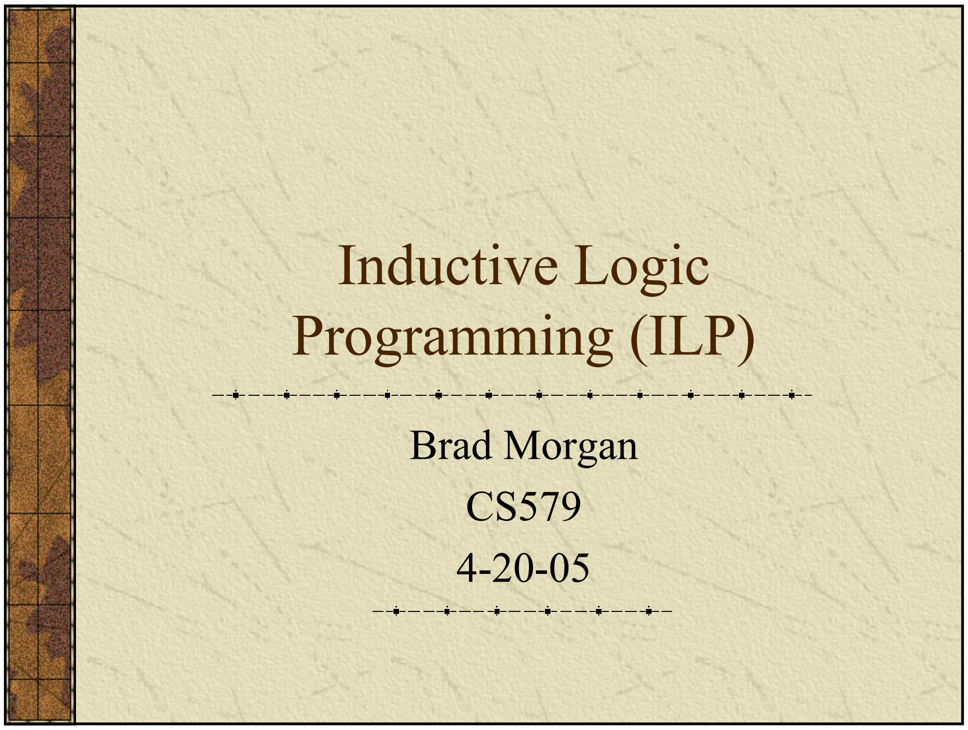
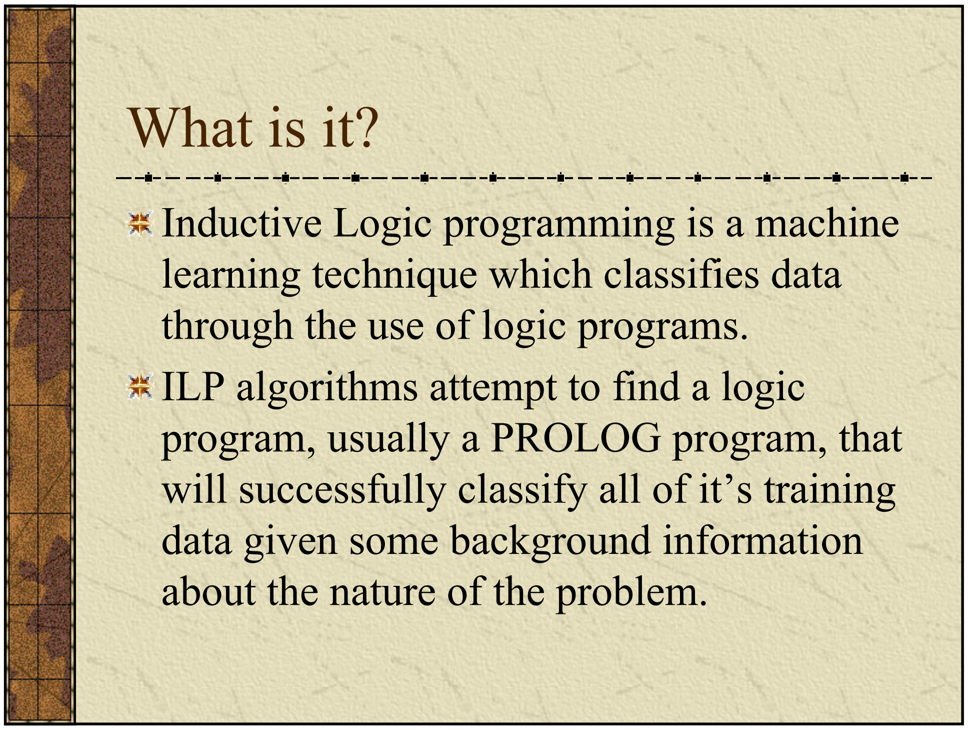


