DRONACHARYA
College of Engineering

Section D

Explanation-Based Learning (EBL)

Explanation-Based Learning (EBL)

One definition:

Learning *general* problem-solving techniques by observing and analyzing human solutions to *specific* problems.

"Hey! Look what Zog do!"

(drawn by Gary Larson)

The EBL Hypothesis

By understanding why an example is a member of a concept, can learn the essential properties of the concept

Trade-off

the need to collect many examples

for

the ability to “explain” single examples (a “domain” theory)

Learning by Generalizing Explanations

Given
- Goal (e.g., some predicate calculus statement)
- Situation Description (facts)
- Domain Theory (inference rules)
- Operationality Criterion

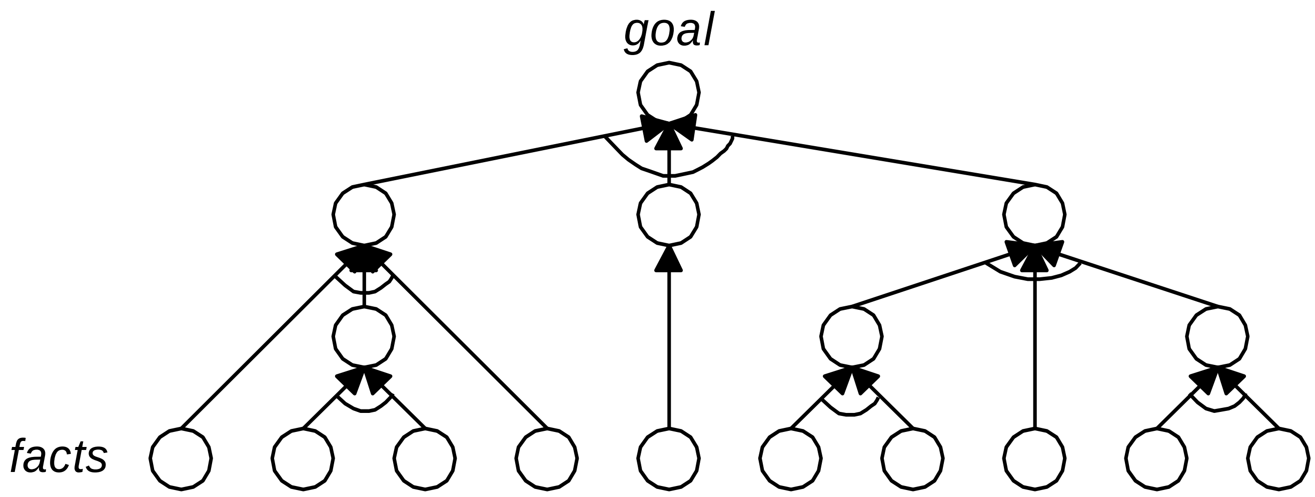
Use problem solver to justify, using the *rules*, the *goal* in terms of the *facts*.

Generalize the justification as much as possible.

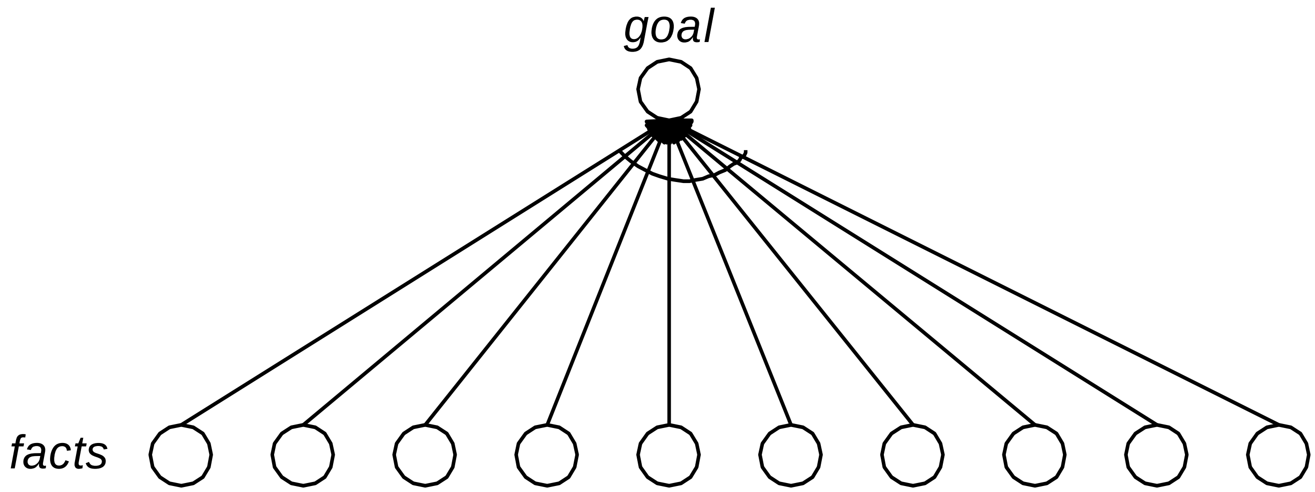
The *operationality criterion* states which other terms can appear in the generalized result.

