Heuristic Based Induction of Answer Set Programs: From Default Theories to Combinatorial problems

28th International Conference on Inductive Logic Programming - ILP 2018

Farhad Shakerin, Gopal Gupta
{fxs130430,gupta}@utdallas.edu

September 2, 2018

The University of Texas at Dallas

Agenda

- ILP Problem Definition
- FOLD Algorithm
- ILP Problem Definition Revisited (1)
- Heuristic-based Non-Observation Learning
- ILP Problem Definition Revisited (2)
- Induction of ASP programs
- Applications

ILP Problem Definition

Given

- Target predicate t, target ground predicates E^+ and E^-, Horn background knowledge B with no even cycles

Find hypothesis H such that:

- $\forall e \in E^+ : B \cup H \models e$
- $\forall e \in E^- : B \cup H \not\models e$
- B and H are consistent

In This Paper: A Heuristic-based algorithm to relax the following limitations

- Only Target Ground Predicates in E^+, E^- $\Longrightarrow$ Non-Observation Predicate Learning
- Horn BK $\Longrightarrow$ Learning ASP programs

Negation-As-Failure Semantics (NAF)

- NAF is essential in common sense reasoning (e.g., default reasoning, incomplete knowledge)
- **Even cycles** (i.e., even number of **not** until reaching the same predicate) are responsible for generating multiple stable models in presence of NAF semantics
- **Example:**

```
p :- not q.
q :- not p.
```

- Once NAF is allowed in ILP, algorithm should be able to handle ASP programs with multiple *stable models*

Motivation

- Learning ASP is not a new problem
- Why a Heuristic-based Algorithm?
  - Greedy search vs. Exhaustive search (Scalability)
  - Noise tolerance (Over-fitting)
- We extend FOIL algorithm (R.Quinlan 90) to induce Answer Set Programs with multiple Stable Models
- FOLD (**F**irst **O**rder **L**earner of **D**efault-theories) is introduced to learn default theories with one stable model
- We then extend it to multiple generate/handle multiple stable models

Agenda

- ILP Problem Definition
- **FOLD Algorithm**
- ILP Problem Definition Revisited (1)
- Heuristic-based Non-Observation Learning
- ILP Problem Definition Revisited (2)
- Induction of ASP programs
- Applications

FOLD algorithm

FOLD algorithm extends FOIL while learning default and possibly multiple exceptions for a concept.

First Order Learner of Default-theories (FOLD)

- Specialize $target(V_1, ...V_m)$:- $true$ only by adding positive literals
- Stop once information gain turns 0 or the *Maximum Description Length* reached (whichever happens first) save current literals $p_1, ..., p_k$
- Switch the current positive and negative examples
- recursively call FOLD to learn a predicate called ab
- Add the following rule to the current hypothesis $target(V_1, ...V_m)$:- $p_1, ..., p_k$, **not** $ab(V_1, ..., V_m)$

FOLD Step by Step

```
fly(X) :- ?
```

Example

$\mathscr{B}$:	$bird(X) \leftarrow penguin(X).$	
	$bird(tweety).$	$bird(woody).$
	$cat(kitty).$	$penguin(polly).$
$\mathscr{E}^+$:	$fly(tweety).$	$fly(woody).$
$\mathscr{E}^-$:	$fly(polly).$	$fly(kitty).$

List of candidate predicates:

```
bird(X),cat(X),penguin(X)
```

Initially...

$fly(X) \leftarrow true.$ $E^+ = [\text{tweety,woody}]$ $E^- = [\text{polly,kitty}]$

FOLD Step by Step

`fly(X) :- ?`

Example

$\mathscr{B}$:	$bird(X) \leftarrow penguin(X).$	
	$bird(tweety).$	$bird(woody).$
	$cat(kitty).$	$penguin(polly).$
$\mathscr{E}^+$:	$fly(tweety).$	$fly(woody).$
$\mathscr{E}^-$:	$fly(polly).$	$fly(kitty).$

List of candidate predicates:

