Learning logic programs by discovering where not to search

Andrew Cropper and Céline Hocquette

Engineering and Physical Sciences Research Council

UNIVERSITY OF OXFORD

What is this talk about?

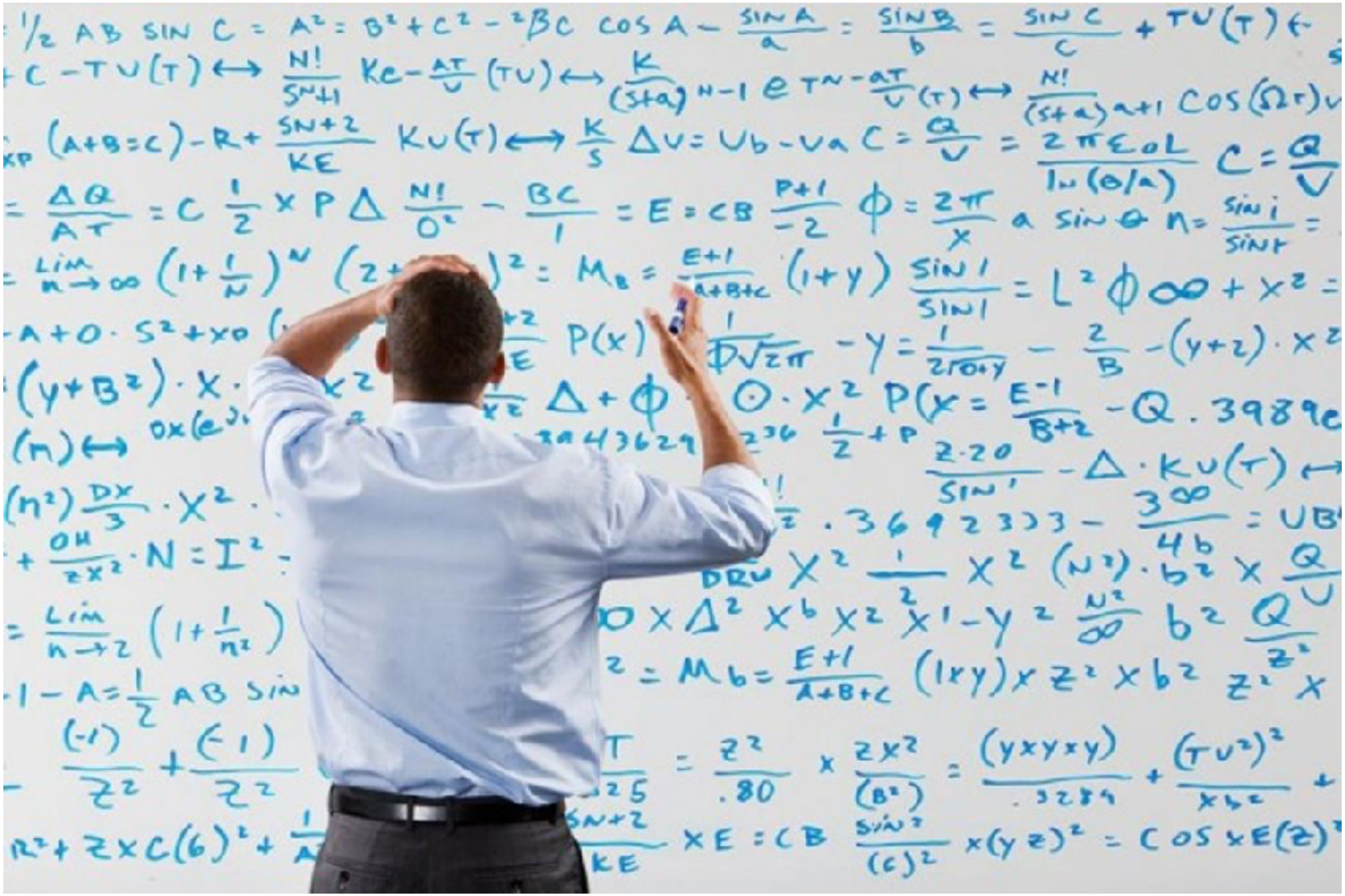
- A simple approach to improve the learning performance of an ILP system

What is this talk about?

