Learning Logic Programs by Combining Programs

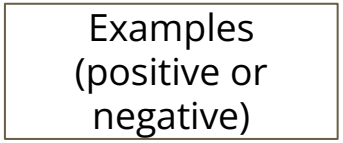
Andrew Cropper, Céline Hocquette
University of Oxford

UK Research and Innovation

UNIVERSITY OF OXFORD

Program Synthesis

Program Synthesis

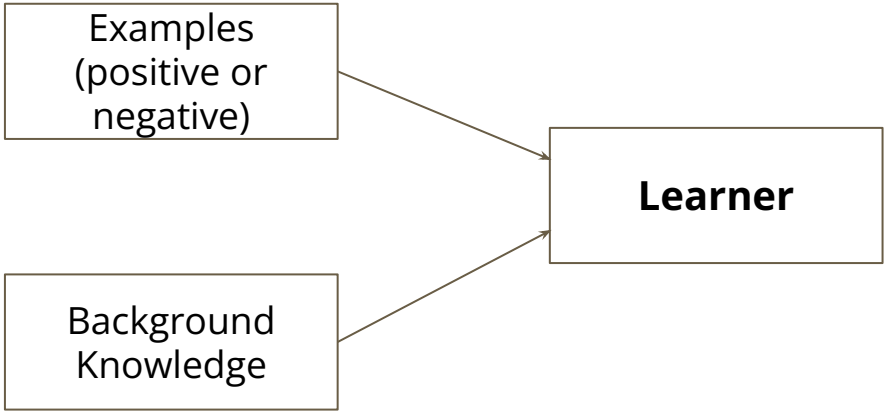


Program Synthesis

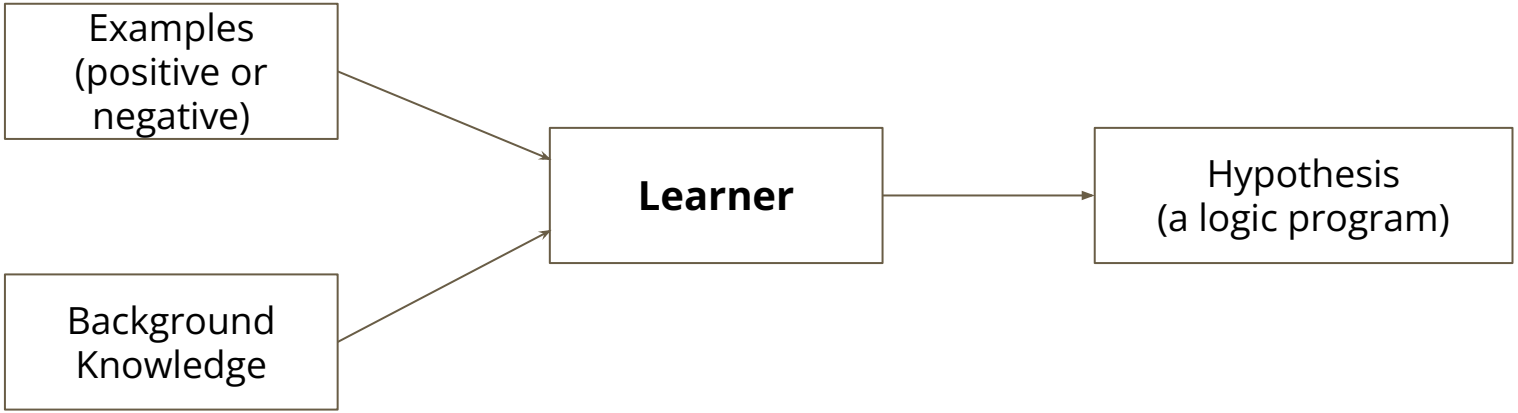
Examples (positive or negative)

