inference
State-Space Representation of a Logic System

- A state-space graph of a logical system consists of nodes (the propositions) and arcs (the logical implications).
- Determining whether a given proposition is a logical consequence of a set of propositions becomes a problem of finding a path from a start node to the goal node.
And/Or Graphs

And/Or graph of the expression $q \lor r \rightarrow p$
And/Or Graphs

And/Or graph of the expression $q \land r \rightarrow p$
And/Or Graphs

And/Or graph of a set of propositional calculus expressions
Forward Chaining (Data-Driven)

• Forward chaining is a reasoning model that works from a set of facts and rules towards a set of conclusions, diagnoses or recommendations

• When a fact matches the antecedent of a rule, the rule fires, and the conclusion of the rule is added to the database of facts
Forward Chaining (Data-Driven)

Let $p$, $q$, $r$, ... be propositions. Assume the following assertions:

$q \rightarrow p, \quad r \rightarrow p, \quad v \rightarrow q, \quad s \rightarrow r, \quad t \rightarrow r, \quad s \rightarrow u, \quad s, \quad t$

Propositions that are given as true ($s$ and $t$) correspond to the given data of the problem.

From this set of assertions and modus ponens, certain proposition ($p$, $r$, and $u$) may be inferred; others (such as $v$ and $q$) may not be so inferred, and indeed do not logically follow from these assertions.
Start with the facts (A and B). These are assumed to be true. Next, fire any rule whose premises are satisfied, and add its conclusion to the KB, until the query (Q) is found.

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Backward Chaining (Goal-Driven)

• In cases where a particular conclusion is to be proved, backward chaining can be more appropriate

• Backward chaining works backward from a conclusion towards the original facts

• When a conclusion matches the conclusion of a rule in the database, the antecedents of the rule are compared with facts in the database
Backward Chaining (Goal-Driven)

Work backwards from the query $Q$. To prove $Q$, check if $Q$ is known already, or prove it is true by checking all premises of some rule concluding $Q$.

$$
P \Rightarrow Q
$$
$$
L \land M \Rightarrow P
$$
$$
B \land L \Rightarrow M
$$
$$
A \land P \Rightarrow L
$$
$$
A \land B \Rightarrow L
$$
$$
A
$$
$$
B
Example of Goal-Driven Search

• A goal-driven graph search is where the goal to be proved true is a predicate calculus expression containing variables
• The axioms are the logical descriptions of a relationship between objects
• The facts and rules for a specific example are given on the following slide
Example of Goal-Driven Search

1. Fred is a collie.
   \texttt{collie(fred)}.

2. Sam is Fred’s master.
   \texttt{master(fred,sam)}.

3. The day is Saturday.
   \texttt{day(saturday)}.

4. It is cold on Saturday.
   \texttt{\neg (warm(saturday))}.

5. Fred is trained.
   \texttt{trained(fred)}.

6. Spaniels are good dogs and so are trained collies.
   \forall X [\texttt{spaniel(X) } \lor (\texttt{collie(X)} \land \texttt{trained(X)}) \rightarrow \texttt{gooddog(X)}]

7. If a dog is a good dog and has a master then he will be with his master.
   \forall (X,Y,Z) [\texttt{gooddog(X) } \land \texttt{master(X,Y)} \land \texttt{location(Y,Z)} \rightarrow \texttt{location(X,Z)}]

8. If it is Saturday and warm, then Sam is at the park.
   \texttt{(day(saturday) } \land \texttt{warm(saturday)) } \rightarrow \texttt{location(sam,park)}.

9. If it is Saturday and not warm, then Sam is at the museum.
   \texttt{(day(saturday) } \land \texttt{\neg (warm(saturday))} ) \rightarrow \texttt{location(sam,museum)}.
Example of Goal-Driven Search

• The goal is the expression: location(fred, X), meaning “where is fred?”

• A goal-driven search examines all possible ways of establishing X (the location of fred)

• Consider rule 7: if fred is a good dog and fred has a master and fred’s master is at a location then fred is at that location also

• The premises of this rule are then examined: what does it mean to be a “good dog”, etc?