`bird(X),cat(X),penguin(X)`

Trying...`bird(X)`

$fly(X) \leftarrow bird(X)$ $\quad E^+ =$ [tweety,woody] $\quad E^- =$ [polly]

FOLD Step by Step

```
fly(X) :- ?
```

Example

$\mathscr{B}$: $bird(X) \leftarrow penguin(X).$

$bird(tweety).$ $bird(woody).$

$cat(kitty).$ $penguin(polly).$

$\mathscr{E}^+$: $fly(tweety).$ $fly(woody).$

$\mathscr{E}^-$: $fly(polly).$ $fly(kitty).$

List of candidate predicates:

`bird(X),cat(X),penguin(X)`

Trying...`cat(X)`

$fly(X) \leftarrow cat(X) \quad E^+ = [] \quad E^- = [\text{kitty}]$

FOLD Step by Step

```
fly(X) :- ?
```

Example

$\mathscr{B}$: $bird(X) \leftarrow penguin(X).$

$bird(tweety).$ $bird(woody).$

$cat(kitty).$ $penguin(polly).$

$\mathscr{E}^+$: $fly(tweety).$ $fly(woody).$

$\mathscr{E}^-$: $fly(polly).$ $fly(kitty).$

List of candidate predicates:

`bird(X),cat(X),penguin(X)`

Trying...`penguin(X)`

$fly(X) \leftarrow penguin(X)$ $E^+ = []$ $E^- = [\text{polly}]$

FOLD Step by Step

```
fly(X) :- ?
```

Example

$\mathscr{B}:$ $bird(X) \leftarrow penguin(X).$

$bird(tweety).$ $bird(woody).$

$cat(kitty).$ $penguin(polly).$

$\mathscr{E}^+:$ $fly(tweety).$ $fly(woody).$

$\mathscr{E}^-:$ $fly(polly).$ $fly(kitty).$

After first iteration

$fly(X) \leftarrow bird(X).$ $E^+ =$ [tweety,woody] $E^- =$[polly]

FOLD Step by Step

```
fly(X) :- ?
```

Example

$\mathscr{B}:$ $bird(X) \leftarrow penguin(X).$

$bird(tweety).$ $bird(woody).$

$cat(kitty).$ $penguin(polly).$

$\mathscr{E}^+:$ $fly(tweety).$ $fly(woody).$

$\mathscr{E}^-:$ $fly(polly).$ $fly(kitty).$

information gain turns 0...

$fly(X) \leftarrow bird(X).$ $E^+ = [\text{tweety,woody}]$ $E^- = [\text{polly}]$

FOLD Step by Step

```
fly(X) :- ?
```

Example

$\mathscr{B}$:	$bird(X) \leftarrow penguin(X).$	
	$bird(tweety).$	$bird(woody).$
	$cat(kitty).$	$penguin(polly).$
$\mathscr{E}^+$:	$fly(tweety).$	$fly(woody).$
$\mathscr{E}^-$:	$fly(polly).$	$fly(kitty).$

At this point, FOLD swaps $E^+ =$ [tweety,woody] and $E^- =$ [polly] and recursively calls FOLD

Initially

$ab(X) \leftarrow true.$ $E^+ =$ [polly] $E^- =$ [tweety,woody]

FOLD Step by Step

```
fly(X) :- ?
```

Example

$\mathscr{B}:$	$bird(X) \leftarrow penguin(X).$	
	$bird(tweety).$	$bird(woody).$
	$cat(kitty).$	$penguin(polly).$
$\mathscr{E}^+:$	$fly(tweety).$	$fly(woody).$
$\mathscr{E}^-:$	$fly(polly).$	$fly(kitty).$

After first iteration

$ab(X) \leftarrow penguin(X).$ $E^+ = [\text{polly}]$ $E^- = []$

FOLD Step by Step

```
fly(X) :- ?
```

Example

$\mathscr{B}$:	$bird(X) \leftarrow penguin(X)$.	
	$bird(tweety)$.	$bird(woody)$.
	$cat(kitty)$.	$penguin(polly)$.
$\mathscr{E}^+$:	$fly(tweety)$.	$fly(woody)$.
$\mathscr{E}^-$:	$fly(polly)$.	$fly(kitty)$.

