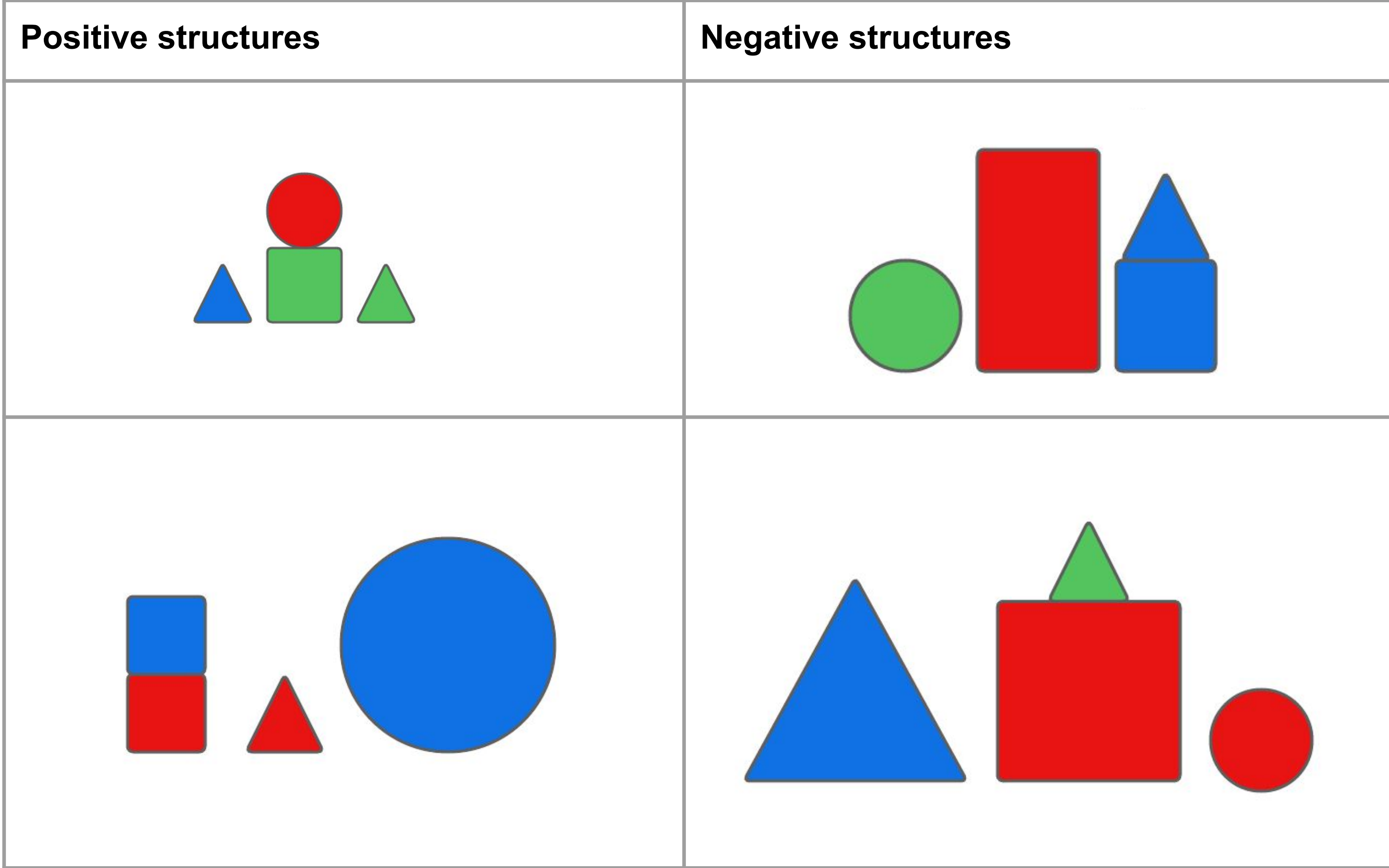
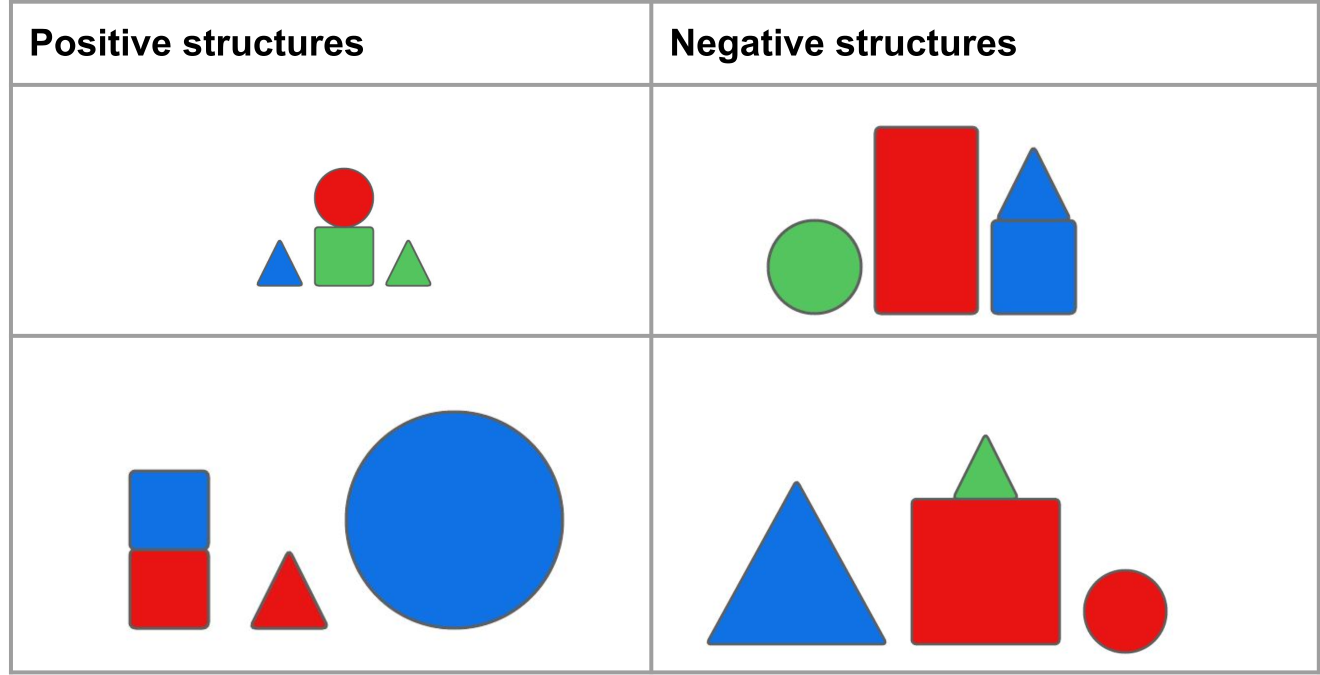
Efficient synthesis of logic programs through problem decomposition

Céline Hocquette
University of Oxford / Southampton

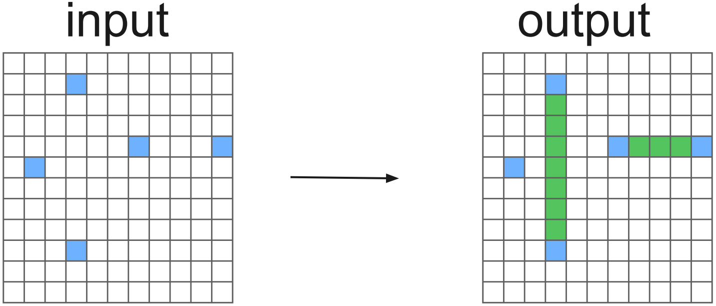


Positive structures
Negative structures

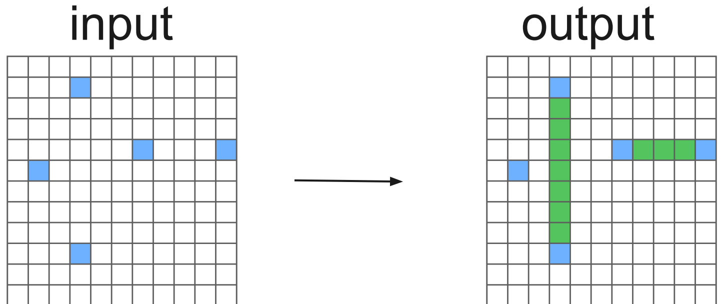


There must be a red piece in contact with a square piece

Abstraction and Reasoning Corpus (ARC) [Chollet, 2019]

