Tutorial on Popper

Céline Hocquette, Andrew Cropper
University of Oxford

UK Research and Innovation

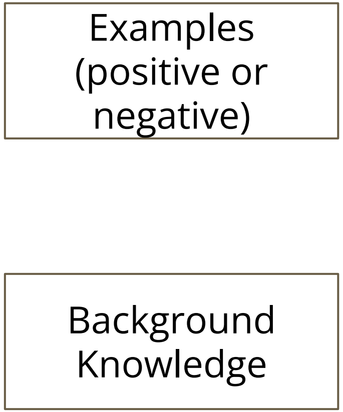
UNIVERSITY OF OXFORD

Inductive Logic Programming

Inductive Logic Programming

Examples
(positive or
negative)

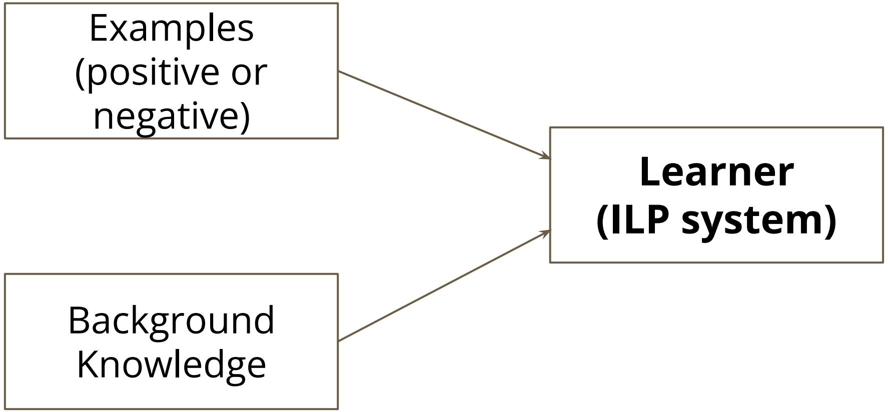
Inductive Logic Programming



Examples (positive or negative)

Background Knowledge

Inductive Logic Programming

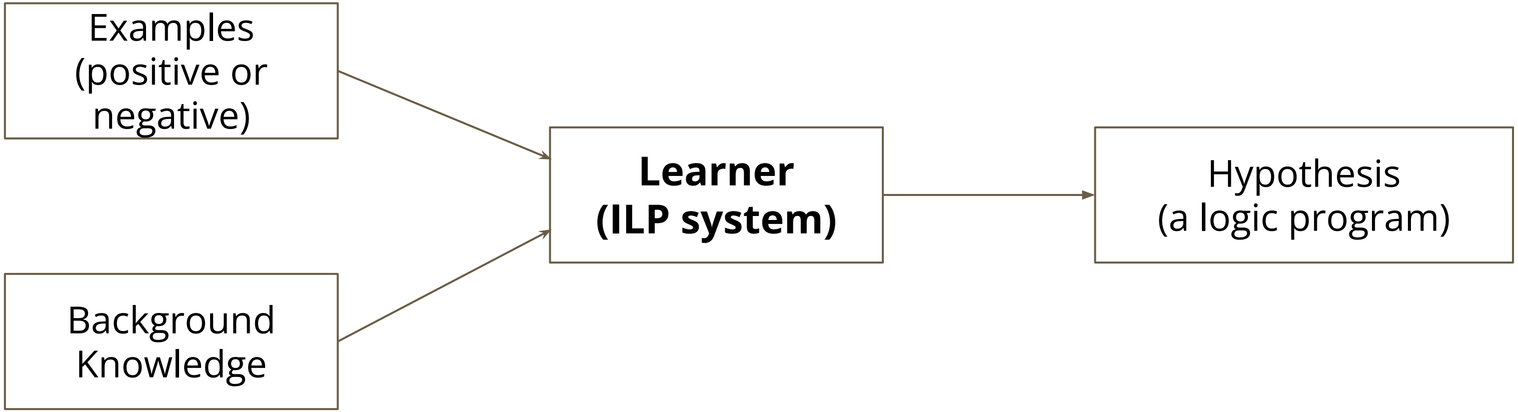


Examples (positive or negative)

Background Knowledge

Learner (ILP system)

Inductive Logic Programming

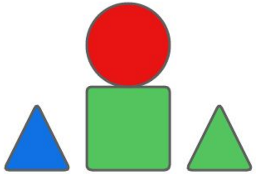
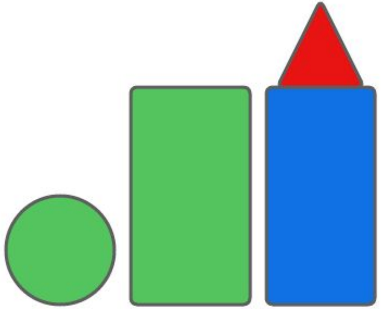
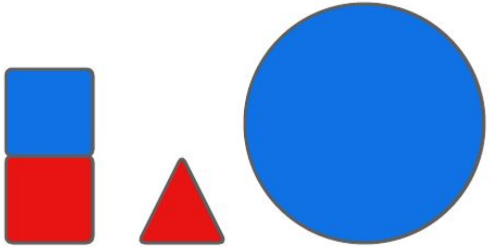
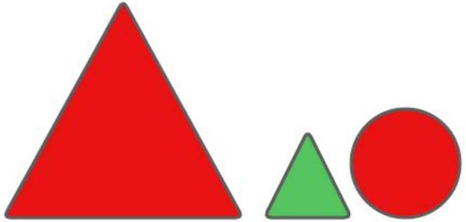