- A simple approach to improve the learning performance of an ILP system

- The idea is to discover where not to search before searching for a solution

No technical details



Inductive logic programming

Inductive logic programming

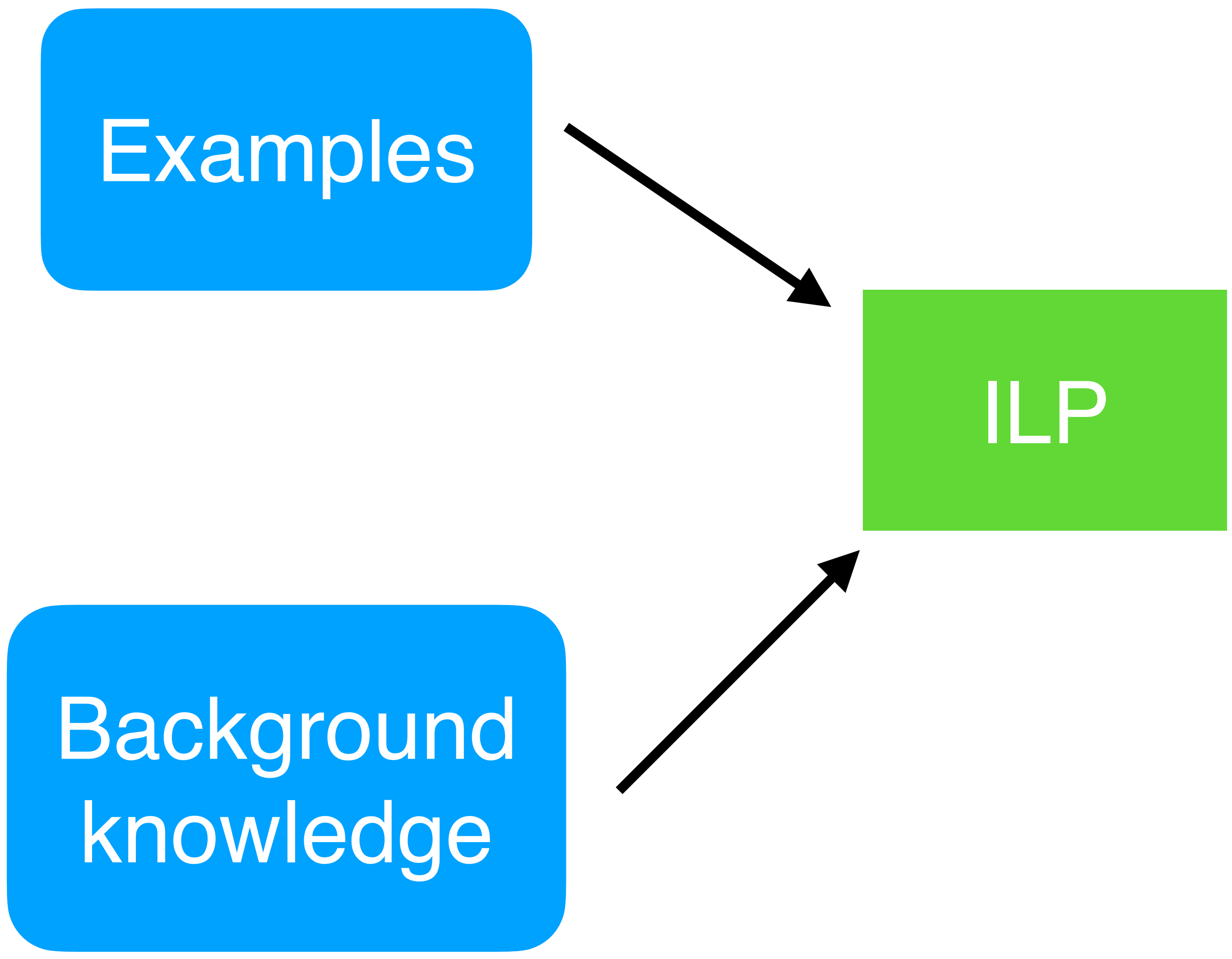
Examples

Inductive logic programming

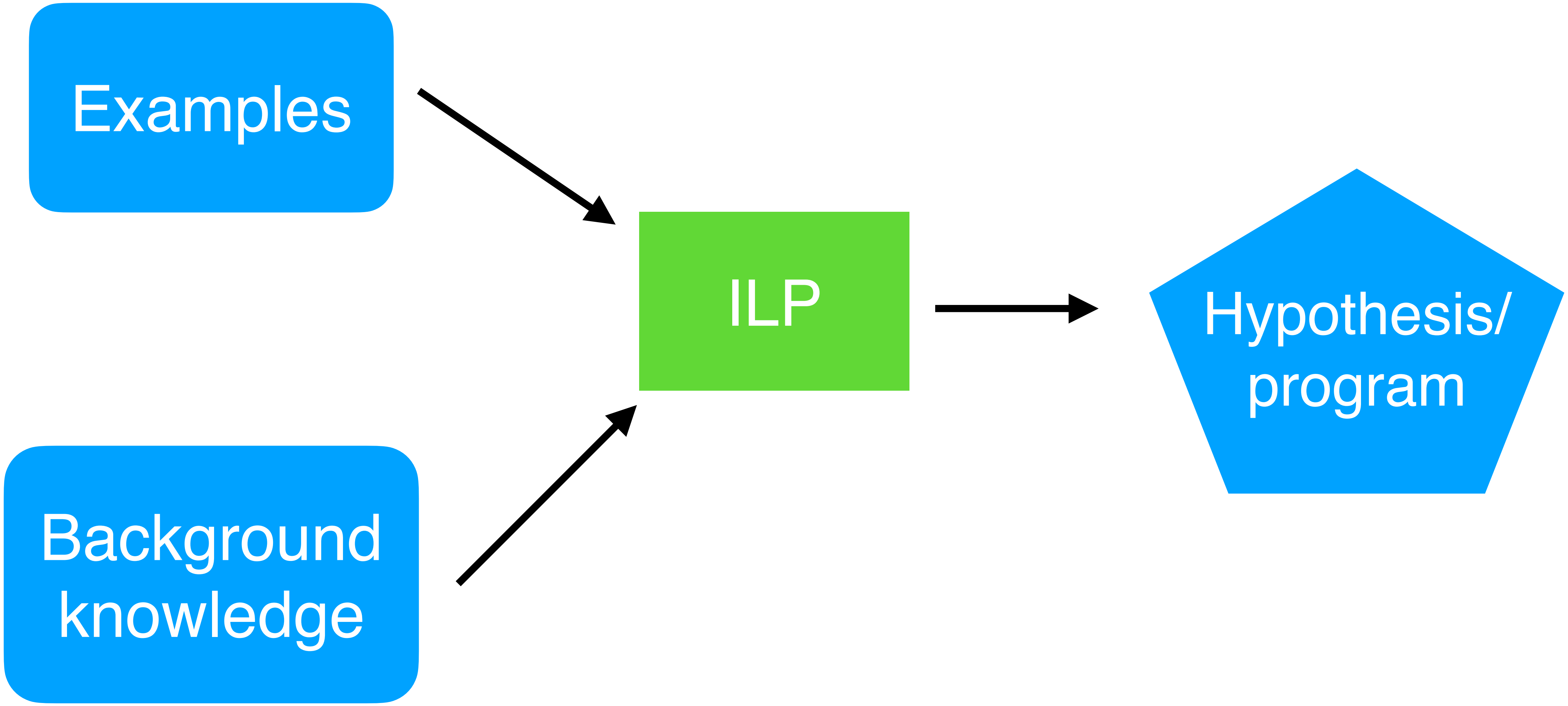
Examples

Background knowledge

Inductive logic programming



Inductive logic programming



Examples

input	outpu
dog	g
sheep	p
chicken	n

Examples

input	outpu
dog	g
sheep	p
chicken	n

representation
last(dog,g)
last(sheep,p)
last(chicken,n)

BK

```
head([H|_],H).
tail([_|T],T).
empty(A).
```

Hypothesis

```
last(A,B):-tail(A,C),empty(C),head(A,B).
last(A,B):-tail(A,C),f(C,B).
```

Motivation

Very large hypothesis spaces

Idea

Can we discover where not to search before searching for a solution?

BK

head(ijcai,i)	*tail(ijcai,jcai)*	*even(2)*
head(ecai,e)	*tail(ecai,cai)*	*even(4)*
head(cai,c)	*tail(jcai,cai)*	*odd(1)*
tail(ai,i)	*tail(cai,ai)*	*odd(3)*

$$r_1 = h \leftarrow tail(A,A)$$
$$r_2 = h \leftarrow tail(A,B),\ tail(B,A)$$
$$r_3 = h \leftarrow tail(A,B),\ tail(B,C),\ tail(A,C)$$
$$r_4 = h \leftarrow tail(A,A),\ head(A,B),\ odd(B)$$
$$r_5 = h \leftarrow head(A,B),\ odd(B),\ even(B)$$

head(ijcai,i)	*tail(ijcai,jcai)*	*even(2)*
head(ecai,e)	*tail(ecai,cai)*	*even(4)*
head(cai,c)	*tail(jcai,cai)*	*odd(1)*
tail(ai,i)	*tail(cai,ai)*	*odd(3)*

tail is not reflexive

$$\cancel{r_1 = h \leftarrow tail(A,A)}$$
$$r_2 = h \leftarrow tail(A,B),\ tail(B,A)$$
$$r_3 = h \leftarrow tail(A,B),\ tail(B,C),\ tail(A,C)$$
$$\cancel{r_4 = h \leftarrow tail(A,A),\ head(A,B),\ odd(B)}$$