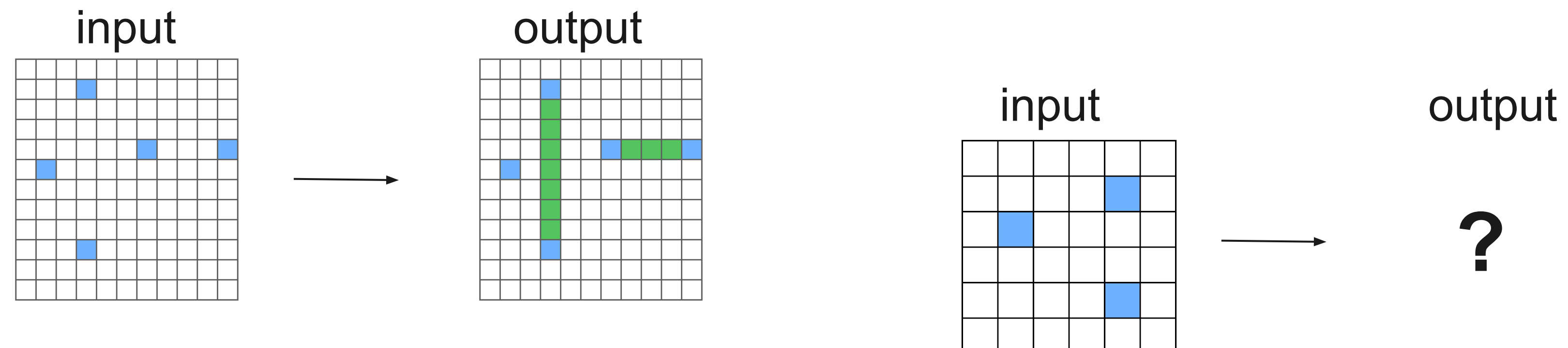


Abstraction and Reasoning Corpus (ARC) [Chollet, 2019]



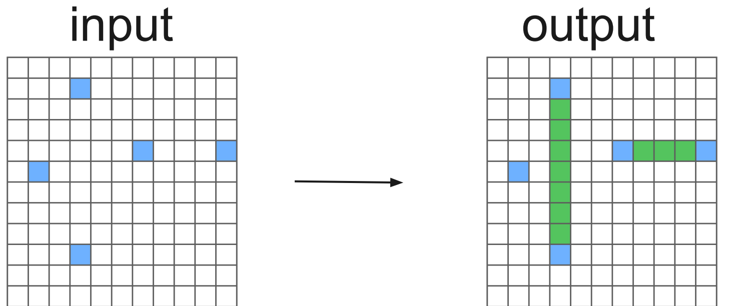
Color in green pixels in between two blue pixels

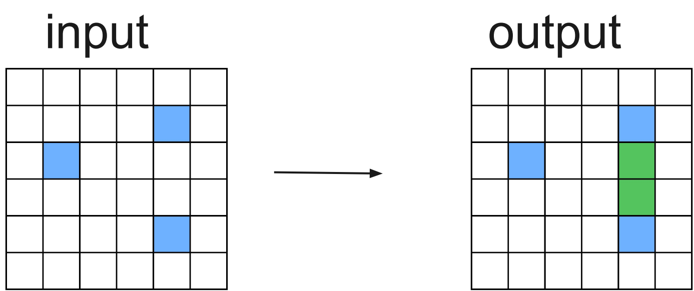
Abstraction and Reasoning Corpus (ARC) [Chollet, 2019]



Color in green pixels in between two blue pixels

Abstraction and Reasoning Corpus (ARC) [Chollet, 2019]





Inductive Logic Programming

Inductive Logic Programming

a form of program synthesis based on logic

Inductive Logic Programming

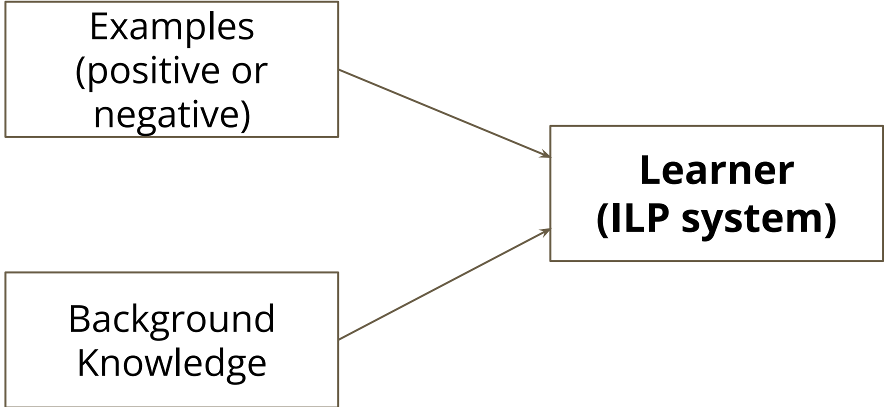
Examples
(positive or
negative)

Inductive Logic Programming

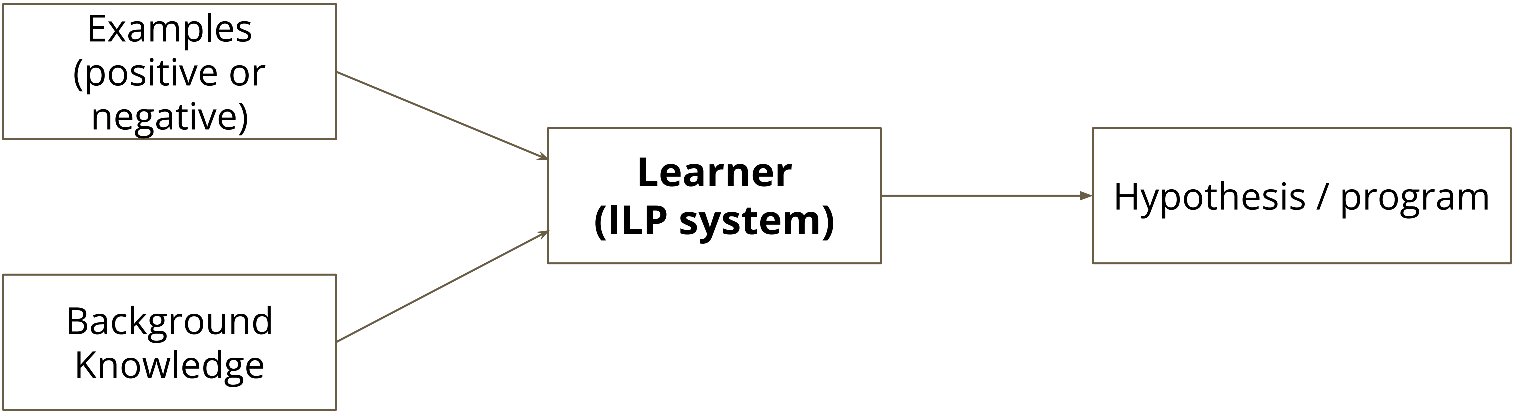
Examples (positive or negative)

