Relational decomposition for program synthesis

Céline Hocquette and **Andrew Cropper**

University of Oxford

What is this talk about?

A different way to look at program synthesis

What is program synthesis?

Given:

- examples, typically input -> output

What is program synthesis?

Given:

- examples, typically input -> output

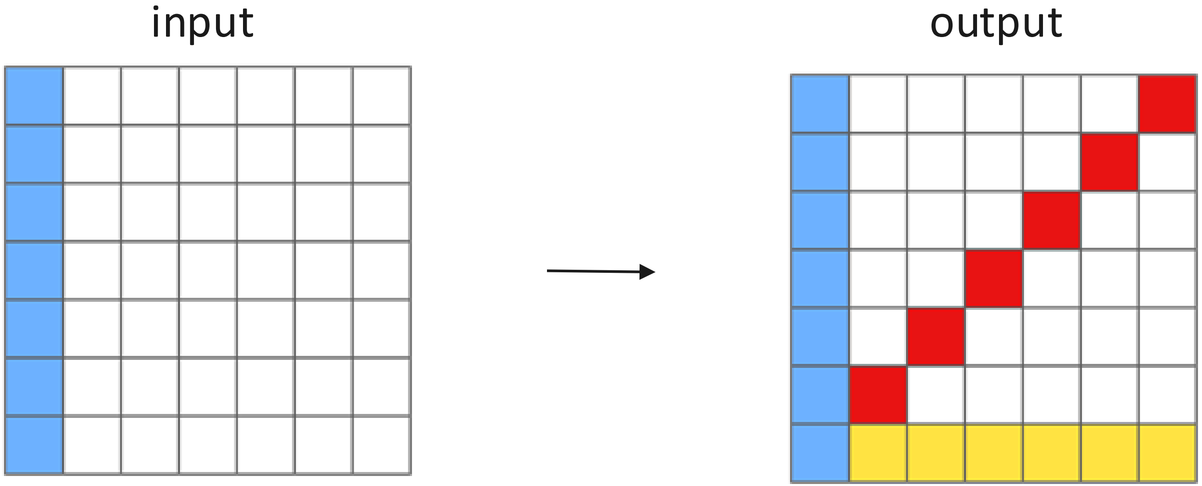
Find:

- a computer program that generalises them

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

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)))
```


input
output

Why is synthesis hard?

Infinite search space of undecidable programs

Why is synthesis hard?

Search space is bigger for harder problems

Standard approach

Find a sequence of functions to map an input to an output

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)))
```

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

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)))))
```

Our idea

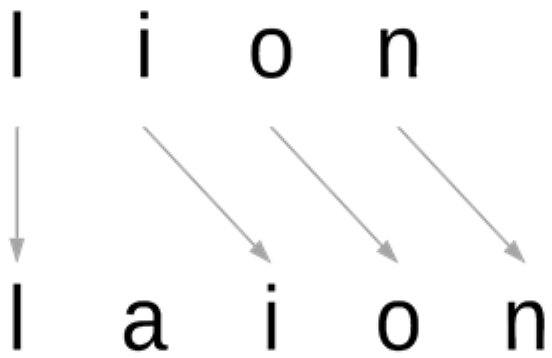
Look a problem differently

Our approach

Find relations between input and output elements

Input	Output
[l, i, o, n]	[l, a, i, o, n]

Input	Output
[l, i, o, n]	[l, a, i, o, n]



Input	Output
[l, i, o, n]	[l, a, i, o, n]

Input	Output
[l, i, o, n]	[l, a, i, o, n]

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

Input	Output
[l, i, o, n]	[l, a, i, o, n]

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

Input	Output
[l, i, o, n]	[l, a, i, o, n]

