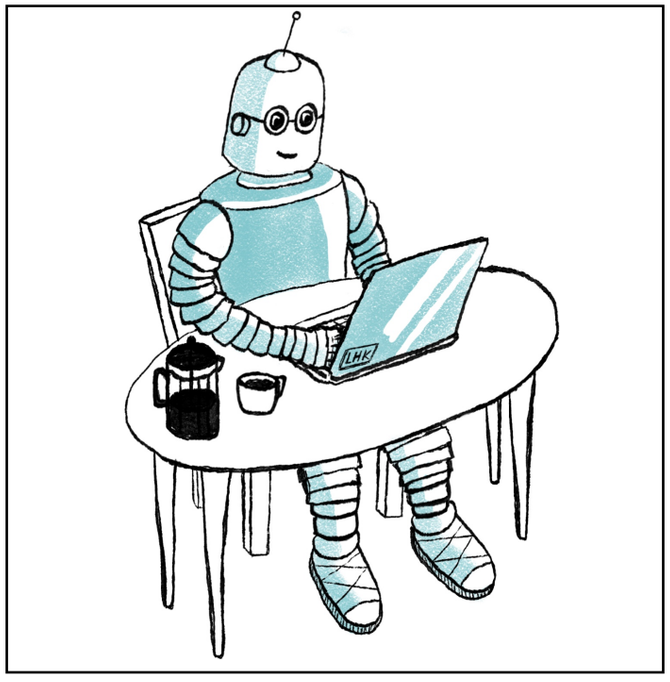
The Automatic Computer Scientist



Andrew Cropper

University of Oxford

What is this talk about?

Introduction to inductive logic programming

Algorithms are ubiquitous!

Algorithms are ubiquitous!

Algorithms are ubiquitous!

Round trip · 1 · Economy

London ⇄ Helsinki · Fri, 5 Feb – Tue, 9 Feb

Stops · Airlines · Bags · Price · Times · Connecting airports · Duration · More

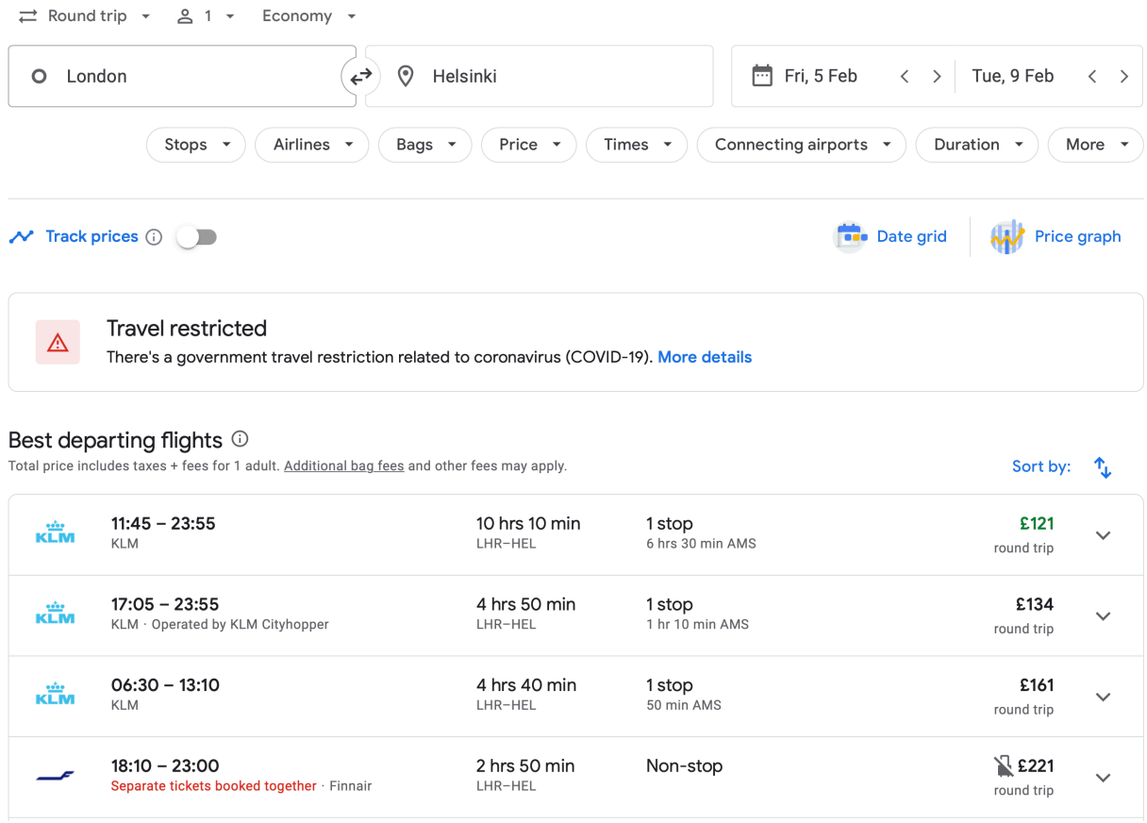
Track prices · Date grid · Price graph

Travel restricted
There's a government travel restriction related to coronavirus (COVID-19). More details

Best departing flights
Total price includes taxes + fees for 1 adult. Additional bag fees and other fees may apply. Sort by:

KLM	11:45 – 23:55 KLM	10 hrs 10 min LHR–HEL	1 stop 6 hrs 30 min AMS	£121 round trip
KLM	17:05 – 23:55 KLM · Operated by KLM Cityhopper	4 hrs 50 min LHR–HEL	1 stop 1 hr 10 min AMS	£134 round trip
KLM	06:30 – 13:10 KLM	4 hrs 40 min LHR–HEL	1 stop 50 min AMS	£161 round trip
Finnair	18:10 – 23:00 Separate tickets booked together · Finnair	2 hrs 50 min LHR–HEL	Non-stop	£221 round trip

Algorithms are ubiquitous!



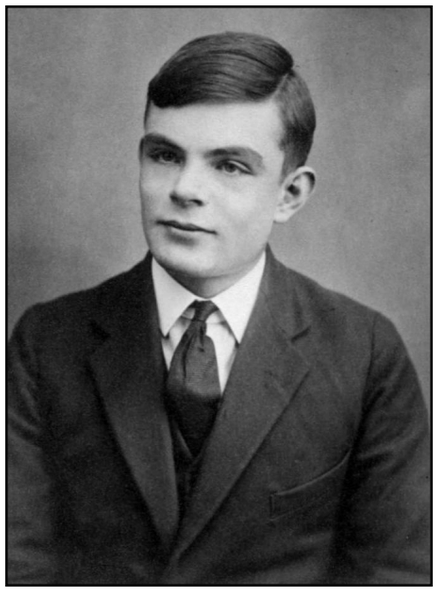
How do we design novel algorithms?

How do we design novel algorithms?

Problem

How do we design novel algorithms?

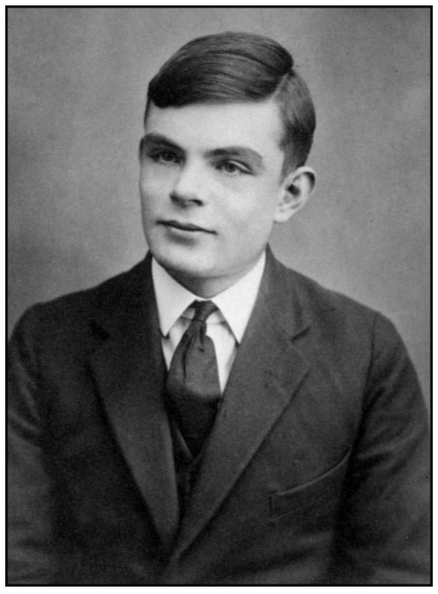
Problem



Computer Scientist

How do we design novel algorithms?