Since no negative example left, recursive call returns. The original call to FOLD returns with the following default theory:

```
fly(X) :- bird(X), not ab(X).
ab(X) :- penguin(X).
```

Agenda

- ILP Problem Definition
- FOLD Algorithm
- **ILP Problem Definition Revisited (1)**
- Heuristic-based Non-Observation Learning
- ILP Problem Definition Revisited (2)
- Induction of ASP programs
- Applications

ILP Problem Definition (Revisited)

Given

- Target predicate `t`, target ground predicates E^+ and E^-, Horn background knowledge B with no even cycles

Find hypothesis H such that:

- $\forall e \in E^+ : B \cup H \models e$
- $\forall e \in E^- : B \cup H \not\models e$
- B and H are consistent

In This Paper: A Heuristic-based algorithm to relax the following limitations

1. **Only Target Ground Predicates in E^+, E^- $\Longrightarrow$ Non-Observation Predicate Learning**
2. Horn BK $\Longrightarrow$ Learning ASP programs

Non-Observation Predicate Learning

- Allows to have examples other than the ground target predicate
- These non-target examples impact the hypothesis indirectly
- **Example:** Learning target = `r(X)` with $E^+ = \{$`p(a), r(c)`$\}$, $E^- = \{$`p(d)`$\}$

```
(1) p(X) :- s(X), not r(X).
(2) s(X) :- q(X,Y), r(Y).
(3) q(a,b).
(4) s(d).
```

- $H \models$ `p(a)` requires: $H \models$`s(a)`, $H \not\models$ `r(a)`
- `s(a)` requires: `r(b)`
- $H \not\models$ `p(d)` requires: $H \not\models$ `r(d)`

Abduction in Goal-Directed ASP

- Non-OPL is realized via s(ASP), a Goal-Directed ASP system
  - Takes an ASP program and a query Q
  - enumerates all answer sets that contain propositions/predicates of Q
  - Enumeration employs co-inductive SLD resolution to systematically compute the elements of the Greatest Fix-Point (GFP) of P.
  - It outputs "partial answer sets" that contain only elements necessary to establish Q.
  - s(ASP) does not ground the ASP program
- s(ASP) allows to run a query Q abductively
- Whenever needed, Q succeeds by assuming facts from a set defined of #abducibles

Abduction in Goal-Directed ASP (2)

- Non-OPL is realized via s(ASP), a Goal-Directed ASP system
  - Takes an ASP program and a query Q
  - enumerates all answer sets that contain propositions/predicates of Q
  - Enumeration employs co-inductive SLD resolution to systematically compute the elements of the Greatest Fix-Point (GFP) of P.
  - It outputs "partial answer sets" that contain only elements necessary to establish Q.
  - s(ASP) does not ground the ASP program
- s(ASP) allows to run a query Q abductively
- Whenever needed, Q succeeds by assuming facts from a set defined of #abducibles

Abduction in Goal-Directed ASP (3)

Example: Learning target = `r(X)` with E^+ ={`p(a), r(c)`}, E^- ={`p(d)`}

```
(1) p(X) :- s(X), not r(X).
(2) s(X) :- q(X,Y), r(Y).
(3) q(a,b).
(4) s(d).
```

We Run s(ASP) System and define #abducible `r(x)`

- `?- p(a)`
- **Partial Answer set** {p(a), q(a,b), r(b), s(a), not r(a)}
- `?- not p(d)`
- **Partial Answer set** {p(d), s(d), r(d)}
- $E^+ =$ {`r(c),r(b),r(d)`}
- $E^- =$ {`r(a)`}

ILP Problem Definition (Revisited)

Given

- Target predicate `t`, target ground predicates E^+ and E^-, Horn background knowledge B with no even cycles

Find hypothesis H such that:

- $\forall e \in E^+ : B \cup H \models e$
- $\forall e \in E^- : B \cup H \not\models e$
- B and H are consistent