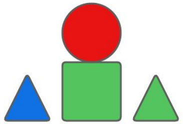
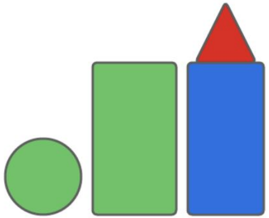
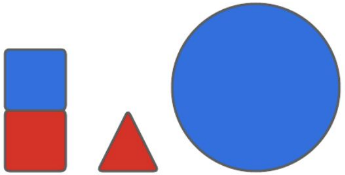
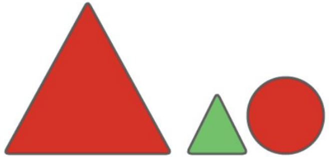
Examples (positive or negative)

Background Knowledge

Learner (ILP system)

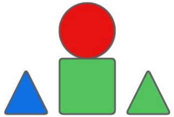
Hypothesis (a logic program)

Positive examples	Negative examples
	
	

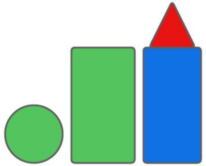
Positive examples	Negative examples
	
	

There must be a red piece in contact with a small piece

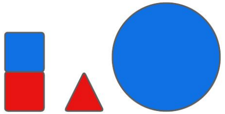
Positive examples	Negative examples
zendo(ex1). zendo(ex2).	zendo(ex3). zendo(ex4).



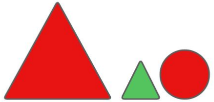
ex1



ex3

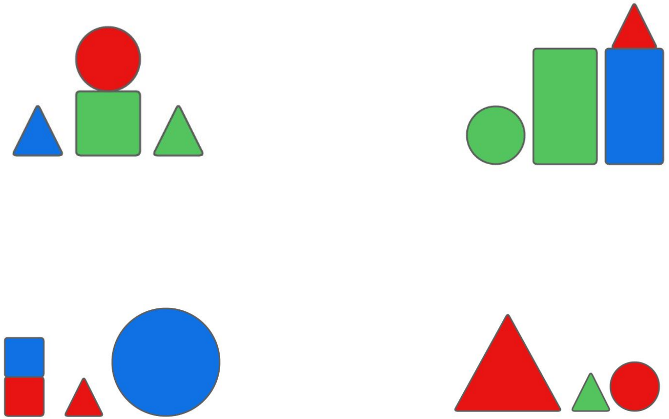


ex2



ex4

Background Knowledge
piece(ex1, p1). piece(ex1, p2). piece(ex1, p3). piece(ex1, p4). blue(p1). triangle(p1). size(p1, 2). small(2). red(p2). round(p2). triangle(p4). contact(p2, p3). on(p2, p3). right(p4, p3). left(p1, p2). ...



Hypothesis

```
zendo(Structure):-
    piece(Structure,Piece1),
    red(Piece1),
    contact(Piece1,Piece2),
    size(Piece2,Size),
    small(Size).
```

Popper: an inductive logic programming system

Why care?

Why care?

- learn globally optimal programs (textually minimal or minimal description length)

Why care?

- learn globally optimal programs (textually minimal or minimal description length)
- learn recursive programs

Why care?

- learn globally optimal programs (textually minimal or minimal description length)
- learn recursive programs
- support predicate invention

Why care?

- learn globally optimal programs (textually minimal or minimal description length)
- learn recursive programs
- support predicate invention
- learn large programs with many rules

Why care?

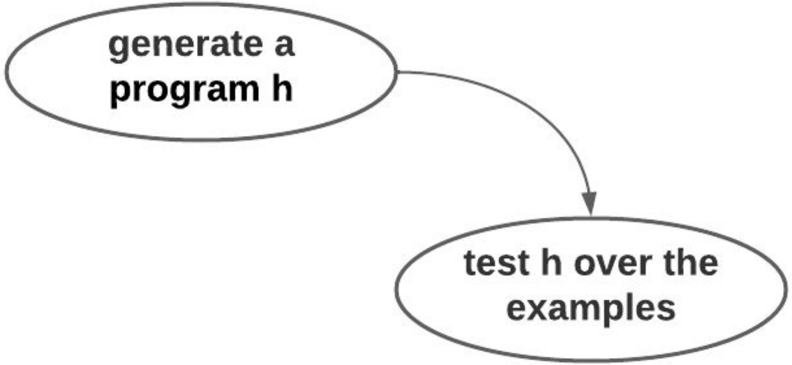
- learn globally optimal programs (textually minimal or minimal description length)
- learn recursive programs
- support predicate invention
- learn large programs with many rules
- support noisy examples

How does it work?

