

- A. Cropper and S. Dumancic. Inductive logic programming at 30: A New Introduction CoRR, abs/2008.07912, 2020.
- S. Muggleton & all.: Meta-interpretive learning: application to grammatical inference, http://www.doc.ic.ac.uk/~shm/FLOC_ILP/Paper03.pdf
- Patsantzis, S., Muggleton, S.H. Top program construction and reduction for polynomial time Meta-Interpretive learning. Mach Learn 110, 755–778 (2021).
- <https://www.cs.ox.ac.uk/activities/programinduction/Aleph/aleph.html>
- <https://github.com/stassa/louise>
- <https://github.com/logic-and-learning-lab/Popper>
- http://www.doc.ic.ac.uk/~shm/FLOC_ILP/Lecture1.1.pdf
- Olga Štěpánková, Luboš Popelínský: Induktivní logické programování <https://www.fi.muni.cz/~popel/lectures/DK-ILP3-fixed.pdf>

Predicate Logic

- Recall the predicate logic.
- CNF, DNF the conjunctive and disjunctive normal form
- **clause**: a disjunction of literals $father(X, Y) \vee \neg parent(X, Y) \vee \neg male(X)$
- **Horn clauses** with at most one positive literal, written as a rule
 - **definite clause** $father(X, Y) : \neg male(X), parent(X, Y)$.
 - **fact** - no negative literal $male(adam)$.
 - **goal clause** - no positive literal $false : \neg father(X, bob)$.
- **Ground term, clause** - a term, a clause without variables.
- We have our data in the form of a set of clauses BK, E^+, E^- ,
 - the background knowledge BK is a set of (Horn) clauses,
 - the positive and examples E^+, E^- are sets of ground literals (facts).

Example (Rule in BK)

$$BK = \left\{ \begin{array}{l} lego_builder(alice). \\ enjoys_lego(A) : \neg lego_builder(A). \\ estate_agent(dave). \\ enjoys_lego(alice). \\ enjoys_lego(claire). \end{array} \right\} \quad \begin{array}{l} E^+ = \{ happy(alice). \} \\ E^- = \left\{ \begin{array}{l} happy(bob). \\ happy(claire). \\ happy(dave). \end{array} \right\} \end{array}$$

Substitution, Subsumption

- Clauses are implicitly generally quantified.
- They should not have a variable with the same name.

Definition (Substitution, Subsumption)

- Given a **substitution** $\theta = \{v_i/t_i\}$ and formula F . $F\theta$ is formed by replacing every variable v_i in F by t_i .
- Substitution θ **unifies** atom A and B in the case $A\theta = B\theta$.
- **Atom A subsumes atom B** , $A \succeq B$, iff there exists a substitution θ such that $A\theta = B$.
- **Clause C subsumes clause D** , $C \succeq D$, iff there exists a substitution θ such that $C\theta \subseteq D$.

Example

- $C_1 = f(A, B) : \neg head(A, B)$.
- $C_2 = f(X, Y) : \neg head(X, Y), empty(Y)$.
- C_1 subsumes C_2 since $C_1\theta \subseteq C_2$ with $\theta = \{A/X, B/Y\}$.

Definition (Generalisation)

- Clause C **is more general than** clause D , iff $C \models D$.
- Clause C is more general than clause D with respect to BK , iff $B, C \models D$.
 - BK is the **background knowledge**.

Whenever a clause C subsumes a clause D , then $C \models D$.

Example (intuitive)

- Statement A: Daffy Duck can fly. $can_fly(daffy)$
- Statement B: All ducks can fly. $can_fly(X) \succeq can_fly(daffy)$.

Example (counter-intuitive)

- Statement C: Marek lives in London.
- Statement D: Marek lives in England.
- $lives(marek, london)$
- $lives(marek, england)$
- Background knowledge $lives(x, england) : \neg lives(x, london)$.
- $BK, C \models D$, 'C is more general than D with respect to BK'.
- $C \succeq D$ with respect to BK .

ILP general logical setting

Definition (Hypothesis Properties)

The background knowledge BK and the hypothesis H should entail E , that is:

Necessity	$BK \not\models E^+$	we need H
Sufficiency, Completeness	$BK \& H \models E^+$	H explains positive examples
Weak consistency	$BK \& H \not\models \perp$	H does not contradict BK
(Strong) consistency	$BK \& H \& E^- \not\models \perp$	... neither negative examples

Definition (ILP task)

The Inductive Logic Programming (ILP) task is

- Given
 - BK background knowledge (logic program)
 - E^+, E^- examples – sets of ground unit clauses
- Given BK, E find a logic program H such that is necessary, sufficient and consistent.
- Often, we assume noisy data and accept some errors, but we try to minimize them.