Inductive Logic Programming (ILP)

Brad Morgan

CS579

4-20-05



What is it?

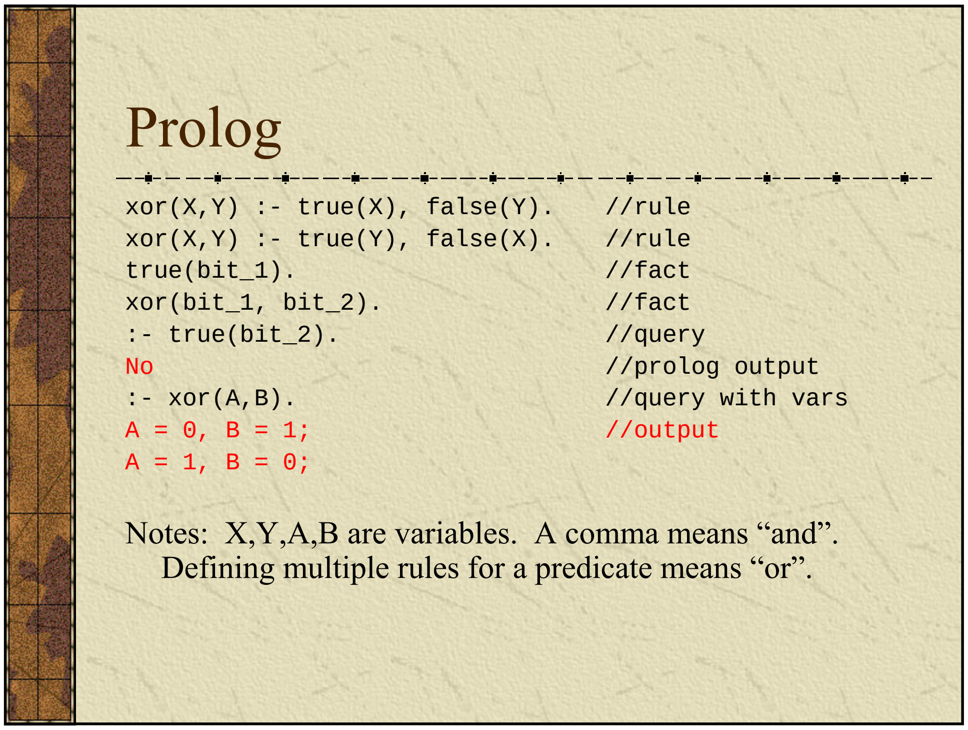
- Inductive Logic programming is a machine learning technique which classifies data through the use of logic programs.
- ILP algorithms attempt to find a logic program, usually a PROLOG program, that will successfully classify all of it’s training data given some background information about the nature of the problem.

Intro to Logic Programming

- Logic programming is non procedural.
- In traditional programming we write an algorithm that computes answers to questions.
- When we write a logic program we are writing the questions (and all of our assumptions) instead of how to answer the questions.
- A logic program interpreter attempts to find possible answers to the questions.