Background Knowledge

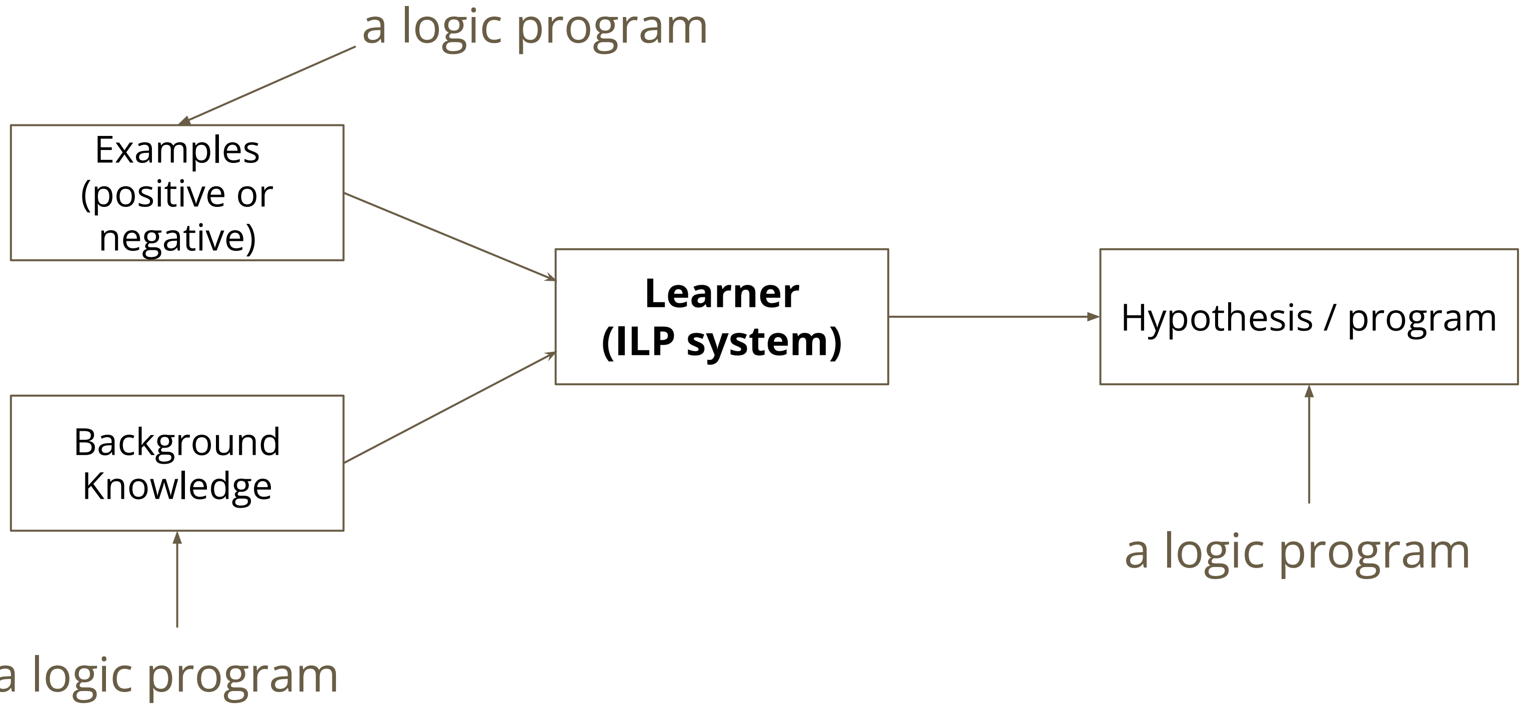
Inductive Logic Programming



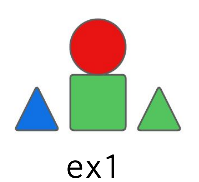
Inductive Logic Programming

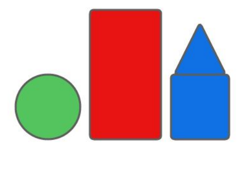


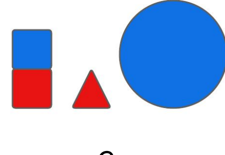
Inductive Logic Programming

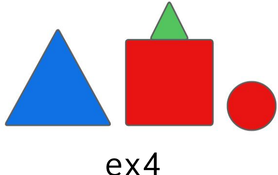


Positive examples	Negative examples
zendo(ex1). zendo(ex2).	zendo(ex3). zendo(ex4).

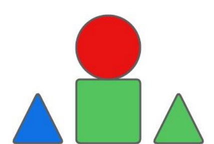

ex1

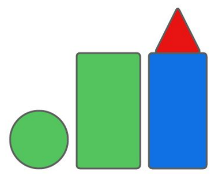

ex3

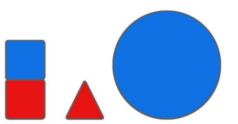

ex2

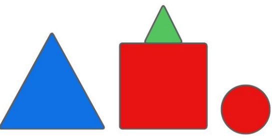

ex4

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). circle(p2). triangle(p4). contact(p2, p3). on(p2, p3). right(p4, p3). left(p1, p2). …



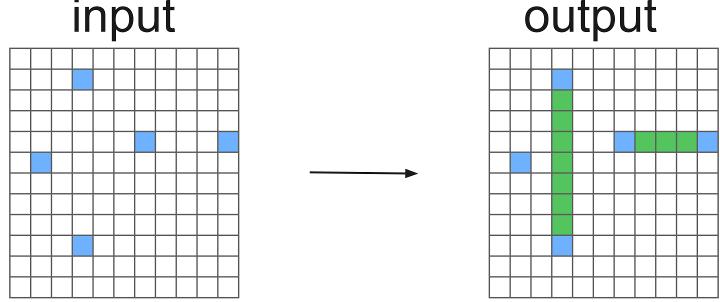






Program

```
zendo(Structure) ←
    piece(Structure,Piece1),
    red(Piece1),
    contact(Piece1,Piece2),
    square(Piece2).
```



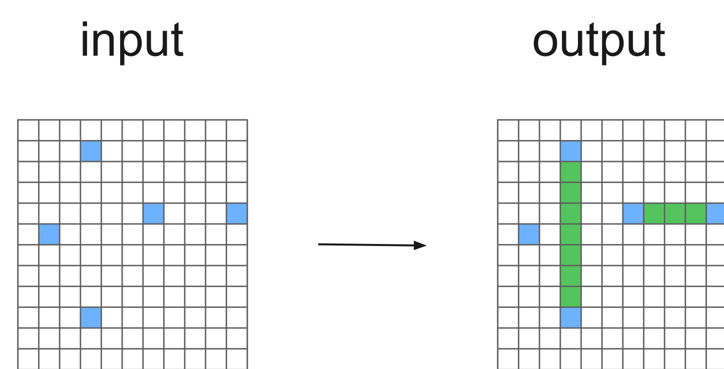
Program
out(X,Y,C) ← in(X,Y,C).
out(X,Y,green) ← in(X1,Y,blue), in(X2,Y,blue), X1<X<X2.
out(X,Y,green) ← in(X,Y1,blue), in(X,Y2,blue), Y1<Y<Y2.

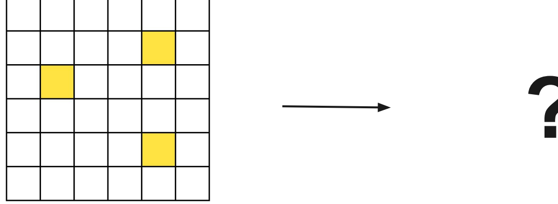
Why ILP?

Why ILP?

- high generalisation ability

```
out(X,Y,Color) ← in(X,Y,Color).
```



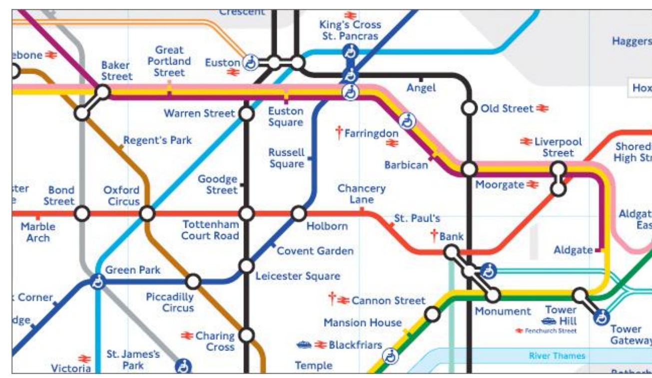


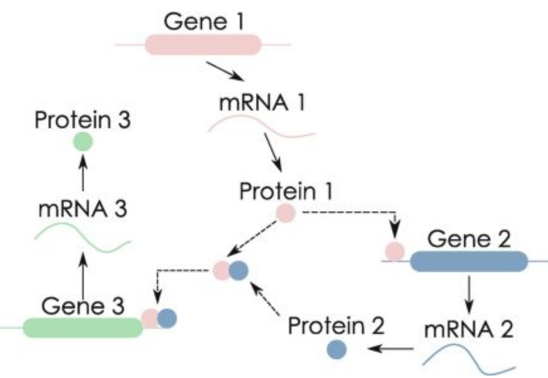
Why ILP?

- high generalisation ability
- learn from small amount of data

Why ILP?

- high generalisation ability
- learn from small amount of data
- learn from highly relational data





Why ILP?

- high generalisation ability
- learn from small amount of data
- learn from highly relational data
- learn explainable programs

Why ILP?

- high generalisation ability
- learn from small amount of data
- learn from highly relational data
- learn explainable programs
- reason about programs

Main challenge

Main challenge

hypothesis space = the set of all programs which may be learned by the learner

Large hypothesis spaces!