Problem

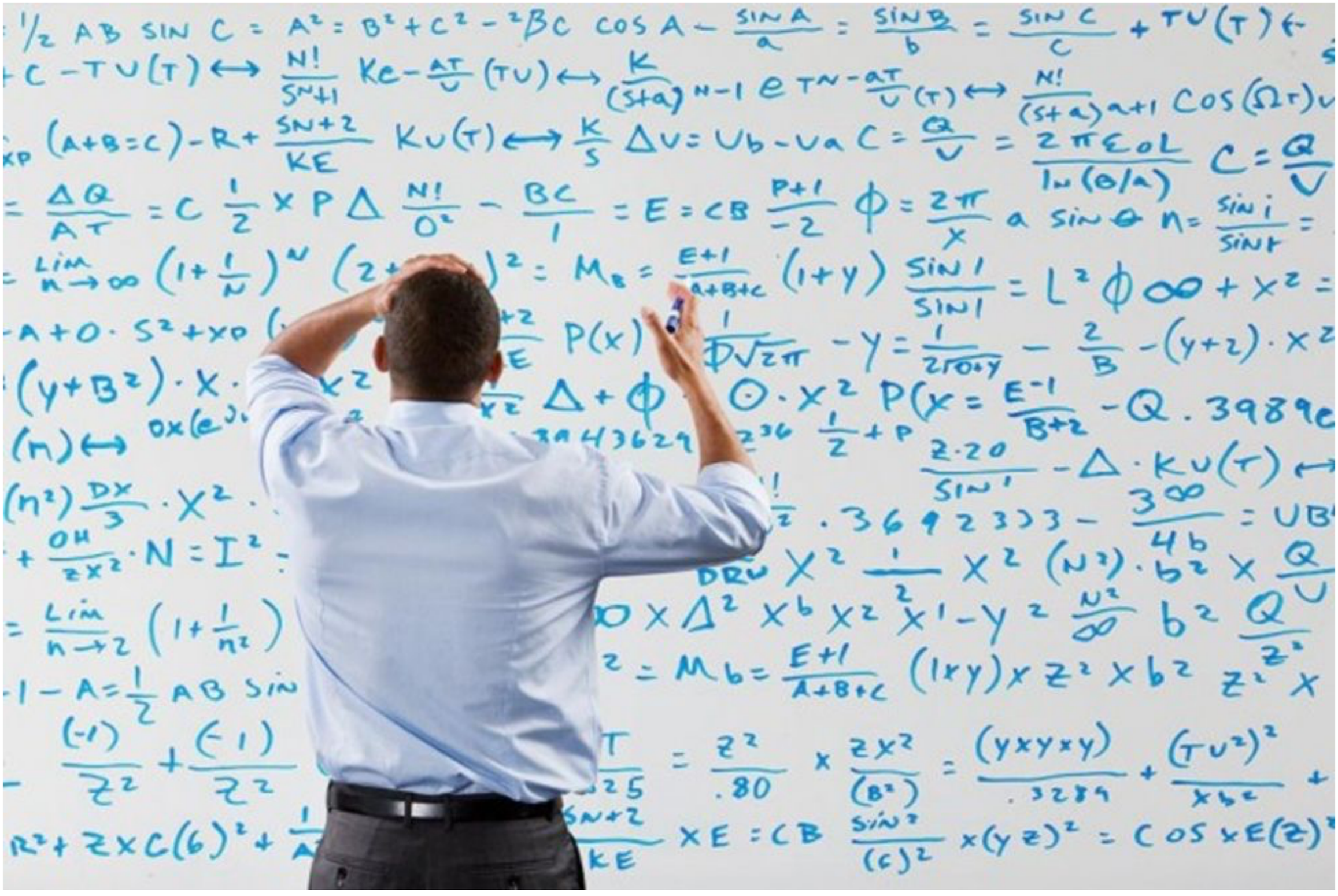


Computer Scientist

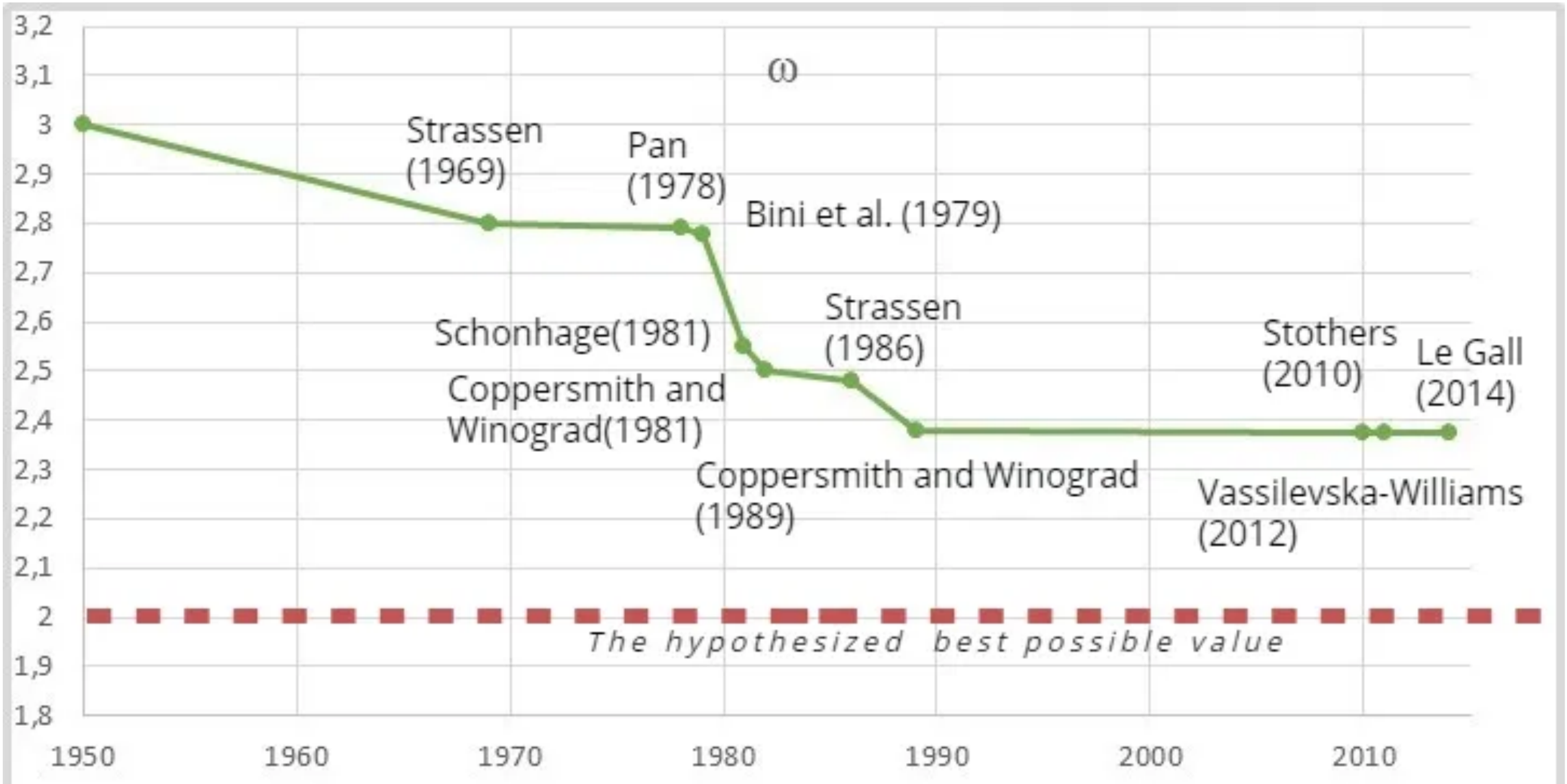
Algorithm

Any problems?

