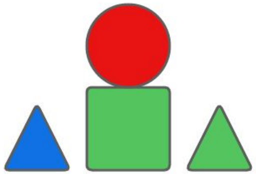
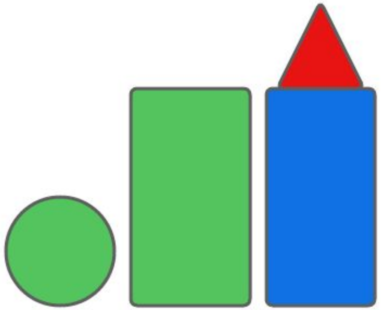
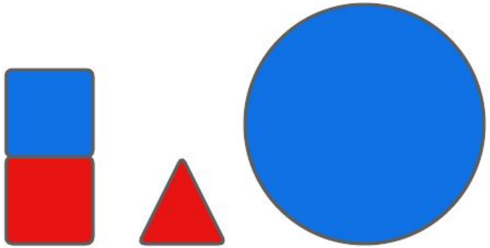
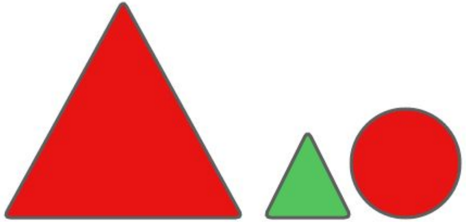
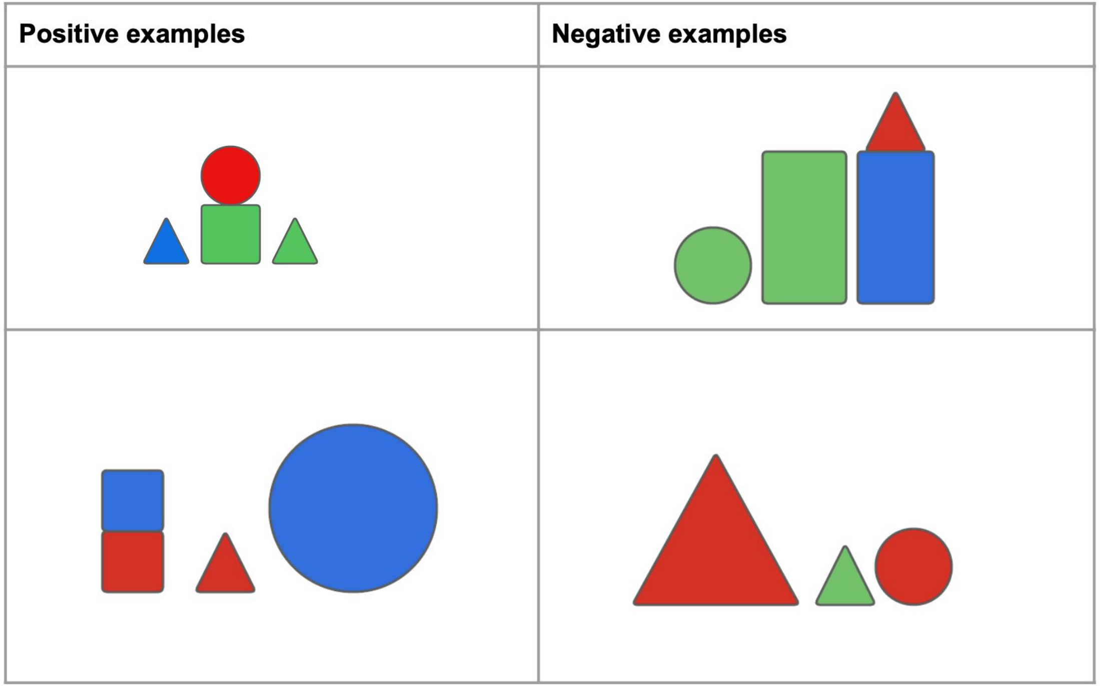
Learning logic programs with Popper

Céline Hocquette
University of Oxford

UK Research and Innovation

UNIVERSITY OF OXFORD

Positive examples	Negative examples
	
	



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

Inductive Logic Programming (ILP)

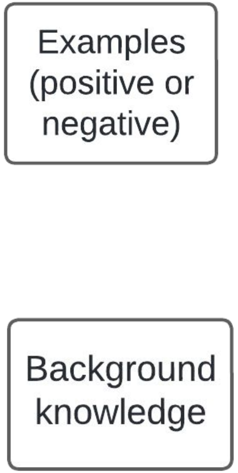
Inductive Logic Programming (ILP)

a form of program synthesis

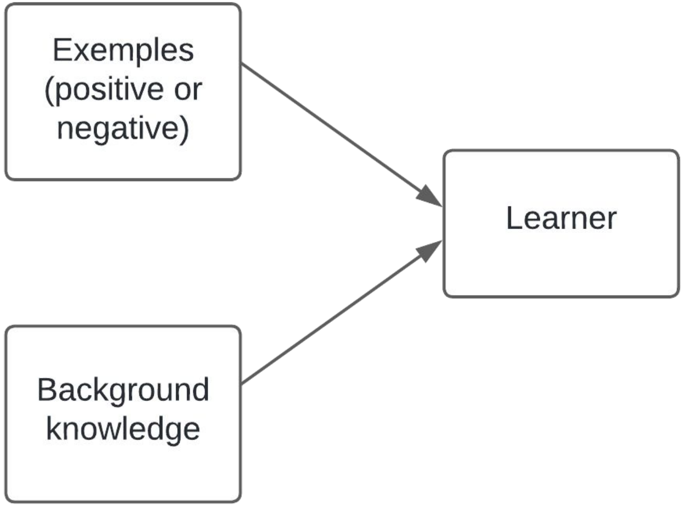
Inductive Logic Programming (ILP)