Main challenge

hypothesis space = the set of all programs which may be learned by the learner

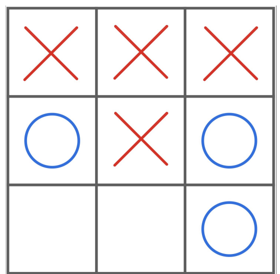
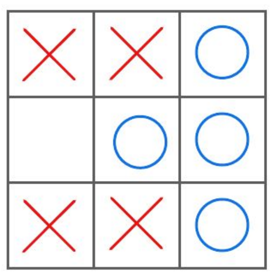
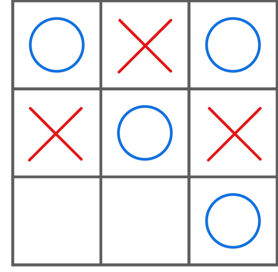
Large hypothesis spaces!

Zendo: 10^8 hypotheses with 1 rule and at most 6 variables and at most 6 literals

In this presentation: problem decomposition

1. Combining rules to learn programs with many rules
2. Joining rules to learn programs with big rules
3. Example decomposition

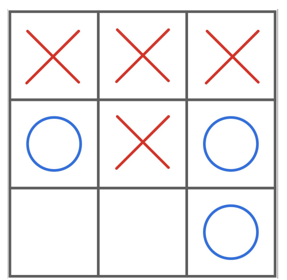
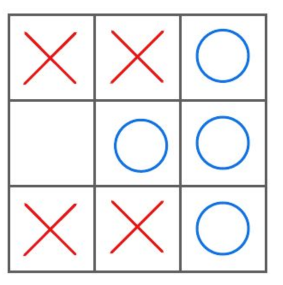
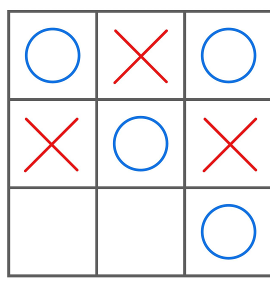
1 - Combining rules to learn programs with many rules

```
win(Board,Player) ← cell(Board,X,0,Player),cell(Board,X,1,Player),cell(Board,X,2,Player)
win(Board,Player) ← cell(Board,0,Y,Player),cell(Board,1,Y,Player),cell(Board,2,Y,Player)
win(Board,Player) ← cell(Board,0,0,Player),cell(Board,1,1,Player),cell(Board,2,2,Player)
win(Board,Player) ← cell(Board,2,0,Player),cell(Board,1,1,Player),cell(Board,0,2,Player)
```

Learning logic programs by combing programs, Andrew Cropper and Céline Hocquette, ECAI, 2023

1 - Combining rules to learn programs with many rules

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

r2: win(Board,Player) ← cell(Board,0,Y,Player),cell(Board,1,Y,Player),cell(Board,2,Y,Player)

r3: win(Board,Player) ← cell(Board,0,0,Player),cell(Board,1,1,Player),cell(Board,2,2,Player)