Problem 1: algorithm design is difficult



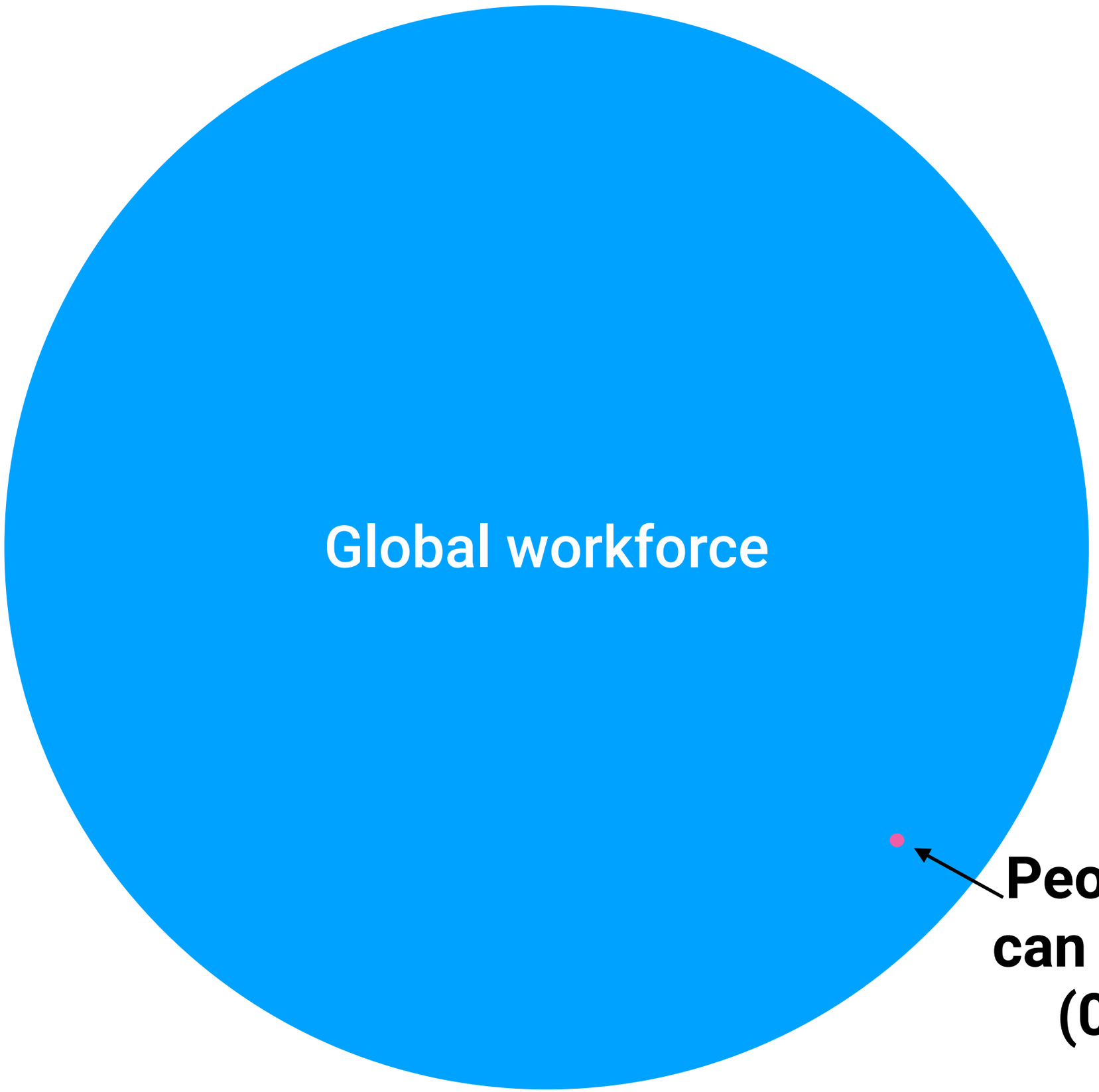
Matrix multiplication



**I understand that these algorithms are not used in practice because of large hidden constants*

Problem 2: few people can program

Problem 2: few people can program

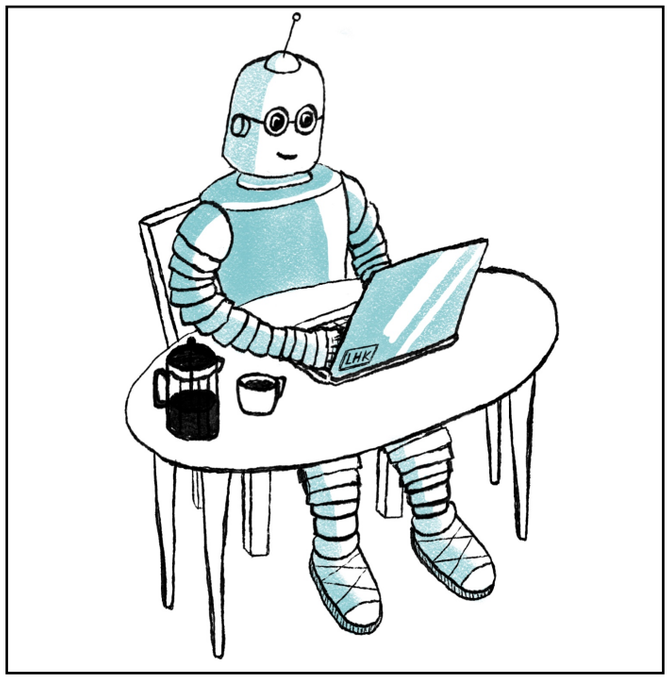


My research

My research

My research

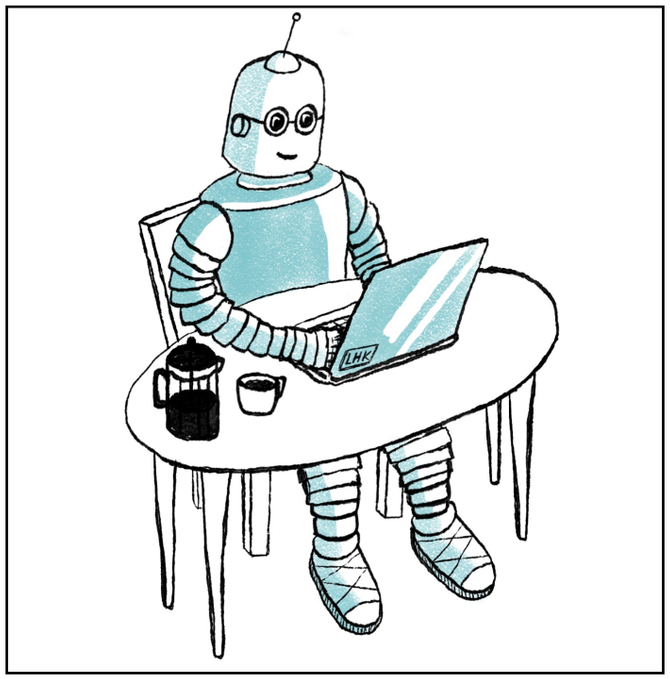
Problem



AutoCS

My research

Problem

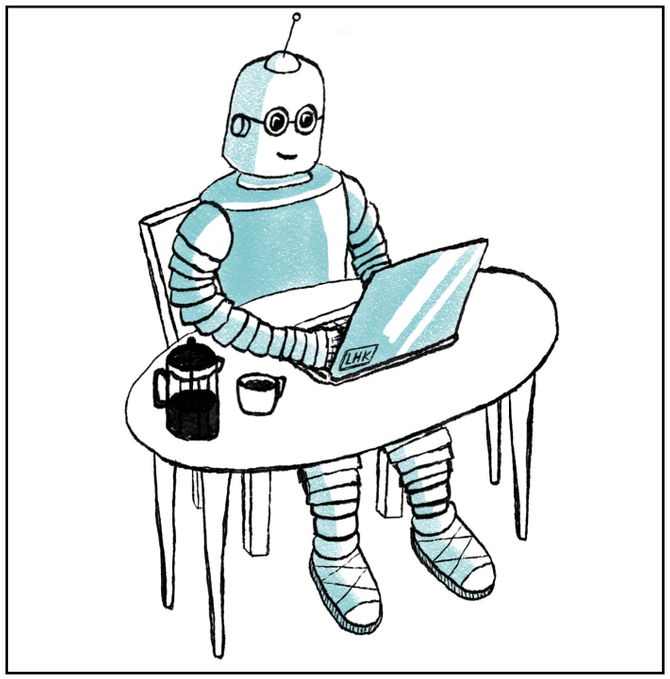


Algorithm

AutoCS

My research

Problem



Algorithm

AutoCS

I design machine learning algorithms that learn algorithms

Algorithm discovery

Algorithm discovery

nature