How does it work?

generate a program h

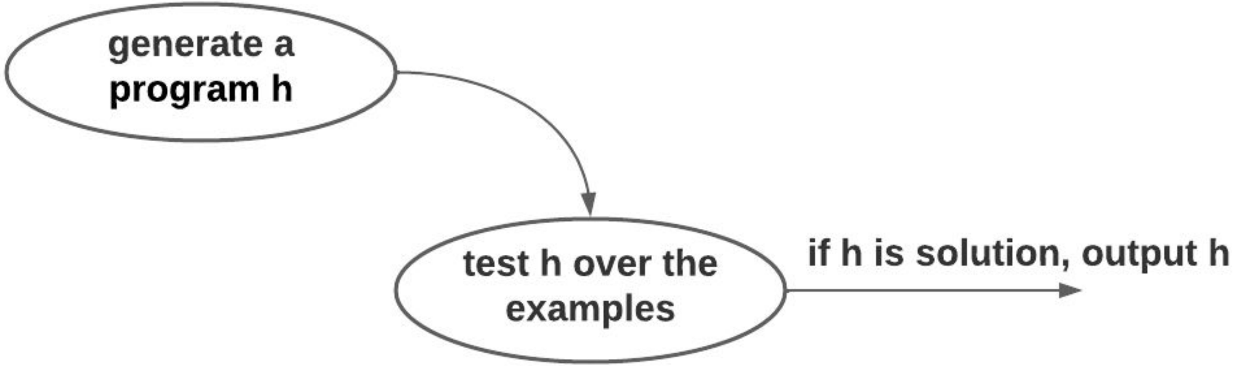
How does it work?



generate a program h

test h over the examples

How does it work?