Background Knowledge

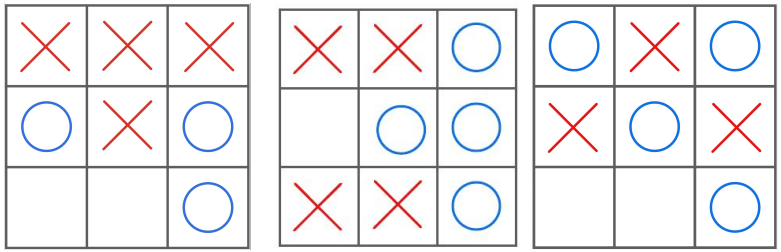
Program Synthesis



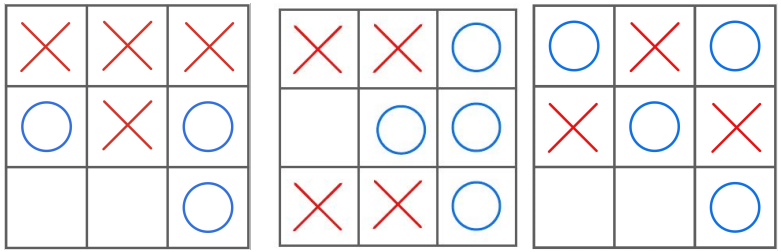
Program Synthesis



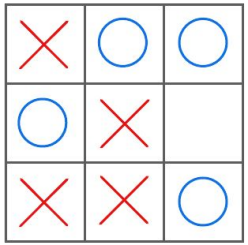
Positive



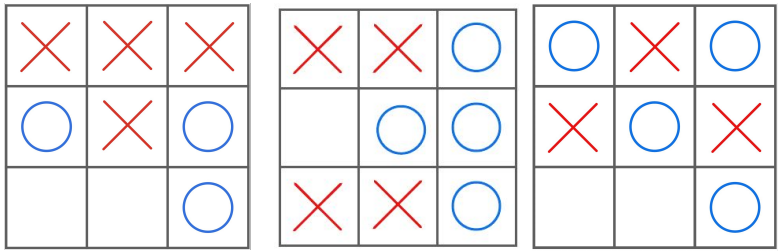
Positive



Negative

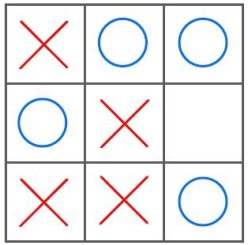


Positive



b1 b2 b3

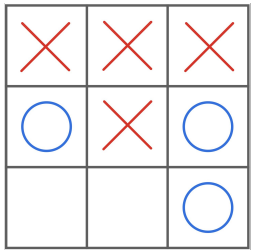
Negative



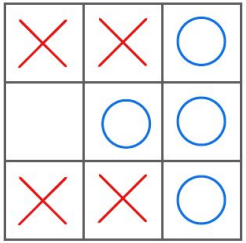
b4

Positive examples	Negative examples
win(b1,x). win(b2,o). win(b3,o).	win(b4,x).

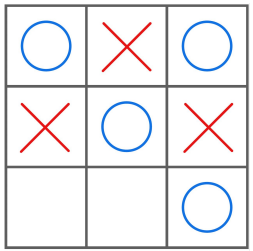
Positive



b1

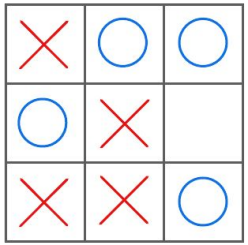


b2



b3

Negative



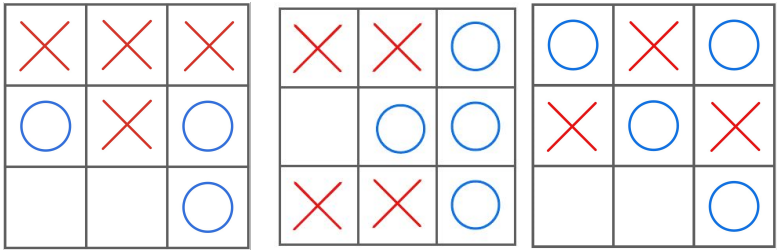
b4

Positive examples	Negative examples
win(b1,x). win(b2,o). win(b3,o).	win(b4,x).

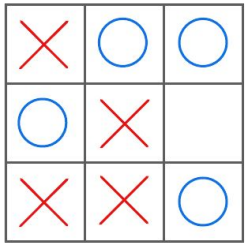
Background Knowledge
cell(b1,0,x). cell(b1,1,x). cell(b1,2,x). ... cell(b2,0,x). ... cell(b3,0,o).