Prolog

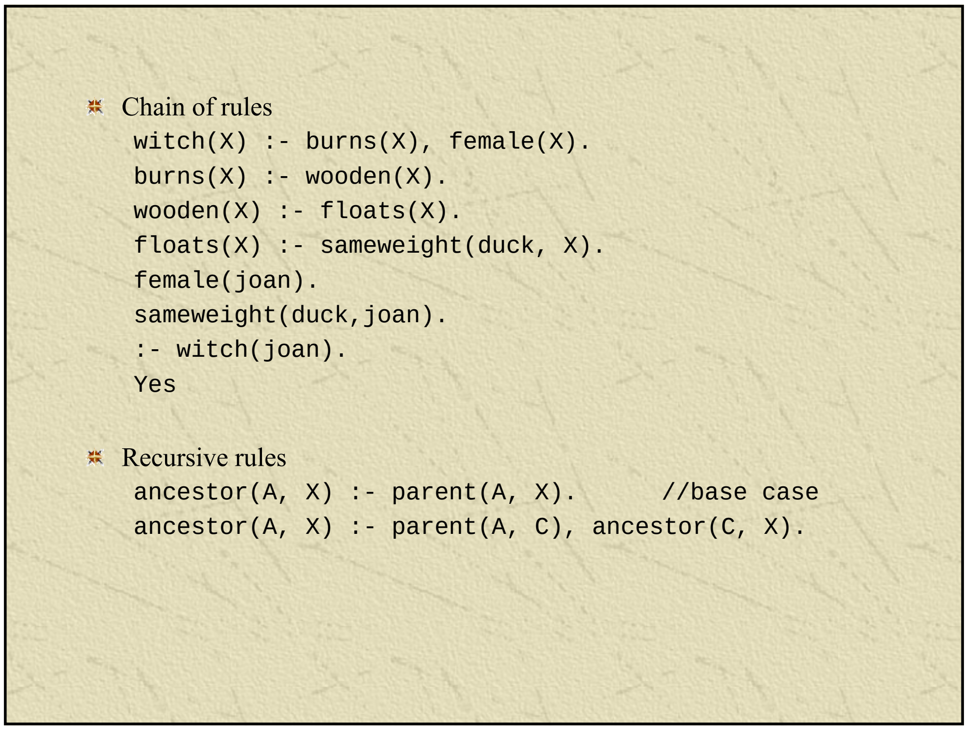
- Prolog is the most popular logic interpreter.
- Prolog programs are constructed from clauses.
  - Clause: “<head> :- <body>.”
  - Both Head and Body are composed of literals/predicates. E.g. “parent(john, X).”
  - “john” and ‘X’ are atoms.
  - The head must be a single predicate but the body can be a list of predicates.
  - A clause can be read as body implies head.
  - Either the head or the body can be omitted to indicate queries, or facts respectively. Otherwise it is a rule.



Prolog

```
xor(X,Y) :- true(X), false(Y).    //rule
xor(X,Y) :- true(Y), false(X).    //rule
true(bit_1).                      //fact
xor(bit_1, bit_2).                //fact
:- true(bit_2).                   //query
No                                //prolog output
:- xor(A,B).                      //query with vars
A = 0, B = 1;                     //output
A = 1, B = 0;
```

Notes: X,Y,A,B are variables. A comma means “and”. Defining multiple rules for a predicate means “or”.



- Chain of rules

```
witch(X) :- burns(X), female(X).
burns(X) :- wooden(X).
wooden(X) :- floats(X).
floats(X) :- sameweight(duck, X).
female(joan).
sameweight(duck,joan).
:- witch(joan).
Yes
```