Examples (positive or negative)

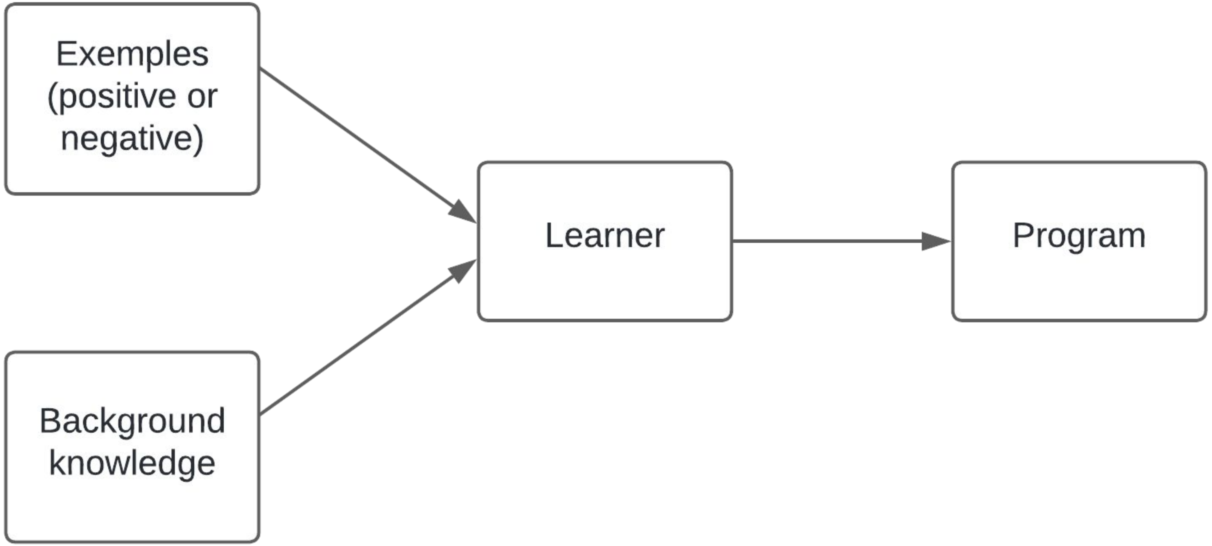
Inductive Logic Programming (ILP)



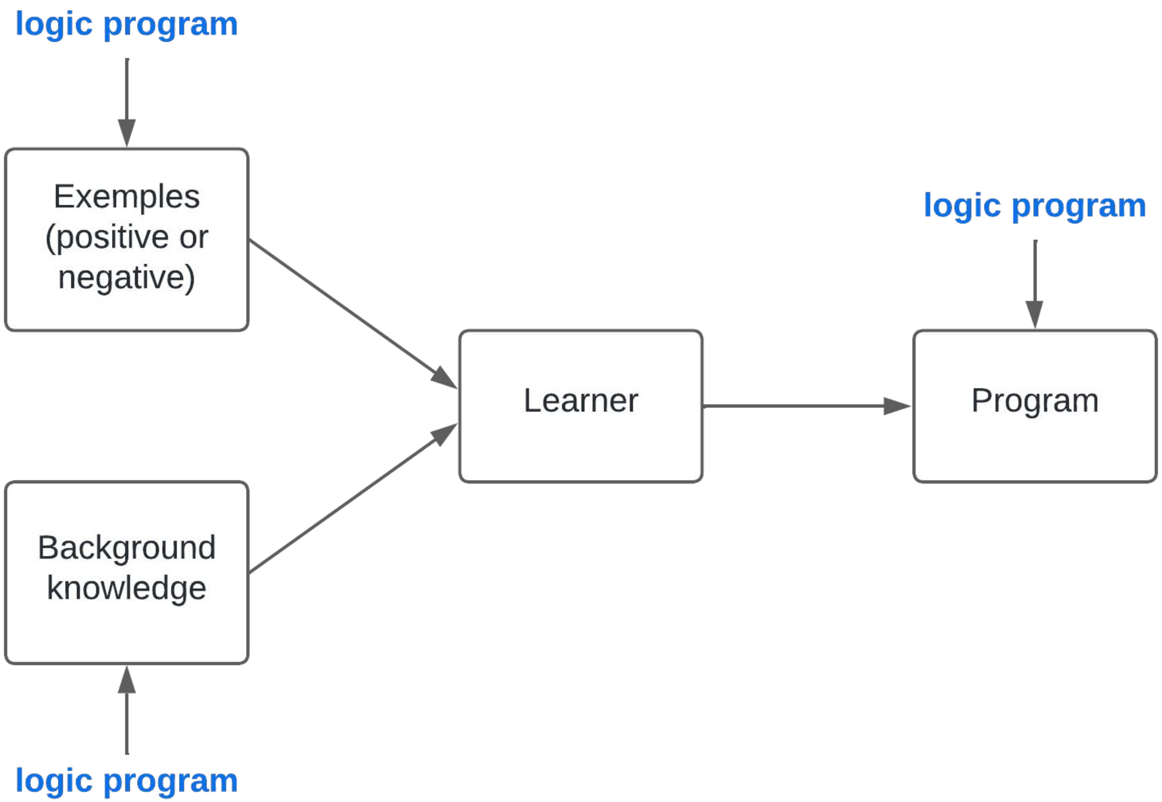
Inductive Logic Programming (ILP)



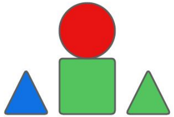
Inductive Logic Programming (ILP)



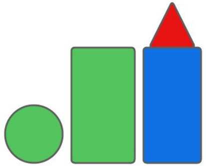
Inductive Logic Programming (ILP)



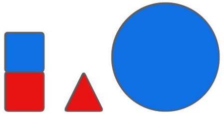
Positive examples	Negative examples
zendo(e_1). zendo(e_2).	zendo(e_{-1}). zendo(e_{-2}).