```
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).
```

Input	Output
[l, i, o, n]	[l, a, i, o, n]

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

Input	Output
[l, i, o, n]	[**l**, a, i, o, n]

the output at index 1 is the input at index 1

the output at index 2 is a

the output at index l > 2 is the input at index l-1

Input	Output
[l, i, o, n]	[l, **a**, i, o, n]

the output at index 1 is the input at index 1

the output at index 2 is a

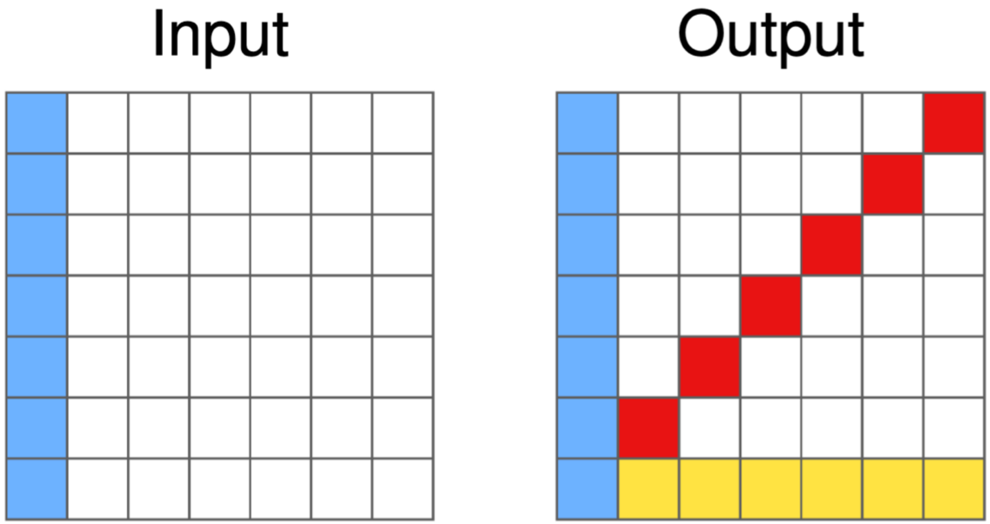
the output at index l > 2 is the input at index l-1

Input	Output
[l, i, o, n]	[l, a, **i, o, n**]

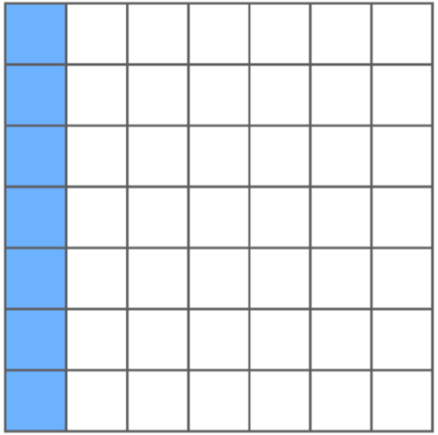
the output at index 1 is the input at index 1

the output at index 2 is a

the output at index I > 2 is the input at index I-1

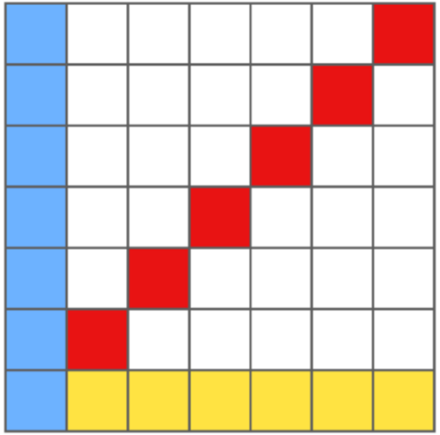

Input
Output

Input

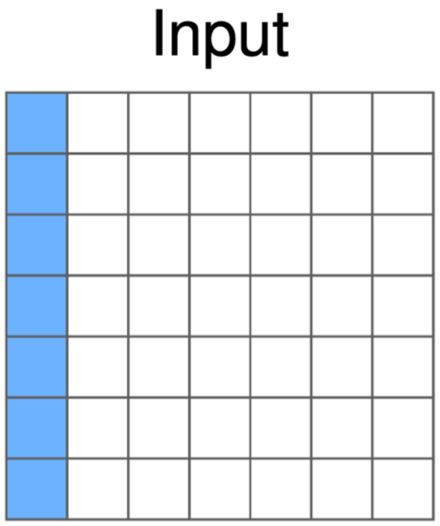


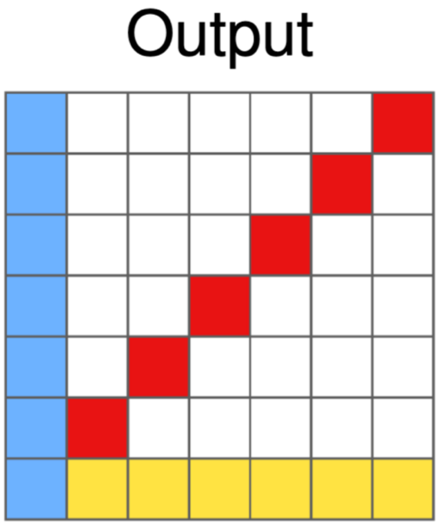
```
in(1,1,blue).      empty(2,1).
in(1,2,blue).      empty(2,1).
in(1,3,blue).      empty(2,1).
...                ... .
```

Output

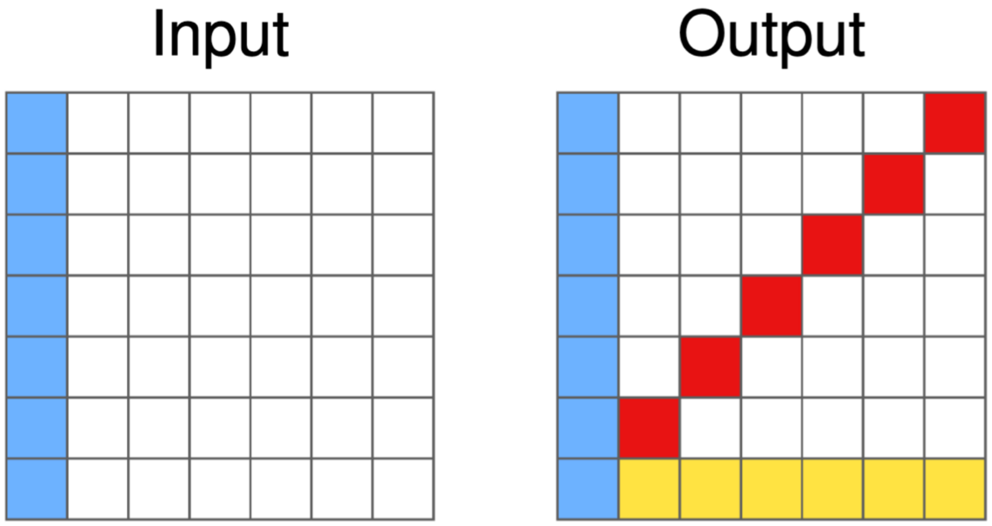