0	1	2
3	4	5
6	7	8

Positive



Negative



Hypothesis
win(Board,Player) ← cell(Board,0,Player), cell(Board,1,Player), cell(Board,2,Player) win(Board,Player) ← cell(Board,0,Player), cell(Board,3,Player), cell(Board,6,Player) win(Board,Player) ← cell(Board,0,Player), cell(Board,4,Player), cell(Board,8,Player)

0	1	2
3	4	5
6	7	8

Challenge

Very large hypothesis spaces

Challenge

Very large hypothesis spaces

Noughts and Crosses:

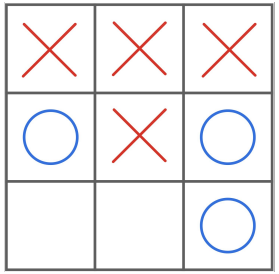
- 10^9 hypotheses with 3 rules
- 10^{20} hypotheses with 9 rules

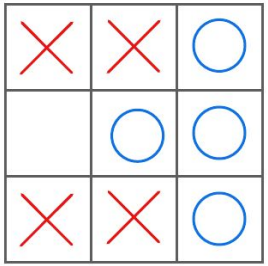
Challenge

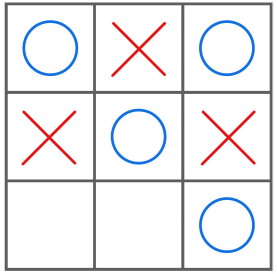
Difficult to learn programs with many rules

Our contribution: a program synthesis approach which learns programs with many rules

Motivation

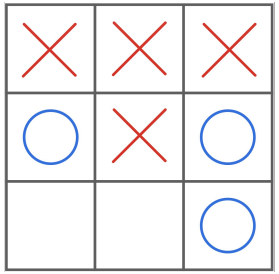


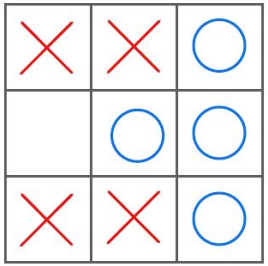


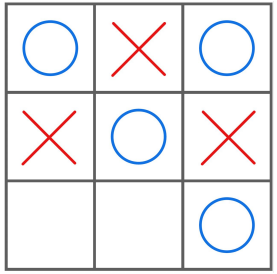


win(Board,Player) ← cell(Board,0,Player), cell(Board,1,Player), cell(Board,2,Player)
win(Board,Player) ← cell(Board,0,Player), cell(Board,3,Player), cell(Board,6,Player)
win(Board,Player) ← cell(Board,0,Player), cell(Board,4,Player), cell(Board,8,Player)

Motivation







r1: win(Board,Player) ← cell(Board,0,Player), cell(Board,1,Player), cell(Board,2,Player)

r2: win(Board,Player) ← cell(Board,0,Player), cell(Board,3,Player), cell(Board,6,Player)

r3: win(Board,Player) ← cell(Board,0,Player), cell(Board,4,Player), cell(Board,8,Player)

r1, r2, and r3 do not depend on each other

Idea

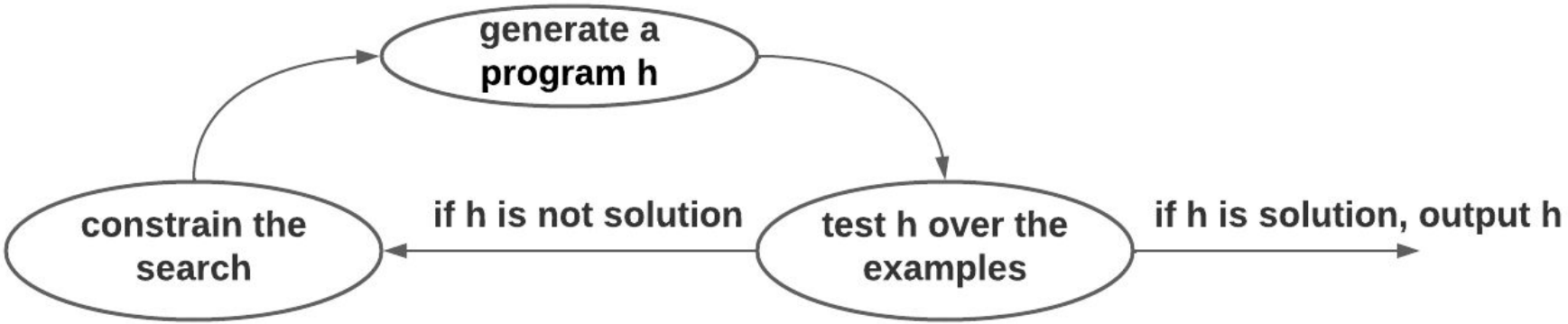
Learn small programs that cover some of the positive examples