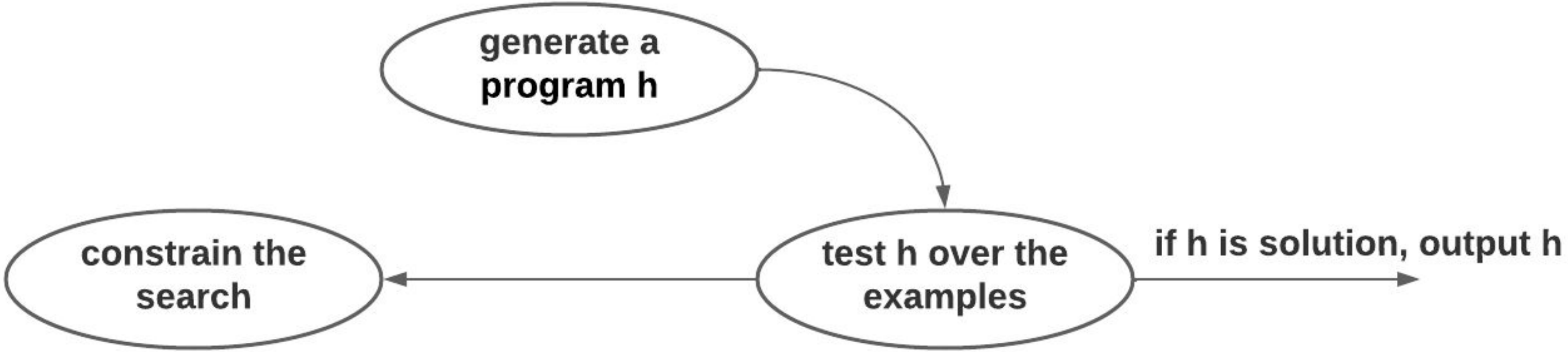


generate a program h

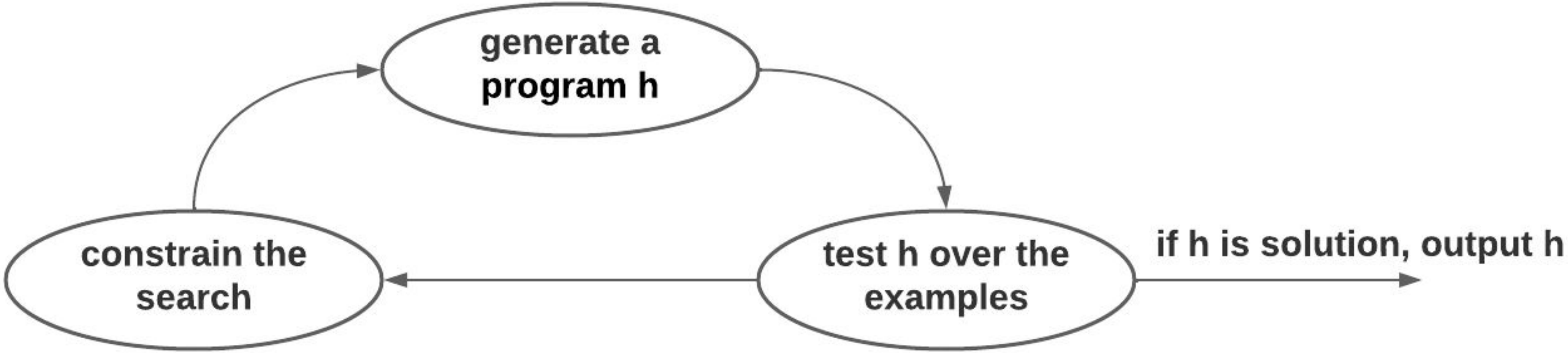
test h over the examples

if h is solution, output h

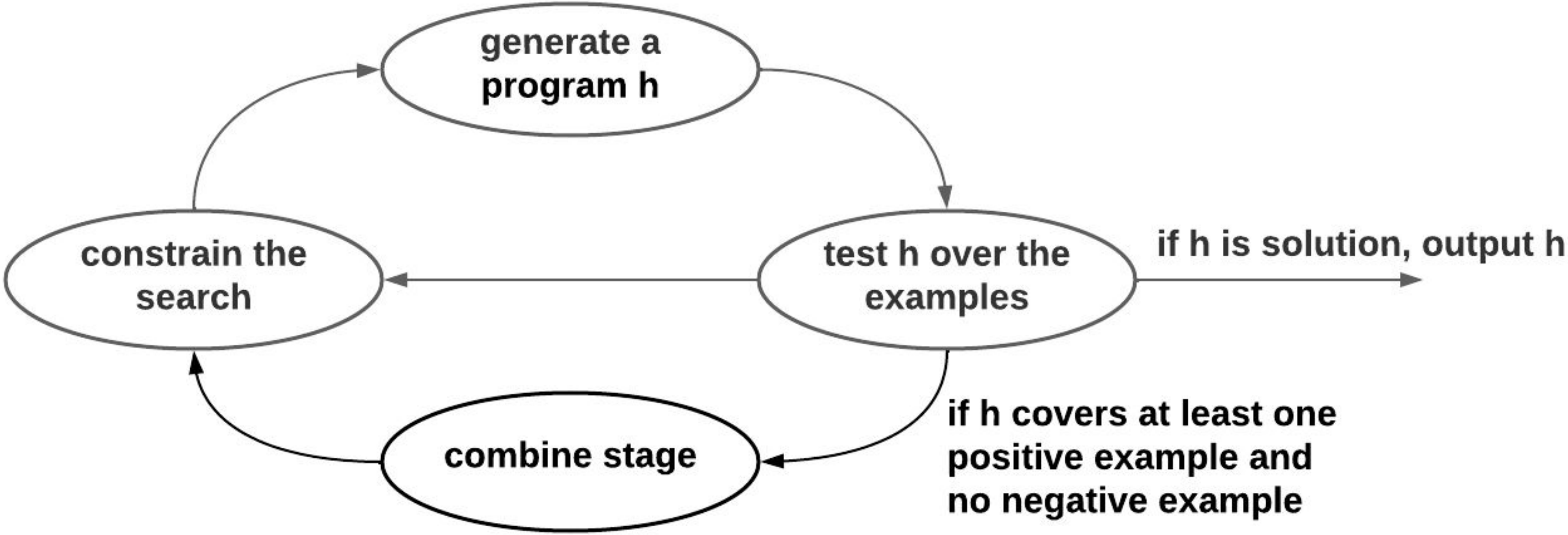
How does it work?



How does it work?



How does it work?


generate a program h
test h over the examples
if h is solution, output h
constrain the search
if h covers at least one positive example and no negative example
combine stage

Learning programs by combining programs, Andrew Cropper and Céline Hocquette, *ECAI, 2023*.

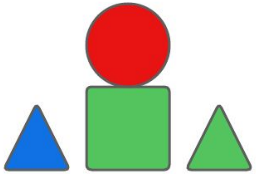
Questions?

https://github.com/logic-and-learning-lab/Popper

Popper input

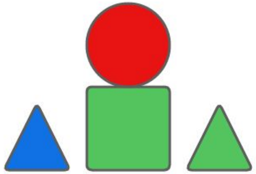
- examples file *exs.pl*
- bk file *bk.pl*
- bias file *bias.pl*

Zendo: exs file



```
pos(zendo(ex0)).      neg(zendo(ex20)).
pos(zendo(ex1)).      neg(zendo(ex21)).
pos(zendo(ex2)).      neg(zendo(ex22)).
pos(zendo(ex3)).      neg(zendo(ex23)).
pos(zendo(ex4)).      neg(zendo(ex24)).
pos(zendo(ex5)).      neg(zendo(ex25)).
pos(zendo(ex6)).      neg(zendo(ex26)).
pos(zendo(ex7)).      neg(zendo(ex27)).
pos(zendo(ex8)).      neg(zendo(ex28)).
pos(zendo(ex9)).      neg(zendo(ex29)).
pos(zendo(ex10)).     neg(zendo(ex30)).
```

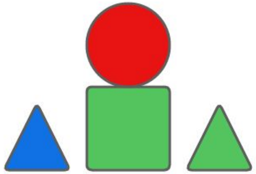
Zendo: bk file



```
piece(ex1, p1).
piece(ex1, p2).
piece(ex1, p3).
piece(ex1, p4).
blue(p1).
triangle(p1).
size(p1, 2).
small(2).
red(p2).
round(p2).
triangle(p4).
contact(p2, p3).
on(p2, p3).
right(p4, p3).
left(p1, p2).
...
```