Explore content Journal information

nature › articles › article

Article | Published: 20 January 2026

Machine-discovered fastest matrix multiplication algorithm

M.L. Algorithm & A. Cropper

Nature **589**, 386–390 (2026) Cite this article

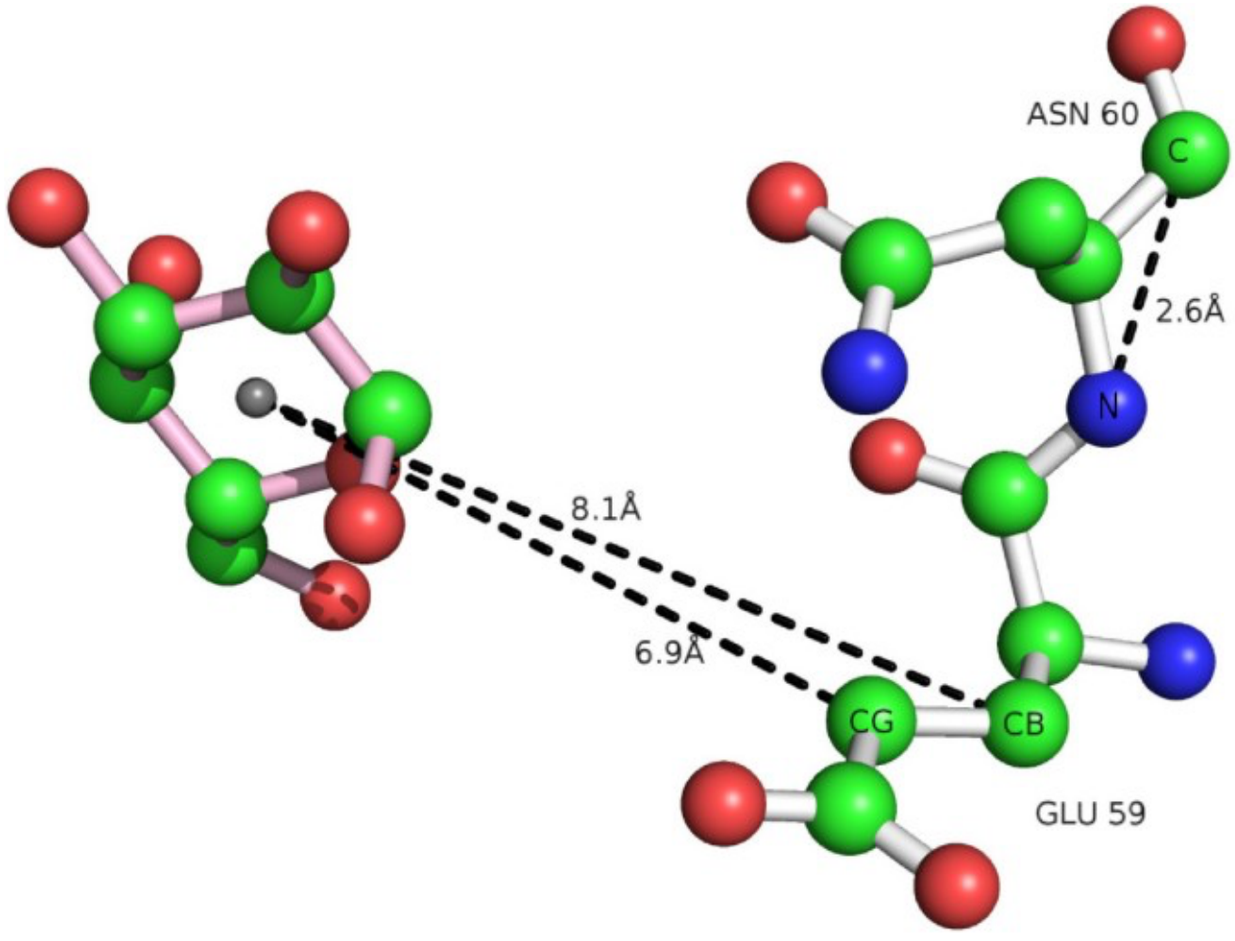
Automate programming

Automate programming

	A	B	C
1	**Participants**	**Country**	
2	Ronnie Anderson, UK	UK	
3	Tom Boone, Canada	Canada	
4	Sally Brook, USA	USA	
5	Jeremy Hill, Australia	Australia	
6	Mattias Waldau, USA	USA	
7	Robert Furlan, France	France	
8	David White, UK	UK	

Scientific discovery

Scientific discovery



What features do we want from an AutoCS?