In This Paper: A Heuristic-based algorithm to relax the following limitations

1. Only Target Ground Predicates in E^+, E^- $\implies$ Non-Observation Predicate Learning
2. **Horn BK** $\implies$ **Learning ASP programs**

Agenda

- ILP Problem Definition
- FOLD Algorithm
- ILP Problem Definition Revisited (1)
- Heuristic-based Non-Observation Learning
- ILP Problem Definition Revisited (2)
- **Induction of ASP programs**
- Applications

Induction of ASP Programs with Multiple Stable Models

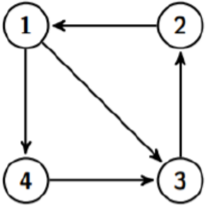
- $B \cup H$ has multiple stable models
- Examples $e \in E^+$ hold only in some of the stable models of $B \cup H$
- We adopt the ILASP[1] *partial interpretation* as the ILP framework to induce programs with multiple stable models

[1] M. Law, A. Russo,K. Broda, Inductive Learning of Answer Set Programs, Logics in Artificial Intelligence,Springer, 2014

Induction of ASP Programs with Multiple Stable Models

Definition A partial interpretation E is a pair $E = \langle E^{inc}, E^{exc} \rangle$ of sets of ground atoms called inclusions and exclusions, respectively. Let $A \in AS(B \cup H)$ denote a stable model of $B \cup H$. A *extends* $\langle E^{inc}, E^{exc} \rangle$ if and only if $(E^{inc} \subseteq A) \wedge (E^{exc} \cap A = \emptyset)$

Example: The graph-coloring problem



- Positive examples:

  $E_1^+ = \{\langle r(1), b(2), g(3), b(4) \rangle, \langle not\ b(1), not\ g(1), not\ r(2),$
  $not\ g(2), not\ r(3), not\ b(3), not\ r(4), not\ g(4) \rangle\}$

  $E_2^+ = \{\langle b(1), r(2), g(3), r(4) \rangle, \langle not\ r(1), not\ g(1), not\ b(2),$
  $not\ g(2), not\ r(3), not\ b(3), not\ b(4), not\ g(4) \rangle\}$

- Negative examples:

  $E_1^- = \{\langle r(1) \rangle, \langle not\ r(2) \rangle\}$ $\quad E_2^- = \{\langle r(1) \rangle, \langle not\ r(3) \rangle\}$

ILP Problem Definition Revisited

Given

- Target predicate `t`
- Two sets of partial interpretations E^+ and E^-
- ASP background knowledge B

Find hypothesis H such that:

- $\forall e^+ \in E^+ \exists A \in AS(B \cup H)$ such that A extends e^+
- $\forall e^- \in E^- \nexists A \in AS(B \cup H)$ such that A extends e^-

FOLD algorithm is extended to solve instances of this problem

Example - “Who goes to the party?”

Background Knowledge

```
conflict(X,Y) :- conflict(Y,X).
:- person(X), works(X), off(X).
person(p1). person(p2). person(p3). person(p4). person(p5).
conflict(p1,p4).
conflict(p2,p3).
```

Target Predicate: `goesToParty(X):- ?`

Positive Examples

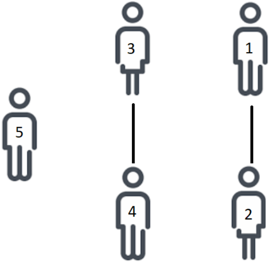
$E_1^+ = \{\langle$ g(p1),g(p2),o(p1),o(p2),w(p3),o(p4),w(p5) $\rangle, \langle$ g(p3),g(p4),g(p5) $\rangle\}$

$E_2^+ = \{\langle$ g(p3),g(p4),g(p5),o(p1),o(p2),o(p3),o(p4),o(p5) $\rangle, \langle$ g(p1),g(p2) $\rangle\}$

$E_3^+ = \{\langle$ g(p1),g(p3),g(p5),o(p1),o(p2),o(p3),w(p4),o(p5) $\rangle, \langle$ g(p2),g(p4) $\rangle\}$

$E_4^+ = \{\langle$ g(p2),g(p5),w(p1),o(p2),w(p3),w(p4),o(p5) $\rangle, \langle$ g(p1),g(p3),g(p4) $\rangle\}$

Example - “Who goes to the party?”