r4: win(Board,Player) ← cell(Board,2,0,Player),cell(Board,1,1,Player),cell(Board,0,2,Player)
```

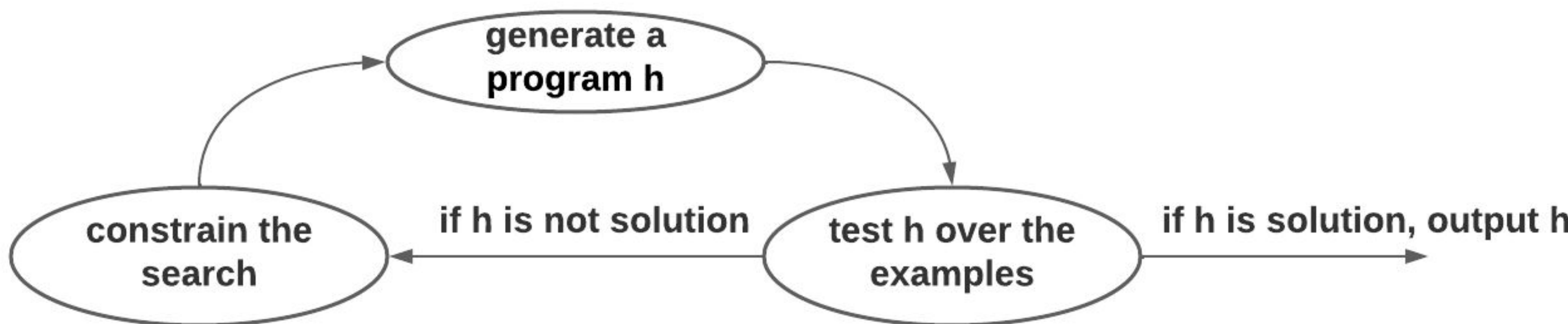
r_1, r_2, r_3 and r_4 do not depend on each other

Idea

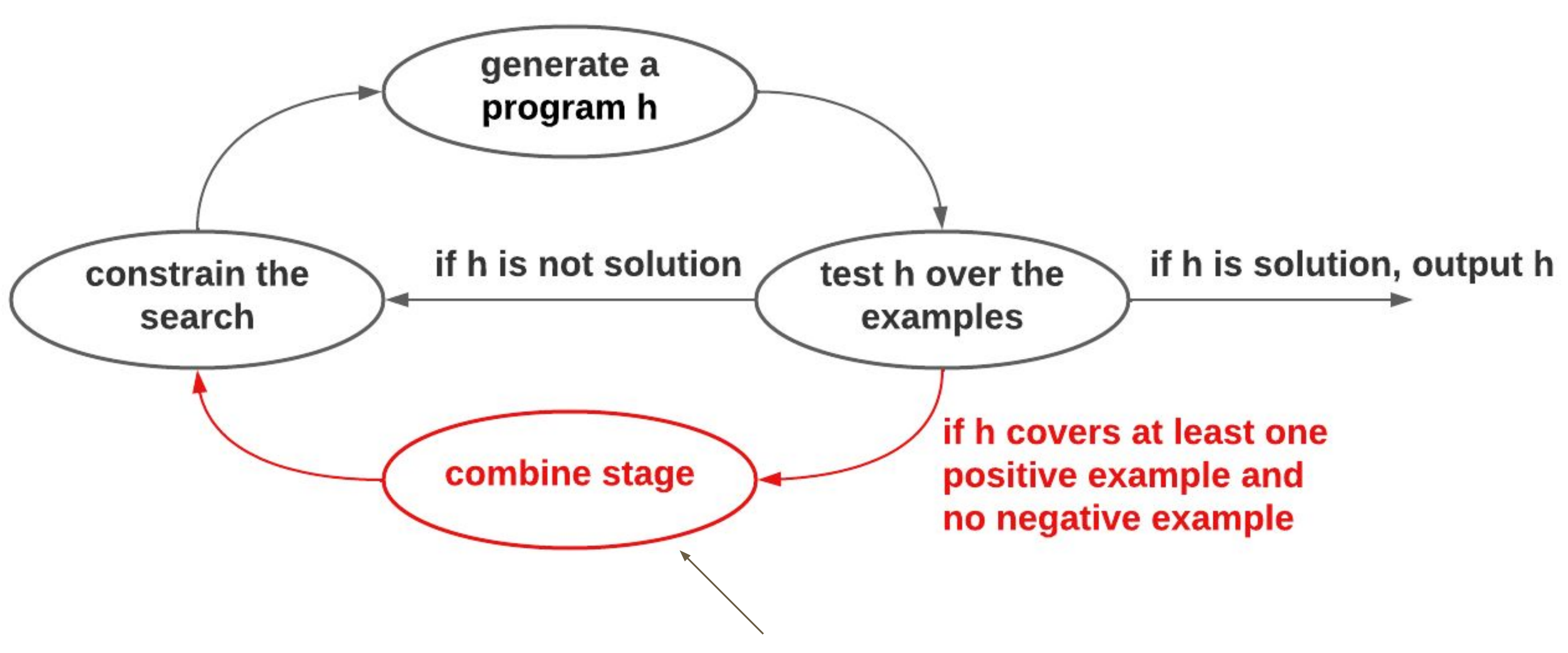
Learn small programs that entail some of the positive examples

Reason about the coverage of programs to find a combination of programs that entails many positive examples

Our approach



Our approach



solved using a constraint optimisation approach

Combine stage

Input:

Program	Positive examples covered	Size
p1	{e1,e2,e3}	3
p2	{e9}	3
p3	{e1,e3,e5,e6,e7}	4
p4	{e2,e6,e7}	4
p5	{e2,e5,e8,e9}	5
p6	{e8,e9}	6

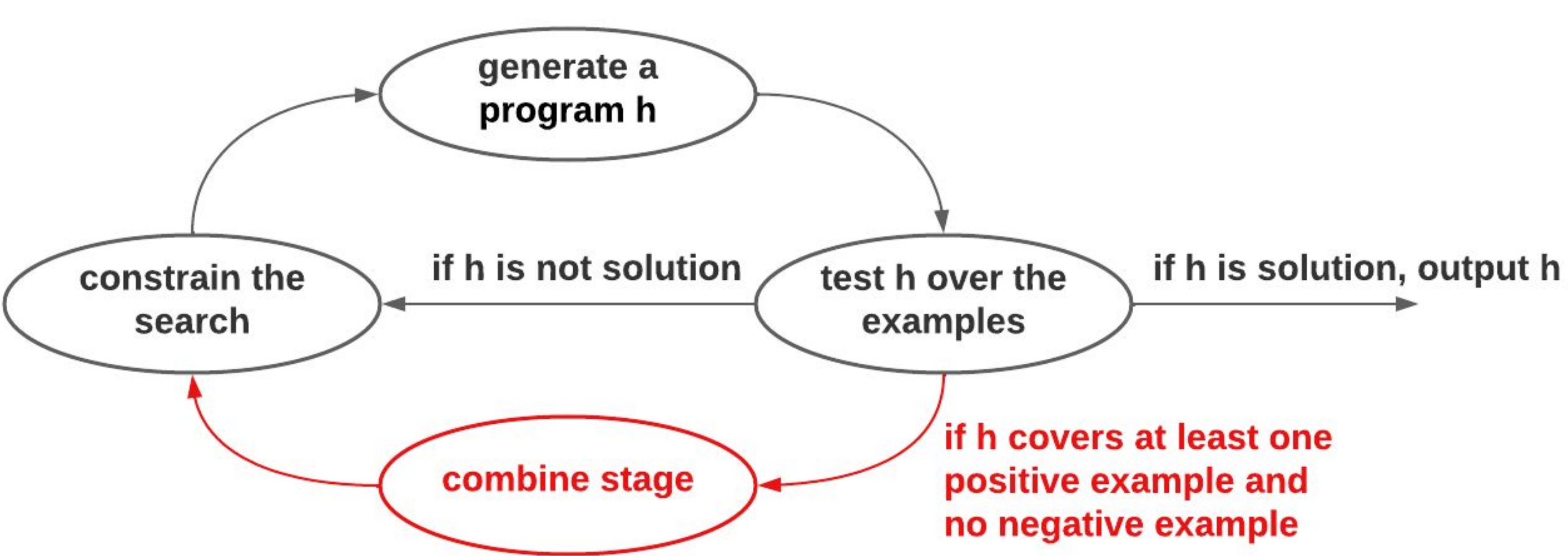
Combine stage

Input:

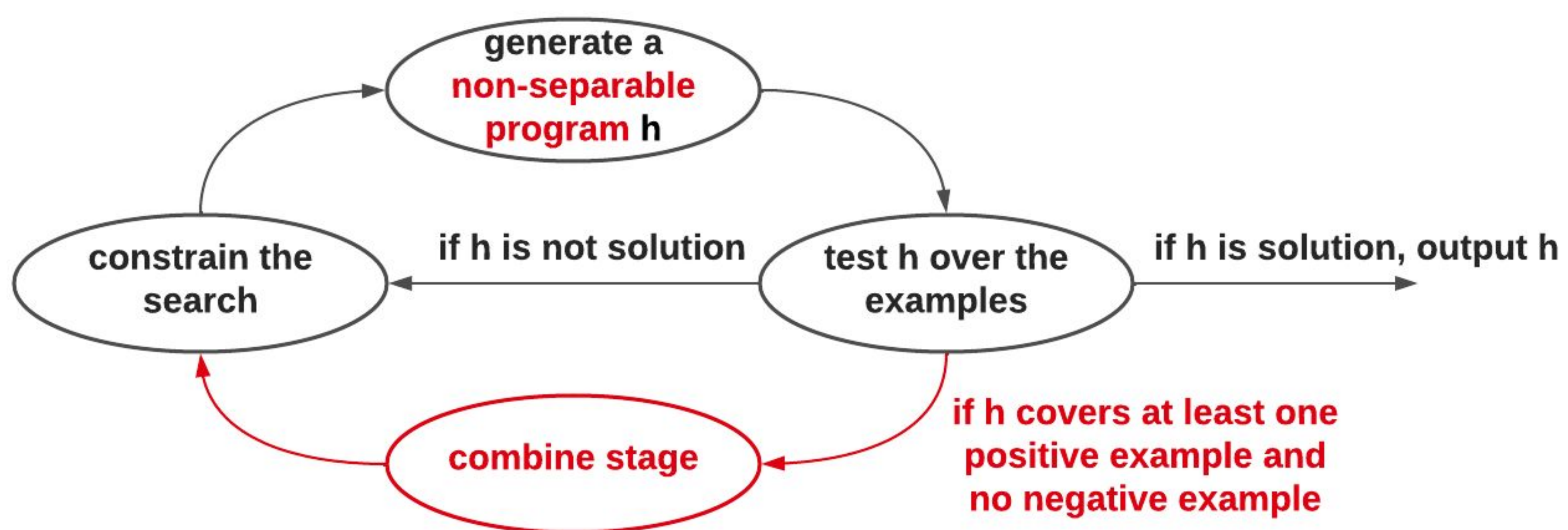
Program	Positive examples covered	Size
p1	{e1,e2,e3}	3
p2	{e9}	3
p3	{e1,e3,e5,e6,e7}	4
p4	{e2,e6,e7}	4
p5	{e2,e5,e8,e9}	5
p6	{e8,e9}	6

Output:
{p1,p3,p5} covers {e1,e2,e3,e5,e6,e7,e8,e9} and has size 12

Our approach



Our approach



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

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

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

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

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

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

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

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

How well does it work?

Task	With combine	Without combine
md	13 ± 1	3357 ± 196
buttons	**23** ± **3**	*timeout*
rps	87 ± 15	*timeout*
coins	**490** ± **35**	*timeout*
buttons-g	**3** ± **0**	*timeout*
coins-g	105 ± 6	*timeout*
attrition	**26** ± **1**	*timeout*
centipede	**9** ± **0**	1102 ± 136

Learning times (s) with a timeout of 60 minutes

Task	With combine	Without combine
md	**100** ± **0**	37 ± 13
buttons	**100** ± **0**	19 ± 0
rps	**100** ± **0**	18 ± 0
coins	**100** ± **0**	17 ± 0
buttons-g	**100** ± **0**	50 ± 0
coins-g	**100** ± **0**	50 ± 0
attrition	**98** ± **0**	2 ± 0
centipede	**100** ± **0**	**100** ± **0**

Predictive accuracies (%)

Why does it work?

- We decompose a learning task into smaller tasks that can be solved separately

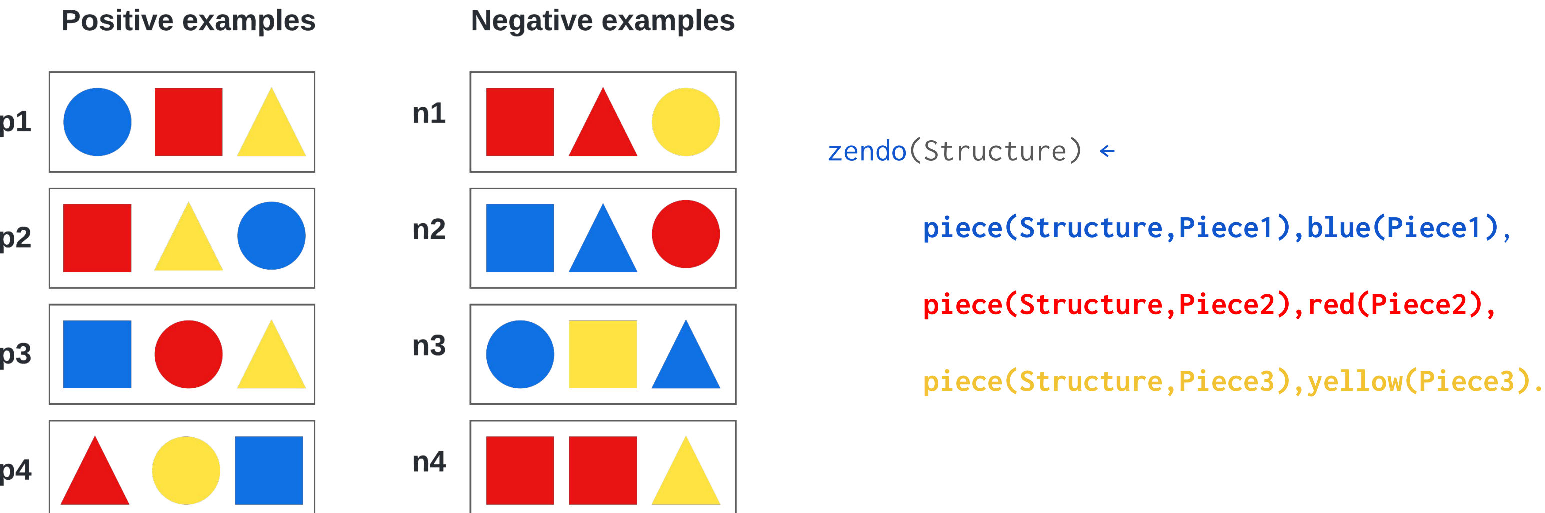
Why does it work?

- We decompose a learning task into smaller tasks that can be solved separately
- 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

2 - Joining rules to learn programs with big rules