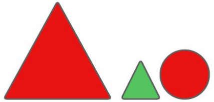
e_1



e_{-1}

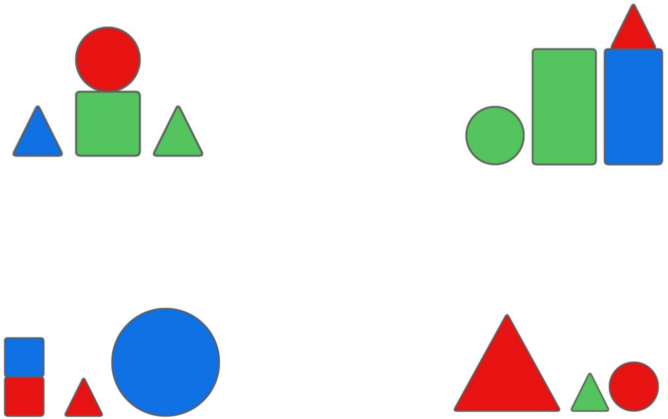


e_2



e_{-2}

Background Knowledge
piece(e_1, p1). piece(e_1, p2). piece(e_1, p3). piece(e_1, 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). …



Program

```
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 and large rules

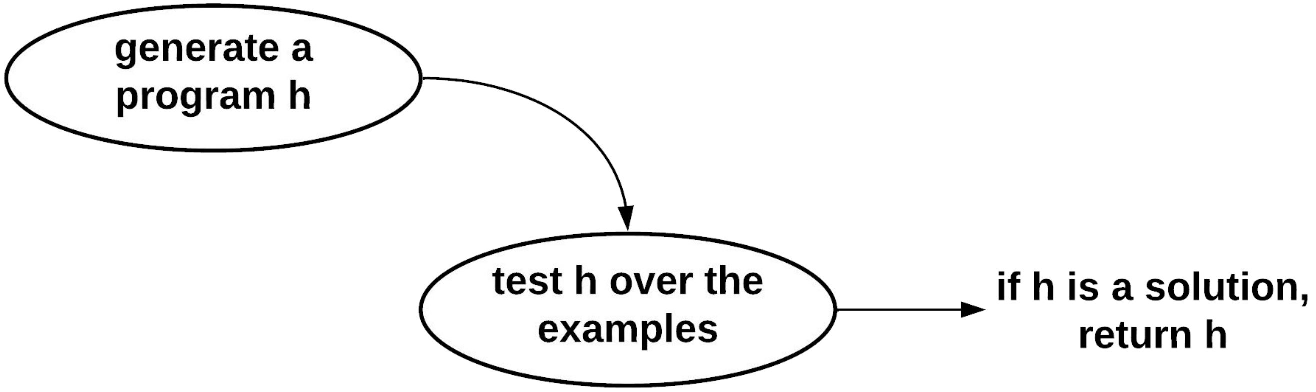
Why care?

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

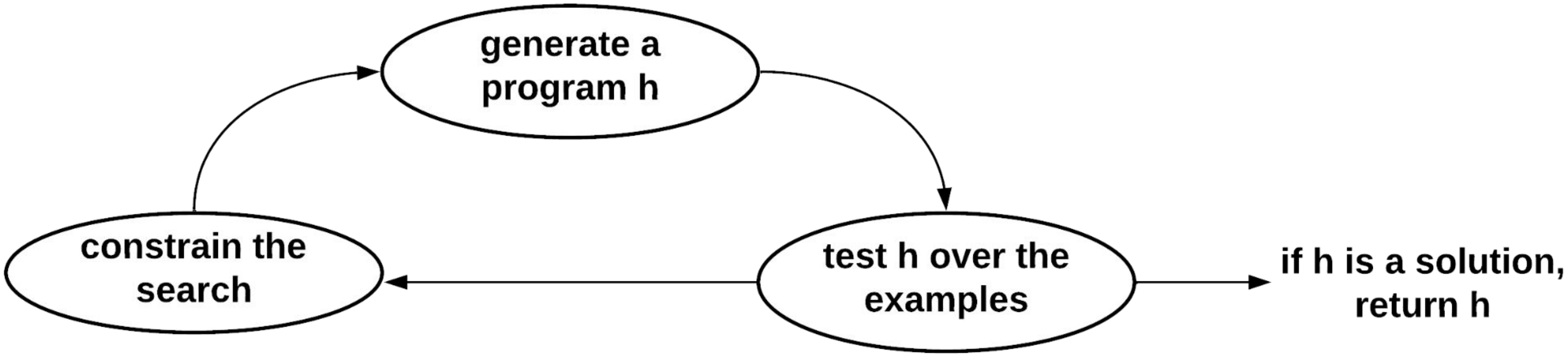
How does it work?

How does it work?

How does it work?

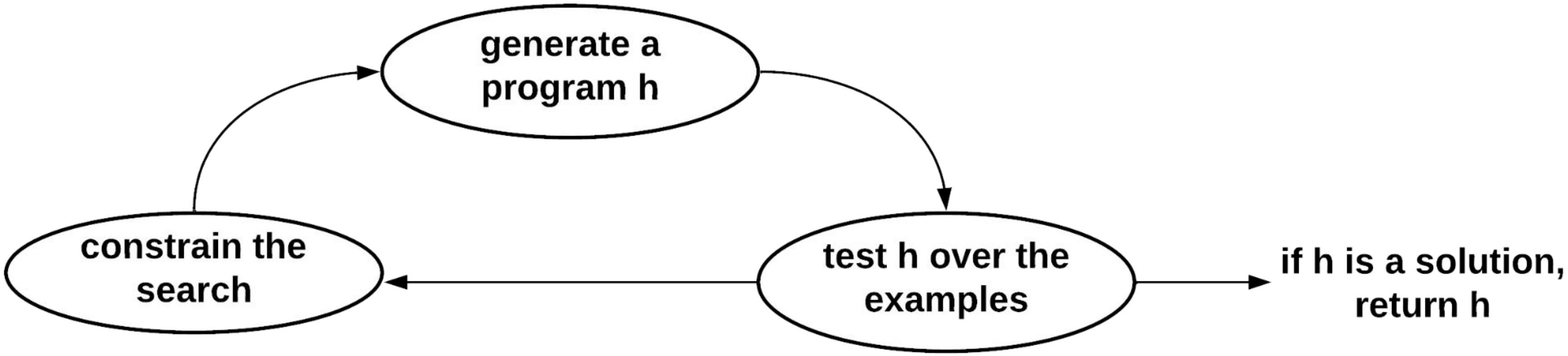


How does it work?



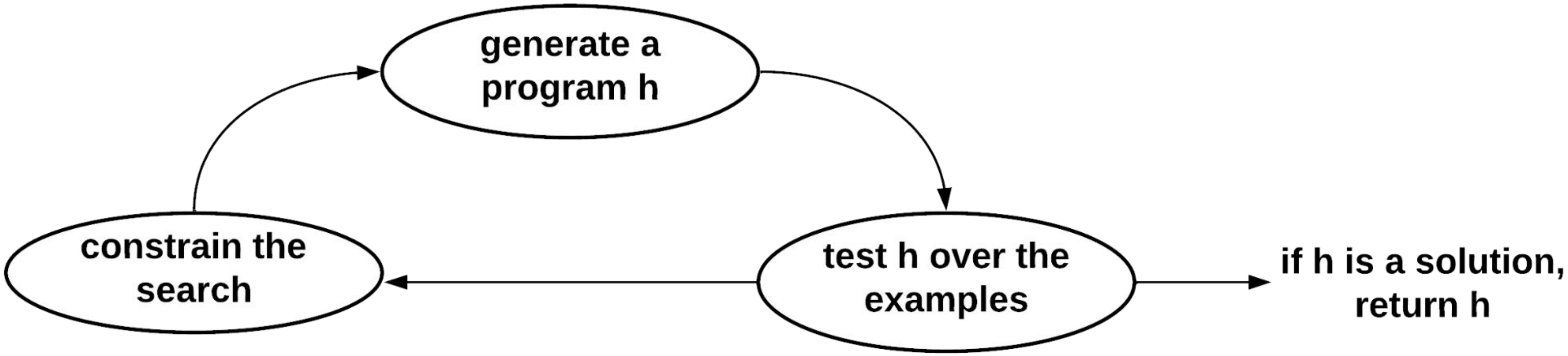
How does it work?