• More formally, the problem is to determine a substitution for the variable X. If such a substitution exists, location(fred, X) will be a logical consequence of the initial assertions
Example of Goal-Driven Search

To do this, clauses that have “location(fred,X)” as their conclusion are examined in the order they are given in the database. The first such clause is clause 7. The conclusion, location(X, Z), is unified with location(fred, X) by the substitutions {fred/X, X/Z}. The premises of this rule, under the same substitution set, form the “and” descendants of the top goal:

\[
gooddog(fred) \land master(fred, Y) \land location(Y, X).\]

This expression may be interpreted as meaning that one way to find fred is to see if fred is a good dog, find out who fred’s master is, and then find out where the master is. The initial goal has thus been replaced by three sub-goals. These are “and” nodes and all of them must be solved.
Example of Goal-Driven Search

To solve the sub-goals, the problem solver first determines whether fred is a good dog. This matches the conclusion of clause 6 using the substitution {fred/X}. The premise of clause 6 is the “or” of two expressions:

spaniel(fred) ∨ (collie(fred) ∧ trained(fred)).

The database does not contain the first assertion, so the problem solver must assume it is false. The other “or” node is true, due to clauses 1 and 5. This proves that gooddog(fred) is true.
Example of Goal-Driven Search

The problem solver then examines the second of the premises of clause 7: master(X, Y). Under the substitution \{fred/X\}, master(X, Y) becomes master(fred, Y), which unifies with the fact from clause 2 of master(fred, sam). This produces the unifying substitution of \{sam/Y\}, which also gives the value sam to the third sub-goal of clause 7, creating the new goal location(sam, X).

Since the problem solver tries rules in order, the goal location(sam, X) will first unify with the conclusion of clause 7. Note that the same rule is being tried with different bindings for X (no global variables). Thus, the multiple occurrences of X in different rules in this example indicate different formal parameters.
Example of Goal-Driven Search

In attempting to solve the premises of clause 7 with these new bindings, the problem solver will fail because sam is not a good dog. Here, the search will backtrack to the goal location(sam, X) and try the next match, the conclusion of clause 8. This will also fail, which will cause another backtrack and a unification with the conclusion of clause 9 location(sam, museum).

Because the premises of clause 9 are supported in the set of assertions (clauses 3 and 4), it follows that the conclusion of 9 is true. This final unification goes all the way back up the tree to finally answer: location(fred, museum).
Example of Goal-Driven Search

Substitutions = \{fred/X, sam/Y, museum/Z\}
Data-Driven vs. Goal-Driven

• Data-driven is best for automatic, “unconscious” processing
  – e.g., object recognition, routine decisions
  – may do lots of work that is irrelevant to the goal
• Goal-driven is more appropriate for problem-solving
  – e.g., Where are my keys? How do I get into a PhD program?
• Complexity of goal-driven can be much less than linear in size of the knowledge base
• Goal-driven is often used in expert systems that are designed for medical diagnosis or other types of diagnostic systems
Rule-Based Production Systems

• A production system is a system that uses knowledge in the form of rules to provide diagnoses or advice on the basis of input data.
• The system consists of a database of rules (knowledge base), a database of facts, and an inference engine which reasons about the facts using the rules.
• Also called an “expert system”
Architecture of an Expert System

- An expert system uses expert knowledge derived from human experts to diagnose illnesses, provide recommendations and solve other problems.
Architecture of an Expert System

- Knowledge base: database of rules (domain knowledge)
- Explanation system: explains the decisions the system makes
- User Interface: the means by which the user interacts with the expert system
- Knowledge base editor: allows the user to edit the information in the knowledge base
Expert System Example

MYCIN: A rule-based expert system implemented in LISP that is used to diagnose bacterial infections and recommend a course of antibiotics according to disease, weight, etc.

IF the infection is primary-bacteremia
AND the site of the culture is one of the sterile sites
AND the suspected portal of entry is the gastrointestinal tract
THEN there is suggestive evidence (0.7) that infection is bacteroid.
Knowledge Engineering

• A knowledge engineer takes knowledge from experts and inputs it into the expert system
• A knowledge engineer will usually choose which expert system shell to use
• The knowledge engineer is also responsible for entering meta rules
Knowledge Engineering

1. Identify the task
2. Assemble the relevant knowledge
3. Decide on a vocabulary of predicates, functions, and constants
4. Encode general knowledge about the domain
5. Encode a description of the specific problem instance
6. Pose queries to the inference procedure and get answers
7. Debug the knowledge base
Conflict Resolution

• Sometimes more than one rule will fire at once, and a conflict resolution strategy must be used to decide which conclusions to use
• One strategy is to give rules priorities (confidences) and to use the conclusion that has the highest priority (confidence)
• Other strategies include applying the rule with the longest antecedent, or applying the rule that was most recently added to the database
Meta Rules

• The rules that determine the conflict resolution strategy are called *meta rules*.
• Meta rules define knowledge about how the system will work.
• For example, meta rules might define that knowledge from Expert A is to be trusted more than knowledge from Expert B.
• Meta rules are treated by the system like normal rules, but are given higher priority.
Example of Expert System