Example (Simple BK_1)

$$BK_2 = \left\{ \begin{array}{l} \text{lego_builder}(\text{alice}). \\ \text{lego_builder}(\text{bob}). \\ \text{estate_agent}(\text{claire}). \\ \text{estate_agent}(\text{dave}). \\ \text{enjoys_lego}(\text{alice}). \\ \text{enjoys_lego}(\text{claire}). \end{array} \right\}$$

$$E^+ = \{ \text{happy}(\text{alice}). \}$$
$$E^- = \left\{ \begin{array}{l} \text{happy}(\text{bob}). \\ \text{happy}(\text{claire}). \\ \text{happy}(\text{dave}). \end{array} \right\}$$

Our hypothesis space:

$$\mathcal{H} = \left\{ \begin{array}{l} h_1 : \text{happy}(A) : \neg \text{lego_builder}(A). \\ h_2 : \text{happy}(A) : \neg \text{estate_agent}(A). \\ h_3 : \text{happy}(A) : \neg \text{enjoys_lego}(A). \\ h_4 : \text{happy}(A) : \neg \text{lego_builder}(A), \text{estate_agent}(A). \\ h_5 : \text{happy}(A) : \neg \text{lego_builder}(A), \text{enjoys_lego}(A). \\ h_6 : \text{happy}(A) : \neg \text{estate_agent}(A), \text{enjoys_lego}(A). \end{array} \right\}$$

- $BK \cup h_1 \models \text{happy}(\text{bob})$ therefore h_1 is inconsistent.
- $BK \cup h_2 \not\models \text{happy}(\text{alice})$ therefore h_2 is incomplete and inconsistent.
- $BK \cup h_3 \models \text{happy}(\text{claire})$ therefore h_3 is inconsistent.
- $BK \cup h_4 \not\models \text{happy}(\text{alice})$ therefore h_4 is incomplete.
- h_5 is both complete and consistent.
- $BK \cup h_6 \not\models \text{happy}(\text{alice})$ therefore h_6 is incomplete.

Hypothesis Space

- To specify (restrict) the hypothesis space usually mode declarations are used.

Definition (Mode declarations)

Mode declarations denote which literals may appear in the head/body of a rule. A mode declaration is of the form:

$$\text{mode}(\text{recall}, \text{pred}(m_1, m_2, \dots, m_a))$$

where *recall* is the maximum number of occurrences of the predicate
 m_i are the argument types and they may be assigned as input +, output −, constant #.

Example

```
modeb(2,parent(+person,-person)).  
modeh(1,happy(+person)).  
modeb(*,member(+list,-element)).  
modeb(1,head(+list,-element)).
```

A. Cropper and S. Dumancic. *Inductive logic programming at 30: a new introduction.*

Non-monotonic reasoning

- In Prolog, there is negation as a failure.

Example

$$\text{Program} = \left\{ \begin{array}{l} \text{sunny.} \\ \text{happy} : \neg \text{sunny}, \text{not weekday.} \end{array} \right\}$$

- Prolog tries to prove *weekday*.
- It does not prove it, therefore it concludes *happy*.
- With additional knowledge *weekday* some of entailments are not true any more.

Definition (Normal logic program)

Normal logic programs may include negated literals in the body of a clause, e.g.

$$h : \neg b_1, \dots, b_n, \text{not } b_{n+1}, \dots, \text{not } b_m.$$

Aleph ILP system (based on Progol)

- Given
 - A set of mode declaration M
 - Background knowledge BK in the form of a normal program allows negation, with the semantics negation as a failure
 - Positive E^+ and negative E^- examples as a set of ground facts
- Return: A normal program hypothesis H that:
 - H is consistent with M
 - $\forall e \in E^+, H \cup BK \models e$ (H is complete)
 - $\forall e \in E^-, H \cup BK \not\models e$ (H is consistent).

Aleph

1. Select a positive example to generalize.
2. Construct the most specific clause consistent with M that entails the example (the bottom clause).
3. Search for the 'best' clause more general than the bottom clause.
4. Add the clause to the hypothesis and remove all examples covered.
5. If a positive example left, return to step 1.