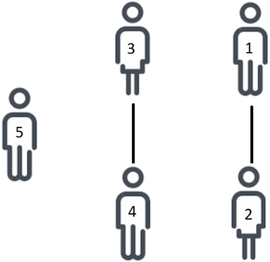
$E_1^+ = \{\langle$ ~~g(p1),g(p2)~~,o(p1),o(p2),w(p3),o(p4),w(p5) $\rangle, \langle$ g(p3),g(p4),g(p5) $\rangle\}$
$E_2^+ = \{\langle$ ~~g(p3),g(p4),g(p5)~~,o(p1),o(p2),o(p3),o(p4),o(p5) $\rangle, \langle$ g(p1),g(p2) $\rangle\}$
$E_3^+ = \{\langle$ ~~g(p1),g(p3),g(p5)~~,o(p1),o(p2),o(p3),w(p4),o(p5) $\rangle, \langle$ g(p2),g(p4) $\rangle\}$

$E_4^+ = \{\langle$ ~~g(p2),g(p5)~~,w(p1),o(p2),w(p3),w(p4),o(p5) $\rangle, \langle$ g(p1),g(p3),g(p4) $\rangle\}$

Trying...

```
goesToParty(X) :- true.
```

Example - "Who goes to the party?"



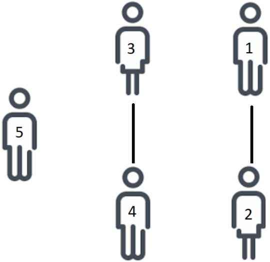
$E_1^+ = \{\langle$ g(p1),g(p2),o(p1),o(p2),w(p3),o(p4),w(p5) $\rangle, \langle$ g(p3),~~g(p4)~~,g(p5) $\rangle\}$
$E_2^+ = \{\langle$ g(p3),g(p4),g(p5),o(p1),o(p2),o(p3),o(p4),o(p5) $\rangle, \langle$ ~~g(p1),g(p2)~~ $\rangle\}$
$E_3^+ = \{\langle$ g(p1),g(p3),g(p5),o(p1),o(p2),o(p3),w(p4),o(p5) $\rangle, \langle$ ~~g(p2)~~,g(p4) $\rangle\}$

$E_4^+ = \{\langle$ g(p2),g(p5),w(p1),o(p2),w(p3),w(p4),o(p5) $\rangle, \langle$ ~~g(p1),g(p3),g(p4)~~ $\rangle\}$

Trying... `works(X)`

```
goesToParty(X) :- works(X).
```

Example - "Who goes to the party?"



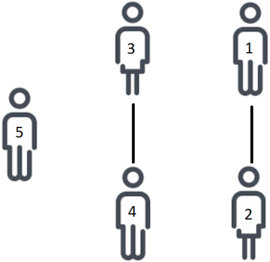
$E_1^+ = \{\langle\ \cancel{g(p1)},\cancel{g(p2)},o(p1),o(p2),w(p3),o(p4),w(p5)\ \rangle,\langle\ \cancel{g(p3)},g(p4),\cancel{g(p5)}\ \rangle\}$
$E_2^+ = \{\langle\ \cancel{g(p3)},\cancel{g(p4)},\cancel{g(p5)},o(p1),o(p2),o(p3),o(p4),o(p5)\ \rangle,\langle\ g(p1),g(p2)\ \rangle\}$
$E_3^+ = \{\langle\ \cancel{g(p1)},\cancel{g(p3)},\cancel{g(p5)},o(p1),o(p2),o(p3),w(p4),o(p5)\ \rangle,\langle\ g(p2),\cancel{g(p4)}\ \rangle\}$

$E_4^+ = \{\langle\ \cancel{g(p2)},\cancel{g(p5)},w(p1),o(p2),w(p3),w(p4),o(p5)\ \rangle,\langle\ \cancel{g(p1)},\cancel{g(p3)},\cancel{g(p4)}\ \rangle\}$

Trying... `off(X)`

```
goesToParty(X) :- off(X).
```

Example - "Who goes to the party?"