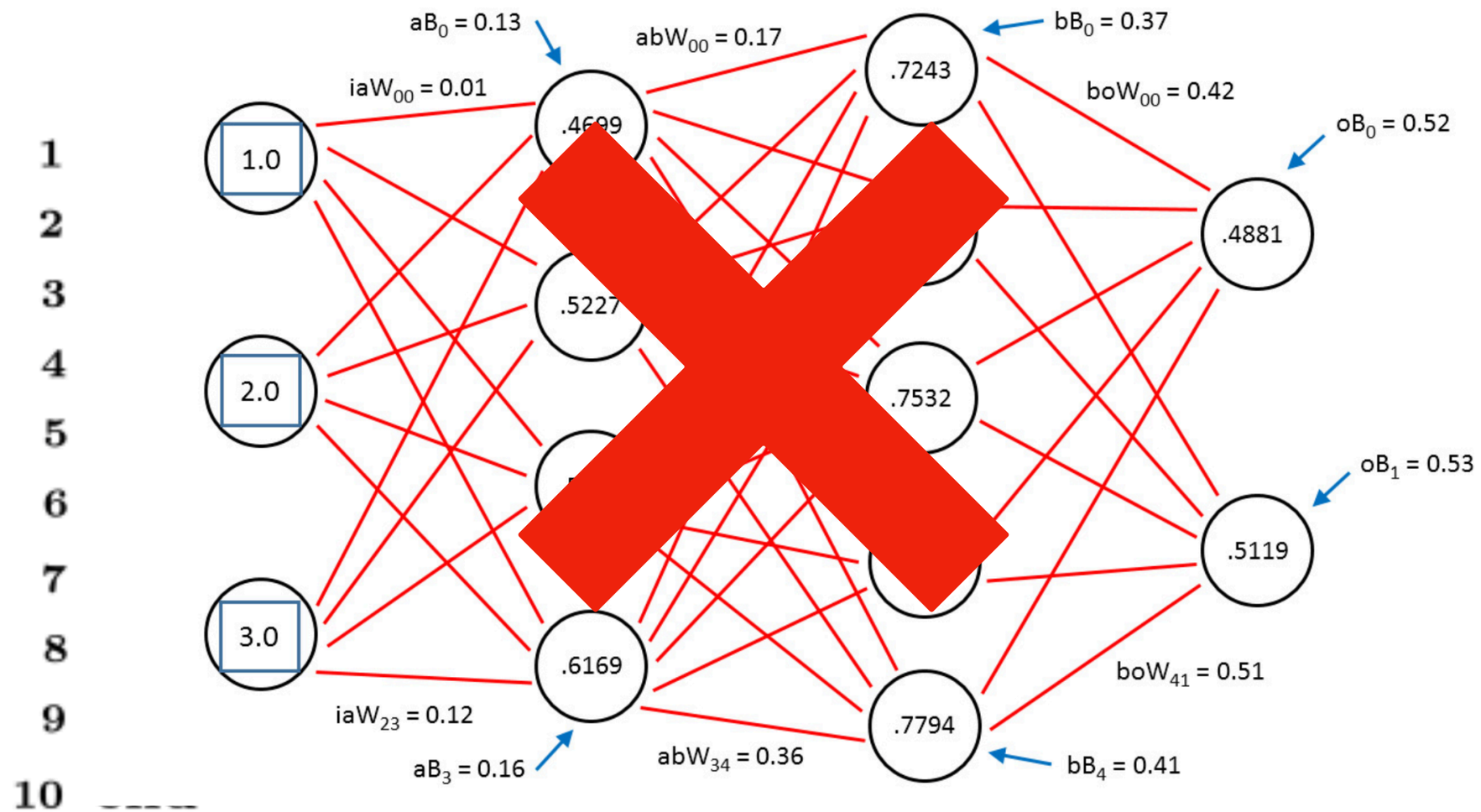
Explainability

To publish an algorithm or scientific discovery, we must **explain** it to others!

Explainability

To publish an algorithm or scientific discovery, we must **explain** it to others!

Algorithm 1: How to write algorithms



Efficiency

Efficiency

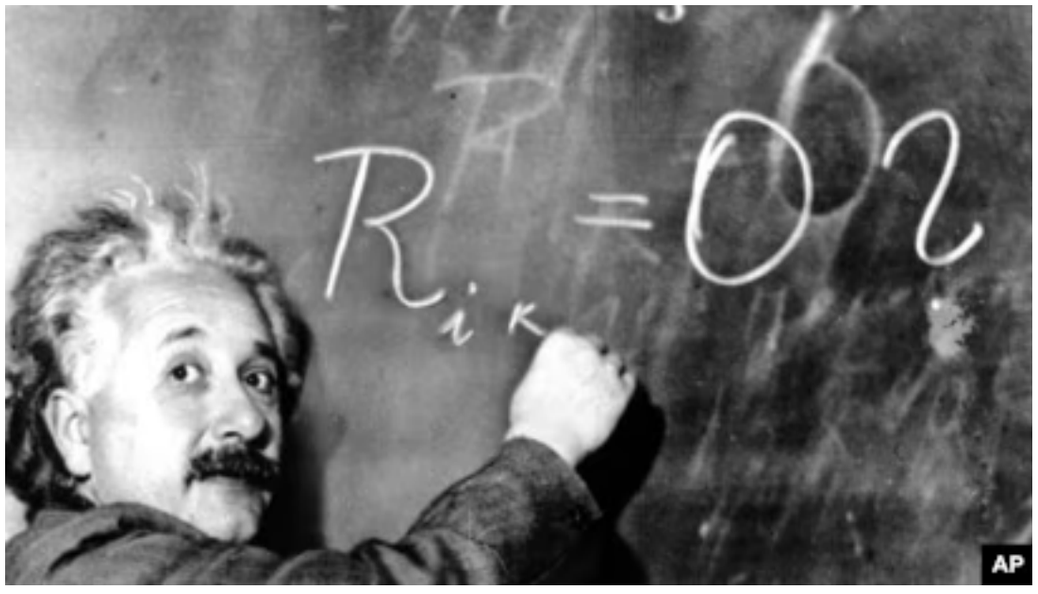
	A	B	C
1	**Participants**	**Country**	
2	Ronnie Anderson, UK	UK	
3	Tom Boone, Canada	Canada	
4	Sally Brook, USA	USA	
5	Jeremy Hill, Australia	Australia	
6	Mattias Waldau, USA	USA	
7	Robert Furlan, France	France	
8	David White, UK	UK	

Users do not want to provide 100+ examples …

Efficiency

Experiments can be expensive!

Expert prior knowledge



How?