```
out(1,1,blue).  out(1,7,yellow).
out(1,2,blue).  out(2,7,yellow).
out(1,3,blue).  out(3,7,yellow).
...             ... .
```

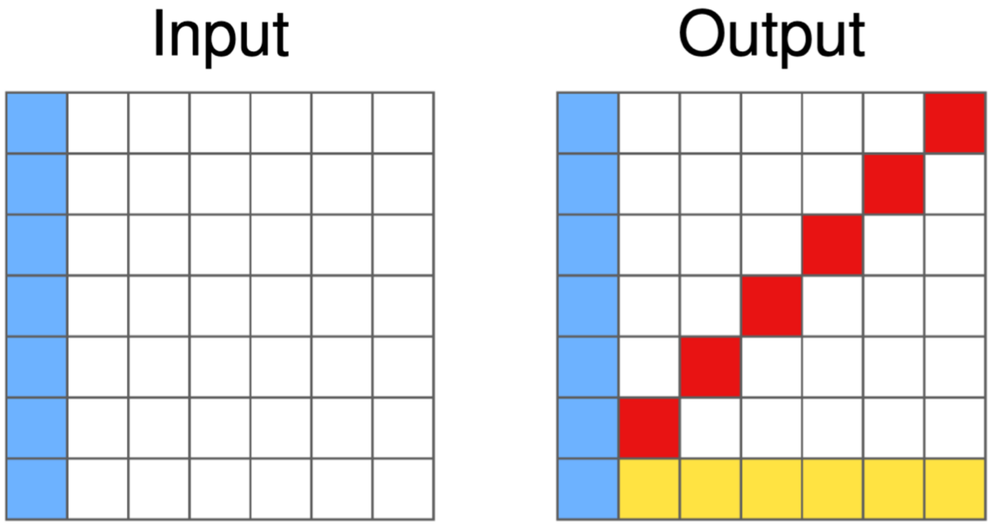




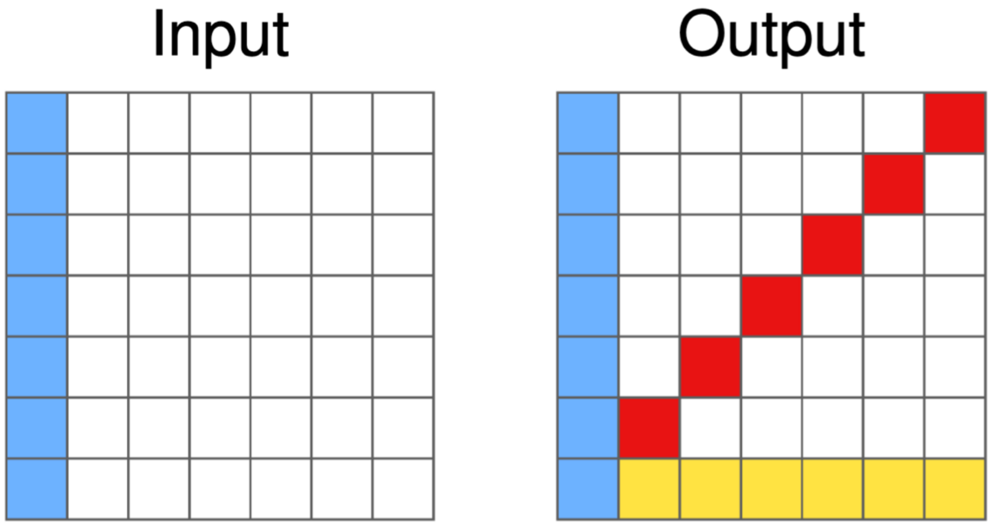