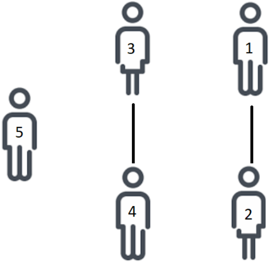


$$E_1^+ = \{\langle\ \cancel{g(p1)},\cancel{g(p2)},o(p1),o(p2),w(p3),o(p4),w(p5)\ \rangle,\langle\ \cancel{g(p3)},g(p4),\cancel{g(p5)}\ \rangle\}$$
$$E_2^+ = \{\langle\ \cancel{g(p3)},\cancel{g(p4)},\cancel{g(p5)},o(p1),o(p2),o(p3),o(p4),o(p5)\ \rangle,\langle\ g(p1),g(p2)\ \rangle\}$$
$$E_3^+ = \{\langle\ \cancel{g(p1)},\cancel{g(p3)},\cancel{g(p5)},o(p1),o(p2),o(p3),w(p4),o(p5)\ \rangle,\langle\ g(p2),\cancel{g(p4)}\ \rangle\}$$
$$E_4^+ = \{\langle\ \cancel{g(p2)},\cancel{g(p5)},w(p1),o(p2),w(p3),w(p4),o(p5)\ \rangle,\langle\ \cancel{g(p1)},\cancel{g(p3)},\cancel{g(p4)}\ \rangle\}$$

After 1st iteration

```
goesToParty(X) :- off(X).
```

Example - “Who goes to the party?”

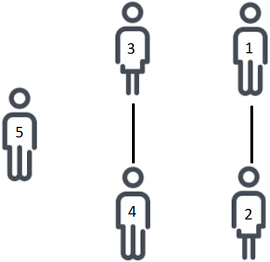


$E_1^+ = \{\langle\ \cancel{g(p1)},\cancel{g(p2)},o(p1),o(p2),w(p3),o(p4),w(p5)\ \rangle,\langle\ \cancel{g(p3)},g(p4),\cancel{g(p5)}\ \rangle\}$
$E_2^+ = \{\langle\ \cancel{g(p3)},\cancel{g(p4)},\cancel{g(p5)},o(p1),o(p2),o(p3),o(p4),o(p5)\ \rangle,\langle\ g(p1),g(p2)\ \rangle\}$
$E_3^+ = \{\langle\ \cancel{g(p1)},\cancel{g(p3)},\cancel{g(p5)},o(p1),o(p2),o(p3),w(p4),o(p5)\ \rangle,\langle\ g(p2),\cancel{g(p4)}\ \rangle\}$

After 1st iteration

```
goesToParty(X) :- off(X).
```

Example - “Who goes to the party?”

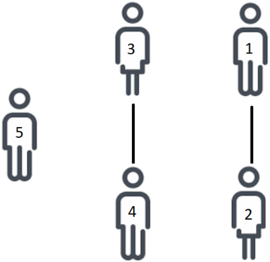


$E_1^+ = \{\langle$ ~~g(p1),g(p2)~~,o(p1),o(p2),w(p3),o(p4),w(p5) $\rangle, \langle$ ~~g(p3)~~,g(p4),~~g(p5)~~ $\rangle\}$
$E_2^+ = \{\langle$ ~~g(p3),g(p4),g(p5)~~,o(p1),o(p2),o(p3),o(p4),o(p5) $\rangle, \langle$ g(p1),g(p2) $\rangle\}$
$E_3^+ = \{\langle$ ~~g(p1),g(p3),g(p5)~~,o(p1),o(p2),o(p3),w(p4),o(p5) $\rangle, \langle$ g(p2),~~g(p4)~~ $\rangle\}$

No more gainful candidate predicate, Swapping E^{inc},E^{exc}

```
goesToParty(X) :- off(X)
```

Example - "Who goes to the party?"