Bottom Clause Construction

Definition (Bottom clause)

Let H be a clausal hypothesis and C be a clause. The bottom clause $\perp(C)$ is the most specific clause such that:

$$H \cup \perp(C) \models C.$$

- The purpose is to bound the search in the step in 3.
- Without mode declarations, the bottom clause may have infinite cardinality.

Example (Bottom clause)

$$M = \left\{ \begin{array}{l} : -modeh(*, pos(+shape)). \\ : -modeb(*, red(+shape)). \\ : -modeb(*, square(+shape)). \\ : -modeb(*, triangle(+shape)). \\ : -modeb(*, polygon(+shape)). \end{array} \right\} \quad BK = \left\{ \begin{array}{l} red(s1). \\ blue(s2). \\ square(s1). \\ triangle(s2). \\ polygon(A) : -rectangle(A). \\ rectangle(A) : -square(A). \end{array} \right\}$$

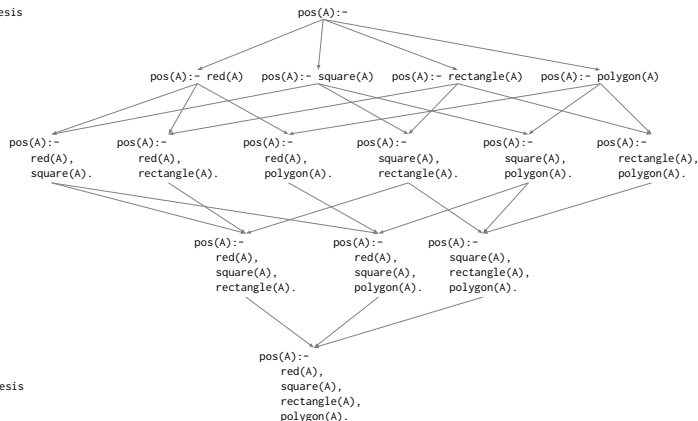
Let e be the positive example $pos(s1)$. Then:

$$\perp(e) = pos(A) : -red(A), square(A), rectangle(A), polygon(A).$$

Clause Search

- Aleph performs a bounded-breadth-first search to enumerate the shorter clauses before the longer ones.
- The search is bounded by several parameters (max. clause size, max. proof depth).

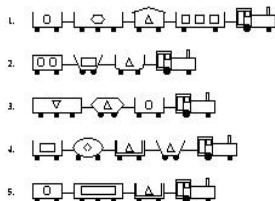
Most general hypothesis



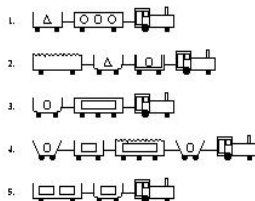
Most specific hypothesis

East-West Trains (1)

1. TRAINS GOING EAST



2. TRAINS GOING WEST

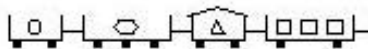


East-West Trains (2)



```
eastbound(east1).           % eastbound train 1
eastbound(east2).           short(car_12). closed(car_12).
eastbound(east3).           long(car_11). open_car(car_11).
eastbound(east4).           ...
eastbound(east5).           shape(car_11,rectangle). shape(car_12,rectangle).
                             ...
eastbound(west6).           load(car_11,rectangle,3). load(car_12,triangle,1).
eastbound(west7).           ...
eastbound(west8).           wheels(car_11,2). wheels(car_12,2).
```

East-West Trains (4)



[bottom clause][literals] [25][saturation time] [0.01]

eastbound(A) :-

has_car(A,B), has_car(A,C), has_car(A,D), has_car(A,E),
short(B), short(D), closed(D), long(C), long(E), open_car(B), open_car(C), open_car(E),
shape(B,rectangle), shape(C,rectangle), shape(D,rectangle), shape(E,rectangle),
wheels(B,2), wheels(C,3), wheels(D,2), wheels(E,2),
load(B,circle,1), load(C,hexagon,1), load(D,triangle,1), load(E,rectangle,3).

[reduce]

eastbound(A). [5/5]

eastbound(A) :- has_car(A,B). [5/5]

eastbound(A) :-

has_car(A,B), short(B). [5/5]

...

eastbound(A) :-

has_car(A,B), wheels(B,3). [3/1]

eastbound(A) :-

has_car(A,B), closed(B). [5/2]

eastbound(A) :- has_car(A,B),

load(B,triangle,1). [5/2]

:- modeh(1, eastbound(+train)).