```
out(X,Y,C):- in(X,Y,C).
out(X,Y,yellow):- empty(X,Y), height(X).
out(X,Y,red):- empty(X,Y), height(X+Y-1).
```



1. An output pixel is colour C if it is colour C in the input



2. An output pixel is yellow if it is in the bottom row and empty in the input



3. An output pixel is red if it is empty and on the diagonal

How?

Inductive logic programming

How?

Logical machine learning

Inductive logic programming

Inductive logic programming

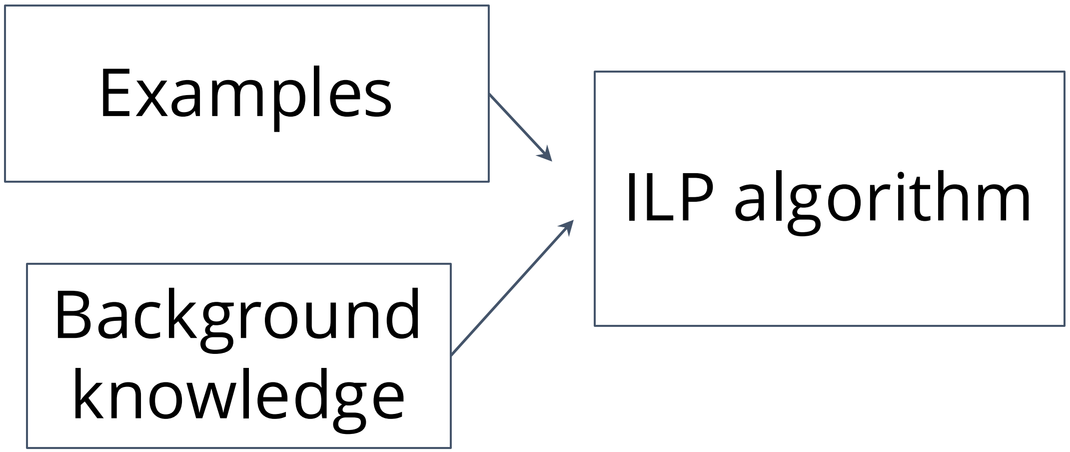
Examples

Inductive logic programming

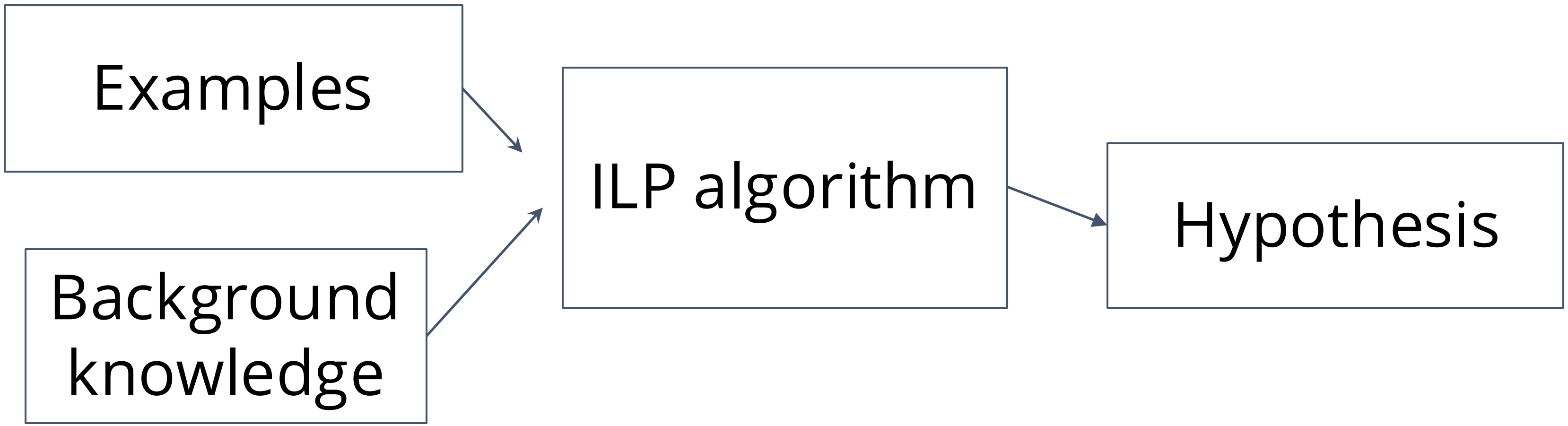
Examples

Background knowledge

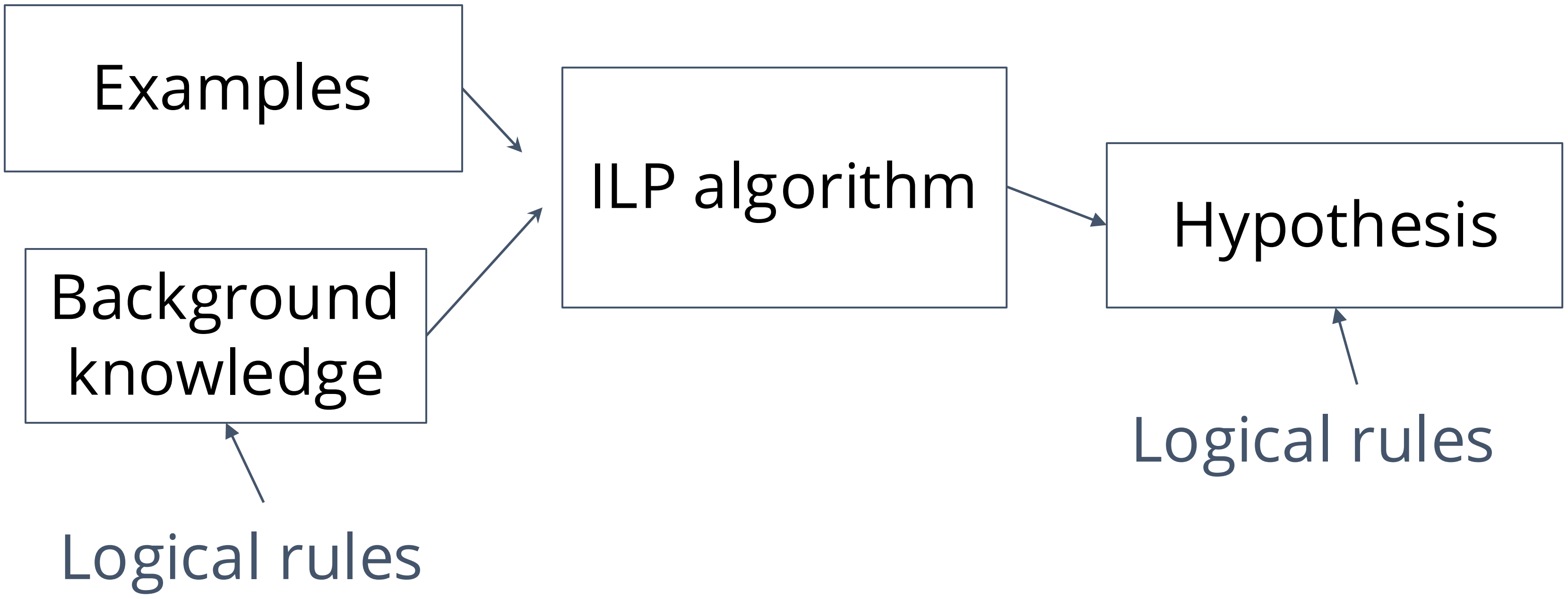
Inductive logic programming



Inductive logic programming



Inductive logic programming



Input	Output
[l, i, o, n]	[l, a, i, o, n]

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

Input	Output
[l, i, o, n]	[l, a, i, o, n]

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