Zendo: bias file

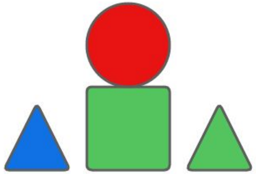
(predicate declarations)



```
head_pred(zendo,1).
body_pred(piece,2).
body_pred(contact,2).
body_pred(coord1,2).
body_pred(coord2,2).
body_pred(size,2).
body_pred(blue,1).
body_pred(green,1).
body_pred(red,1).
body_pred(small,1).
body_pred(medium,1).
body_pred(large,1).
body_pred(upright,1).
body_pred(lhs,1).
body_pred(rhs,1).
body_pred(strange,1).
```

Zendo: bias file

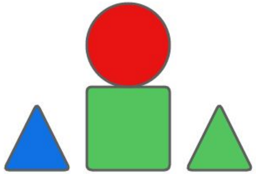
(optional types)



```
type(zendo,(state,)).
type(piece,(state,piece)).
type(contact,(piece,piece)).
type(coord1,(piece,real)).
type(coord2,(piece,real)).
type(size,(piece,real)).
type(blue,(piece,)).
type(green,(piece,)).
type(red,(piece,)).
type(small,(real,)).
type(medium,(real,)).
type(large,(real,)).
type(upright,(piece,)).
type(lhs,(piece,)).
type(rhs,(piece,)).
type(strange,(piece,)).
```

Zendo: bias file

(optional types)



all or none of the types must be provided (Popper does not support partial typing)

```
type(zendo,(state,)).
type(piece,(state,piece)).
type(contact,(piece,piece)).
type(coord1,(piece,real)).
type(coord2,(piece,real)).
type(size,(piece,real)).
type(blue,(piece,)).
type(green,(piece,)).
type(red,(piece,)).
type(small,(real,)).
type(medium,(real,)).
type(large,(real,)).
type(upright,(piece,)).
type(lhs,(piece,)).
type(rhs,(piece,)).
type(strange,(piece,)).
```

Popper

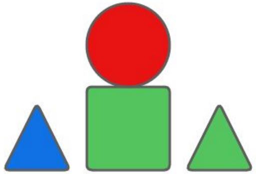
```
python popper.py <input-dir>
```

Popper

```
python popper.py ./examples/zendo1
```

Zendo

```
python popper.py ./examples/zendo1
12:06:37 Generating programs of size: 3
12:06:37 Generating programs of size: 4
12:06:37 Generating programs of size: 5
```



```
12:06:41 Generating programs of size: 6
12:06:41 ********************
12:06:41 New best hypothesis:
12:06:41 tp:19 fn:1 tn:20 fp:0 size:20
12:06:41 zendo(A):- piece(A,B),contact(B,C),lhs(B),strange(C).
12:06:41 zendo(A):- piece(A,B),rhs(B),contact(B,C),blue(C).
12:06:41 zendo(A):- piece(A,B),contact(B,C),red(C),lhs(C).
12:06:41 zendo(A):- piece(A,B),contact(B,C),upright(C),red(C).
```

```
12:06:41 ********************
********** SOLUTION **********
Precision:1.00 Recall:1.00 TP:20 FN:0 TN:20 FP:0 Size:6
zendo(A):- piece(A,C),red(C),contact(C,B),size(B,D),small(D).
******************************
```

Total execution time: 4.96s

```
python popper.py ./examples/zendo2
```

Zendo:
a more
difficult task

Zendo: a more difficult task

```
11:47:01 Generating programs of size: 3
11:47:01 Generating programs of size: 4
11:47:02 ********************
11:47:02 New best hypothesis:
11:47:02 tp:47 fn:53 tn:100 fp:0 size:4
11:47:02 zendo(A):- piece(A,B),lhs(B),green(B).
11:47:02 ********************
```

```
11:47:02 Generating programs of size: 5
11:47:05 Generating programs of size: 6
11:47:05 ********************
11:47:05 New best hypothesis:
11:47:05 tp:52 fn:48 tn:100 fp:0 size:14
11:47:05 zendo(A):- piece(A,B),lhs(B),green(B).
11:47:05 zendo(A):- piece(A,B),green(B),contact(B,C),red(C).
11:47:05 zendo(A):- piece(A,B),contact(B,C),upright(C),strange(B).
11:47:05 ********************
```

```
11:47:31 Generating programs of size: 7
11:47:31 ********************
11:47:31 New best hypothesis:
11:47:31 tp:57 fn:43 tn:100 fp:0 size:27
11:47:31 zendo(A):- piece(A,B),lhs(B),green(B).
11:47:31 zendo(A):- piece(A,B),green(B),contact(B,C),red(C).
11:47:31 zendo(A):- piece(A,C),strange(C),contact(C,B),green(B),blue(C).
11:47:31 zendo(A):- piece(A,C),contact(C,B),lhs(B),piece(A,D),green(D).
11:47:31 zendo(A):- piece(A,C),coord2(C,B),strange(C),size(C,B),green(C).
11:47:31 ********************
```

```
11:51:29 ********************
11:51:29 New best hypothesis:
11:51:29 tp:100 fn:0 tn:100 fp:0 size:14
11:51:29 zendo(A):- piece(A,C),green(C),piece(A,B),coord1(B,D),lhs(B),coord1(C,D).
11:51:29 zendo(A):- piece(A,C),piece(A,D),piece(A,B),red(B),green(C),blue(D).
11:51:29 ********************
```

```
********** SOLUTION **********
Precision:1.00 Recall:1.00 TP:100 FN:0 TN:100 FP:0 Size:14
zendo(A):- piece(A,C),green(C),piece(A,B),coord1(B,D),lhs(B),coord1(C,D).
zendo(A):- piece(A,C),piece(A,D),piece(A,B),red(B),green(C),blue(D).
******************************
```