```
zendo(Structure):- piece(Structure,Piece),black(Piece).
```



How does it work?

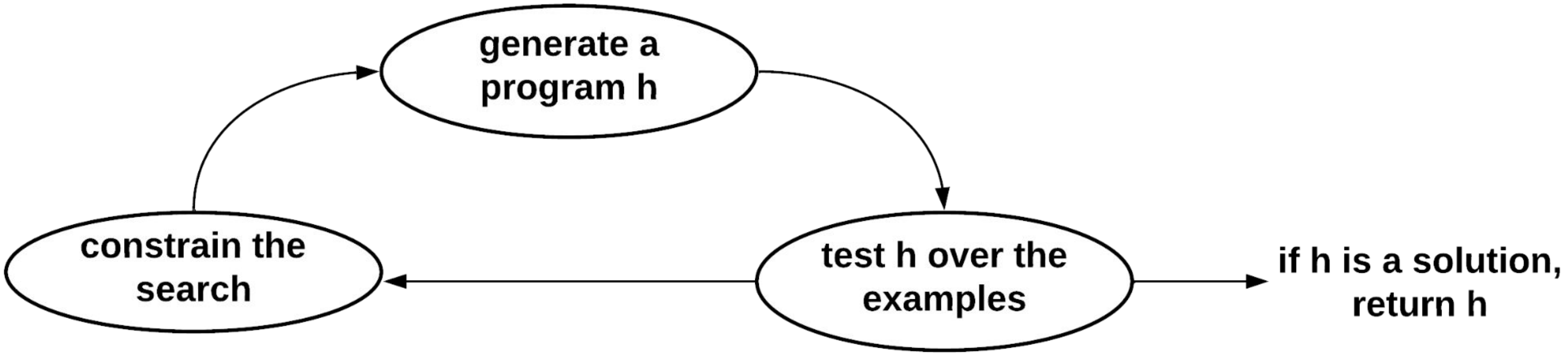
```
zendo(Structure):- piece(Structure,Piece),black(Piece).
```



This program does not entail any positive example

How does it work?