$$r_5 = h \leftarrow head(A,B),\ odd(B),\ even(B)$$

head(ijcai,i)	*tail(ijcai,jcai)*	*even(2)*
head(ecai,e)	*tail(ecai,cai)*	*even(4)*
head(cai,c)	*tail(jcai,cai)*	*odd(1)*
tail(ai,i)	*tail(cai,ai)*	*odd(3)*

tail is not symmetric

$$\cancel{r_1 = h \leftarrow tail(A,A)}$$
$$\cancel{r_2 = h \leftarrow tail(A,B),\ tail(B,A)}$$
$$r_3 = h \leftarrow tail(A,B),\ tail(B,C),\ tail(A,C)$$
$$\cancel{r_4 = h \leftarrow tail(A,A),\ head(A,B),\ odd(B)}$$
$$r_5 = h \leftarrow head(A,B),\ odd(B),\ even(B)$$

head(ijcai,i)	*tail(ijcai,jcai)*	*even(2)*
head(ecai,e)	*tail(ecai,cai)*	*even(4)*
head(cai,c)	*tail(jcai,cai)*	*odd(1)*
tail(ai,i)	*tail(cai,ai)*	*odd(3)*

tail is not transitive

$$\cancel{r_1 = h \leftarrow tail(A,A)}$$

$$\cancel{r_2 = h \leftarrow tail(A,B),\ tail(B,A)}$$

$$\cancel{r_3 = h \leftarrow tail(A,B),\ tail(B,C),\ tail(A,C)}$$

$$\cancel{r_4 = h \leftarrow tail(A,A),\ head(A,B),\ odd(B)}$$

$$r_5 = h \leftarrow head(A,B),\ odd(B),\ even(B)$$

head(ijcai,i)	*tail(ijcai,jcai)*	*even(2)*
head(ecai,e)	*tail(ecai,cai)*	*even(4)*
head(cai,c)	*tail(jcai,cai)*	*odd(1)*
tail(ai,i)	*tail(cai,ai)*	*odd(3)*

odd(B) and even(B) are unsatisfiable

~~$r_1 = h \leftarrow tail(A,A)$~~

~~$r_2 = h \leftarrow tail(A,B), tail(B,A)$~~

~~$r_3 = h \leftarrow tail(A,B), tail(B,C), tail(A,C)$~~

~~$r_4 = h \leftarrow tail(A,A), head(A,B), odd(B)$~~

~~$r_5 = h \leftarrow head(A,B), odd(B), even(B)$~~

head(ijcai,i)	*tail(ijcai,jcai)*	*even(2)*
head(ecai,e)	*tail(ecai,cai)*	*even(4)*
head(cai,c)	*tail(jcai,cai)*	*odd(1)*
tail(ai,i)	*tail(cai,ai)*	*odd(3)*

We have eliminated hypotheses before even considering the examples

How?