Total execution time: 112.94s

```
python popper.py ./examples/zendo2 —bkcons
```

Total execution time: 65.18s

Learning from noisy data

Learning from noisy data

minimum description length

Learning MDL Logic Programs From Noisy Data, Céline Hocquette, Andreas Niskanen, Matti Järvisalo, Andrew Cropper.

Learning from noisy data

minimum description length: trade-off model complexity (program size) and data fit (training accuracy)

Learning MDL Logic Programs From Noisy Data, Céline Hocquette, Andreas Niskanen, Matti Järvisalo, Andrew Cropper.

Learning from noisy data

minimum description length: trade-off model complexity (program size) and data fit (training accuracy)

$$mdl(h) = size(h) + fp(h) + fn(h)$$

Learning MDL Logic Programs From Noisy Data, Céline Hocquette, Andreas Niskanen, Matti Järvisalo, Andrew Cropper.

Zendo: with 10% noise added

```
python popper.py ./examples/noisy-zendo2-10 --noisy
```

```
19:59:09 Generating programs of size: 3
19:59:09 ********************
19:59:09 New best hypothesis:
19:59:09 tp:86 fn:13 tn:30 fp:71 size:3 mdl:87
19:59:09 zendo(A):- piece(A,B),red(B).
19:59:09 ********************
19:59:09 ********************
19:59:09 New best hypothesis:
19:59:09 tp:95 fn:4 tn:40 fp:61 size:3 mdl:68
19:59:09 zendo(A):- piece(A,B),green(B).
19:59:09 ********************
```

```
19:59:09 Generating programs of size: 4
19:59:09 ********************
19:59:09 New best hypothesis:
19:59:09 tp:40 fn:59 tn:97 fp:4 size:4 mdl:67
19:59:09 zendo(A):- piece(A,B),lhs(B),green(B).
19:59:09 ********************
```