```
zendo(Structure):- piece(Structure,Piece),black(Piece).
```



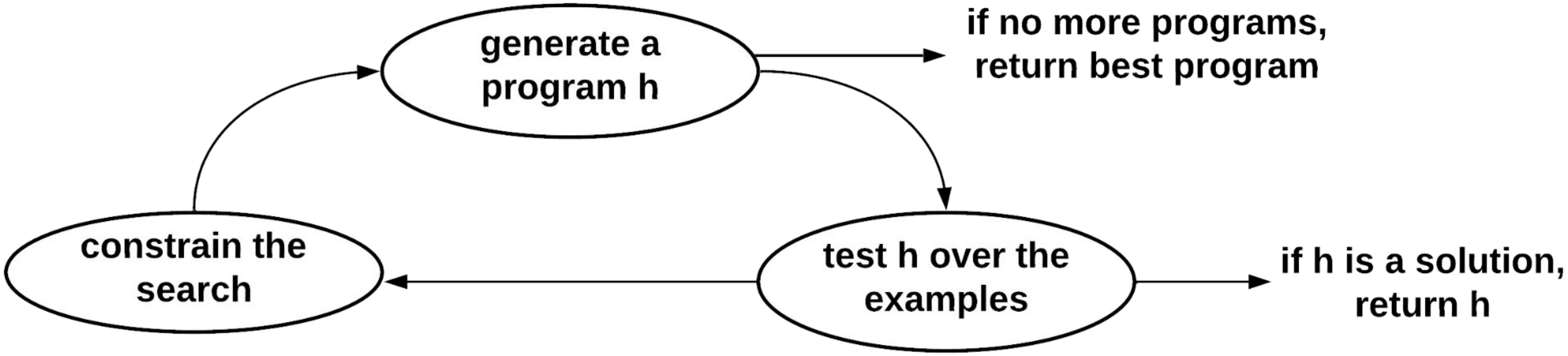
This program does not entail any positive example

We can prune its specialisations, such as:

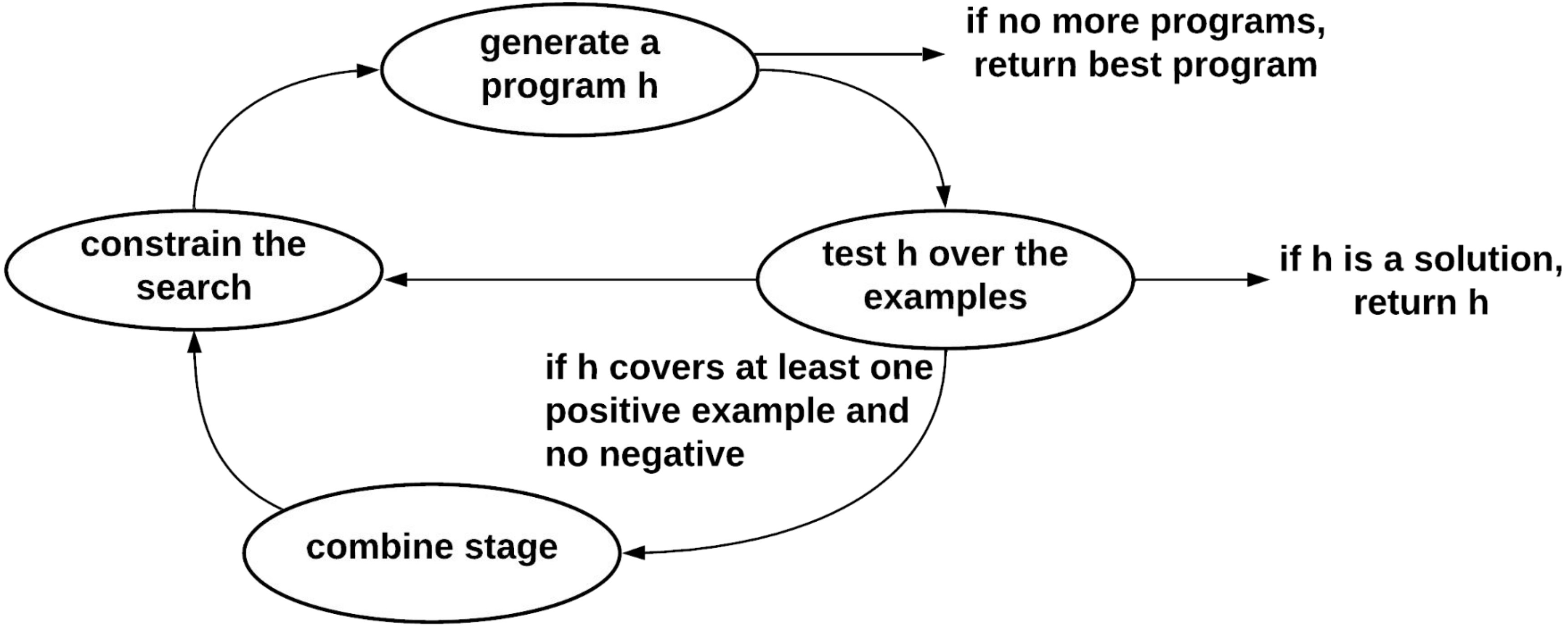
```
zendo(Structure):- piece(Structure,Piece),black(Piece),contact(Piece,Piece1),blue(Piece1).
zendo(Structure):- piece(Structure,Piece),black(Piece),round(Piece).
zendo(Structure):- piece(Structure,Piece),black(Piece),size(Piece,Size),small(Size).
```

…

How does it work?



How does it work?



We use a MaxSAT solver to search for an optimal combination (a union) of programs.

Learning programs by combining programs, A. Cropper and C. Hocquette, *ECAI, 2023*.

Combine stage