How?

1. Preprocess the BK to discover constraints

How?

1. Preprocess the BK to discover constraints

2. Use the constraints to bootstrap a constraint-driven ILP system

Constraint discovery

Name	Property	Constraint	Example
Irreflexive	$\neg p(A,A)$	$\leftarrow p(A,A)$	$\leftarrow$ *brother(A,A)*
Antitransitive	$p(A,B),\ p(B,C) \rightarrow \neg p(A,C)$	$\leftarrow p(A,B),\ p(B,C),\ p(A,C)$	$\leftarrow$ *succ(A,B), succ(B,C), succ(A,C)*
Antitriangular	$p(A,B),\ p(B,C) \rightarrow \neg p(C,A)$	$\leftarrow p(A,B),\ p(B,C),\ p(C,A)$	$\leftarrow$ *tail(A,B), tail(B,C), tail(C,A)*
Injective	$p(A,B),\ p(C,B) \rightarrow A{=}C$	$\leftarrow p(A,B),\ p(C,B),\ A{\neq}C$	$\leftarrow$ *succ(A,B), succ(C,B)*, $A{\neq}C$
Functional	$p(A,B),\ p(A,C) \rightarrow B{=}C$	$\leftarrow p(A,B),\ p(A,C),\ B{\neq}C$	$\leftarrow$ *length(A,B), length(A,C)*, $B{\neq}C$
Asymmetric	$p(A,B) \rightarrow \neg p(B,A)$	$\leftarrow p(A,B),\ p(B,A)$	$\leftarrow$ *mother(A,B), mother(B,A)*
Exclusive	$p(A) \rightarrow \neg q(A)$	$\leftarrow p(A),\ q(A)$	$\leftarrow$ *odd(A), even(A)*

Constraint discovery

Name	Property	Constraint	Example
Irreflexive	$\neg p(A,A)$	$\leftarrow p(A,A)$	$\leftarrow brother(A,A)$
Antitransitive	$p(A,B), p(B,C) \rightarrow \neg p(A,C)$	$\leftarrow p(A,B), p(B,C), p(A,C)$	$\leftarrow succ(A,B), succ(B,C), succ(A,C)$
Antitriangular	$p(A,B), p(B,C) \rightarrow \neg p(C,A)$	$\leftarrow p(A,B), p(B,C), p(C,A)$	$\leftarrow tail(A,B), tail(B,C), tail(C,A)$
Injective	$p(A,B), p(C,B) \rightarrow A{=}C$	$\leftarrow p(A,B), p(C,B), A{\neq}C$	$\leftarrow succ(A,B), succ(C,B), A{\neq}C$
Functional	$p(A,B), p(A,C) \rightarrow B{=}C$	$\leftarrow p(A,B), p(A,C), B{\neq}C$	$\leftarrow length(A,B), length(A,C), B{\neq}C$
Asymmetric	$p(A,B) \rightarrow \neg p(B,A)$	$\leftarrow p(A,B), p(B,A)$	$\leftarrow mother(A,B), mother(B,A)$
Exclusive	$p(A) \rightarrow \neg q(A)$	$\leftarrow p(A), q(A)$	$\leftarrow odd(A), even(A)$

Use ASP programs to discover the constraints

Functional property

$$p(A,B),\ p(A,C) \rightarrow B=C$$

Functional constraint

$$\leftarrow p(A,B),\ p(A,C),\ B!=C$$

Functional example

$\leftarrow length(A,B),\ length(A,C),\ B!=C$

Bootstrapping

We use the constraints to bootstrap Popper, a constraint-driven ILP system.

Does it work?

Does it work?

How long does BK preprocessing take?