```
% bk
in(1,l).
in(2,i).
in(3,o).
in(4,n).
<(2,3).
<(2,4).
>(5,2).
...
```

Input	Output
[l, i, o, n]	[l, a, i, o, n]

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

```
% bk
in(1,l).
in(2,i).
in(3,o).
in(4,n).
<(2,3).
<(2,4).
>(5,2).
...
```

ILP
algorithm

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

```
% bk
in(1,l).
in(2,i).
in(3,o).
in(4,n).
<(2,3).
<(2,4).
>(5,2).
...
```

ILP algorithm

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

Does it work?

Does it work?

Our approach:

- Off-the-shelf ILP system
- Learn from raw data + simple arithmetic

Does it work?

Others (non-relational):

- Domain-specific algorithms
- Domain-specific functions

1D-ARC

Figure 4: The *denoise* task from *1D-ARC*.

Task include *mirror*, *fill*, or *hollow*.

1D-ARC

Us

Time	ARGA	BEN	MB	HL	Decom
1	**93±6**	na	0±0	0±0	59±7
10	**94±6**	na	0±0	0±0	63±7
60	**94±6**	na	0±0	0±0	69±6

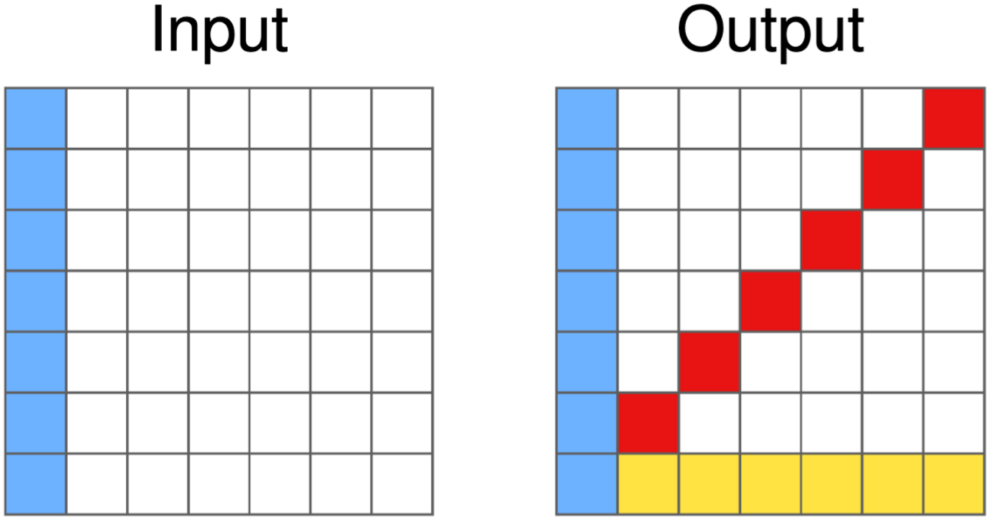
1D-ARC

Us

Time	ARGA	BEN	MB	HL	Decom
1	**93±6**	na	0±0	0±0	59±7
10	**94±6**	na	0±0	0±0	63±7
60	**94±6**	na	0±0	0±0	69±6

ARGA uses 15 domain-specific operators, including *mirror*, *fill*, and *hollow*.

ARC1



ARC1

Us

Time	ARGA	BEN	MB	HL	Decom
1	8±1	6±na	0±0	0±0	**15 ±1**
10	11±2	**25±na**	0±0	0±0	20±1
60	12±2	na	0±0	0±0	**22±1**

BEN uses domain-specific object detection features and domain-specific functions, such as *denoise*, *mirror, rotate*

List functions

the list [8, 2, 7, 0, 3] (0.78)