Idea

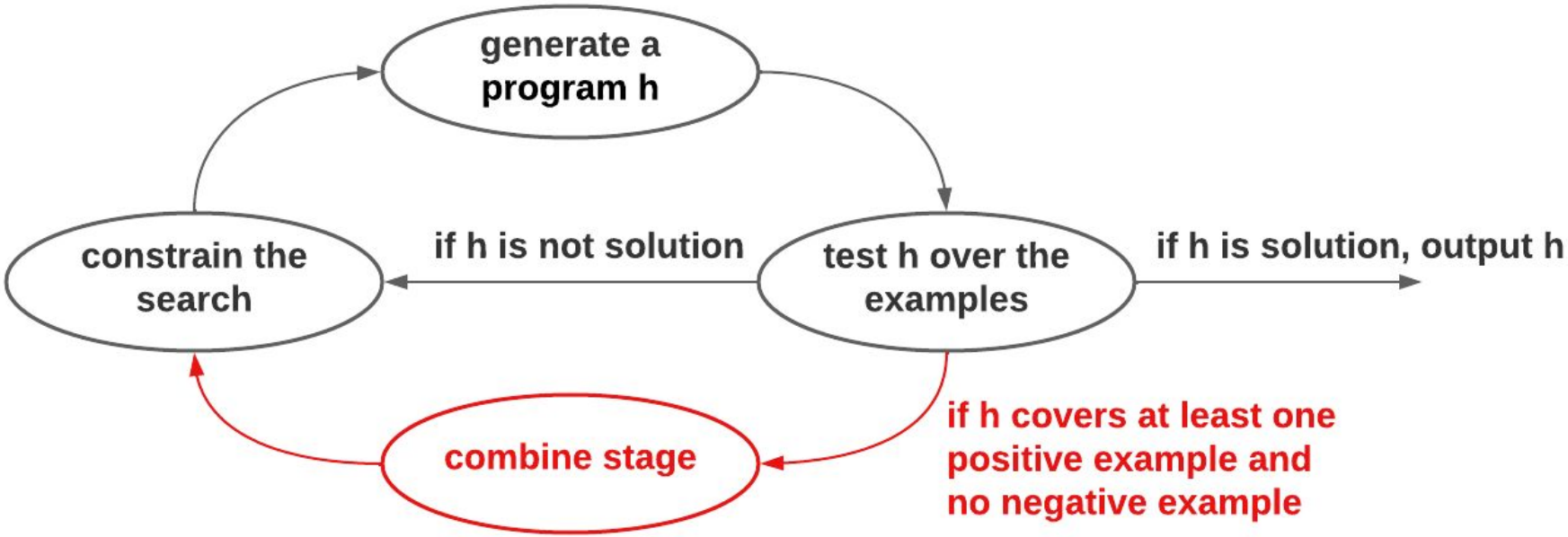
Learn small programs that cover some of the positive examples