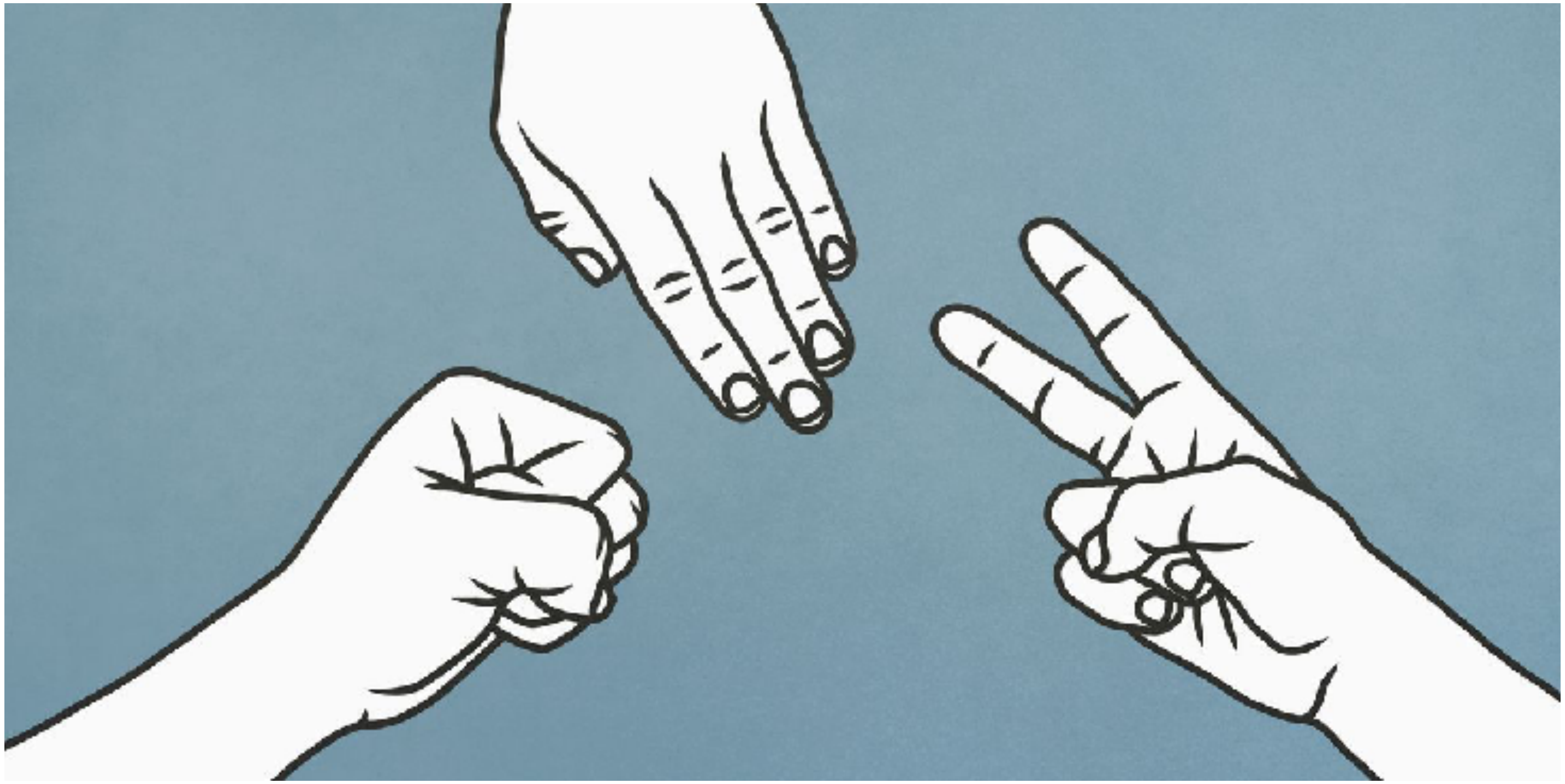
Domain	**Time**
trains	0.22 ± 0.00
zendo	0.03 ± 0.00
imdb	0.02 ± 0.00
krk	0.10 ± 0.00
rps	0.02 ± 0.00
centipede	0.02 ± 0.00
md	0.01 ± 0.00
buttons	0.02 ± 0.00
attrition	0.01 ± 0.00
coins	0.03 ± 0.00
synthesis	4.00 ± 0.40

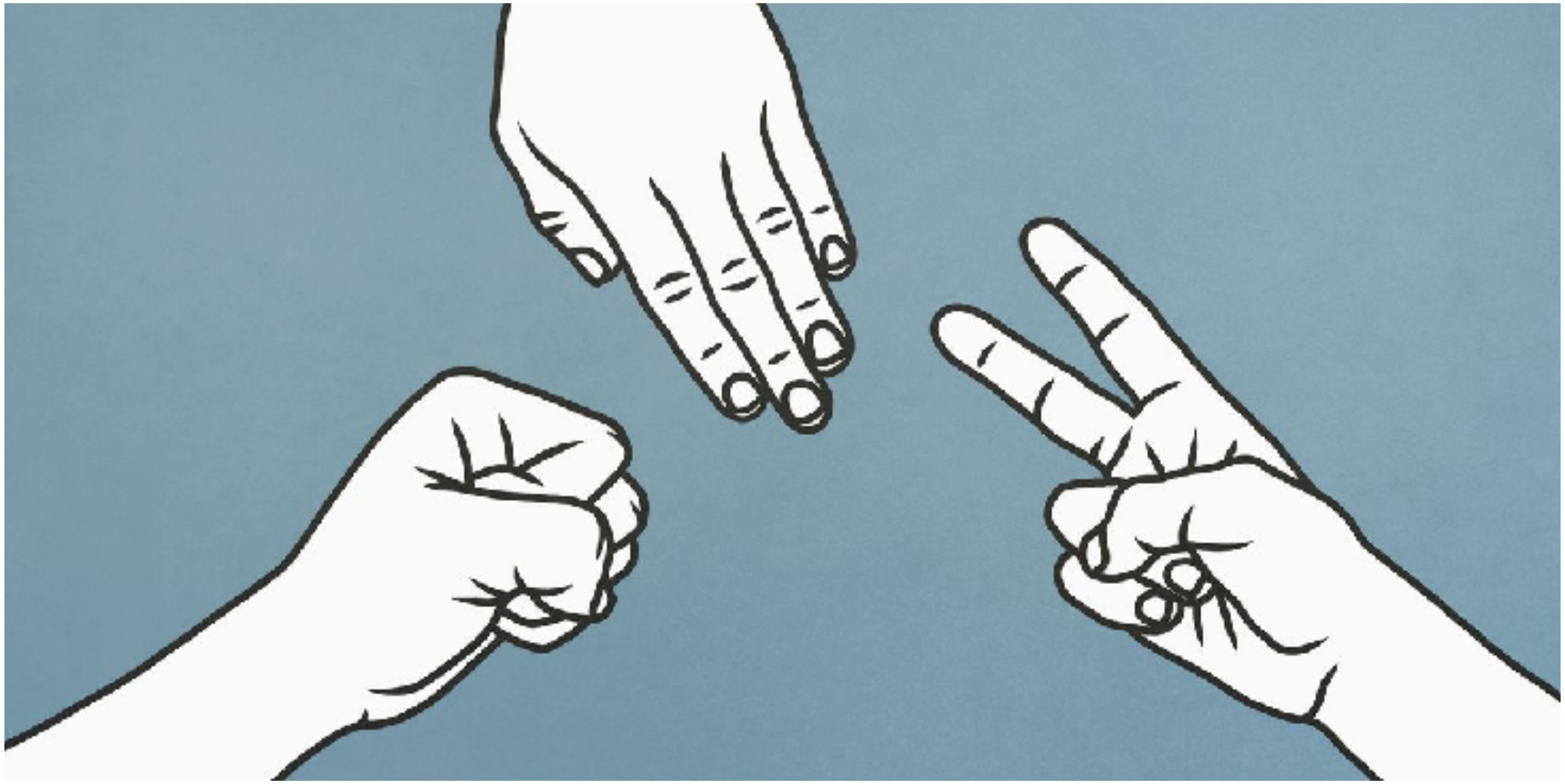
Does it work?