```
[1,4,9,7]           → [8,2,7,0,3]
[8,4,5,0,2,1,9,3,7] → [8,2,7,0,3]
[4,9,1,8,5,0]       → [8,2,7,0,3]
[1,5,4,7,0]         → [8,2,7,0,3]
[6]                 → [8,2,7,0,3]
```

add 2 to every item (0.68)

```
[7,3,4]         → [9,5,6]
[2,3,1,0,5]     → [4,5,3,2,7]
[3,6,5,7,1,2,4] → [5,8,7,9,3,4,6]
[5,6,1,0]       → [7,8,3,2]
[3,4,1,2,0,7]   → [5,6,3,4,2,9]
```

swap the first and last items (0.63)

```
[0,9,1,8]     → [8,9,1,0]
[8,1,4,7,5,2] → [2,1,4,7,5,8]
[6,2,8,7]     → [7,2,8,6]
[0,6]         → [6,0]
[6,9,5]       → [5,9,6]
```

remove all but item 3 (0.49)

```
[7,39,31,4]         → [31]
[23,57,65,52,51]    → [65]
[9,35]              → []
[79,97,98,4,6,89]   → [98]
[48]                → []
```

replace item 6 with a 3 (0.35)

```
[6,1,9,0,7,2]     → [6,1,9,0,7,3]
[7,2,1,6]         → [7,2,1,6]
[9,6,2,0,8,7,4,5] → [9,6,2,0,8,3,4,5]
[1,9,7,0,5,4,8]   → [1,9,7,0,5,3,8]
[4,0,7]           → [4,0,7]
```

first N items after item 1, N is the value of item 1 (0.26)

```
[3,1,9,0,7]           → [1,9,0]
[2,1,3,4,6,9]         → [1,3]
[4,1,2,3,5,0,7,6,9,8] → [1,2,3,5]
[1,5,4,2,8,3,0,6]     → [5]
[5,2,1,0,4,3,7,6]     → [2,1,0,4,3]
```

List functions

Us

Time	ARGA	BEN	MB	HL	Decom
1	0±0	na	0±0	**31±2**	27±2
10	0±0	na	7±1	33±2	**46±2**
60	0±0	na	8±1	35±3	**52±2**

HL is designed for the list function dataset

Why does it work?

- Decomposes a training example into multiple examples

What is missing?

- Better search

What is missing?

- Go beyond raw data (add counting)

Conclusion

- Looking at a problem differently can improve learning performance

Interested?

ILP system Popper

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

(Just Google Popper + ILP)