:- modeb(*, has_car(+train, -car)).

:- modeb(1, short(+car)).

:- modeb(1, load(+car, #shape, #int)).

...

Aleph 2, Popper, PyGol

- Aleph default evaluation function is **coverage** defined as $P - N$,
 - P is the number of positive examples covered by the clause
 - N is the number of negative examples covered by the clause
 - that means it accepts some noise.
- It starts from the most general one $pos(A) : -$.
- It tries to specialize the clause
 - by adding literals to the body of it, which it selects from the bottom clause
- or by instantiating variables.
- Each specialization is called **refinement**.
- Aleph Advantages
 - one Prolog file, easy to download and use.
 - <https://www.cs.ox.ac.uk/activities/programinduction/Aleph/aleph.html>
 - It has good empirical performance.
 - Allows numerical reasoning, user defined cost functions, handles noisy data.
- Aleph Disadvantages
 - It has many parameters to tune.
 - It struggles to learn recursive programs and optimal programs
 - since it learns only a single clause a time.

- Given
 - A set of metarules M
 - Background knowledge BK in the form of a normal program
 - Positive E^+ and negative E^- examples as a set of facts (atoms).
- Return: A definite program hypothesis H that:
 - H is consistent with M
 - $\forall e \in E^+, H \cup B \models e$ (H is complete)
 - $\forall e \in E^-, H \cup B \not\models e$ (H is consistent)
 - $\forall h \in H, \exists m \in M$ such that $h = m\theta$
 - where θ is a substitution that grounds all the existentially quantified variables in m .

Example (Metarule)

- An example is the **chain** metarule $P(A, B) \leftarrow Q(A, C), R(C, B)$
- that allows Metagol to induce programs such as

$$\begin{aligned} f(A, B) &: - \quad tail(A, C), tail(C, B). \\ grandparent(A, B) &: - \quad parent(A, C), parent(C, B). \end{aligned}$$

- Metagol is a form of ILP based on a Prolog meta-interpreter.

Metagol

1. Select a positive example to generalize.
 - If none exists, test the hypothesis on the negative examples.
 - If the hypothesis does not entail any negative example
stop and return the hypothesis.
 - otherwise backtrack to a choice point at step 2 and continue.
2. Try to prove the atom by:
 - using given BK or an already induced clauses
 - unifying the atom with the head of a metarule
 - binding the variables in the metarule to symbols in the predicate and constant signatures
 - save the substitution
 - try to prove the body of the metarule
by treating the body atoms as examples and applying step 2 to them.

Recursion

- Metagol can learn recursive programs.

Example (Reachability)

Consider learning the concept of *reachability* in a graph. Without recursion, with the maximal depth 4 we could learn:

$$\text{reachable}(A, B) : - \text{edge}(A, B).$$
$$\text{reachable}(A, B) : - \text{edge}(A, C), \text{edge}(C, B).$$
$$\text{reachable}(A, B) : - \text{edge}(A, C), \text{edge}(C, D), \text{edge}(D, B).$$
$$\text{reachable}(A, B) : - \text{edge}(A, C), \text{edge}(C, D), \text{edge}(D, E), \text{edge}(E, B).$$

With recursion, we can learn:

$$\text{reachable}(A, B) : - \text{edge}(A, B).$$
$$\text{reachable}(A, B) : - \text{edge}(A, C), \text{reachable}(C, B).$$

iterative deepening

- Metagol uses iterative deepening to search for hypotheses.
 - at depth $d = 1$, at most one metasub.
 - at iteration d , it introduces $d - 1$ new predicate symbols and is allowed to use d clauses.

Metagol Example

Example (Kinship example)

$$BK = \left\{ \begin{array}{l} \text{mother}(\text{ann}, \text{amy}).\text{mother}(\text{ann}, \text{andy}). \\ \text{mother}(\text{amy}, \text{amelia}).\text{mother}(\text{amy}, \text{bob}). \\ \text{mother}(\text{linda}, \text{gavin}). \\ \text{father}(\text{steve}, \text{amy}).\text{father}(\text{steve}, \text{andy}). \\ \text{father}(\text{andy}, \text{spongebob}).\text{father}(\text{gavin}, \text{amelia}). \end{array} \right\}$$

metarule(*ident*, [P, Q], [P, A, B], [[Q, A, B]]).