- Recursive rules

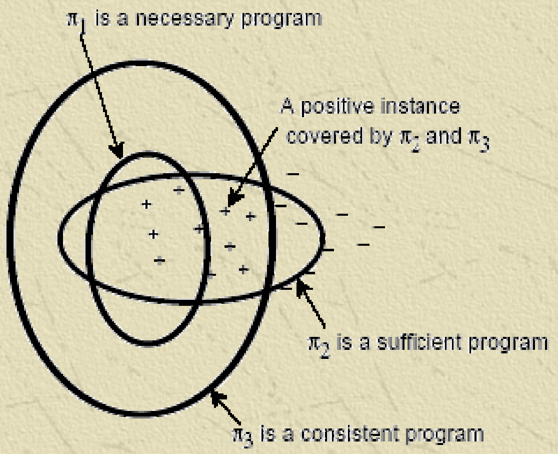
```
ancestor(A, X) :- parent(A, X).      //base case
ancestor(A, X) :- parent(A, C), ancestor(C, X).
```

Inductive Logic Programming

- In Inductive Logic Programming (ILP), we want to learn a logic program that satisfies the training data. Then we can use that logic program to classify future instances. This program is called the **"concept"**
- I will limit my discussion of ILP to problems where we are classifying something as true or false.
- Therefore our training data is a set of true examples, and a set of false examples.
- An ILP must also be given information about the nature of the problem called background information before a logic program can be generated.

ILP terminology

- When a concept returns true for a set of arguments we say the program **covers** those arguments.
- A concept is **sufficient** if it covers all of the positive examples in the training set.
- A concept is **necessary** if it covers none of the negative training examples.
- If a concept is both sufficient and necessary, then it is **consistent**.
- The goal is to find a concept that is consistant.


π_1 is a necessary program
A positive instance covered by π_2 and π_3
π_2 is a sufficient program
π_3 is a consistent program

- Where π is a concept

Concepts

- If a concept is sufficient but not necessary, then we can make it cover fewer examples by specializing it.
- It the opposite is true then we generalize it to make it accept more examples.
- The most general concept can be written in prolog by simply making it a fact for all possible inputs. E.g. “is_carny(X) :- true.”
- Similarly the most special prolog program is “is_carny(X) :- false”

ILP Algorithm

- Some possible ways to search for a concept are...
  1. Start with the most general clause, and specialize until the concept is consistent.
  2. Start with the most specific and generalize until the concept is consistent.
  3. A common approach combines the two. Start with the most general, specialize until the concept is necessary, then generalize until the concept is more sufficient. Repeat until the concept is consistant.

ILP Algorithm

- There are three ways to generalize a concept.
  1. Replace some terms with variables
  2. Remove literals from the body of a clause
  3. Add clause to the program.
- Analogously there are three ways to specialize
  1. Replace some variables with terms
  2. Add literals to the body of a clause
  3. Remove a clause.

Clause :- literal(VARX, term), literal.

ILP Algorithm

- Since we are using an iterative search for the concept, we don't want to lose any information that we gained from the previous iteration.
- Therefore, we can add a clause to the concept to generalize, and add predicates to the body of a clause to specialize it.
- E.g.
  - "equal(X,Y) := X<=y." -> "equal(X,Y) := (X<=Y), (X>=Y)."
  - "equal(X,Y) := X<Y.
    equal(X,Y) := X>Y)."

Predicate/literal restrictions

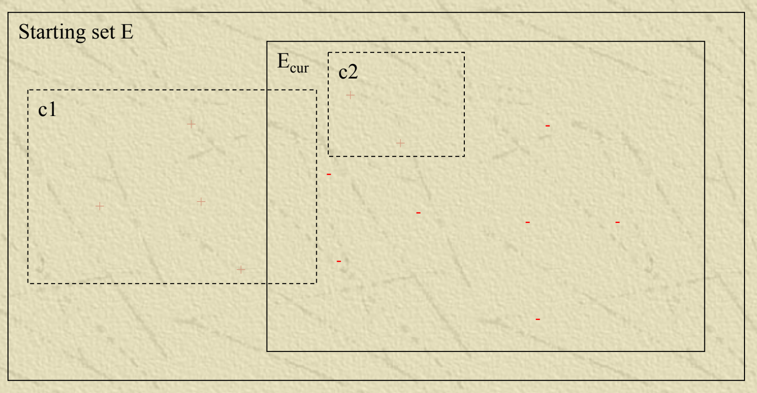
1. Literals used in the background knowledge…
   - whose arguments are a subset of those in the head of the clause.
   - that introduce a new distinct variable different from those in the head of the clause.
2. A literal that equates a variable in the head of the clause with another such variable or with a term mentioned in the background knowledge.
3. A literal must contain at least one existing variable
4. A recursive literal with restrictions on it's arguments to prevent infinite recursion.