Combine these programs to learn large programs with many rules and literals

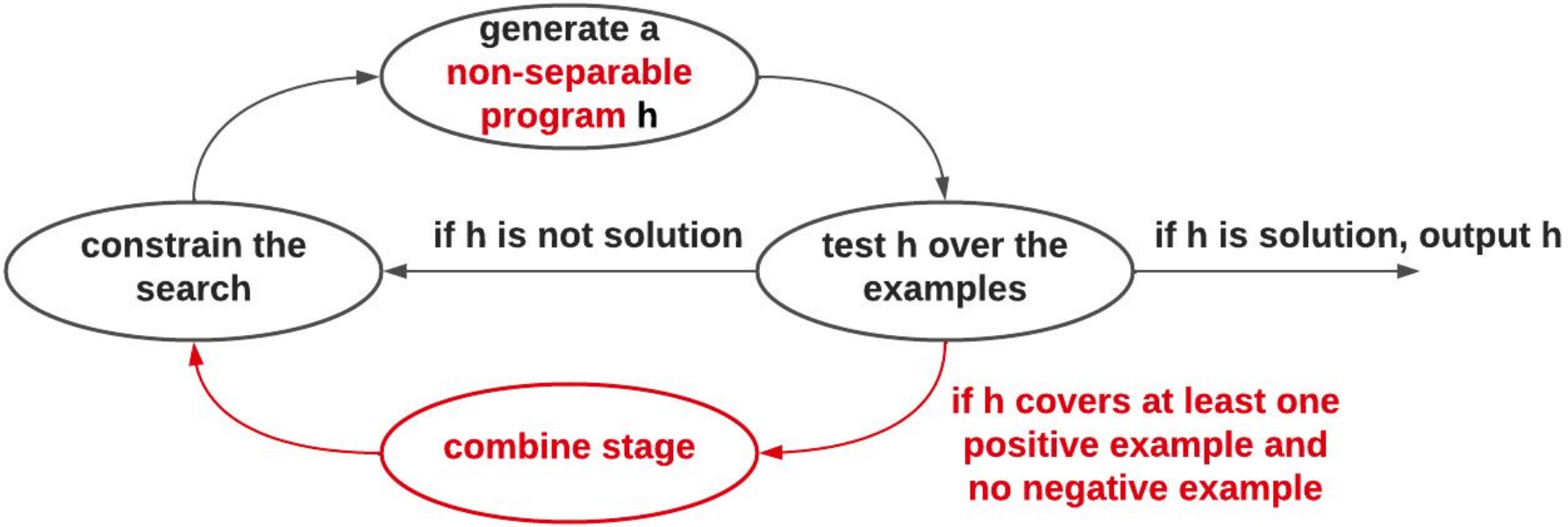
Our approach



Our approach



Our approach



win(Board,Player) ← cell(Board,0,Player), cell(Board,1,Player), cell(Board,2,Player)

win(Board,Player) ← cell(Board,0,Player), cell(Board,3,Player), cell(Board,6,Player)

win(Board,Player) ← cell(Board,0,Player), cell(Board,4,Player), cell(Board,8,Player)

win(Board,Player) ← cell(Board,0,Player), cell(Board,1,Player), cell(Board,2,Player)

win(Board,Player) ← cell(Board,0,Player), cell(Board,3,Player), cell(Board,6,Player)

win(Board,Player) ← cell(Board,0,Player), cell(Board,4,Player), cell(Board,8,Player)

Separable program

line(Board,0,Player) ← cell(Board,0,Player)
line(Board,Cell,Player) ← cell(Board,Cell,Player), above(Cell,Cell1), **line**(Board,Cell1,Player)

line(Board,0,Player) ← cell(Board,0,Player)
line(Board,Cell,Player) ← cell(Board,Cell,Player), above(Cell,Cell1), **line**(Board,Cell1,Player)

Non-separable program

Why should it help?

Why should it help?

Searching over non-separable programs only can vastly reduce the hypothesis space.

Why should it help?

Searching over non-separable programs only can vastly reduce the hypothesis space.

m rules in the hypothesis space,
at most k rules in a program

separable	non-separable
m^k	m

Combine stage

We search for a combination (a union) of programs that covers as many positive examples as possible, and is minimal in size.

Combine stage

r1: win(Board,Player) ← cell(Board,0,Player), cell(Board,1,Player), cell(Board,2,Player)

r2: win(Board,Player) ← cell(Board,0,Player), cell(Board,1,Player), cell(Board,5,Player)

r3: win(Board,Player) ← cell(Board,0,Player), cell(Board,3,Player), cell(Board,6,Player)

r4: win(Board,Player) ← cell(Board,0,Player), cell(Board,4,Player), cell(Board,8,Player)

r5: win(Board,Player) ← cell(Board,0,o), cell(Board,1,x), cell(Board,8,Player)

r6: win(Board,Player) ← cell(Board,0,Player), cell(Board,1,Player), next(Player,Player1), cell(Board,5,Player)

r7: win(Board,Player) ← cell(Board,8,Player), next(Player,Player1), cell(Board,6,Player1), cell(Board,7,Player1)

r8: win(Board,Player) ← cell(Board,4,Player), next(Player,Player1), cell(Board,1,Player1), cell(A,2,o)

r9: win(Board,Player) ← cell(Board,0,o), next(Board,1,Player), next(Player,Player1), cell(Board,7,Player1), cell(Board,8,Player1)

...