metarule(*chain*, [P, Q, R], [P, A, B], [[Q, A, C], [R, C, B]]).

$$E^+ = \left\{ \begin{array}{l} \text{grandparent}(\text{ann}, \text{amelia}). \\ \text{grandparent}(\text{steve}, \text{amelia}). \\ \text{grandparent}(\text{ann}, \text{spongebob}). \\ \text{grandparent}(\text{linda}, \text{amelia}). \end{array} \right\}$$

$$E^- = \{ \text{grandparent}(\text{amy}, \text{amelia}). \}$$

Tracing Metagol

- It select the first example to generalize $grandparent(ann, amelia)$.
- It tries to prove it from BK and induced clauses. It fails.
- Metagol tries to use the first metarule:

$$grandparent(ann, amelia) : \neg Q(ann, amelia).$$

stores $sub(ident, [grandparent, Q])$

- and tries to unify Q , but fails.
- Metagol tries to use the second metarule:

$$grandparent(ann, amelia) : \neg Q(ann, C), R(C, amelia).$$

stores $sub(chain, [grandparent, Q, R])$

- and recursively tries to prove $Q(ann, C)$ and $R(C, amelia)$.
- It succeeds with the metasum $sub(chain, [grandparent, mother, mother])$
- and induces the first clause;

$$grandparent(A, B) : \neg mother(A, C), mother(C, B).$$

Metagol Trace 2

- Then, it select the second example to generalize *grandparent(steve, amelia)*.
- It tries to prove it from BK and induced clauses. It fails.
- Metagol can again use the second metarule with another substitution:
stores *sub(chain, [grandparent, father, mother])*
- and induces the second clause;

$$\text{grandparent}(A, B) : - \text{father}(A, C), \text{mother}(C, B).$$

- Given no bound on the program size, the Metagol would prove the other two examples the same way and form the program:

$$\text{grandparent}(A, B) : - \text{mother}(A, C), \text{mother}(C, B).$$
$$\text{grandparent}(A, B) : - \text{father}(A, C), \text{mother}(C, B).$$
$$\text{grandparent}(A, B) : - \text{father}(A, C), \text{father}(C, B).$$
$$\text{grandparent}(A, B) : - \text{mother}(A, C), \text{father}(C, B).$$

With predicate invention, it learns:

$$\text{grandparent}(A, B) : - \text{grandparent_1}(A, C), \text{grandparent_1}(C, B).$$
$$\text{grandparent_1}(A, B) : - \text{father}(A, B).$$
$$\text{grandparent_1}(A, B) : - \text{mother}(A, B).$$

Tail Recursive Metarule

Example (Tail Recursive Metarule)

- An example is the **tail recursive** metarule $P(A, B) \leftarrow Q(A, C), P(C, B)$
- Metagol can also learn mutually recursive programs, such:

```
even(0).  
even(A)  : -   successor(A, B), even_1(B).  
even_1(A) : -   successor(A, B), even(B).
```

We even do not have to provide the concept of an odd number. We can let the Metagol to invent such predicate (*even_1*).

ASPAL algorithm

- ASPAL uses Answer Set Programming.
- ASP program can have one, many, or none models (answer sets).
- Computation in ASP is the process of finding models.
- We may specify the range of the number of clauses from a set being true.
 $0\{\textit{sunny.}, \textit{weekday.}, \textit{happy}(A) : \neg \textit{lego_builder}(A)\}3$
- We may specify an evaluation function to optimize (like to minimize the number of 'true' clauses, e.g. the size of the hypothesis).

ASPAL

- Generate all possible rules consistent with the given mode declarations. Assign each rule a unique identifier and add an guessable atom in each rule.
- Use an ASP solver to find a minimal subset of the rules by formulating the problem as an ASP optimization problem.

Example (ASPAL)

$$BK = \left\{ \begin{array}{l} \text{bird}(\text{alice}). \\ \text{bird}(\text{betty}). \\ \text{can}(\text{alice}, \text{fly}). \\ \text{can}(\text{betty}, \text{swim}). \\ \text{ability}(\text{fly}). \\ \text{ability}(\text{swim}). \end{array} \right\} \quad M = \left\{ \begin{array}{l} \text{modeh}(1, \text{penguin}(+\text{bird})). \\ \text{modeb}(1, \text{bird}(+\text{bird})). \\ \text{modeb}(*, \text{not can}(+\text{bird}, \#ability)). \end{array} \right\}$$

$$E^+ = \{\text{penguin}(\text{betty}).\}$$

$$E^- = \{\text{penguin}(\text{alice}).\}$$

Given the modes, the possible rules are:

$\text{penguin}(X) \quad : - \quad \text{bird}(X).$

$\text{penguin}(X) \quad : - \quad \text{bird}(X), \text{not can}(X, \text{swim}).$

$\text{penguin}(X) \quad : - \quad \text{bird}(X), \text{not can}(X, \text{fly}).$

$\text{penguin}(X) \quad : - \quad \text{bird}(X), \text{not can}(X, \text{swim}), \text{not can}(X, \text{fly}).$

ASPAL replaces constants and adds extra literal:

$\text{penguin}(X) \quad : - \quad \text{bird}(X), \text{rule}(r1).$

$\text{penguin}(X) \quad : - \quad \text{bird}(X), \text{not can}(X, C1), \text{rule}(r2, C1).$

$\text{penguin}(X) \quad : - \quad \text{bird}(X), \text{not can}(X, C1), \text{not can}(X, C2), \text{rule}(r3, C1, C2).$

ASPAL passes to an ASP solver:

bird(alice).

bird(betty).

can(alice, fly).

can(betty, swim).

ability(fly).

ability(swim).

penguin(X) : -bird(X), rule(r1).

penguin(X) : -bird(X), not can(X, C1), rule(r2, C1).

penguin(X) : -bird(X), not can(X, C1), not can(X, C2), rule(r3, C1, C2).

0{rule(r1), rule(r2, fly), rule(r2, swim), rule(r3, fly, swim)}4

goal : -penguin(betty), not penguin(alice).

: -not goal.

- The answer is: *rule(r2, c(fly))*
- Which is translated to a program:

penguin(A) : -bird(A), not can(A, fly).

Python implementations PyGol

- Most ILP implementations are in Prolog.
- If you insist on Python, there is also one.

PyILP is older, user-friendly Python interface for an Inductive Logic Programming (ILP). There are incorporated two distinct ILP algorithms: Aleph and Metagol .

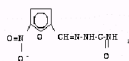
PyGol is a novel Inductive Logic Programming(ILP) system based on Meta Inverse Entailment(MIE) using Python programming language. MIE is similar to Mode-Directed Inverse Entailment (MDIE) but

- does not require mode declarations
 - each hypothesis clause is derived from a meta theory generated automatically from background knowledge.
 - Meta theory can also be viewed as a higher-order language bias that defines the hypothesis space.
- <https://github.com/danyvarghese/PyGol/tree/main>

- Bioinformatics

- ILP can make predictions based on the (sub)structured biological data.
- Predict mutagenic activity of molecules and alert the causes of chemical cancers
- learning protein folding signatures.

Pozitivní



Negativní



- Robot scientist.

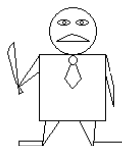
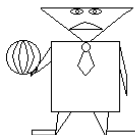
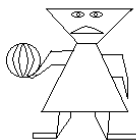
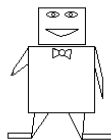
- BK knowledge represents the relationship between protein-coding sequences, enzymes, and metabolites in pathway.
- Automatically generates hypotheses, runs experiments, and iterates results.

- Games

- Sokoban
- Bridge
- Checkers.

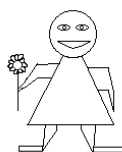
Computer Game Application (Luboš Popelínský & all.)

? Differentiate the friendly agent (4 left) from the enemies (4 right).



přatelští

nepřatelští



Automata Example

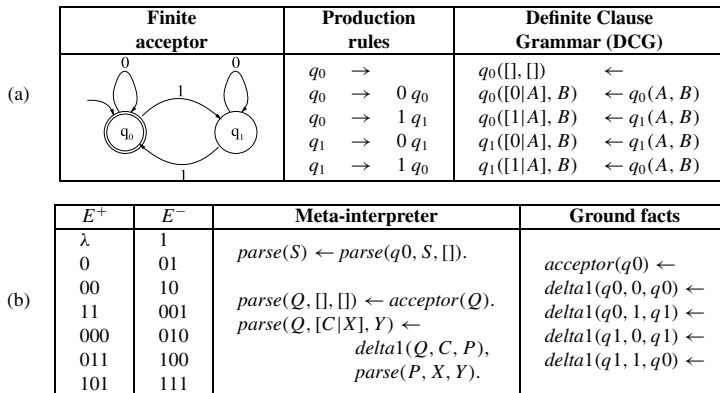


Fig. 1 (a) Parity acceptor with associated production rules, DCG; (b) positive examples (E^+) and negative examples (E^-), Meta-interpreter and ground facts representing the Parity grammar

http://www.doc.ic.ac.uk/~shm/FLOC_ILP/Paper03.pdf

Louise Example Grammar Learning

Example (Module)

```
:-module(anbn, [background_knowledge/2  
    ,metarules/2  
    ,positive_example/2  
    ,negative_example/2  
    ,a/2  
    ,b/2  
]).
```