Q1. Can BK constraint discovery reduce learning times?

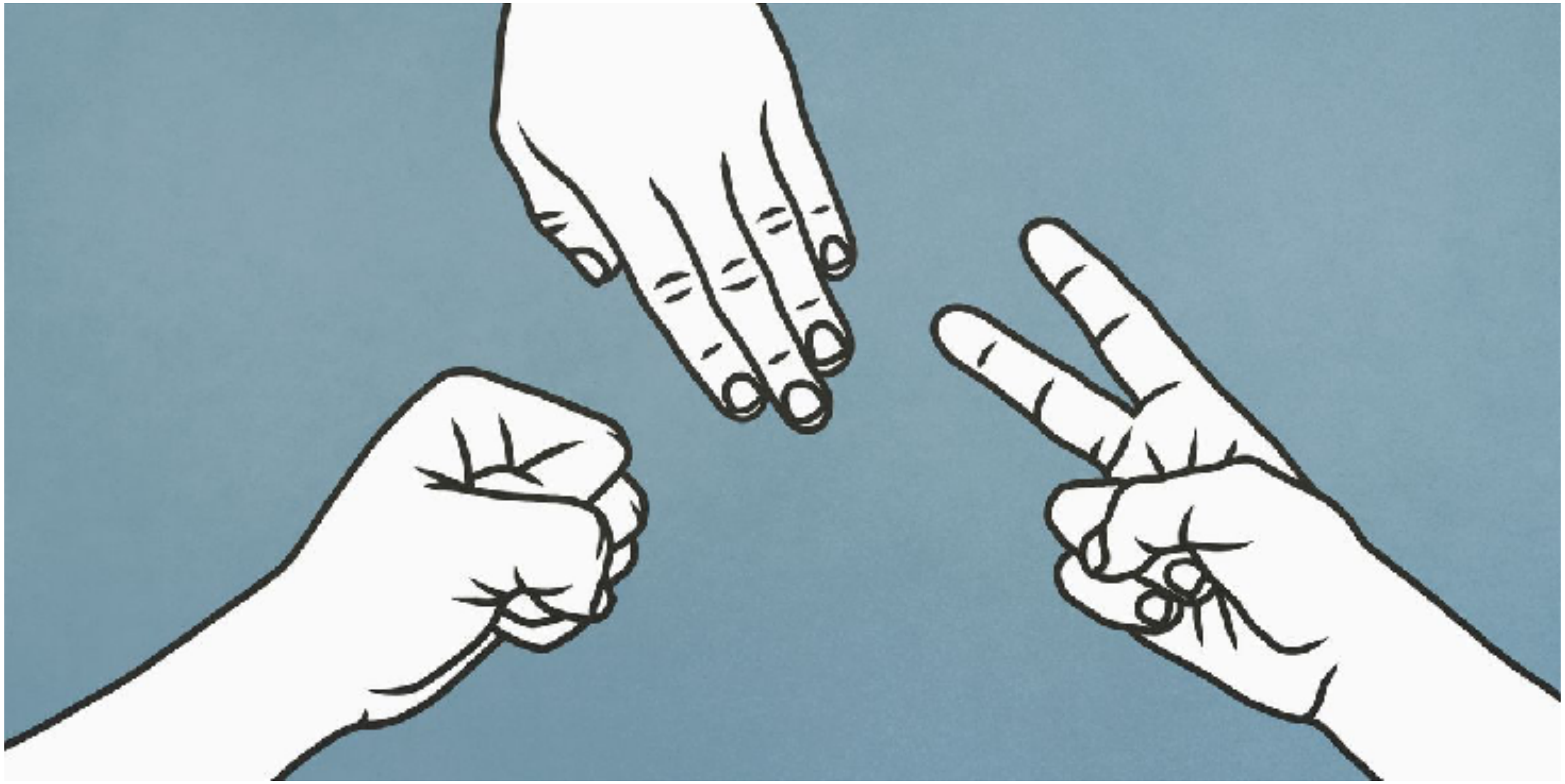
Does it work?

