Learning by Answer Sets

Chiaki Sakama

Wakayama University, Japan

Presented at AAAI Spring Symposium on Answer Set Programming, March 2001

Inductive Logic Programming (ILP)

Goal: Inductive construction of first-order clausal theories from examples and background knowledge.

State of the art: mostly handles Horn logic programs or classical clausal theories as background knowledge.

Problems of Horn Logic Programs

- Horn logic programs are monotonic, i.e., adding a new sentence to a program never invalidates previous conclusion.
- Horn LPs are not sufficiently expressive to represent incomplete knowledge and do not characterize nonmonotonic human thinking.

Nonmonotonic Logic Programs (NMLP)

Goal: Representing incomplete knowledge and reasoning with commonsense.

State of the art: non-Horn extensions of logic programming such as programs with negation as failure, explicit negation, disjunction, hypothesis, etc.

Nonmonotonicity in Induction

- Induction is nonmonotonic in the sense that once induced hypotheses might be changed by the introduction of new evidence.
- Induction problems assume background knowledge which is incomplete, otherwise there is no need to learn.
  - Representing and reasoning with incomplete knowledge are vital issues in ILP.

Limitations of Horn ILP

Negation as failure (NAF) is useful to effectively specify exceptions.

B: bird(x) ← penguin(x).
bird(a). bird(b). bird(c). penguin(d).

positive ex.: fly(a), fly(b), fly(c).

negative ex.: fly(d).

a possible hypothesis:

fly(x) ← bird(x), not penguin(x).

- It would require indirect and unsuccinct representaion to specify the same hypothesis using a Horn program.

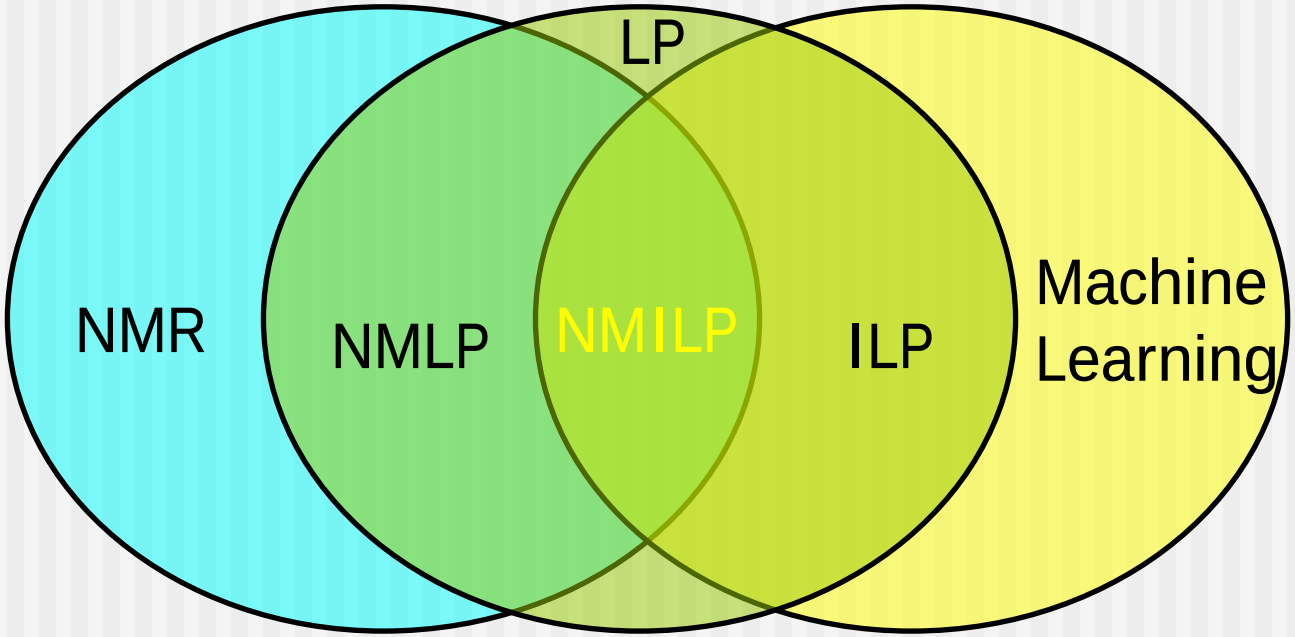
Limitations of Horn ILP

Negation as failure appears in many basic and practical Prolog programs.

e.g. Computing set differences:
$diff_{R-S}(X) \leftarrow R(X), not\ S(X).$

- Horn ILP cannot handle a program containing such a rule with NAF.

Nonmonotonic ILP (NMILP)



- NMILP provides a framework for realizing both commonsense reasoning and inductive learning.

NMILP: Perspectives

- Extend the representation language
- Enable reasoning and learning from default knowledge
- Use well-established theoretical and procedural tools in NMLP
- Open new applications in nonmonotonic problems

What is the problem for NMILP?