A basic ILP algorithm

```
Initialize $\Xi_{cur} := \Xi$.
Initialize $\pi :=$ empty set of clauses.
**repeat** [The outer loop works to make $\pi$ sufficient.]
    Initialize $c := \rho :-$ .
    **repeat** [The inner loop makes $c$ necessary.]
        Select a literal $l$ to add to $c$. [This is a nondeterministic choice point.]
        Assign $c := c, l$.
    **until** $c$ is necessary. [That is, until $c$ covers no negative instances in $\Xi_{cur}$.]
    Assign $\pi := \pi, c$. [We add the clause $c$ to the program.]
    Assign $\Xi_{cur} := \Xi_{cur} -$ (the positive instances in $\Xi_{cur}$ covered by $\pi$).
**until** $\pi$ is sufficient.
```

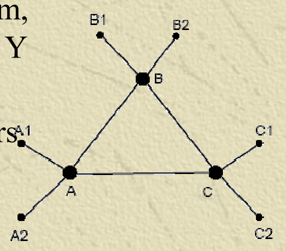
- Σ is the training set. Σ_{cur} is a subset of the training set
- π is the concept we are learning.
- C is the current clause in π that is being specialized.

Example

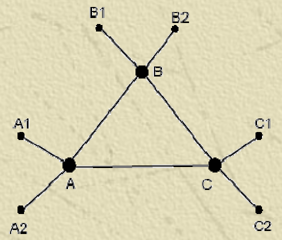


An Example Problem

- We would like to find a logic program, nonstop(X,Y) that will tell us if X to Y is a nonstop flight.
- We give the ILP a training set of pairs of cities that have direct flights. <A,C>,<B,C>,<B2,B>…etc.
- We also give it background information on the structure of the map. Such as satellite(B2,B), and hub(B).



- Training set.
  - E^+={AB, BA, B_2B, BC, BB_1, BB_2, C_2C}
  - E^-={BC_2, B_2A_2, A_1C, AB_2, …}



1. Initialize “nonstop(X,Y) :- true.”
2. Specialize, “nonstop(X,Y):- hub(X).”
   - Covers some negatives {BC_2, AB_2, …}
3. Specialize, “nonstop(X,Y):-hub(X),hub(Y).”
   - Positives not covered {B_2B,BB_1,C_2C}
4. Set E_{cur} = {B_2B,BB_1,C_2C} and E^-
5. Generalize by adding another clause.
6. Specialize again on E_{cur}. “nonstop(X,Y):-satellite(X,Y).”
   - This covers {B_2B, C_2C} but not {BB_1, BB_2}
7. Add “nonstop(X,Y):- satellite(Y,X).”
   - Now all of the positives in the training set are classified, and none of the negatives.

Choosing the Best Literal

- At each step of the inner loop we need to choose a literal from many possible literals. So we want the literal we choose to be a good one.
- We would like to maximize the probability that an example drawn at random from those covered by the new clause is positive.
- This means that to test a literal we need to run the interpreter on the new clause.
  - quality = (# positive examples covered) / (total # covered).

Recursion

- This algorithm can work with recursive clauses, if we are careful about what recursive literals we allow.
- This type of clause must be avoided.

  recur_forever(X,Y) :- recur_forever(X,Y).
- This clause will work

  less_than(X,Y) :- add_1(X,Z), less_than(Z,Y).
  - x<y if there exists a z=x+1, and z<y.
- If we require that there is a partial ordering on an argument of the recursive literal, established by the previous literals.
- Of course a base case would also be good to have.