```
zendo(Structure) ←
    piece(Structure,Piece1),blue(Piece1),
    piece(Structure,Piece2),red(Piece2),
    piece(Structure,Piece3),yellow(Piece3).
```

Learning big logical rules by joining small rules, Céline Hocquette, Andreas Niskanen, Rolf Morel, Matti Järvisalo, and Andrew Cropper, IJCAI, 2024.

Idea

Learn small rules that entail some positive and some negative examples

```
zendo1(Structure) ← piece(Structure,Piece1),blue(Piece1).

zendo2(Structure) ← piece(Structure,Piece2),red(Piece2).

zendo3(Structure) ← piece(Structure,Piece3),yellow(Piece3).
```

Idea

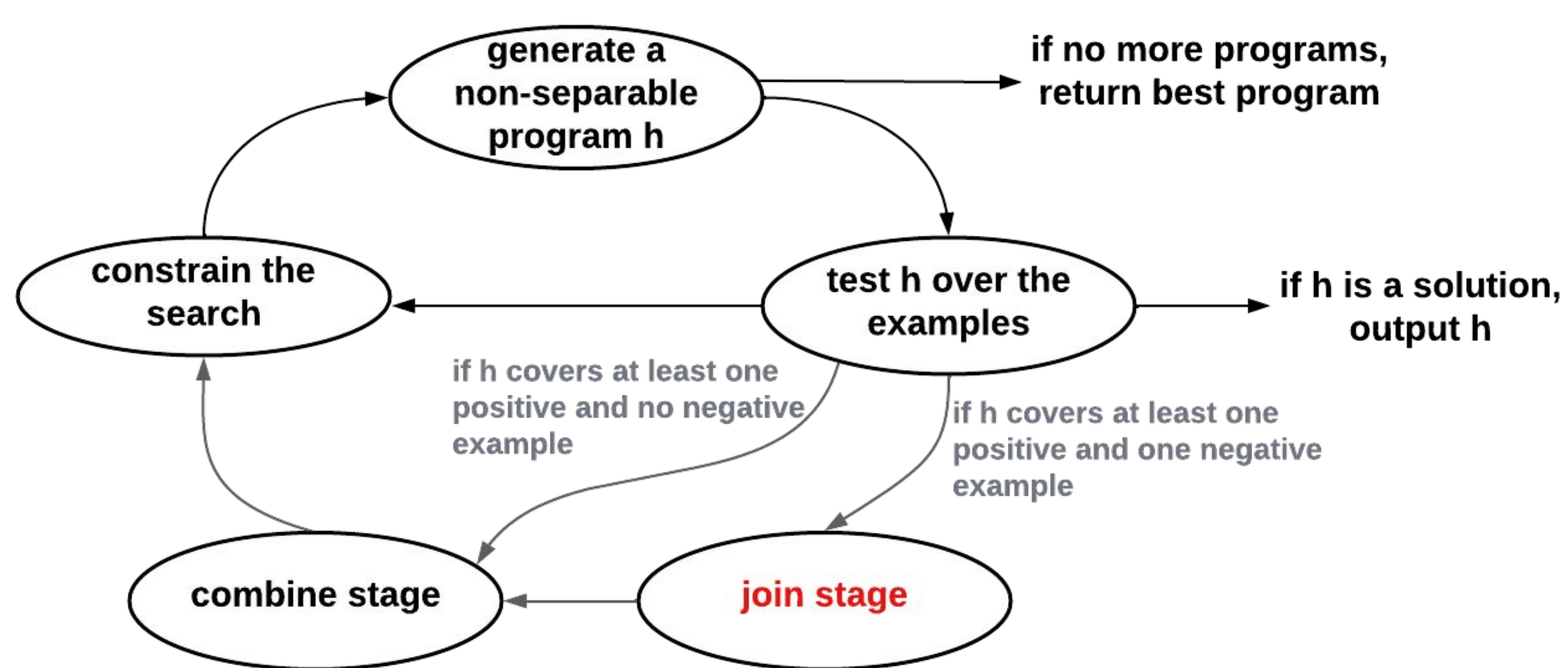
Learn small rules that entail some positive and some negative examples

```
zendo1(Structure) ← piece(Structure,Piece1),blue(Piece1).

zendo2(Structure) ← piece(Structure,Piece2),red(Piece2).

zendo3(Structure) ← piece(Structure,Piece3),yellow(Piece3).
```

Reason about the coverage of programs to find conjunctions that entail some positive examples and no negative examples