- NMLP is different from classical logic, then existing techniques of Horn ILP are not directly applicable to NMILP.
- Extension of the present framework and introduction of new techniques for NMILP are necessary.

Purpose of this Research

- Consider extended logic programs (ELP) as background KB
- Build a theory of nonmonotonic inductive logic programming
- Develop a procedure for inducing nonmonotonic rules to explain positive/negative examples

Contents of This Talk

- Preliminary Definitions
- Learning from Positive Example
- Learning from Negative Example
- Discussion

Preliminary Definitions

- An extended logic program (ELP) is a set of rules of the form:

$$L_0 \leftarrow L_1, \ldots, L_m, \mathit{not}\ L_{m+1}, \ldots, \mathit{not}\ L_n$$

where *not* represents negation as failure(NAF)

- *pred(L)* denotes the predicate of L, and *const(L)* denotes the set of constants in L.
- The semantics of an ELP is given by the answer set semantics (Gelfond&Lifschitz).

Preliminary Definitions

- A program is consistent if it has a consistent answer set.
- A consistent program which has a single answer set is called categorical.
- When a rule R is satisfied in every answer set of a program P, it is written as $P \models R$.

Preliminary Definitions

- A literal L or an NAF formula not L is called an LP-literal.
- Let Lit be the set of all ground literals and $S \subseteq Lit$, then define

  $$S^{+} = S \cup \{ \text{not } L \mid L \in Lit \setminus S \}.$$

- For an LP-literal K,

  $|K| = K$ if K is a literal;

  $|K| = L$ if $K = \text{not } L$.

Learning from Positive Examples: Problem Setting

Given:

- a background knowledge base P as a function-free & categorical ELP
- a positive example L as a ground literal s.t. $P \not\models L$

Find: a rule R satisfying: $P \cup \{R\} \models L$

Proposition

Let P be a program and R a rule s.t. $P \cup \{R\}$ is consistent.

For any ground literal L, $P \cup \{R\} \models L$ and $P \models R$ imply $P \models L$.

Relevance

- For two ground LP-literals L_1 and L_2, $L_1 \sim L_2$ if $pred(L_1) = pred(L_2)$ and $const(L_1) = const(L_2)$.
- Let L be a ground LP-literal and S a set of ground LP-literals. L_1 in S is relevant to L if either (i) $L_1 \sim L$ or (ii) L_1 shares a constant with L_2 in S where L_2 is relevant to L.
- L is involved in a program P if |L| appears in the ground instance of P.

Constructing Hypothesis

Let S be the answer set of P.

Then, $P \cup \{R\} \models L$ and $P \not\models L$ imply $P \not\models R$ (by Proposition), thereby

$$S \not\models R \ .$$

Consider the integrity constraint

$$\leftarrow \Gamma$$

where Γ is the set of ground LP-literals in S^+ s.t. every element in Γ is relevant to L and is involved in $P \cup \{L\}$.

Constructing Hypotheses

As S does not satisfy the constraint,

$S \not\models \leftarrow \Gamma$.

By $S \not\models L$, not L is in S^+ and also in Γ.

Shifting L to the head, we get

$L \leftarrow \Gamma'$

where $\Gamma' = \Gamma \setminus \{ \text{not } L \}$.

Finally, construct a rule R* s.t.

$R^*\theta = L \leftarrow \Gamma'$ for some substitution θ.

Example

P: bird(x)← penguin(x),

bird(tweety)←, penguin(polly)←.

L: flies(tweety).

Initially, P|≠flies(tweety).

S^+ = {bird(t), bird(p), peng(p),
not peng(t), not flies(t), not flies(p),
not¬ bird(t/p), not¬ peng(t/p),
not¬ flies(t/p) } (t/p means t or p).

Example (cont.)

Picking up LP-literals which are relevant to L and are involved in $P \cup \{L\}$:

$\leftarrow$ bird(t),not peng(t),not flies(t).

Shifting L=flies(t) to the head:

flies(t)$\leftarrow$ bird(t), not peng(t).

Replacing tweety by a variable *x*:

R* = flies(x)$\leftarrow$ bird(x), not peng(x)

where $P \cup \{R^*\} \models$ flies(tweety) holds.

Correctness Theorem

A rule R is negative-cycle-free if for any not L in body(R), $pred(L) \neq pred(head(R))$.

Let P be a categorical program, L a ground literal, and R* a rule obtained as above. If R* is negative-cycle-free and pred(L) appears nowhere in P, then $P \cup \{R^*\} \models L$.

Learning from Negative Examples: Problem Setting

Given:

- a background knowledge base P as a function-free & categorical ELP
- a negative example L as a ground literal s.t. $P \models L$

Find: a rule R satisfying:

$$P \cup \{R\} \not\models L$$

Target Predicate

- A target predicate is a pre-specified predicate which is subject to learn.
- In case of positive examples, a target predicate is identified with the one appearing in the example.
- In case of negative examples, a negative example L is already entailed from P. The purpose is then to block the derivation of L by introducing some rule R to P.
- We set a target predicate which is different from the one appearing in L.