Post-processing

- It is possible that some of the literals that are generated end up being necessary.
- If we eliminate such literals the program can be made more general.
- A simple way to get rid of useless literals is to test all of the literals against the data. We just check to see if removing a literal will produce the same results.

Noise

- If we have noise it is better to allow some negative examples than to continue until all examples are classified correctly.
- In ILP, the decision to stop is based on the number of bits needed to encode a clause vs the number of bits needed to encode the examples covered by the clause. A good clause should never need as many bits as the examples.
- Intuitively, if we end up with a literal for every positive example we classify, the clause isn't usefull. If this happens we stop.

Bit Encoding

- Number of bits needed to encode examples is

$$\log_2(|T|) + \log_2\left(\binom{|T|}{p}\right)$$

- Number of bits needed to encode a clause is

1	(to indicate whether negated)
$+\log_2$ (number of relations)	(to indicate which relation)
$+\log_2$ (number of possible arguments)	(to indicate which variables)

Advantages of ILP

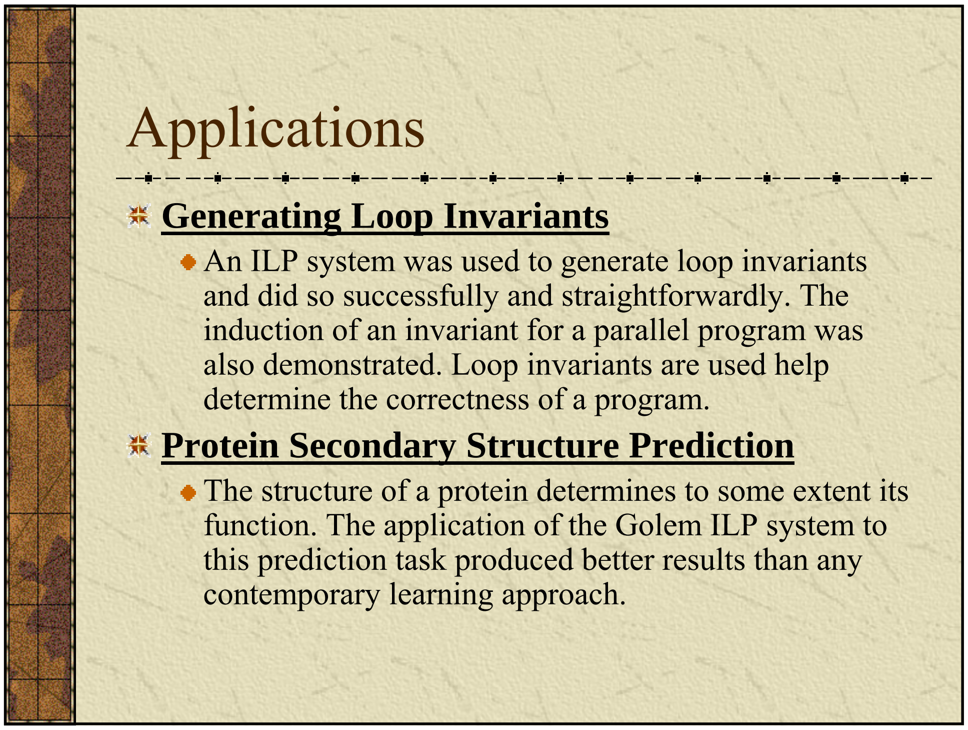
- The biggest advantage ILP has over other machine learning methods it that the results concept, or concepts produced by an ILP algorithm can be understood by the user.
- Some data can better be described using background logic rather than attribute, value pairs. E.g. What if we want to induce a sorting algorithm? This can be done with enough background info.
- Training data can be in the form of logic

Disadvantages

- ILPs generate logic programs. Logic programs can be slow or even intractable to interpret.
- ILPs have a fairly specific domain. ILP would not be used for image recognition for example.