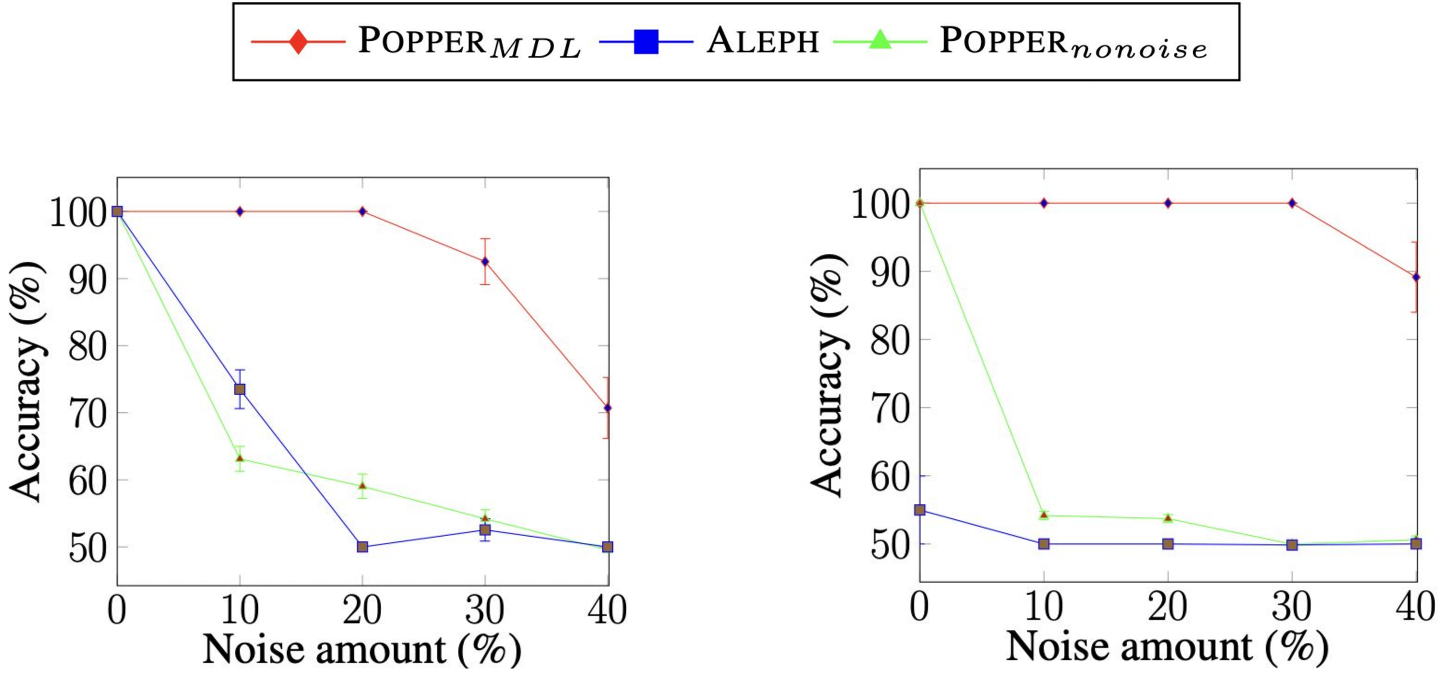
```
19:59:09 Generating programs of size: 5
19:59:09 ********************
19:59:09 New best hypothesis:
19:59:09 tp:49 fn:50 tn:94 fp:7 size:8 mdl:65
19:59:09 zendo(A):- piece(A,B),lhs(B),green(B).
19:59:09 zendo(A):- piece(A,B),contact(B,C),green(C).
19:59:09 ********************
19:59:10 New best hypothesis:
19:59:10 tp:76 fn:23 tn:66 fp:35 size:5 mdl:63
19:59:10 zendo(A):- piece(A,B),green(B),piece(A,C),blue(C).
19:59:10 ********************
19:59:10 New best hypothesis:
19:59:10 tp:82 fn:17 tn:67 fp:34 size:5 mdl:56
19:59:10 zendo(A):- piece(A,B),green(B),piece(A,C),red(C).
19:59:10 ********************
```

```
19:59:11 Generating programs of size: 6
19:59:11 ********************
19:59:11 New best hypothesis:
19:59:11 tp:90 fn:9 tn:67 fp:34 size:9 mdl:52
19:59:11 zendo(A):- piece(A,B),lhs(B),green(B).
19:59:11 zendo(A):- piece(A,B),green(B),piece(A,C),red(C).
19:59:11 ********************
```

```
19:59:22 Generating programs of size: 7
20:01:18 ********************
20:01:18 New best hypothesis:
20:01:18 tp:67 fn:32 tn:93 fp:8 size:7 mdl:47
20:01:18 zendo(A):- piece(A,B),green(B),piece(A,D),piece(A,C),red(C),blue(D).
20:01:18 ********************
20:05:40 New best hypothesis:
20:05:40 tp:90 fn:9 tn:91 fp:10 size:14 mdl:33
20:05:40 zendo(A):- piece(A,B),green(B),piece(A,D),piece(A,C),red(C),blue(D).
20:05:40 zendo(A):-piece(A,C),coord1(C,D),green(C),piece(A,B),lhs(B),coord1(B,D).
20:05:40 ********************
```

```
********** SOLUTION **********
Precision:0.90 Recall:0.91 TP:90 FN:9 TN:91 FP:10 Size:14 MDL:33
zendo(A):- piece(A,B),green(B),piece(A,D),piece(A,C),red(C),blue(D).
zendo(A):- piece(A,C),coord1(C,D),green(C),piece(A,B),lhs(B),coord1(B,D).
```

Total execution time: 121.91s



iggp-rps

dropk

Popper can support up to 20-30% of noise in the examples

Recursion: dropk

Positive	Negative
f([40, 58, 10, 9, 89, 64],1,[58, 10, 9, 89, 64]). f([15, 93, 40],3,[]). f([66, 17, 39, 79, 35, 18, 45, 37],5,[18, 45, 37]).	f([17, 37, 97],0,[37, 97]). f([23, 51, 98, 73, 72, 26],5,[23, 51, 98, 73, 72, 26]).

Recursion: dropk

Positive	**Negative**
f([40, 58, 10, 9, 89, 64],1,[58, 10, 9, 89, 64]). f([15, 93, 40],3,[]). f([66, 17, 39, 79, 35, 18, 45, 37],5,[18, 45, 37]).	f([17, 37, 97],0,[37, 97]). f([23, 51, 98, 73, 72, 26],5,[23, 51, 98, 73, 72, 26]).

Hypothesis

```
f(Input,K,Output):- tail(Input,Output),one(K).
f(Input,K,Output):- tail(Input,List),decrement(K,K1),f(List,K1,Output).
```

Recursion: dropk 20% noise

```
python popper.py ./examples/dropk-20 —noisy
```

Recursion: dropk 20% noise

```
10:56:34 Generating programs of size: 3
10:56:34 ********************
10:56:34 New best hypothesis:
10:56:34 tp:10 fn:84 tn:101 fp:5 size:3 mdl:92
10:56:34 f(A,B,C):- tail(A,C),odd(B).
10:56:34 ********************
10:56:34 ********************
10:56:34 New best hypothesis:
10:56:34 tp:10 fn:84 tn:104 fp:2 size:3 mdl:89
10:56:34 f(A,B,C):- tail(A,C),one(B).
10:56:34 ********************
```