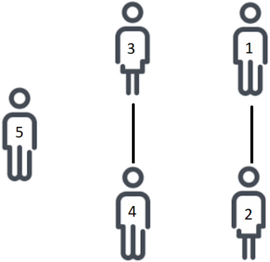


$E_1^+ = \{\langle$ -g(p4),g(p1),g(p2),o(p1),o(p2),w(p3),o(p4),w(p5) $\rangle, \langle$ -g(p1),-g(p2) $\rangle\}$
$E_2^+ = \{\langle$ -g(p1),-g(p2),g(p3),g(p4),g(p5),o(p1),o(p2),o(p3),o(p4),o(p5) $\rangle, \langle$ -g(p3),-g(p4),-g(p5) $\rangle\}$
$E_3^+ = \{\langle$ -g(p2),g(p1),g(p3),g(p5),o(p1),o(p2),o(p3),w(p4),o(p5) $\rangle, \langle$ -g(p1),-g(p2) $\rangle\}$

Learn -goesToParty(X)

```
goesToParty(X) :- off(X),not -goesToParty(X)
-goesToParty(X) :- true.
```

Example - "Who goes to the party?"

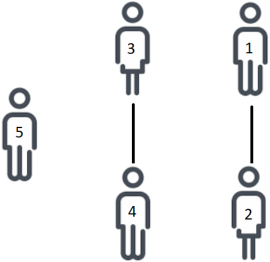


$E_1^+ = \{\langle$ g(p1),g(p2),o(p1),o(p2),w(p3),o(p4),w(p5) $\rangle, \langle$ -g(p1),-g(p2) $\rangle\}$
$E_2^+ = \{\langle$ g(p3),g(p4),g(p5),o(p1),o(p2),o(p3),o(p4),o(p5) $\rangle, \langle$ -g(p3),-g(p4),-g(p5) $\rangle\}$
$E_3^+ = \{\langle$ g(p1),g(p3),g(p5),o(p1),o(p2),o(p3),w(p4),o(p5) $\rangle, \langle$ -g(p1),-g(p2) $\rangle\}$

Trying... `conflict(X,Y)`

```
goesToParty(X) :- off(X),not -goesToParty(X)
-goesToParty(X) :- conflict(X,Y),
```

Example - “Who goes to the party?”

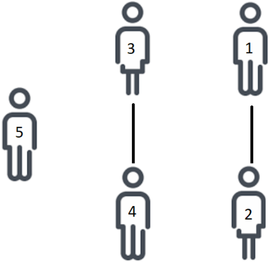


$E_1^+ = \{\langle$ g(p1),g(p2),o(p1),o(p2),w(p3),o(p4),w(p5) $\rangle, \langle\ \rangle\}$
$E_2^+ = \{\langle$ g(p3),g(p4),g(p5),o(p1),o(p2),o(p3),o(p4),o(p5) $\rangle, \langle\ \rangle\}$
$E_3^+ = \{\langle$ g(p1),g(p3),g(p5),o(p1),o(p2),o(p3),w(p4),o(p5) $\rangle, \langle\ \rangle\}$

Trying... `goesToParty(Y)`

```
goesToParty(X) :- off(X),not -goesToParty(X)
-goesToParty(X) :- conflict(X,Y),goesToParty(Y)
```

Example - "Who goes to the party?"



$E_1^+ = \{\langle$ g(p1),g(p2),o(p1),o(p2),w(p3),o(p4),w(p5) $\rangle, \langle\ \rangle\}$
$E_2^+ = \{\langle$ g(p3),g(p4),g(p5),o(p1),o(p2),o(p3),o(p4),o(p5) $\rangle, \langle\ \rangle\}$
$E_3^+ = \{\langle$ g(p1),g(p3),g(p5),o(p1),o(p2),o(p3),w(p4),o(p5) $\rangle, \langle\ \rangle\}$