- The Financial Advisor: help a user decide whether to invest in a savings account or the stock market
- Some investors may want to split their money between the two
- The investment that will be recommended for individual investors depends on their income and the current amount they have saved
Example of Expert System

• Individuals with an inadequate savings account should always make increasing the amount saved their first priority, regardless of their income
• Individuals with an adequate savings account and an adequate income should consider a riskier but potentially more profitable investment in the stock market
• Individuals with a lower income who already have an adequate savings account may want to consider splitting their surplus income between savings and stocks, to increase the cushion in savings while attempting to increase their income through stocks
Example of Expert System

The adequacy of both savings and income is determined by the number of dependents an individual must support. The rule is to have at least $5,000 in savings for each dependent. An adequate income must be a steady income and supply at least $15,000 per year plus an additional $4000 per year for each dependent.

The first step is to translate these guidelines into the predicate calculus using the following predicates:

- savings_account(adequate)
- savings_account(inadequate)
- income(adequate)
- income(inadequate)
Example of Expert System

Using these predicates, the different investment strategies are represented by implications:

\[
\begin{align*}
savings\_account(\text{inadequate}) & \rightarrow investment(\text{savings}). \\
savings\_account(\text{adequate}) \land income(\text{adequate}) & \rightarrow investment(\text{stocks}). \\
savings\_account(\text{adequate}) \land income(\text{inadequate}) & \rightarrow investment(\text{combination}).
\end{align*}
\]

Conclusions are represented by the predicate investment, with possible values of its arguments being stocks, savings, or combination:

\[
investment(X).
\]

where we seek to bind X
Example of Expert System

To determine the minimum adequate savings, a function called minsavings is defined that takes one argument, the number of dependents, and returns 5000 times that argument.

\[
\text{amount\_saved}(X) \land (\text{dependents}(Y) \land \text{greater}(X, \text{minsavings}(Y))) \\
\quad \rightarrow \text{savings\_account(adequate)}.
\]

\[
\text{amount\_saved}(X) \land (\text{dependents}(Y) \land \neg\text{greater}(X, \text{minsavings}(Y))) \\
\quad \rightarrow \text{savings\_account(inadequate)}.
\]
Example of Expert System

In addition, minincome is used to compute the minimum adequate income when given the number of dependents. The investor’s current income is represented by a predicate, earnings. Because an adequate income must be both steady and above the minimum, earnings takes two arguments: the amount earned and either “steady” or “unsteady”.

\[
\text{earnings}(X, \text{steady}) \land (\text{dependents}(Y) \land \text{greater}(X, \text{minincome}(Y))) \\
\rightarrow \text{income}(\text{adequate}).
\]

\[
\text{earnings}(X, \text{steady}) \land (\text{dependents}(Y) \land \neg\text{greater}(X,\text{minincome}(Y))) \\
\rightarrow \text{income}(\text{inadequate}).
\]

\[
\text{earnings}(X, \text{unsteady}) \rightarrow \text{income}(\text{inadequate}).
\]
Example of Expert System

In order to perform a calculation, a description of a particular investor is added to this set of predicate calculus sentences using the predicates amount_saved, earnings, and dependents. Thus, an individual with 3 dependents, $22,000 in savings, and a steady income of $25,000 would be described by:

\[
\begin{align*}
\text{amount\_saved}(22000). \\
\text{earnings}(25000, \text{steady}). \\
\text{dependents}(3).
\end{align*}
\]

In addition,

\[
\begin{align*}
\text{minsavings}(X) &\equiv 5000 \times X \\
\text{minincome}(X) &\equiv 15000 + (4000 \times X)
\end{align*}
\]
Example of Expert System

1. savings_account(inadequate) → investment(savings).
2. savings_account(adequate) ∧ income(adequate) → investment(stocks).
3. savings_account(adequate) ∧ income(inadequate)
   → investment(combination).
4. amount_saved(X) ∧ (dependents(Y) ∧ greater(X, minsavings(Y)))
   → savings_account(adequate).
5. amount_saved(X) ∧ (dependents(Y) ∧ ¬greater(X, minsavings(Y)))
   → savings_account(inadequate).
6. earnings(X, steady) ∧ (dependents(Y) ∧ greater(X, minincome(Y)))
   → income(adequate).
7. earnings(X, steady) ∧ (dependents(Y) ∧ ¬greater(X, minincome(Y)))
   → income(inadequate).
8. earnings(X, unsteady) → income(inadequate).
9. amount_saved(22000).
10. earnings(25000, steady).
11. dependents(3).
Example of Expert System

Using unification and modus ponens, a correct investment strategy for this individual may be inferred as a logical consequence of these descriptions.