Task	Popper	Disco	Change
trains1	5 ± 0.1	4 ± 0.1	**-20%**
trains2	5 ± 0.2	4 ± 0.3	**-20%**
trains3	27 ± 0.8	22 ± 0.6	**-18%**
trains4	24 ± 0.8	20 ± 0.5	**-16%**
zendo1	8 ± 2	6 ± 1	**-25%**
zendo2	32 ± 2	31 ± 2	**-3%**
zendo3	33 ± 2	31 ± 1	**-6%**
zendo4	24 ± 3	24 ± 3	0%
imdb1	1 ± 0	1 ± 0	0%
imdb2	2 ± 0.1	2 ± 0	0%
imdb3	366 ± 23	287 ± 17	**-21%**
krk	48 ± 6	9 ± 0.6	**-81%**
rps	37 ± 1	6 ± 0.2	**-83%**
centipede	47 ± 2	9 ± 0.2	**-80%**
md	142 ± 7	13 ± 0.4	**-90%**
buttons	686 ± 109	25 ± 1	**-96%**
attrition	410 ± 20	57 ± 2	**-86%**
coins	496 ± 19	345 ± 18	**-30%**
buttons-goal	11 ± 0.2	5 ± 0.1	**-54%**
coins-goal	122 ± 6	76 ± 2	**-37%**
dropk	4 ± 0.3	3 ± 0.2	**-25%**
droplast	41 ± 3	23 ± 2	**-43%**
evens	33 ± 7	9 ± 1	**-72%**
finddup	51 ± 8	32 ± 4	**-37%**
last	4 ± 0.4	3 ± 0.2	**-25%**
len	31 ± 5	16 ± 2	**-48%**
sorted	74 ± 5	23 ± 1	**-68%**
sumlist	554 ± 122	320 ± 40	**-42%**



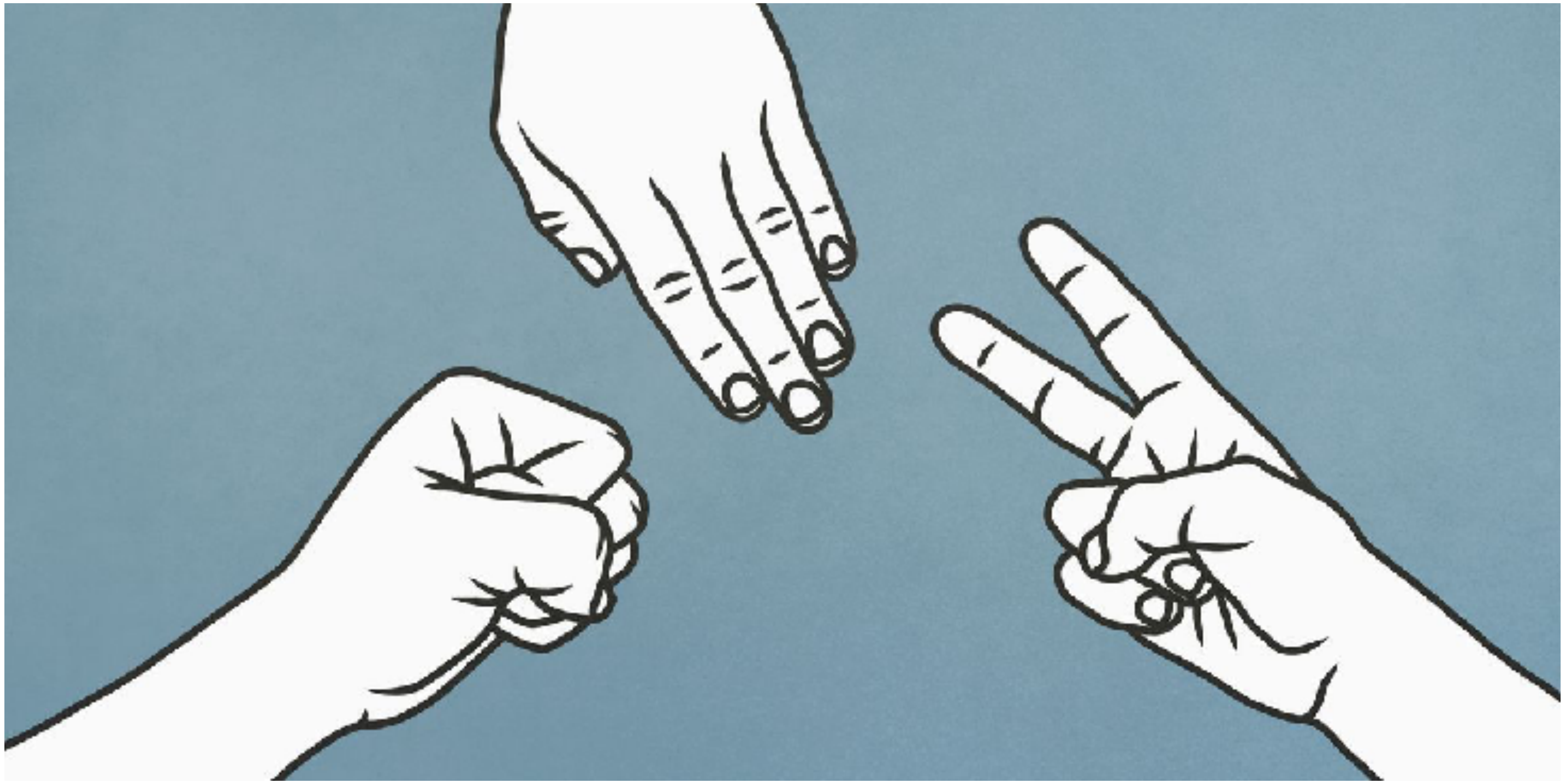


succ/2



succ/2

irreflexive, injective, functional, antitransitive, antitriangular, and asymmetric



succ/2

irreflexive, injective, functional, antitransitive, antitriangular, and asymmetric