Inductive logic programming

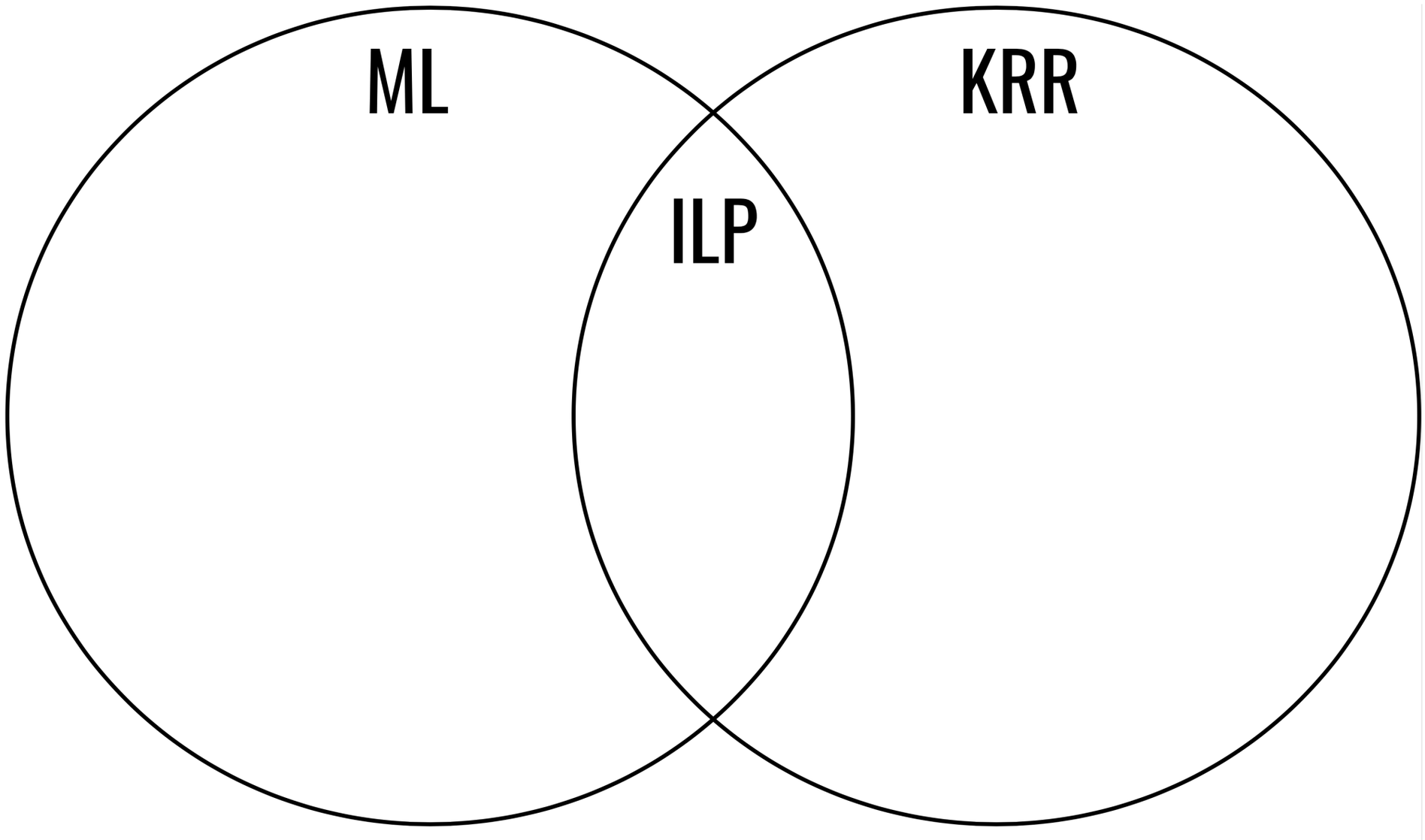
Logic-based machine learning

Inductive logic programming

Combines machine learning and logical reasoning

Inductive logic programming

Combines machine learning and logical reasoning



Inductive logic programming

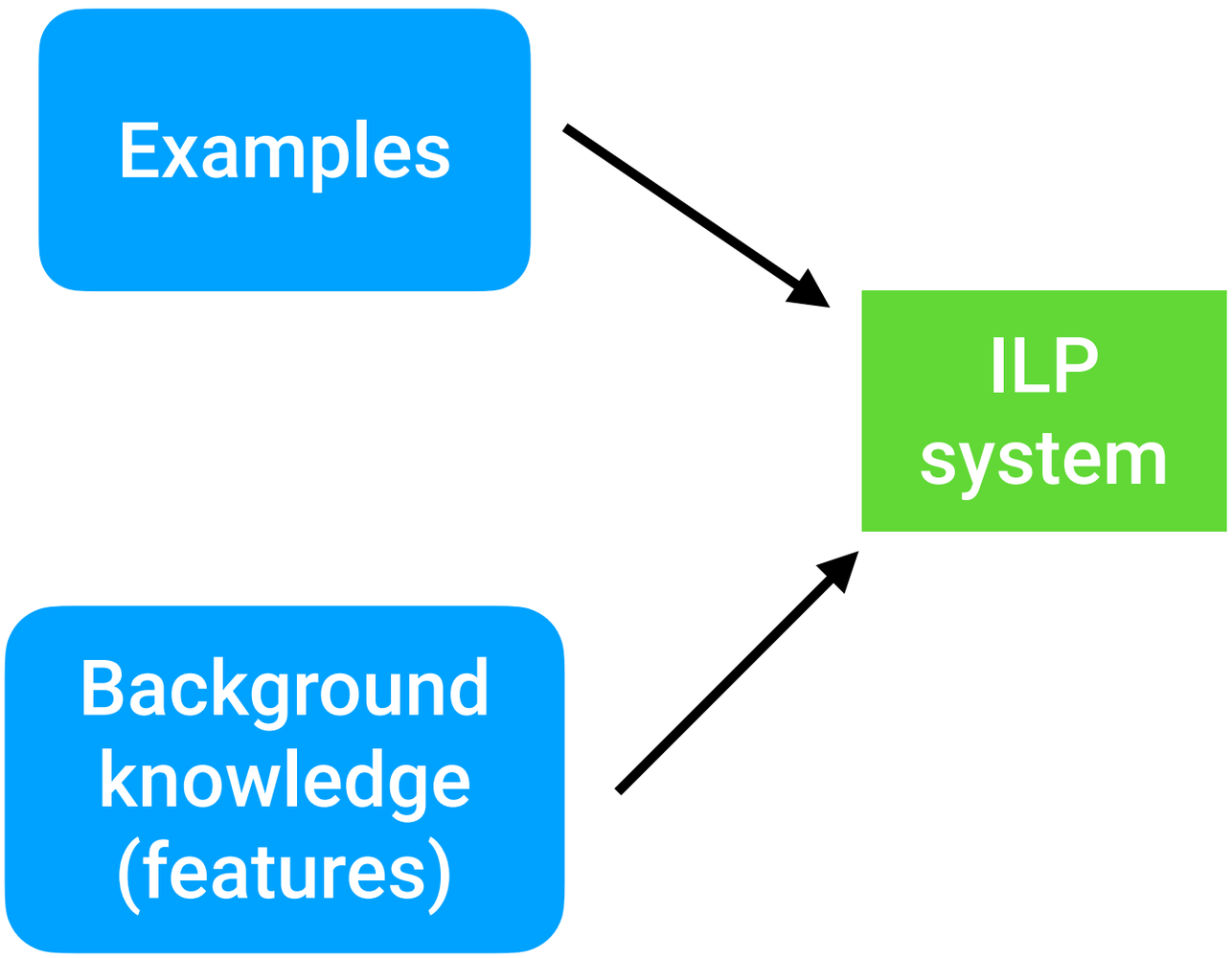
Examples

Inductive logic programming

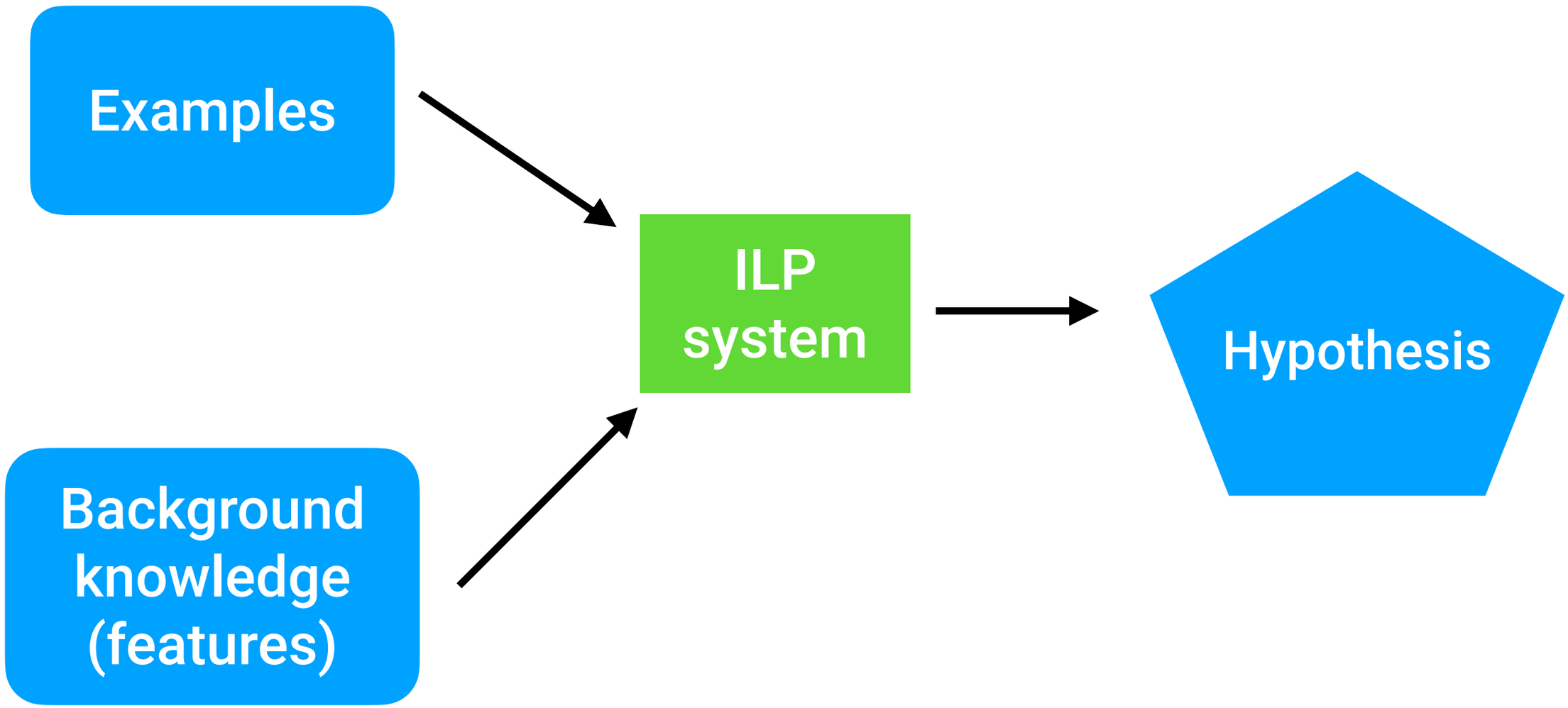
Examples

Background knowledge (features)