```
p1: zendo(Structure):- piece(Structure,Piece),blue(Piece).                               covers {e1,e3,e7,e9} size 3
p2: zendo(Structure):- piece(Structure,Piece),yellow(Piece).                             covers {e2,e3} size 3
p3: zendo(Structure):- piece(Structure,Piece),red(Piece),square(Piece).               covers {e2,e4,e6} size 4
p4: zendo(Structure):- piece(Structure,Piece1),contact(Piece1,Piece2),yellow(Piece2).   covers {e5,e8,e9} size 4
p5: zendo(Structure):- piece(Structure,Piece),size(Piece,Size),small(Size).             covers {e7,e8,e9} size 4
p6: zendo(Structure):- piece(Structure,Piece1),blue(Piece1),piece(Structure,Piece2),red(Piece2).  covers {e5} size 5
p7: zendo(Structure):- piece(Structure,Piece),green(Piece),size(Piece,Size),large(Size).          covers {e4,e5} size 5
p8: zendo(Structure):- piece(Structure,Piece),contact(Piece1,Piece2),red(Piece2),square(Piece2).     covers {e6,e7} size 5
p9: zendo(Structure):- piece(Structure,Piece),red(Piece1),contact(Piece1,Piece2),blue(Piece2),round(Piece2). covers {e8} size 6
…
```

Combine stage

p1: zendo(Structure):- piece(Structure,Piece),blue(Piece). covers {e_1,e_3,e_7,e_9} size 3

~~p2: zendo(Structure):- piece(Structure,Piece),yellow(Piece). covers {e_2,e_3} size 3~~