Standard Approach to EBL

An Explanation (detailed proof of goal)



After Learning (go directly from facts to solution):



Unification-Based Generalization

- An explanation is an inter-connected collection of "pieces" of knowledge (inference rules, rewrite rules, etc.)
- These "rules" are connected using *unification*, as in Prolog
- The generalization task is to compute the *most general unifier* that allows the "knowledge pieces" to be connected together as generally as possible

The EGGS Algorithm (Mooney, 1986)

```
bindings = { }

FOR EVERY equality between
  patterns P and Q in explanation DO
    bindings = unify(P,Q,bindings)

FOR EVERY pattern P DO
  P = substitute-in-values(P,bindings)

Collect leaf nodes and the goal node
```

Sample EBL Problem

Initial Domain Theory

knows(?x,?y) AND nice-person(?y) -> likes(?x,?y)

animate(?z) -> knows(?z,?z)

human(?u) -> animate(?u)

friendly(?v) -> nice-person(?v)

happy(?w) -> nice-person(?w)

Specific Example

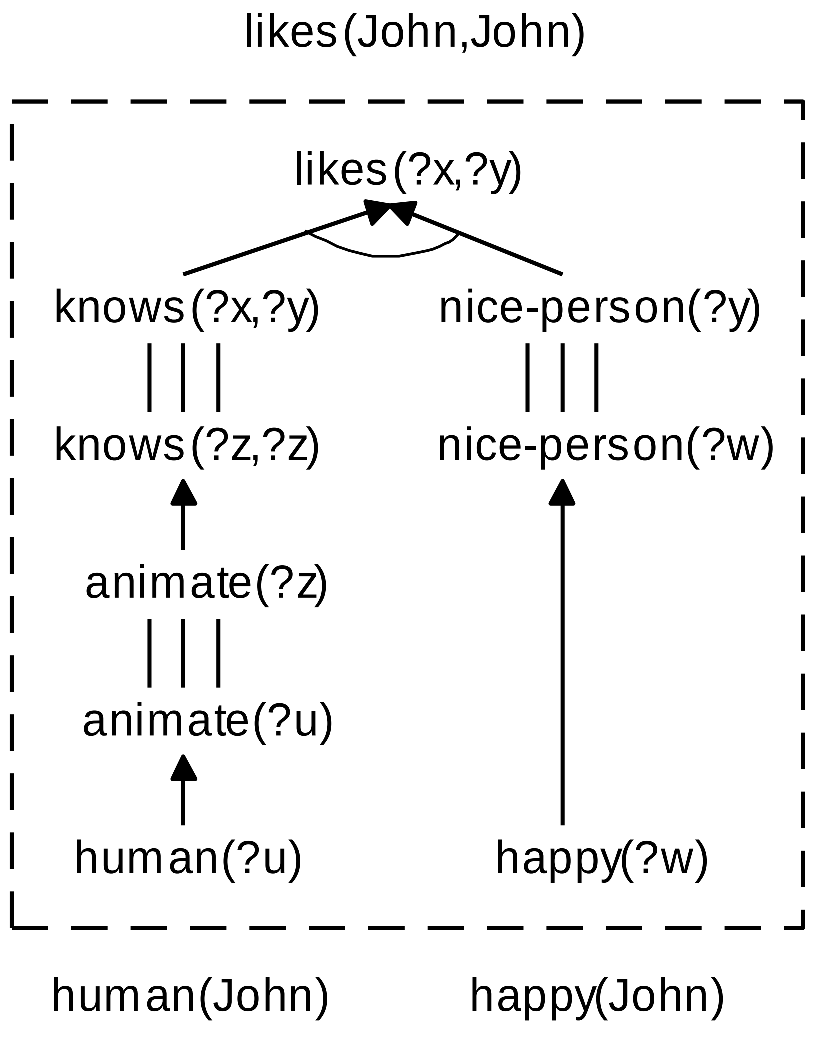
Given human(John) AND happy(John) AND male(John),

show that likes(John,John)