Combine stage

r1: win(Board,Player) ← cell(Board,0,Player), cell(Board,1,Player), cell(Board,2,Player)

~~r2: win(Board,Player) ← cell(Board,0,Player), cell(Board,1,Player), cell(Board,5,Player)~~

r3: win(Board,Player) ← cell(Board,0,Player), cell(Board,3,Player), cell(Board,6,Player)

r4: win(Board,Player) ← cell(Board,0,Player), cell(Board,4,Player), cell(Board,8,Player)

~~r5: win(Board,Player) ← cell(Board,0,o), cell(Board,1,x), cell(Board,8,Player)~~

~~r6: win(Board,Player) ← cell(Board,0,Player), cell(Board,1,Player), next(Player,Player1), cell(Board,5,Player)~~

~~r7: win(Board,Player) ← cell(Board,8,Player), next(Player,Player1), cell(Board,6,Player1), cell(Board,7,Player1)~~

~~r8: win(Board,Player) ← cell(Board,4,Player), next(Player,Player1), cell(Board,1,Player1), cell(A,2,o)~~

~~r9: win(Board,Player) ← cell(Board,0,o), next(Board,1,Player), next(Player,Player1), cell(Board,7,Player1), cell(Board,8,Player1)~~

...

Implementation

We implement our approach in Combo.

Implementation

We implement our approach in Combo.

Theorem: Combo always returns an optimal (minimal size) solution if one exists.

Does it work?

Q1. Can combining non-separable programs improve predictive accuracies and learning times?

Q2. How does COMBO compare against other approaches?

Game playing

Game playing

Task	COMBO	POPPER	DCC	ALEPH
md	13 ± 1	3357 ± 196	*timeout*	**4 ± 0**
buttons	**23 ± 3**	*timeout*	*timeout*	99 ± 0
rps	87 ± 15	*timeout*	*timeout*	**20 ± 0**
coins	**490 ± 35**	*timeout*	*timeout*	*timeout*
buttons-g	**3 ± 0**	*timeout*	*timeout*	86 ± 0
coins-g	105 ± 6	*timeout*	*timeout*	**9 ± 0**
attrition	**26 ± 1**	*timeout*	*timeout*	678 ± 25
centipede	**9 ± 0**	1102 ± 136	2104 ± 501	12 ± 0

Learning times (s) with a timeout of 60 minutes

Game playing

Task	COMBO	POPPER	DCC	ALEPH
md	13 ± 1	3357 ± 196	*timeout*	**4 ± 0**
buttons	**23 ± 3**	*timeout*	*timeout*	99 ± 0
rps	87 ± 15	*timeout*	*timeout*	**20 ± 0**
coins	**490 ± 35**	*timeout*	*timeout*	*timeout*
buttons-g	**3 ± 0**	*timeout*	*timeout*	86 ± 0
coins-g	105 ± 6	*timeout*	*timeout*	**9 ± 0**
attrition	**26 ± 1**	*timeout*	*timeout*	678 ± 25
centipede	**9 ± 0**	1102 ± 136	2104 ± 501	12 ± 0

Learning times (s) with a timeout of 60 minutes

Task	COMBO	POPPER	DCC	ALEPH
md	**100 ± 0**	37 ± 13	**100 ± 0**	94 ± 0
buttons	**100 ± 0**	19 ± 0	**100 ± 0**	96 ± 0
rps	**100 ± 0**	18 ± 0	**100 ± 0**	**100 ± 0**
coins	**100 ± 0**	17 ± 0	**100 ± 0**	17 ± 0
buttons-g	**100 ± 0**	50 ± 0	86 ± 1	**100 ± 0**
coins-g	**100 ± 0**	50 ± 0	90 ± 6	**100 ± 0**
attrition	**98 ± 0**	2 ± 0	2 ± 0	**98 ± 0**
centipede	**100 ± 0**	**100 ± 0**	81 ± 6	**100 ± 0**

Predictive accuracies

Buttons

```
next(A,B):- p(B), c(C), does(A,D,C), true(A,B), input(D,C).
next(A,B):- input(C,E), p(D), true(A,D), b(E), does(A,C,E), q(B).
next(A,B):- input(C,D), not_true(A,B), does(A,C,D), p(B), a(D).
next(A,B):- a(C), does(A,D,C), true(A,B), q(B), input(D,C).
next(A,B):- input(C,E), p(B), true(A,D), b(E), does(A,C,E), q(D).
next(A,B):- c(D), true(A,C), r(B), role(E), does(A,E,D), q(C).
next(A,B):- true(A,C), my_succ(C,B).
next(A,B):- input(C,D), does(A,C,D), true(A,B), r(B), b(D).
next(A,B):- input(C,D), does(A,C,D), true(A,B), r(B), a(D).
next(A,B):- true(A,E), c(C), does(A,D,C), q(B), r(E), input(D,C).
```