Example (Background knowledge)

```
background_knowledge(s/2, [a/2,b/2]).  
a([a|T],T).  
b([b|T],T).
```

Example (Metarules)

```
metarules(s/2,[chain]).  
% (Chain)  $\exists P,Q,R \forall x,y,z: P(x,y) \leftarrow Q(x,z),R(z,y)$ 
```

Louise Example Continued

Example (Positive Examples)

```
positive_example(s/2,E):-  
    member(E, [%s([a,b],[])  
              s([a,a,b,b],[])  
              ]).
```

Example (Negative Examples)

```
negative_example(s/2,E):-  
    member(E, [s([a,a],[])  
              ,s([b,b],[])  
              ,s([a,a,b],[])  
              ,s([a,b,b],[])  
              ]).
```

Example (Parameter Tuning)

```
:- auxiliaries:set_configuration_option(clause_limit, [3]).  
:- auxiliaries:set_configuration_option(max_invented, [1]).  
:- auxiliaries:set_configuration_option(reduction, [none]).
```


Louise Example Continued 2

Example (Learned Program)

```
'$1' (A,B):-a(A,C),a(C,B).  
'$1' (A,B):-a(A,C),s(C,B).  
'$1' (A,B):-b(A,C),b(C,B).  
'$1'(A,B):-s(A,C),b(C,B).  
s(A,B):-'$1' (A,C), '$1' (C,B).  
s(A,B):-'$1' (A,C), b(C,B).  
s(A,B):-a(A,C), '$1'(C,B).  
s(A,B):-a(A,C),b(C,B).  
true.
```

Example (Learned Program)

```
?- auxiliaries:set_configuration_option(unfold_invented, [true]).  
?-make.  
?- learn(s/2).  
s(A,B):-a(A,C),b(C,B).  
s(A,B):-a(A,C),s(C,D),b(D,B).  
true.
```

List of topics

- ① Linear, ridge, lasso regression, k-nearest neighbors, overfitting, curse of dimensionality, (LARS)
- ② Splines - the base, natural splines, smoothing splines; kernel smoothing: kernel average, Epanechnikov kernel.
- ③ Logistic regression, Linear discriminant analysis, (generalized additive models)
- ④ Train/test error and data split, square error, 0-1, cross-entropy, AIC, BIC, cross-validation, one-leave-out CV, biased estimate example
- ⑤ decision trees, information gain/entropy/gini, CART α pruning
- ⑥ random forest (+bagging), OOB error, Variable importance, boosting (Adaboost and gradient boosting), stacking, (MARS),
- ⑦ Bayesian learning: MAP, ML hypothesis, Bayesian optimal prediction !formulas!, EM algorithm
- ⑧ Clustering: k-means, Silhouette, k-medoids, hierarchical
- ⑨ Apriori algorithm, Association rules, support, confidence, lift
- ⑩ Inductive logic programming basic: hypothesis space search, background knowledge, necessity, sufficiency and consistency of a hypothesis, Aleph
- ⑪ Undirected graphical models, Graphical Lasso procedure, (deviance, MRF)
- ⑫ Gaussian processes: estimation of the function and its variance !figures, ideas!
- ⑬ Support Vector Machines (optimal separating hyperplane, slacks, kernels).

Table of Contents

- 1 Basic Notions and Linear Models for Regression
- 2 Kernel Methods, Basis Expansion and regularization
- 3 Linear Methods for Classification
- 4 Model Assessment and Selection
- 5 Additive Models, Trees, and Related Methods
- 6 Ensemble Methods
- 7 Bayesian learning, EM algorithm
- 8 Clustering
- 9 Association Rules, Apriori
- 10 Inductive Logic Programming
- 11 Undirected Graphical Models
- 12 Gaussian Processes
- 13 (Support Vector Machines)
- 14 Further Models