Explanation to Solve Problem

$$\begin{tikzcd}
 & \text{likes(John,John)} & \\
\text{knows(John,John)} \arrow[ur] & & \text{nice-person(John)} \arrow[ul] \\
\text{animate(John)} \arrow[u] & & \\
\text{human(John)} \arrow[u] & & \text{happy(John)} \arrow[uu]
\end{tikzcd}$$

Explanation Structure



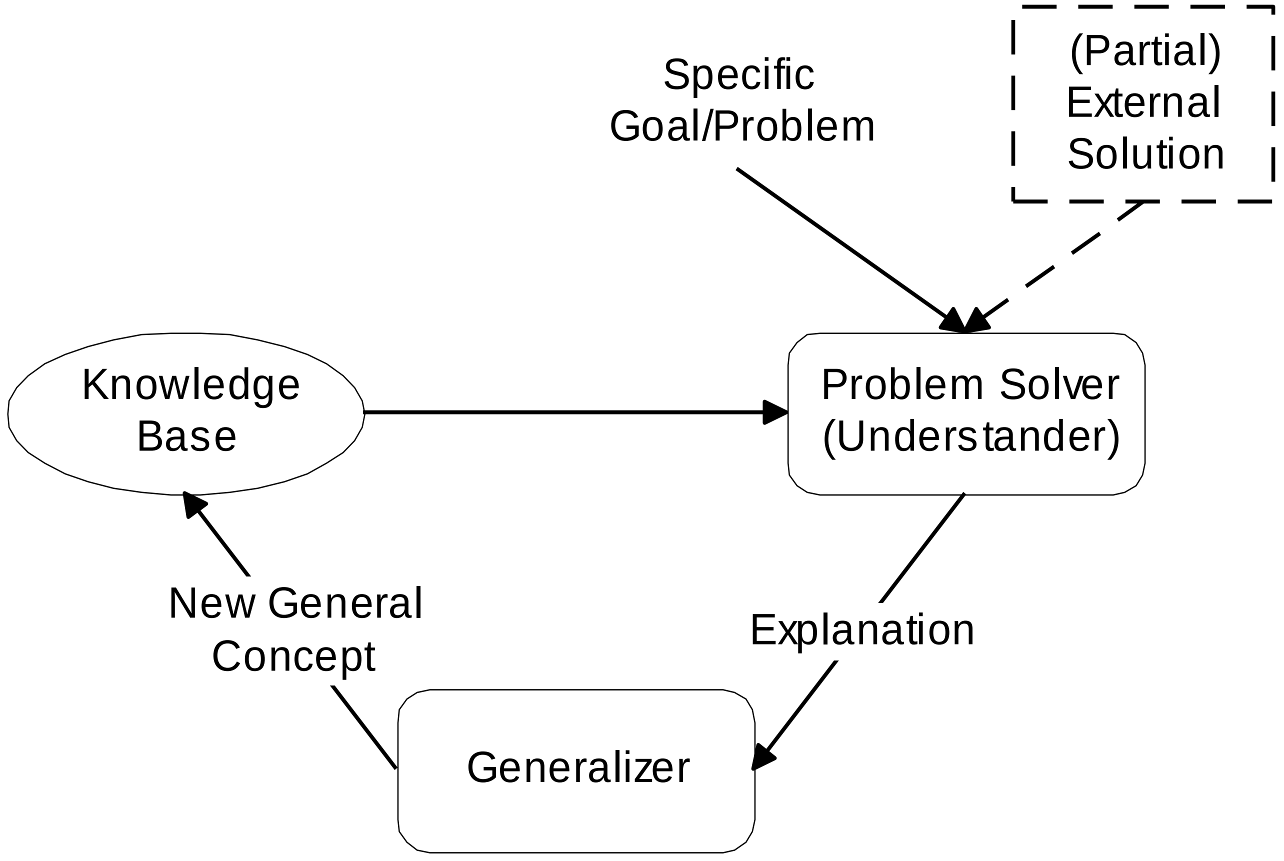
Necessary Unifications:

All variables must match ?z

Resulting Rule:

human(?z) AND happy(?z) -> likes(?z,?z)

Prototypical EBL Architecture



Imperfect Theories and EBL

Incomplete Theory Problem

- Cannot build explanations of specific problems because of missing knowledge

Intractable Theory Problem

- Have enough knowledge, but not enough computer time to build specific explanation

Inconsistent Theory Problem

- Can derive inconsistent results from a theory (e.g., because of default rules)

Some Complications

Inconsistencies and Incompleteness may be due to abstractions and assumptions that make a theory tractable.