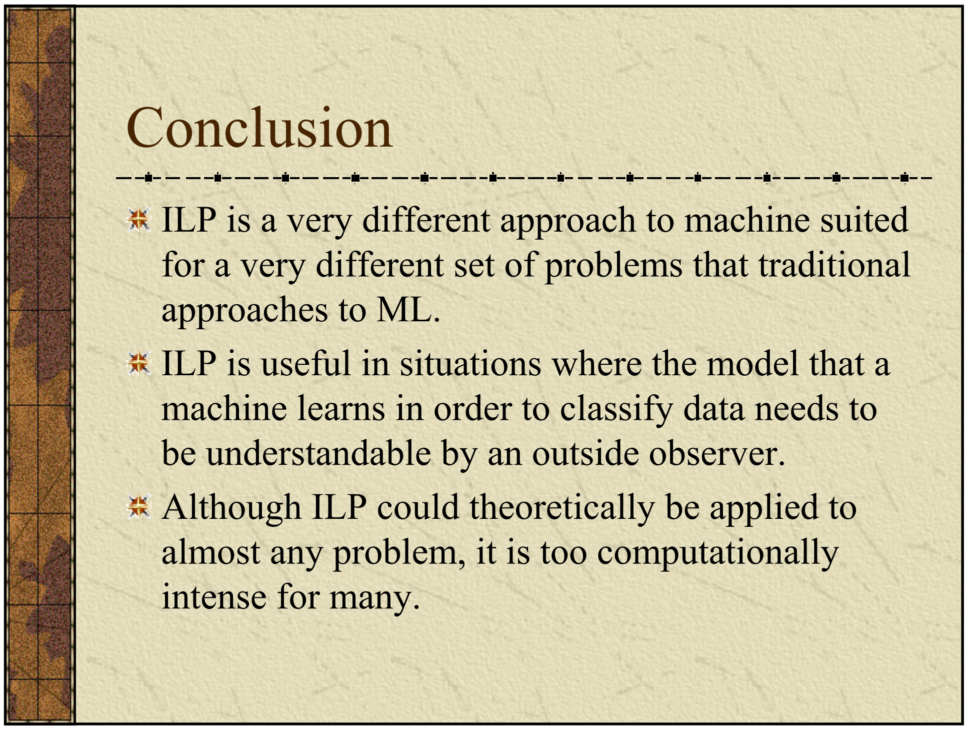
Applications

- **Finite Element Mesh Design**
  - used by engineers to analyze stresses in physical structures. ILP was used to determine rules for the mesh resolution of the structure in terms of certain properties of the structure being modeled. Achieved roughly 78% accuracy
- **Predictive Toxicology**
  - ILP is used to develop a model for determining the toxicity of drugs. In one application, ILP was used to determine toxic molecule, and it's chemical structure. It was 88% accurate.



Applications

- **Generating Loop Invariants**
  - An ILP system was used to generate loop invariants and did so successfully and straightforwardly. The induction of an invariant for a parallel program was also demonstrated. Loop invariants are used help determine the correctness of a program.
- **Protein Secondary Structure Prediction**
  - The structure of a protein determines to some extent its function. The application of the Golem ILP system to this prediction task produced better results than any contemporary learning approach.



Conclusion

- ILP is a very different approach to machine suited for a very different set of problems that traditional approaches to ML.
- ILP is useful in situations where the model that a machine learns in order to classify data needs to be understandable by an outside observer.
- Although ILP could theoretically be applied to almost any problem, it is too computationally intense for many.

References

- Introduction to Machine Learning, Ch. 7, Nills J. Nilsson. http://ai.stanford.edu/people/nilsson/mlbook.html
- Learning Logical Definitions from Relations, J.R. Quinlan (1990)
- Inductive Logic Programming: Techniques and Applications. Nada Lavrac, Saso Dzerosky, (1994)
- Inductive Logic Programming: theory and methods. Stephen Muggleton, Luc De Raedt,
- http://www.doc.ic.ac.uk/~sgc/teaching/v231/lecture14.html