Graph problems

Task	COMBO	POPPER	DCC	ALEPH
adj_red	**2 ± 0**	5 ± 0	6 ± 0	479 ± 349
connected	**5 ± 1**	112 ± 71	735 ± 478	435 ± 353
cyclic	**35 ± 13**	1321 ± 525	1192 ± 456	1120 ± 541
colouring	**2 ± 0**	6 ± 0	5 ± 0	2373 ± 518
undirected	**2 ± 0**	6 ± 0	6 ± 0	227 ± 109
2children	**2 ± 0**	7 ± 0	6 ± 0	986 ± 405

Learning times (s) with a timeout of 60 minutes

Task	COMBO	POPPER	DCC	ALEPH
adj_red	**100 ± 0**	**100 ± 0**	**100 ± 0**	50 ± 0
connected	**98 ± 0**	81 ± 7	82 ± 7	51 ± 0
cyclic	**89 ± 3**	80 ± 7	85 ± 5	50 ± 0
colouring	**98 ± 1**	**98 ± 1**	**98 ± 1**	50 ± 0
undirected	**100 ± 0**	**100 ± 0**	**100 ± 0**	50 ± 0
2children	**100 ± 0**	99 ± 0	**100 ± 0**	50 ± 0

Predictive accuracies

Conclusion

- Approach which learning small **non-separable** programs that cover some of the examples and then **combines** these programs to learn **large** programs.

- Our approach can drastically improve learning performance.

Limitations

- Learn programs from noisy examples by finding combinations that cover as many positive examples and as few negative examples as possible

References

- Muggleton, S. ‘Inductive logic programming’, New Generation Computing, 8(4), 295–318, (1991).
- Cropper, A. and Dumancic, S., ‘Inductive logic programming at 30: A new introduction’, J. Artif. Intell. Res., 74, 765–850, (2022).
- Cropper, A., Morel, R.: Learning programs by learning from failures. Machine Learning 110(4), 801–856 (2021).
- Cropper, A.: Learning logic programs through divide, constrain, and conquer. AAAI 2022.
- Srinivasan, A.: The ALEPH manual. Machine Learning at the Computing Laboratory (2001).
- Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T. ‘Clingo = ASP + control: Preliminary report’, CoRR, (2014).

Tasks

```
next(A,B):-c_p(B),c_c(C),does(A,D,C),my_true(A,B),my_input(D,C).
next(A,B):-my_input(C,E),c_p(D),my_true(A,D),c_b(E),does(A,C,E),c_q(B).
next(A,B):-my_input(C,D),not_my_true(A,B),does(A,C,D),c_p(B),c_a(D).
next(A,B):-c_a(C),does(A,D,C),my_true(A,B),c_q(B),my_input(D,C).
next(A,B):-my_input(C,E),c_p(B),my_true(A,D),c_b(E),does(A,C,E),c_q(D).
next(A,B):-c_c(D),my_true(A,C),c_r(B),role(E),does(A,E,D),c_q(C).
next(A,B):-my_true(A,C),my_succ(C,B).
next(A,B):-my_input(C,D),does(A,C,D),my_true(A,B),c_r(B),c_b(D).
next(A,B):-my_input(C,D),does(A,C,D),my_true(A,B),c_r(B),c_a(D).
next(A,B):-my_true(A,E),c_c(C),does(A,D,C),c_q(B),c_r(E),my_input(D,C).
```

```
zendo4(A):- piece(A,C),contact(C,B),strange(B),upright(C).
zendo4(A):- piece(A,D),contact(D,C),coord2(C,B),coord2(D,B).
zendo4(A):- piece(A,D),contact(D,C),size(C,B),red(D),medium(B).
zendo4(A):- piece(A,D),blue(D),lhs(D),piece(A,C),size(C,B),small(B).
```

```
goal(A,B,C):- not_my_true(A,D),int_0(C),role(B),prop_q(D).
goal(A,B,C):- not_my_true(A,D),prop_r(D),int_0(C),role(B).
goal(A,B,C):- int_0(C),not_my_true(A,D),role(B),prop_p(D).
goal(A,B,C):- agent_robot(B),prop_p(E),true(A,E),true(A,F),prop_r(D),prop_7(F),int_100(C),true(A,D).
```