Since there are 3 rules that conclude about investments, the facts will unify with the conclusion of one of these rules. If we select rule 1 for initial exploration, its premise, savings_account(inadequate) becomes the sub-goal, i.e., the child node that will be expanded next. In generating this child, the only rule that may be applied is rule 5. This produces the “and” node:

\[
\text{amount\_saved}(X) \land \text{dependents}(Y) \land \neg \text{greater}(X, \text{minsavings}(Y)).
\]
Example of Expert System

If we attempt to satisfy these in left-to-right order, amount_saved(X) is taken as the first sub-goal. With amount_saved(22000) as a fact, the first sub-goal will succeed, with unification substituting 22000 for X. Note that because an “and” node is being searched, a failure here would eliminate the need to examine the remainder of the expression.

Similarly, the sub-goal dependents(Y) matches the expression dependents(3) with the substitution {3/Y}. With these substitutions, the search next evaluates the truth of:

\[ \neg \text{greater}(22000, \text{minsavings}(3)). \]
Example of Expert System

This evaluates to false, causing failure of the entire “and” node. The search then backtracks to the parent node, savings_account(inadequate) and attempts to find an alternative way to prove that node true. This corresponds to the generation of the next child in the search. Because no other rules conclude this sub-goal, search fails back to the top-level goal, investment(X). The next rule whose conclusions unify with this goal is rule 2, producing the new sub-goals:

\[
\text{savings\_account(adequate)} \land \text{income(adequate)}.\]
Example of Expert System

Continuing the search, savings_account(adequate) is proved true as the conclusion of rule 4 because

\[
\text{amount\_saved}(22000) \land \text{dependents}(3)
\]

unifies with the first two elements of the premise of assertion 4 under the substitution \{22000/X, 3/Y\}, yielding the implication:

\[
\text{amount\_saved}(22000) \land (\text{dependents}(3) \land \text{greater}(22000, \text{minsavings}(3))) \\
\rightarrow \text{savings\_account}(\text{adequate}).
\]

Here, evaluating the function \text{minsavings}(3) yields:

\[
\text{amount\_saved}(22000) \land (\text{dependents}(3) \land \text{greater}(22000, 15000)) \\
\rightarrow \text{savings\_account}(\text{adequate}).
\]
Example of Expert System

However, income(adequate) is proved false due to rule 6. Because no other rules conclude sub-goal 2, search fails back to the top-level goal once again, investment(X). The next rule whose conclusions unify with this goal is rule 3, producing the new sub-goals:

\[
\text{savings\_account(adequate)} \land \text{income(inadequate)}.
\]

savings\_account(adequate) has already been shown to be true, so income(inadequate) must be verified next, using rule 7.
Example of Expert System

Unifying 10 and 11 with the first two components of the premise of 7:

\[
\text{earnings}(25000, \text{steady}) \land \text{dependents}(3)
\]

unifies with \( \text{earnings}(X, \text{steady}) \land \text{dependents}(Y) \)
under the substitutions \( \{25000/X, 3/Y\} \).

This substitution yields the new implication:
\[
\text{earnings}(25000, \text{steady}) \land \text{dependents}(3) \land \neg \text{greater}(25000, \text{minincome}(3)) \rightarrow \text{income}(\text{inadequate}).
\]
Example of Expert System

Evaluating the function minincome yields:

\[
\text{earnings}(25000, \text{steady}) \land \text{dependents}(3) \land \neg \text{greater}(25000, 27000) \\
\rightarrow \text{income(inadequate)}. 
\]

Because all three components of the premise are individually true, by 10, 3, and the mathematical definition of greater, their conjunction is true and the entire premise of rule 7 is true:

\[
\text{income(inadequate)}. 
\]

Modus ponens may therefore be applied, yielding the conclusion:

\[
\text{investment(combination)}. 
\]
Logic Programming and PROLOG

• PROLOG uses resolution on *Horn clauses*
• A Horn clause has at most one positive literal:
  \[ \neg B \lor \neg C \lor \neg D \lor \neg E \lor A \ldots \]
• This can also be written as an implication:
  \[ B \land C \land D \land E \rightarrow A \]
• In PROLOG, this is written:
  \[ A :- B, C, D, E \]
• If a set of clauses is valid, PROLOG will definitely prove it using resolution and backward chaining