p3: zendo(Structure):- piece(Structure,Piece),red(Piece),square(Piece). covers {e_2,e_4,e_6} size 4

p4: zendo(Structure):- piece(Structure,Piece1),contact(Piece1,Piece2),yellow(Piece2). covers {e_5,e_8,e_9} size 4

~~p5: zendo(Structure):- piece(Structure,Piece),size(Piece,Size),small(Size). covers {e_7,e_8,e_9} size 4~~

~~p6: zendo(Structure):- piece(Structure,Piece1),blue(Piece1),piece(Structure,Piece2),red(Piece2). covers {e_5} size 5~~

~~p7: zendo(Structure):- piece(Structure,Piece),green(Piece),size(Piece,Size),large(Size). covers {e_4,e_5} size 5~~

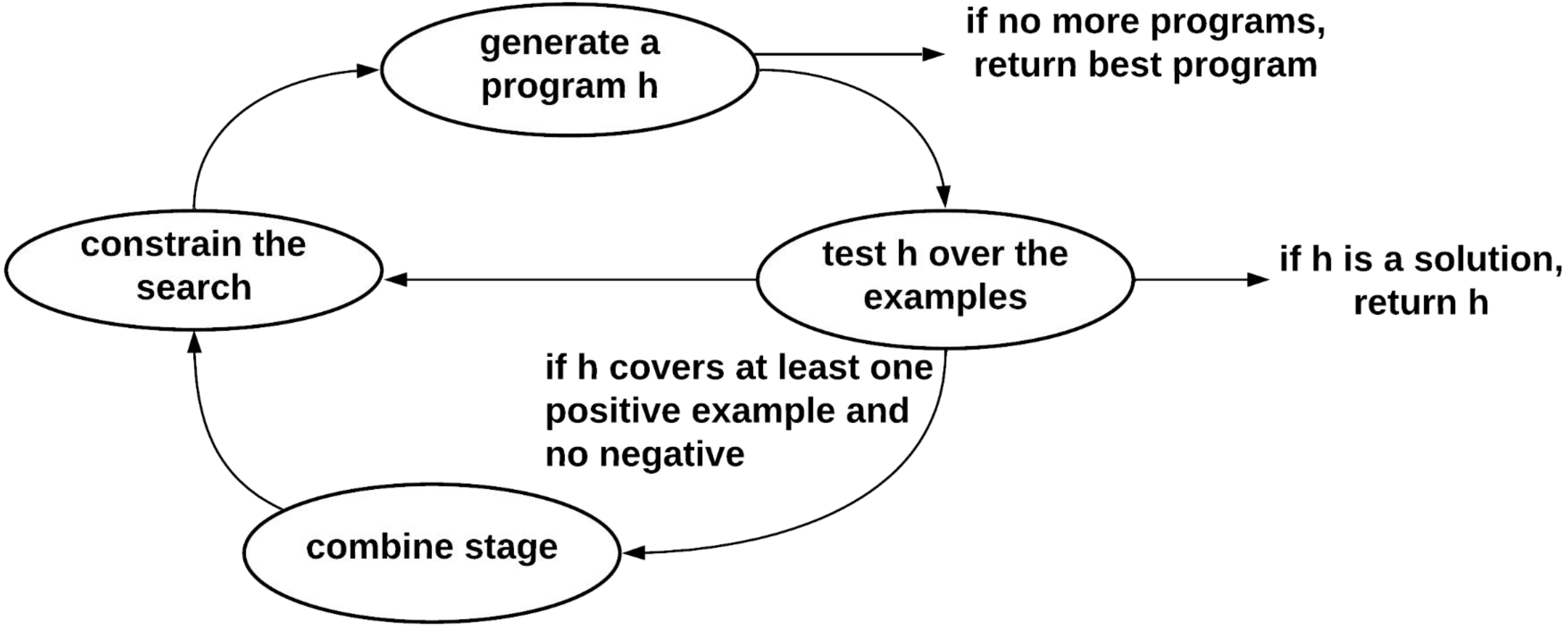
~~p8: zendo(Structure):- piece(Structure,Piece),contact(Piece1,Piece2),red(Piece2),square(Piece2). covers {e_6,e_7} size 5~~

~~p9: zendo(Structure):- piece(Structure,Piece),red(Piece1),contact(Piece1,Piece2),blue(Piece2),round(Piece2). covers {e_8} size 6~~

…

{p1, p3, p4} entails all the positive examples and has minimal size

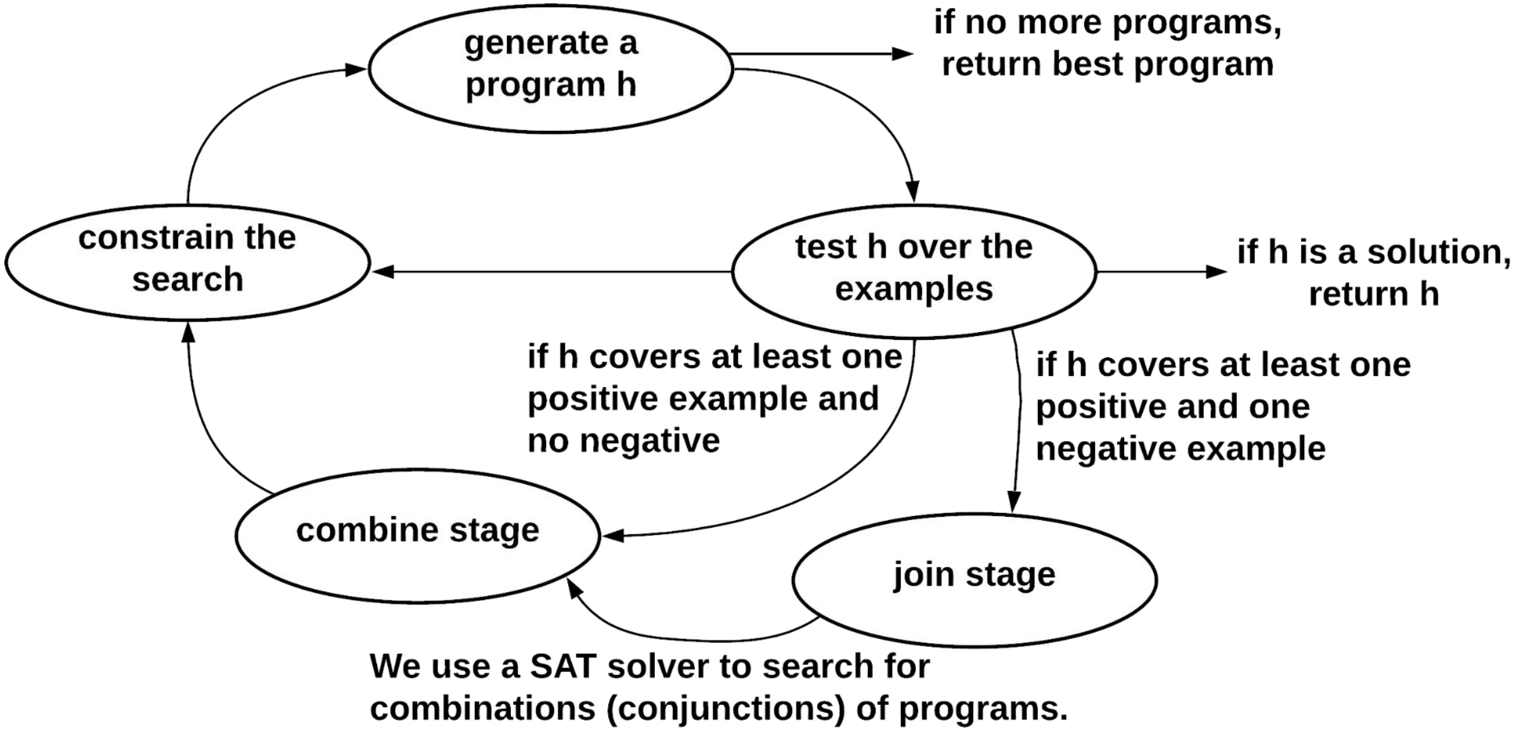
How does it work?



We use a MaxSAT solver to search for an optimal combination (a union) of programs.

Learning programs by combining programs, A. Cropper and C. Hocquette, *ECAI, 2023*.

How does it work?



Learning big logical rules by joining small rules, C. Hocquette, A. Niskanen, R. Morel, M. Järvisalo, and A. Cropper, *AAAI, 2023*.

Join stage