Selection of Target Predicate

- In a predicate dependency-graph, p1 strongly depends on p2 if for any path containing p1, p1 depends on p2.
- p1 negatively depends on p2 if any path from p1 to p2 contains an odd number of negative edges.
- We select a target predicate from predicates in P on which pred(L) strongly and negatively depends.

Proposition

Let P be a program and R a rule s.t. $P \cup \{R\}$ is consistent.

For any ground literal L, $P \cup \{R\} \not\models L$ and $P \models R$ imply $P \not\models L$.

Constructing Hypothesis

Let S be the answer set of P.

Then, $P \cup \{R\} \not\models L$ and $P \models L$ imply $P \not\models R$ (by Proposition), thereby

$$S \not\models R \ .$$

Consider the integrity constraint

$$\leftarrow \Gamma$$

where Γ is the set of ground LP-literals in S^+ s.t. every element in Γ is relevant to L and is involved in $P \cup \{L\}$.

Constructing Hypotheses

As S does not satisfy the constraint,

$$S \not\models \leftarrow \Gamma.$$

If Γ contains not K which has the target predicate, shifting K to the head, we get

$$K \leftarrow \Gamma'$$

where $\Gamma' = \Gamma \setminus \{ \text{not } K \}$.

Finally, construct a rule R* s.t.

$R^*\theta = L \leftarrow \Gamma'$ for some substitution θ.

Example

P: bird(x)← bird(x), not ab(x),

bird(x)← penguin(x),

bird(tweety)←, penguin(polly)←.

L: flies(polly).

Initially, P|= flies(polly).

S^+ = {bird(t), bird(p), peng(p), not peng(t), flies(t), flies(p), not ab(t), not ab(p), not¬ bird(t/p), not¬ peng(t/p), not¬ flies(t/p), not¬ ab(t/p) }.

Example (cont.)

Picking up LP-literals which are relevant to L and are involved in $P \cup \{L\}$:

$\leftarrow$ bird(p),peng(p),flies(p),not ab(p).

Let ab be the target predicate on which flies strongly and negatively depends. Shifting L=ab(p) to the head:

ab(p)$\leftarrow$ bird(p),peng(p),flies(p).

Replacing polly by *x*:

R* = ab(x)$\leftarrow$ bird(x),peng(x),flies(x).

Example (cont.)

In this case, however, $P \cup \{R^*\}$ is inconsistent. To get a consistent program, dropping flies(x) from R*, we get

$R^{**} = ab(x) \leftarrow bird(x), peng(x)$

where $P \cup \{R^{**}\} \not\models flies(p)$ holds.

The rule R** is further simplified as

$ab(x) \leftarrow peng(x)$

using the second rule in P.

Correctness Theorem

Let P be a categorical program, L a ground literal, and R* a rule obtained as above. If $P \cup \{R^*\}$ is consistent and $P \cup \{R^*\theta\} \models L$ for some substitution θ, then $P \cup \{R^*\} \models L$.

Discussion
Learning from Multiple Ex's

- Our algorithm is applicable to learning from a set of examples by iteratively applying the procedure to each example.
- The result of induction depends on the order of examples in general.

Learning from Non-Categorical Programs

- When a program has more than one answer set, different rules are induced by each answer set.
- For example, the program

  $p(a) \leftarrow not\ q(a), \quad q(a) \leftarrow not\ p(a)$

has two answer sets: $\{p(a)\},\{q(a)\}$.

Given the positive example $r(a)$, applying the procedure to each AS produces

$r(x) \leftarrow p(x), not\ q(x)$ and

$r(x) \leftarrow q(x), not\ p(x).$

Correctness and Completeness

- We provided sufficient conditions to guarantee the correctness of the procedure.
- On the other hand, the procedure is not complete in general.
- There exist possibly infinite solutions for explaining examples, so that selecting meaningful hypotheses is important.
- In our algorithm, conditions of relevance and involvement are used to filter out useless hypotheses.

Computability

- We consider function-free and categorical ELPs.
- With the f-f setting, S^+ is finite and the selection of relevant and involved literals from S^+ is done in polynomial time.
- An important class of categorical programs is stratified programs.
  In a f-f stratified program, an inductive hypothesis R* is efficiently constructed from S^+.

Connection to Answer Set Programming (ASP)

- ASP solves a problem by computing answer sets which correspond to solutions.
- We constructed inductive hypotheses by computing answer sets of an ELP.
- This enables us to use existing ASP solvers for computing induction in NMLP.

Summary

- This paper introduced a method of inductive learning in nonmonotonic LPs.
- The result combines techniques of the two important fields of logic programming, NMLP and ILP, and contributes to a theory of NMILP.
- The paper is available from my HP:

  http://www.wakayama-u.ac.jp/~sakama