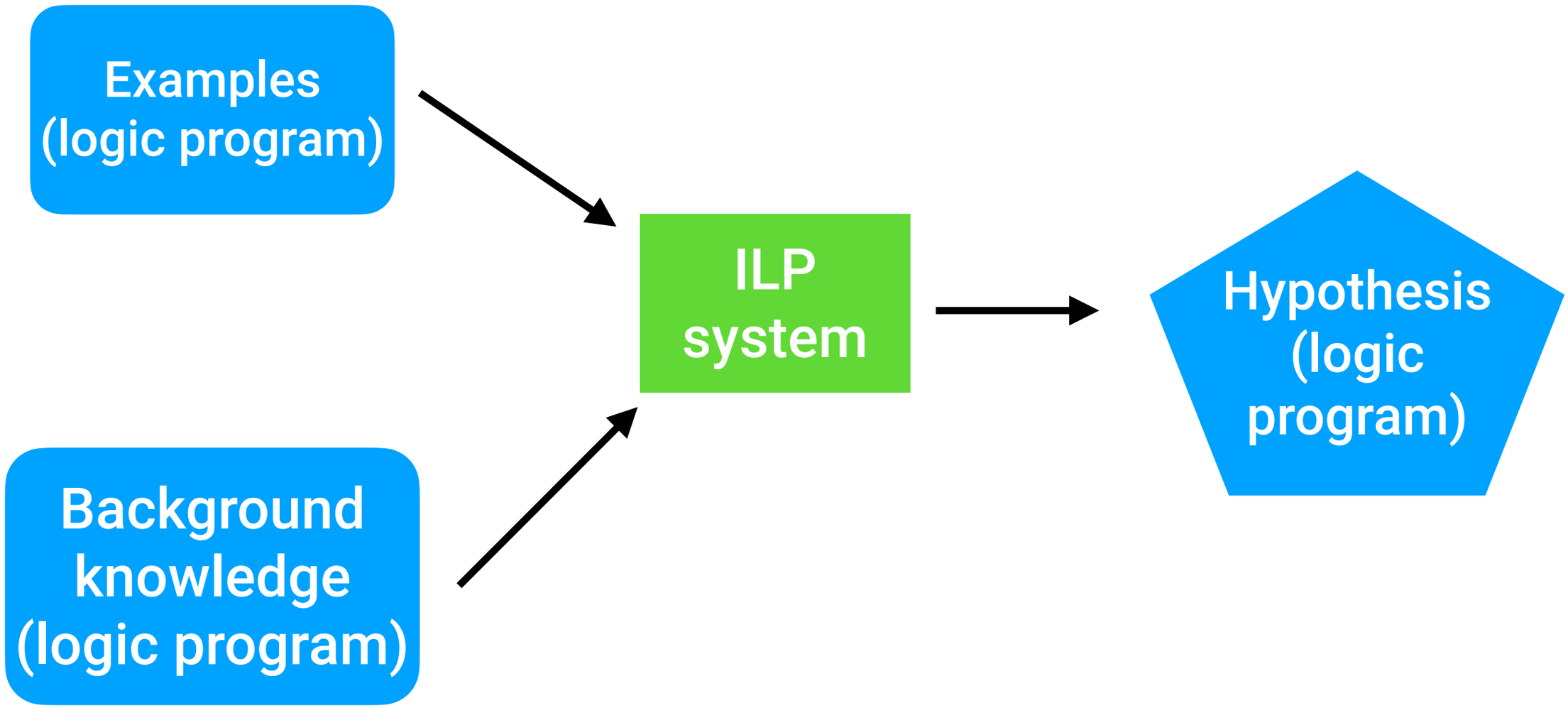
Inductive logic programming



Inductive logic programming



Inductive logic programming



Data are **programs** rather than tables/vectors

Why ILP?

- Data efficient ✓
- Interpretable hypotheses ✓
- Complex prior knowledge ✓

FCAI

Key objectives: data efficiency, trust & ethics, understandability

To create Real AI, we have set up three grand scientific objectives: data efficiency, trust & ethics, and understandability.

Present-day AI solutions work well only in a very small subset of simple domains. In order to expand the deployment of AI further, we need new AI tools that overcome the key shortcomings of current AI systems.

DATA EFFICIENCY

Current AI solutions can be very successful in domains where tasks are relatively simple and well-defined and an abundance of high-quality, properly annotated data are available. Existing AI methods do not, however, easily extend to domains where such data are not available or are difficult or expensive to acquire. Real AIs will be able to work with real-world scarce data – ill-defined, hard to acquire or unavailable.

TRUST & ETHICS

We will create AIs that are secure, give trustworthy results, preserve privacy, are fair, and whose use is ethically sustainable. We will develop the required privacy-preserving and secure methods to address challenges related to susceptibility to manipulation, information stealing and unethical approaches. We will provide new resilient deep learning approaches for the currently popular and successful deep neural networks.

UNDERSTANDABILITY

AI does not yet understand users. We need to open the “black box” of many AI methods: to understand how methods such as neural networks operate and what are the uncertainties inherent to their outputs. Modeling the user and the interaction will help the AI understand the user and vice versa. The outcome is AIs that are able to augment human capabilities in a multitude of tasks.

Why logic programs?

Please ask me in the Q&A section

A very very quick overview of logic programs …

$$\forall \mathbf{A}.\ \text{works\_in}(\mathbf{A}, \text{finland}) \rightarrow \text{happy}(\mathbf{A})$$

*If **A** works in Finland then **A** is happy*

$$\forall \mathbf{A}.\ \text{works\_at}(\mathbf{A},\text{finland}) \wedge \neg\text{winter} \rightarrow \text{happy}(\mathbf{A})$$

*If **A** works in Finland and it is not winter then **A** is happy*

$$\forall \mathbf{A}.\ \text{happy}(\mathbf{A}) \leftarrow \text{works\_in}(\mathbf{A},\text{finland}) \wedge \neg\text{winter}$$

__A__ is happy if __A__ works at Finland and it is not winter

happy(**A**) ← works_in(**A,**finland) ∧ ¬winter

__A__ is happy if __A__ works in Finland and it is not winter

```
happy(A):- works_in(A,finland), not winter.
```

```
happy(A):- works_in(A,finland), not winter.
happy(A):- works_at(A,finland), owns_sad_light(A).
```

***A** is happy if **A** works in Finland and it is not winter*

or

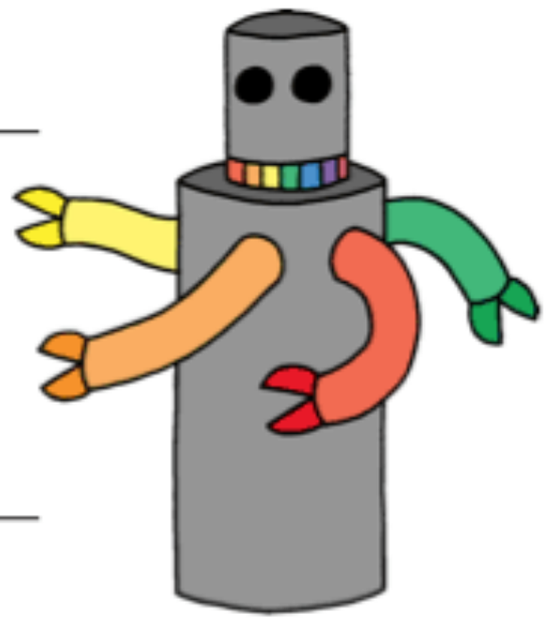
***A** is happy if **A** works in Finland and owns a SAD light*

CONGRATULATIONS!