```
p1: zendo(Structure):- piece(Structure,Piece),blue(Piece).
p2: zendo(Structure):- piece(Structure,Piece),yellow(Piece).
p3: zendo(Structure):- piece(Structure,Piece),red(Piece),square(Piece).
p4: zendo(Structure):- piece(Structure,Piece),contact(Piece,Piece1),blue(Piece1).
…
```

covers $\{e_1,e_3,e_7,e_9\}$ $\{e_{-1},e_{-2}\}$ size 3

covers $\{e_1,e_3,e_7,e_9\}$ $\{e_{-4},e_{-5}\}$ size 3

covers $\{e_1,e_2,e_3,e_4,e_6\}$ $\{e_{-6}\}$ size 4

covers $\{e_7,e_8\}$ $\{e_{-6}\}$ size 4

Join stage

```
p1: zendo(Structure):- piece(Structure,Piece),blue(Piece).
p2: zendo(Structure):- piece(Structure,Piece),yellow(Piece).
p3: zendo(Structure):- piece(Structure,Piece),red(Piece),square(Piece).
p4: zendo(Structure):- piece(Structure,Piece),contact(Piece,Piece1),blue(Piece1).
…
```

p1 covers $\{\mathbf{e_1},\mathbf{e_3},e_7,e_9\}$ $\{e_{-1},e_{-2}\}$ size 3

p2 covers $\{\mathbf{e_1},\mathbf{e_3},e_7,e_9\}$ $\{e_{-4},e_{-5}\}$ size 3

p3 covers $\{\mathbf{e_1},e_2,\mathbf{e_3},e_4,e_6\}$ $\{e_{-6}\}$ size 4

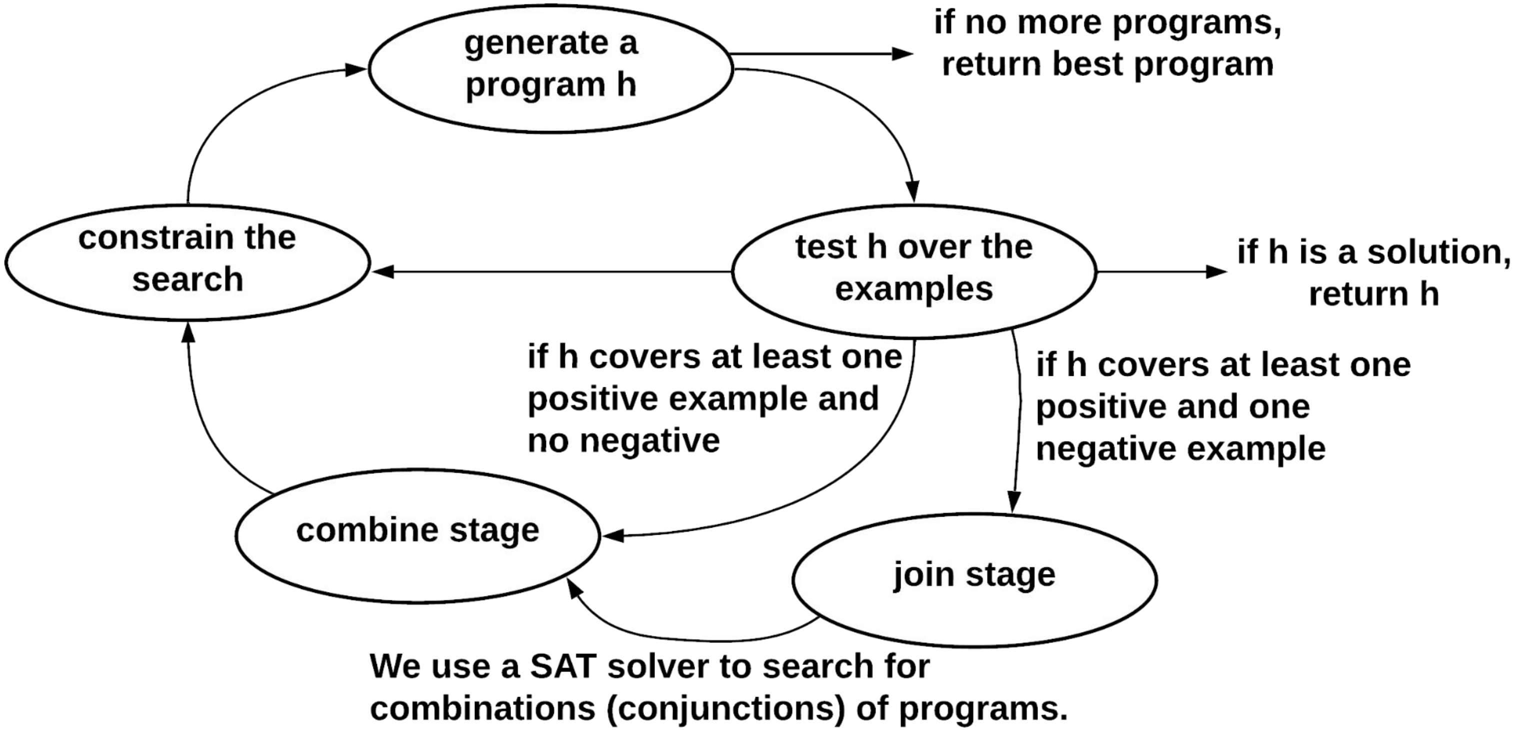
p4 covers $\{e_7,e_8\}$ $\{e_{-6}\}$ size 4

$p = p_1 \cap p_2 \cap p_3$

```
p: zendo(Structure):- piece(Structure,Piece),blue(Piece),
                      piece(Structure,Piece),yellow(Piece),
                      piece(Structure,Piece),red(Piece),square(Piece).
```

p covers $\{e_1,e_3\}$ and no negative example

How does it work?



Learning big logical rules by joining small rules, C. Hocquette, A. Niskanen, R. Morel, M. Järvisalo, and A. Cropper, *AAAI, 2023*.

Correctness

Popper learns an optimal solution (a textually minimal program).

Learning from noisy data

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, C. Hocquette, A. Niskanen, M. Järvisalo, A. Cropper, AAAI24.

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

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:
        - recursive
        - predicate invention
        - optimal programs (mdl or textually minimal)
        - noisy data
        - anytime
        - infinite domains and numerical reasoning
    - 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

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
- avoid recursion if possible
- avoid predicate invention if possible
- use a sat solver for the combine stage

Thank you!