The resulting constraints reduce the number of rules in the hypothesis space from **1,189,916** to **70,270**

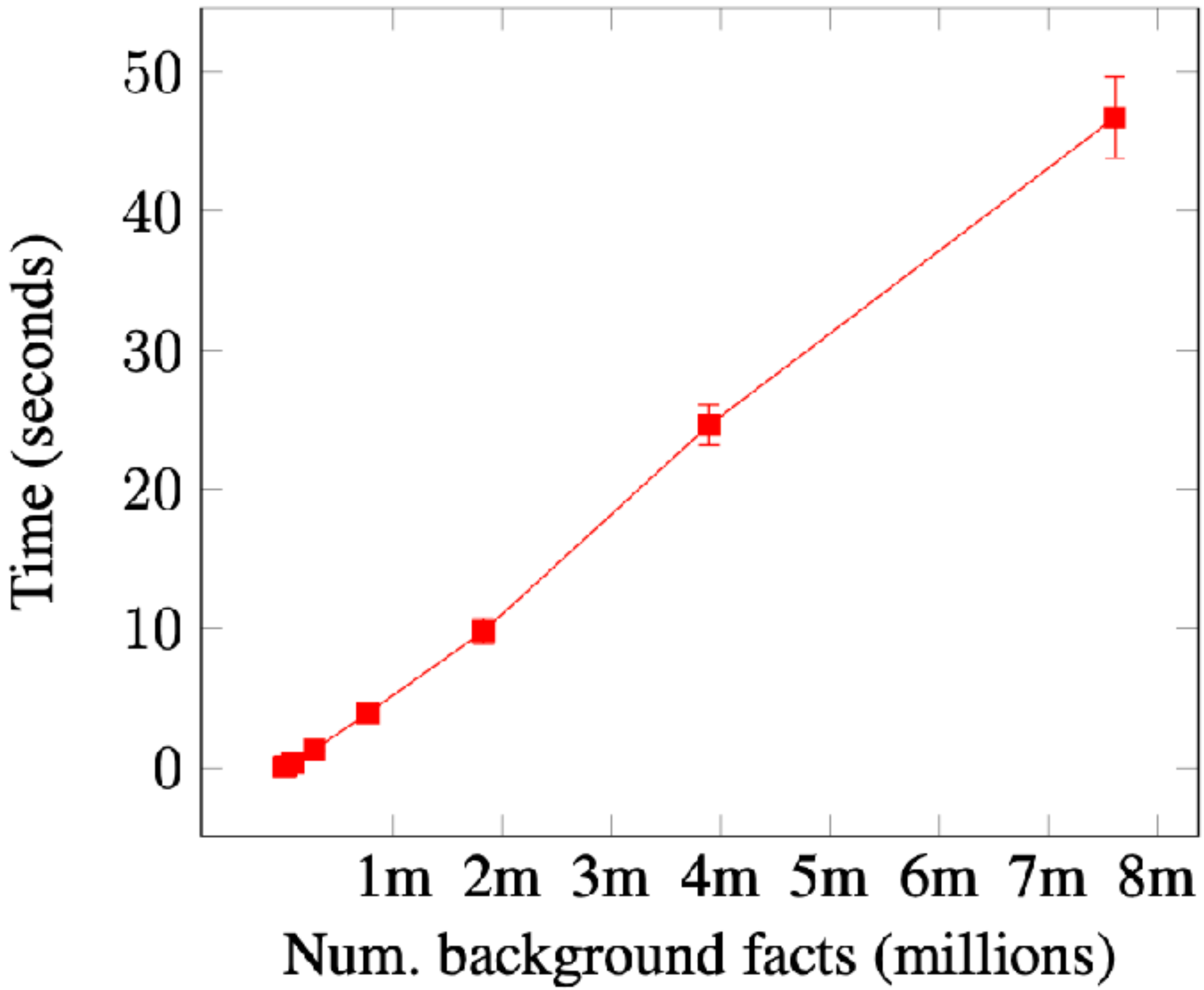
Does it work?

Q2. What effect does BK constraint discovery have on learning times given larger hypothesis spaces?

Size	**Popper**	**Disco**	**Change**
5	12 ± 0.9	3 ± 0.3	**-75%**
6	113 ± 2	10 ± 0.1	**-91%**
7	864 ± 156	23 ± 0.9	**-97%**
8	*timeout*	47 ± 2	**-96%**
9	*timeout*	48 ± 3	**-96%**
10	*timeout*	52 ± 0.1	**-95%**

Does it work?

Q3. How long does BK constraint discovery take given larger BK?



50
40
30
20
10
0
Time (seconds)
1m
2m
3m
4m
5m
6m
7m
8m
Num. background facts (millions)

Why does this idea work?

We only need one counter-example to eliminate a property

Why does this idea work?

Our properties are small

Why care?

Simple, general, and performs well

Why care?

Simple, general, and performs well

Can easily be made better

What can be improved?

Assume a finite domain (Datalog programs)

What can be improved?

Assume given properties to discover

Questions?

Poster ID 191

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