Recursion: dropk 20% noise

```
10:56:34 Generating programs of size: 4
10:56:35 Generating programs of size: 5
10:56:36 Generating programs of size: 6
```

```
10:56:39 Generating programs of size: 7
10:56:50 ********************
10:56:50 New best hypothesis:
10:56:50 tp:45 fn:49 tn:76 fp:30 size:7 mdl:86
10:56:50 f(A,B,C):- tail(A,C),odd(B).
10:56:50 f(A,B,C):- tail(A,D),f(D,B,E),tail(E,C).
10:56:50 ********************
10:56:50 ********************
10:56:50 New best hypothesis:
10:56:50 tp:83 fn:11 tn:71 fp:35 size:7 mdl:53
10:56:50 f(A,B,C):- tail(A,C),odd(B).
10:56:50 f(A,B,C):- decrement(B,D),f(A,D,E),tail(E,C).
10:56:50 ********************
```

```
10:56:51 ********************
10:56:51 New best hypothesis:
10:56:51 tp:83 fn:11 tn:89 fp:17 size:7 mdl:35
10:56:51 f(A,B,C):- tail(A,C),one(B).
10:56:51 f(A,B,C):- decrement(B,D),f(A,D,E),tail(E,C).
10:56:51 ********************
```

```
10:57:03 Generating programs of size: 8
10:59:44 Generating programs of size: 9
```

```
11:06:34 TIMEOUT OF 600 SECONDS EXCEEDED
********** SOLUTION **********
Precision:0.83 Recall:0.88 TP:83 FN:11 TN:89 FP:17 Size:7 MDL:35
f(A,B,C):- tail(A,C),one(B).
f(A,B,C):- decrement(B,D),f(A,D,E),tail(E,C).
******************************
```

Learning large programs

```
python popper.py ./examples/iggp-buttons

********** SOLUTION **********
Precision:1.00 Recall:1.00 TP:98 FN:0 TN:432 FP:0 Size:61
next(A,B):- my_succ(C,B),my_true(A,C).
next(A,B):- my_true(A,B),c_b(C),c_r(B),my_input(D,C),does(A,D,C).
next(A,B):- role(D),c_p(B),does(A,D,C),my_true(A,B),c_c(C).
next(A,B):- c_p(B),my_input(D,C),does(A,D,C),not_my_true(A,B),c_a(C).
next(A,B):- c_r(B),my_true(A,B),c_a(C),does(A,D,C),my_input(D,C).
next(A,B):- c_q(B),my_input(D,C),my_true(A,B),c_a(C),does(A,D,C).
next(A,B):- my_true(A,E),my_input(D,C),c_q(B),c_b(C),c_p(E),does(A,D,C).
next(A,B):- my_true(A,E),does(A,D,C),c_q(E),my_input(D,C),c_r(B),c_c(C).
next(A,B):- c_b(C),c_p(B),c_q(E),my_true(A,E),does(A,D,C),my_input(D,C).
next(A,B):- my_true(A,E),c_c(C),my_input(D,C),c_q(B),c_r(E),does(A,D,C).
******************************
```

Total execution time: 9.00s

```
python popper.py ./examples/eight_puzzle_legal_move

********** SOLUTION **********
Precision:1.00 Recall:1.00 TP:1383 FN:0 TN:3117 FP:0 Size:20
legal_move(A,B,C,D):- input_move(B,E,D),succ(E,C),true_cell(A,E,D,F),cell_type_b(F).
legal_move(A,B,C,D):- true_cell(A,C,F,E),cell_type_b(E),succ(D,F),input_move(B,D,F).
legal_move(A,B,C,D):- succ(F,D),input_move(B,D,F),cell_type_b(E),true_cell(A,C,F,E).
legal_move(A,B,C,D):- cell_type_b(F),true_cell(A,E,D,F),input_move(B,E,D),succ(C,E).
******************************
```

Total execution time: 573.80s

Conclusion

- Popper, an ILP algorithm

Conclusion

- Popper, an ILP algorithm
  - feature-rich:
    - recursive
    - predicate invention
    - optimal programs (mdl or textually minimal)
    - noisy data
    - anytime
    - infinite domains and numerical reasoning

Conclusion

- Popper, an ILP algorithm
    - feature-rich
    - can learn moderately large programs (largish rules and many rules)

Limitations

Limitations

- Very large datasets with lots of BK and lots of examples (10k+)

Limitations

- Very large datasets with lots of BK and lots of examples (10k+)
- Learn rules with many variables (long-chains of reasoning)

Limitations

- Very large datasets with lots of BK and lots of examples (10k+)
- Learn rules with many variables (long-chains of reasoning)
- Invent complex abstractions

Limitations

- Very large datasets with lots of BK and lots of examples (10k+)
- Learn rules with many variables (long-chains of reasoning)
- Invent complex abstractions
- Negation

Tips

Tips

- try no more than 6 variables first (10 is infeasible)

Tips

- try no more than 6 variables first (10 is infeasible)
- if possible, use datalog BK

Tips

- try no more than 6 variables first (10 is infeasible)
- if possible, use datalog BK
- avoid recursion if possible

Tips

- try no more than 6 variables first (10 is infeasible)
- if possible, use datalog BK
- avoid recursion if possible
- avoid predicate invention if possible

Tips

- try no more than 6 variables first (10 is infeasible)
- if possible, use datalog BK and use the –bkcons flag
- avoid recursion if possible
- avoid predicate invention if possible
- use a sat solver for the combine stage

Thank you!