```
zendo(Structure) ← zendo1(Structure),zendo2(Structure),zendo3(Structure).
```

Our approach



Join stage

Input:

Program	Positive examples covered	Negative examples covered	Size
p1	{e1}	{n3}	2
p2	{e2}	{n3}	2
p3	{e1,e2}	{n1,n2}	3
p4	{e1,e2}	{n1,n3}	5
p5	{e1,e2}	{n1,n2}	5

Join stage

Input:

Program	Positive examples covered	Negative examples covered	Size
p1	{e1}	{n3}	2
p2	{e2}	{n3}	2
p3	{e1,e2}	{n1,n2}	3
p4	{e1,e2}	{n1,n3}	5
p5	{e1,e2}	{n2,n3}	5

Output:
c1={p3,p4,p5} covers {e1,e2} and has size 13

Join stage

Input:

Program	Positive examples covered	Negative examples covered	Size
p1	{e1}	{n3}	2
p2	{e2}	{n3}	2
p3	{e1,e2}	{n1,n2}	3
p4	{e1,e2}	{n1,n3}	5
p5	{e1,e2}	{n1,n2}	5

Output:

c1={p3,p4,p5} covers {e1,e2} and has size 13

c2={p1,p3} covers {e1} and has size 5

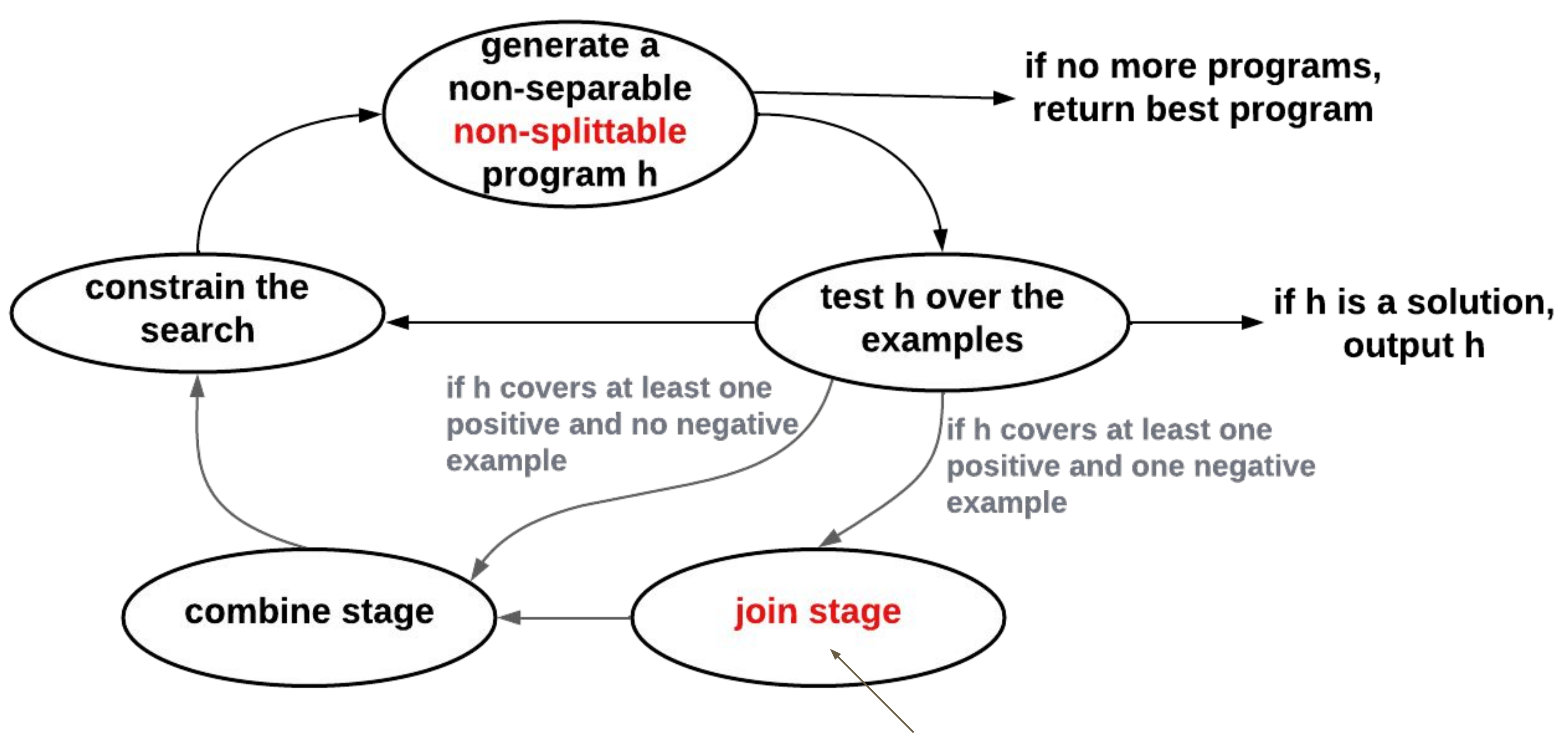
Join stage

Input:

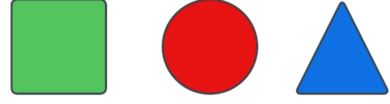
Program	Positive examples covered	Negative examples covered	Size
p1	{e1}	{n3}	2
p2	{e2}	{n3}	2
p3	{e1,e2}	{n1,n2}	3
p4	{e1,e2}	{n1,n3}	5
p5	{e1,e2}	{n1,n2}	5

Output:

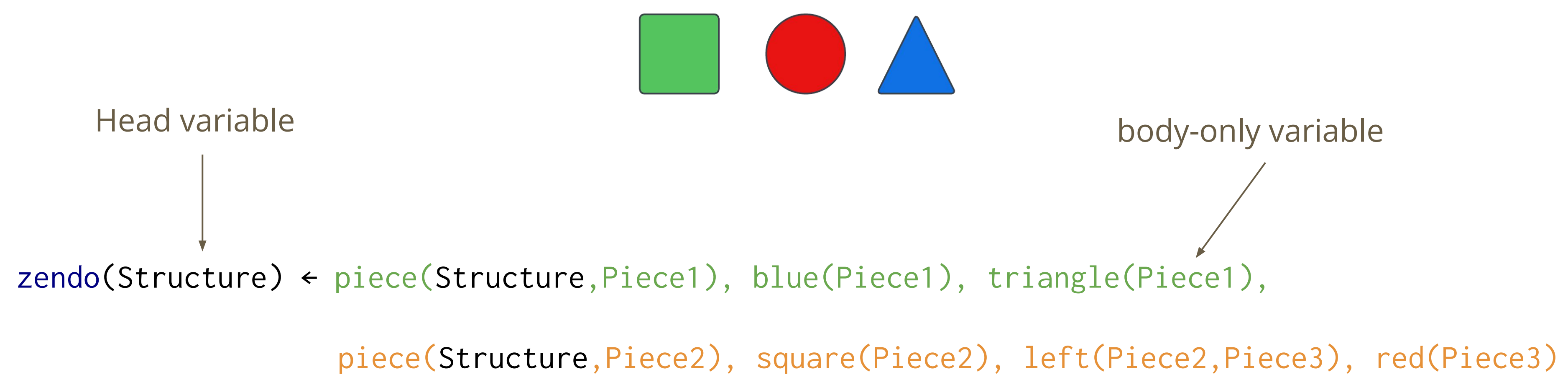
c1={p3,p4,p5} covers {e1,e2} and has size 13
c2={p1,p3} covers {e1} and has size 5
c3={p2,p3} covers {e2} and has size 5



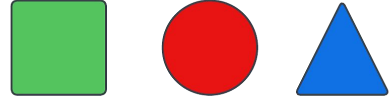
solved using a constraint satisfaction approach



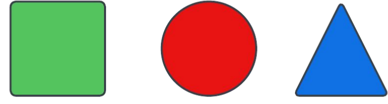
```
zendo(Structure) ← piece(Structure,Piece1), blue(Piece1), triangle(Piece1),
                   piece(Structure,Piece2), square(Piece2), left(Piece2,Piece3), red(Piece3)
```



Splittable program



```
zendo(Structure) ← piece(Structure,Piece1), blue(Piece1), triangle(Piece1),
                   piece(Structure,Piece2), square(Piece2), left(Piece2,Piece3), red(Piece3)
                   left(Piece1,Piece2)
```

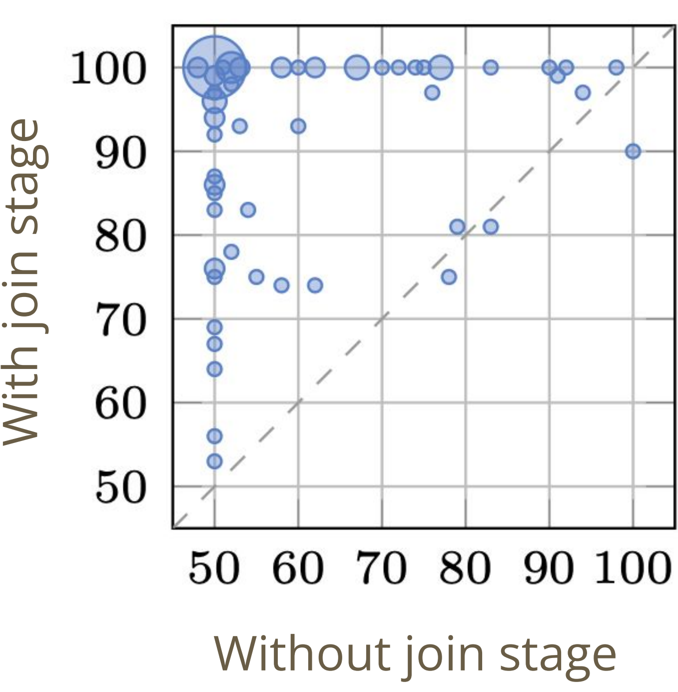


```
zendo(Structure) ← piece(Structure,Piece1), blue(Piece1), triangle(Piece1),
                   piece(Structure,Piece2), square(Piece2), left(Piece2,Piece3), red(Piece3)
                   left(Piece1,Piece2)
```

Non-splittable program

How well does it work?

Predictive accuracies (%)



Why does it work?

- We decompose a learning task into smaller tasks that can be solved separately
- Searching over non-splittable programs only can vastly reduce the hypothesis space.

3 - Example decomposition

Insert the letter a at position 2

Input	Output
[l, i, o, n]	[l, a, i, o, n]
[t, i, g, e, r]	[t, a, i, g, e, r]

Relational decomposition for program synthesis, Céline Hocquette, and Andrew Cropper, under review at AAAI.

3 - Example decomposition

Insert the letter a at position 2

Input	Output
[l, i, o, n]	[l, a, i, o, n]
[t, i, g, e, r]	[t, a, i, g, e, r]