Inconsistencies may arise from missing knowledge (incompleteness).

- e.g., making the closed-world assumption

Issues with Imperfect Theories

- Detecting imperfections
  - “broken” explanations (missing clause)
  - contradiction detection (proving P and not P)
  - multiple explanations (but expected!)
  - resources exceeded
- Correcting imperfections
  - experimentation - motivated by failure type (explanation-based)
  - make approximations/assumptions - assume something is true

EBL as Operationalization (Speedup Learning)

Assuming a complete problem solver and unlimited time, EBL already knows how to recognize all the concepts it will know.

What it learns is how to make its knowledge operational (Mostow).

Is this learning?

Isn’t 99% of human learning of this type?

Knowledge-Level Learning (Newell, Dietterich)

- Knowledge closure
  - all things that can be inferred from a collection of rules and facts
- "Pure" EBL only learns how to solve faster, not how to solve problems previously insoluble.
- Inductive learners make inductive leaps and hence can solve more after learning.
- What about considering resource-limits (e.g., time) on problem solving?

Negative Effects of Speedup Learning

The “Utility Problem”

Time wasted checking “promising” rules

- rules that almost match waste more time than obviously irrelevant ones

General, broadly-applicable rules mask more efficient special cases

Defining Utility (Minton)

- Utility = (AvgSav * ApplFreq) - AvgMatchCost
  - where
  - AvgSav - time saved when rule used
  - ApplFreq - probability rule succeeds given its preconditions tested
  - AvgMatchCost - cost of checking rule's preconditions
- Rules with negative utility are discarded
  - estimated on training data

Learning for Search-Based Planners

Two options

- 1) Save composite collections of primitive operators, called MACROPS
  - explanation turned into rule added to knowledge base
- 2) Have a domain theory about your problem solver
  - use explicit declarative representation
  - build explanations about how problems were solved
    - which choices lead to failure, success, etc.
    - learn evaluation functions (prefer pursuing certain operations in certain situations)

Reasons for Control Rules

- Improve search efficiency (prevent going down "blind alleys")
- To improve solution quality (don't necessarily want first solution found via depth-first search)
- To lead problem solver down seemingly unpromising paths
  - overcome default heuristics designed to keep problem solver from being overly combinatoric

PRODIGY - Learning Control Knowledge (Minton, 1989)

Have domain theory about specific problem

AND another about the problem solver itself

Choices to be made during problem solving:

- which node in current search tree to expand
- which sub-goal of overall goal to explore
- relevant operator to apply
- binding of variables to operators

Control rules can

- lead to the choice/rejection of a candidate
- lead to a partial ordering of candidates (preferences)

SOAR
(Rosenbloom, Laird, and Newell, 1986)

Production system that chunks productions via EBL

Production system - forward chaining rule system for problem solving

Key Idea: IMPASSES

- occur when system cannot decide which rule to apply
- solution to impasse generalized into new rule

Summary of SOAR

A “Production System” with three parts:

- A general-purpose forward search procedure
- A collection of operator-selection rules that help decide which operator to apply
- A look-ahead search procedure invoked when at an impasse

When the impasse occurs, can learn new rules to add to collection of operator-selection rules

Reasoning by Analogy

- Create a description of a situation with a known solution and then use that solution in structurally similar situations
- Problem: a doctor can use a beam of radiation to destroy a cancer, but at the high amount needed, it will also destroy the healthy tissue in any path it follows
- Idea: find a similar (some how) situation and use it to create a solution

Reasoning by Analogy Story

- Similar story: a general needs to send his troops to a particular city for a battle by a particular time, but there is no road wide enough to accommodate all of his troops in the time remaining (even though there are several roads)
- Solution: break up the troops into smaller groups and send each group down a different road
- How to solve the radiation situation??