Done...

```
goesToParty(X) :- off(X),not -goesToParty(X)
-goesToParty(X) :- conflict(X,Y),goesToParty(Y)
```

Agenda

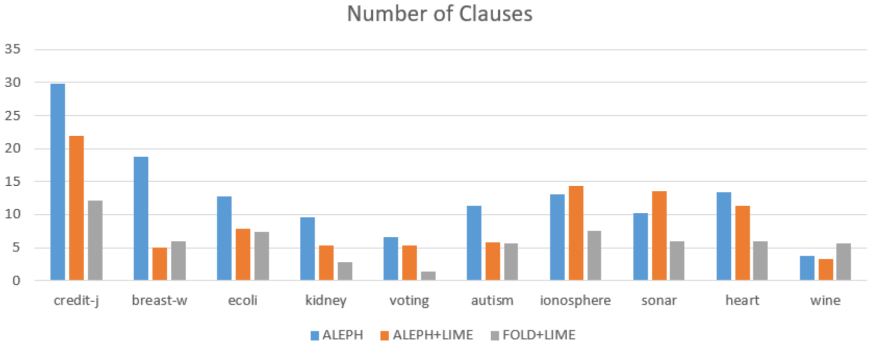
- ILP Problem Definition
- FOLD Algorithm
- ILP Problem Definition Revisited (1)
- Heuristic-based Non-Observation Learning
- ILP Problem Definition Revisited (2)
- Induction of ASP programs
- **Applications**

FOLD Algorithm Applications

- Inductive Learning of Default Theories
- Learning *generate and test* ASP programs for combinatorial Problems (e.g., Graph Coloring and N-Queen)
  - *generate* part is learned from positive examples E^+
  - *test* part is learned from negative examples E^+, in form of ASP constraints (even loops)
- **Induction of Non-monotonic Logic Programs to Explain Boosted Tree Models Using LIME**
  - LIME (**L**ocally **I**nterpret **M**odel-agnostic **E**xplanations) explains complex classifiers decisions
  - For every training example LIME filters out irrelevant features
  - FOLD learns very succinct set of clauses from the data set transformed by LIME

FOLD+LIME[2]

FOLD and ALEPH comparison on UCI Standard benchmarks transformed by LIME



[2] http://arxiv.org/abs/1808.00629

FOLD+LIME[3]

	Algorithm											
Data Set	ALEPH				ALEPH+LIME				**FOLD+LIME**			
credit-j	Prec.	Recall	Acc.	F1	Prec.	Recall	Acc.	F1	Prec.	Recall	Acc.	F1
breast-w	92.8	0.87	0.93	0.89	**0.98**	0.65	0.87	0.76	0.94	**0.92**	**0.95**	**0.92**
ecoli	0.85	0.75	0.84	0.80	0.95	0.84	0.92	0.89	**0.95**	**0.88**	**0.93**	**0.91**
kidney	0.96	0.92	0.93	0.94	**0.99**	0.95	**0.96**	**0.97**	0.93	**0.95**	0.93	0.94
voting	0.97	0.94	0.95	0.95	0.98	0.95	0.96	0.96	**0.98**	**0.96**	**0.97**	**0.97**
autism	0.73	0.43	0.79	0.53	**0.88**	0.38	0.81	0.52	0.84	**0.88**	**0.91**	**0.86**
ionosphere	0.89	0.87	0.85	0.88	0.92	0.85	0.86	0.88	**0.91**	**0.86**	**0.86**	**0.89**
sonar	0.74	0.56	0.66	0.64	0.81	0.72	0.74	0.76	**0.87**	**0.75**	**0.78**	**0.80**
heart	0.76	0.75	0.78	0.75	0.79	0.70	0.79	0.74	**0.82**	0.74	**0.82**	**0.78**
wine	0.94	**0.86**	0.93	0.89	0.91	0.85	0.92	0.88	**0.98**	0.85	**0.93**	**0.91**

Table 1: Evaluation of ALEPH,ALEPH+LIME and FOLD+LIME on 10 UCI Datasets

[3] http://arxiv.org/abs/1808.00629

Thank you