```
def f(xs):
    return cons(head(xs),cons('a',tail(xs))
```

Relational decomposition for program synthesis, Céline Hocquette, and Andrew Cropper, under review at AAAI.

3 - Example decomposition

Insert the letter a at position 3

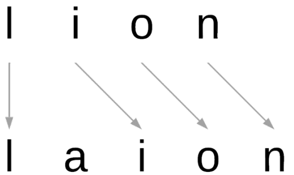
Input	Output
[l, i, o, n]	[l, i, a, o, n]
[t, i, g, e, r]	[t, i, a, g, e, r]

```
def f(xs):
    return cons(head(xs),cons(head(tail(xs)),cons('a',tail(tail(xs)))))
```

3 - Example decomposition

Insert the letter a at position 2

Input	Output
[l, i, o, n]	[l, a, i, o, n]
[t, i, g, e, r]	[t, a, i, g, e, r]



t i g e r

t a i g e r

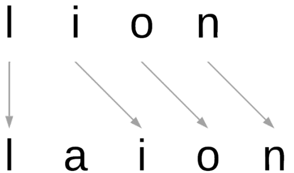
```
in(1,l).
in(2,i).
in(3,o).
in(4,n).
```

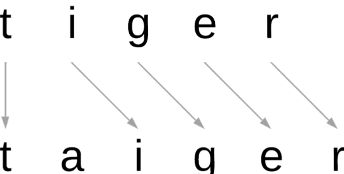
```
out(1,l).
out(2,a).
out(3,i).
out(4,o).
out(5,n).
```

3 - Example decomposition

Insert the letter a at position 2

Input	Output
[l, i, o, n]	[l, a, i, o, n]
[t, i, g, e, r]	[t, a, i, g, e, r]



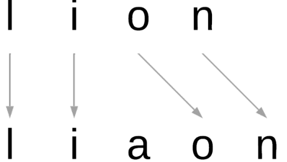


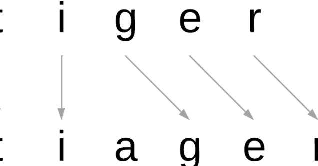

```
out(I,V) ← I<2, in(I,V).
out(2,a).
out(I,V) ← I>2, in(I-1,V).
```

3 - Example decomposition

Insert the letter a at position 3

Input	Output
[l, i, o, n]	[l, i, a, o, n]
[t, i, g, e, r]	[t, i, a, g, e, r]

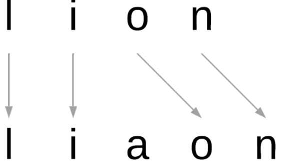


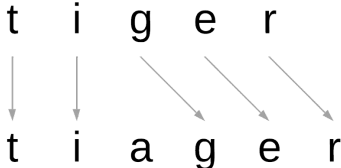


3 - Example decomposition

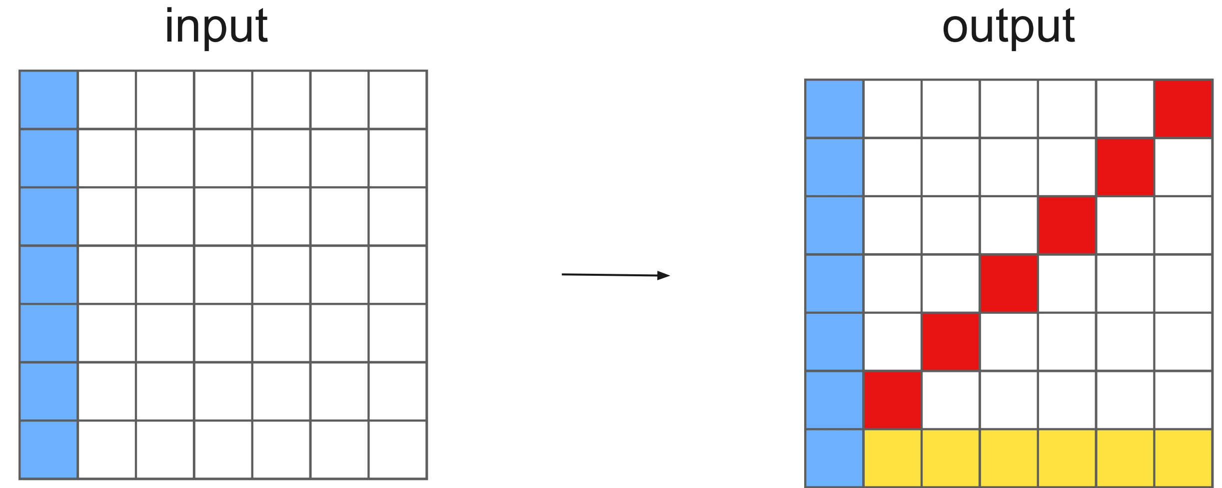
Insert the letter a at position 3

Input	Output
[l, i, o, n]	[l, i, a, o, n]
[t, i, g, e, r]	[t, i, a, g, e, r]

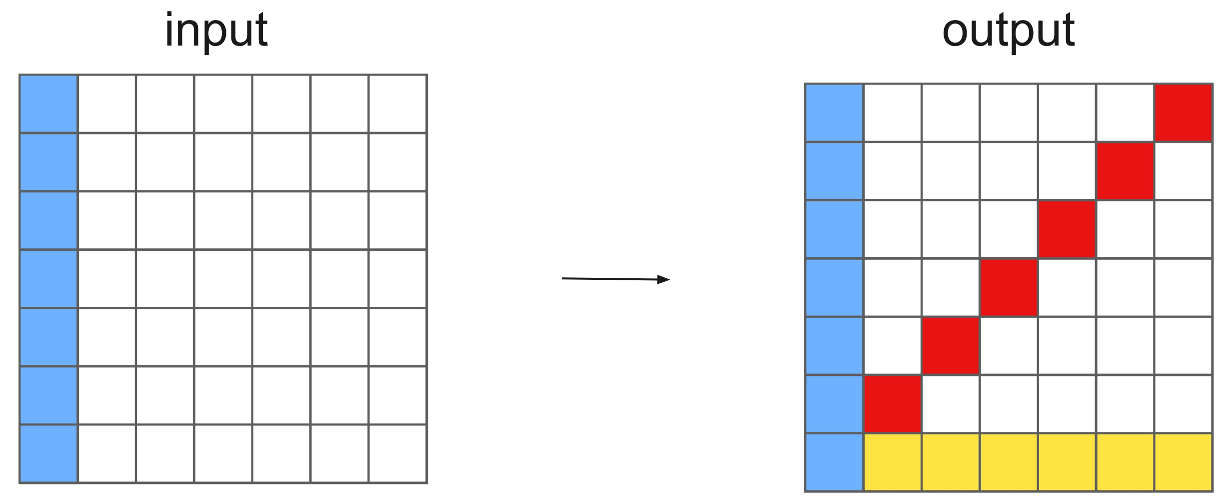





```
out(I,V) ← I<3, in(I,V).
out(3,a).
out(I,V) ← I>3, in(I-1,V).
```



Relational decomposition for program synthesis, Céline Hocquette, and Andrew Cropper, under review at AAAI.




```
out(X,Y,C) ← in(X,Y,C).
out(X,Y,yellow) ← empty(X,Y), height(X).
out(X,Y,red) ← empty(X,Y), height(X+Y-1).
```

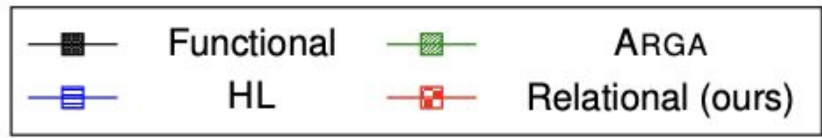
Relational decomposition for program synthesis, Céline Hocquette, and Andrew Cropper, under review at AAAI.

input

output

```
out(X,Y,C) ← in(X,Y,C).
out(X,Y,red) ← empty(X,Y), in(X1,Y,C), in(X,Y1,C).
```

How well does it work?



List functions

Accuracy (%)

0
20
40

1
10
20

Maximum learning time (mins)

Visual reasoning

Accuracy (%)

0
5
10
15
20

1
10
20

Maximum learning time (mins)

Why does it work?

- Decomposes a synthesis task into smaller ones by decomposing each training example into multiple examples

Why does it work?

- Decomposes a synthesis task into smaller ones by decomposing each training example into multiple examples
- Learn relations between elements / pixels

What is missing?

- More core primitives (we used only basic arithmetic relations).
- Better search

Conclusion

Decomposing a synthesis task into smaller ones can improve learning performance

Interested?

Open-source ILP system Popper

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

Thank you! Questions?

celinehocquette@gmail.com