This is to certify that

FCAI

has completed **level 1** of Logic Club



Signed: **A. Cropper**

Club Leader

Two toy examples

input	output
Häkkinen	ä
Rosberg	o
Räikkönen	**?**

input	output
Häkkinen	ä
Rosberg	o
Räikkönen	**ä**

input	output
Häkkinen	ä
Rosberg	o
Räikkönen	ä

everything after the first element of A

```
def f(A):
    C = tail(A)
    B = head(C)
    return B
```

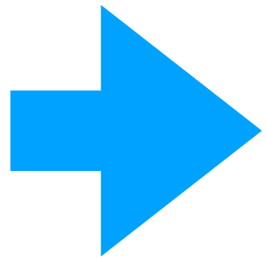
the first element of C

Functions to relations

input	output
Häkkinen	ä
Rosberg	o
Räikkönen	ä

Functions to relations

input	output
Häkkinen	ä
Rosberg	o
Räikkönen	ä



logical representation
f(Häkkinen,ä)
f(Rosberg,o)
f(Räikkönen,ä)

Examples

f(Häkkinen,n)

f(Rosberg,g)

f(Räikkönen,n)

Examples

f(Häkkinen,n)

f(Rosberg,g)

f(Räikkönen,n)

BK

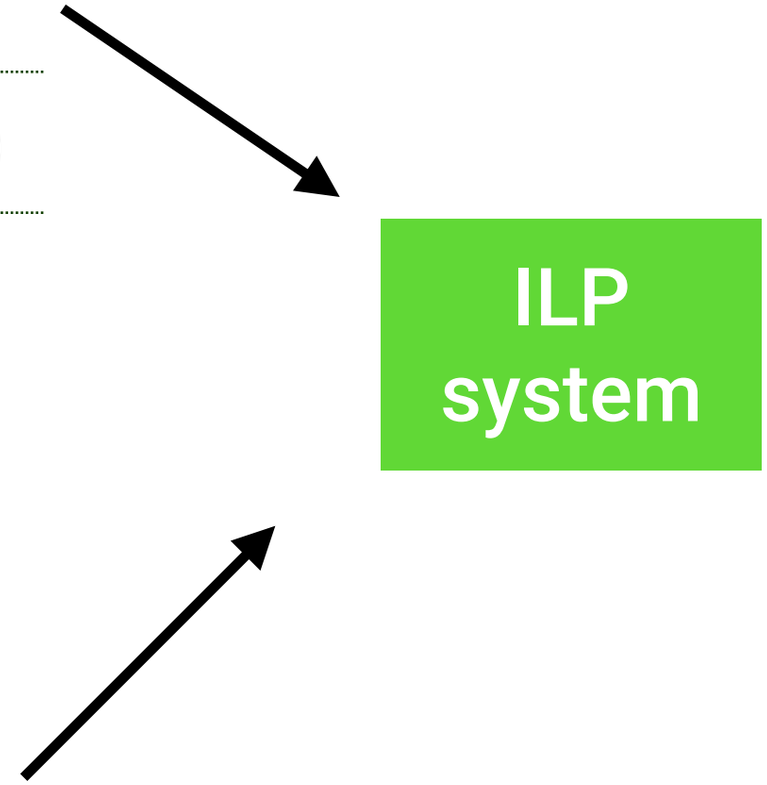
```
head(fcai,f).
head(cai,c).
head(ai,a).
…
tail(fcai,cai).
tail(cai,ca).
…
```

Examples

f(Häkkinen,n)

f(Rosberg,g)

f(Räikkönen,n)



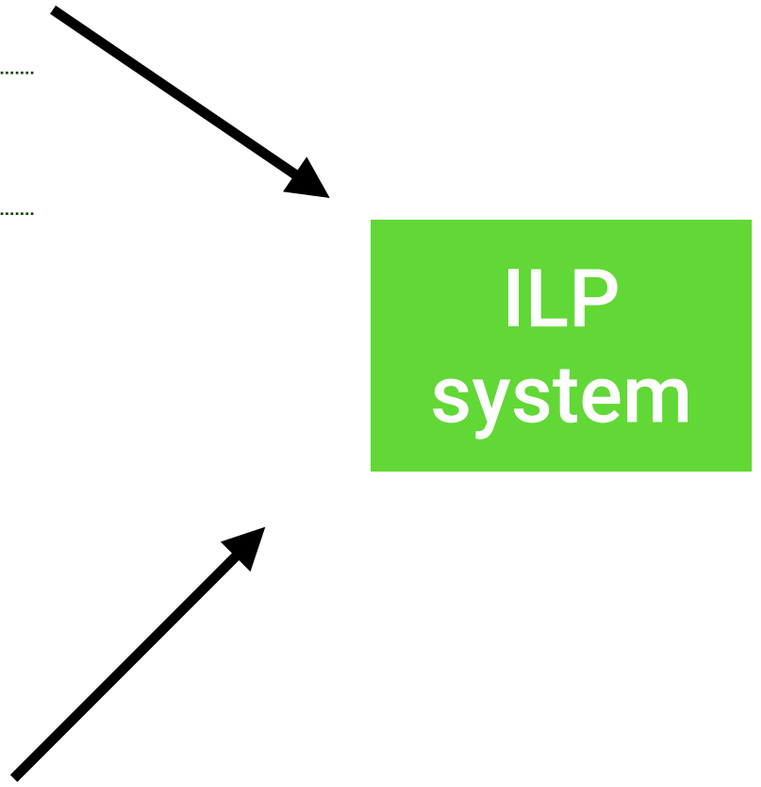
BK

Examples

f(Häkkinen,n)

f(Rosberg,g)

f(Räikkönen,n)



Hypothesis

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

BK

Hypothesis

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

*“**B** is the second element of **A** if it is the head of the tail of **A**”*

input	output
Jyväskylä	ä
Helsinki	i
Espoo	**?**

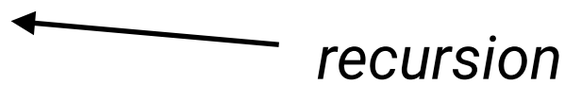
input	output
Jyväskylä	ä
Helsinki	i
Espoo	**o**

examples
f(Jyväskylä,ä)
f(Helsinki,i)
f(Espoo,o)

examples

f(Jyväskylä,ä)
f(Helsinki,i)
f(Espoo,o)

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



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

***B** is the last element of **A** if*

__B__ is the head of __A__ and the tail of __A__ is empty

or

__C__ is the tail of __A__ and __B__ is the last element of __C__

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

***B** is the last element of **A** if*

***B** is the head of **A** and the tail of **A** is empty*

or

C** is the tail of **A** and **B** is the last element of **C

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

***B** is the last element of **A** if*

__B__ is the head of __A__ and the tail of __A__ is empty

or

C** is the tail of **A** and **B** is the last element of **C

How do we do it?

Journal of Artificial Intelligence Research 74 (2022) 765-850 Submitted 11/2021; published 06/2022

Inductive Logic Programming At 30: A New Introduction

Andrew Cropper ANDREW.CROPPER@CS.OX.AC.UK
University of Oxford, Oxford, OX1 2JD, United Kingdom

Sebastijan Dumančić S.DUMANCIC@TUDELFT.NL
TU Delft, Van Mourik Broekmanweg 6, 2628 XE Delft, The Netherlands

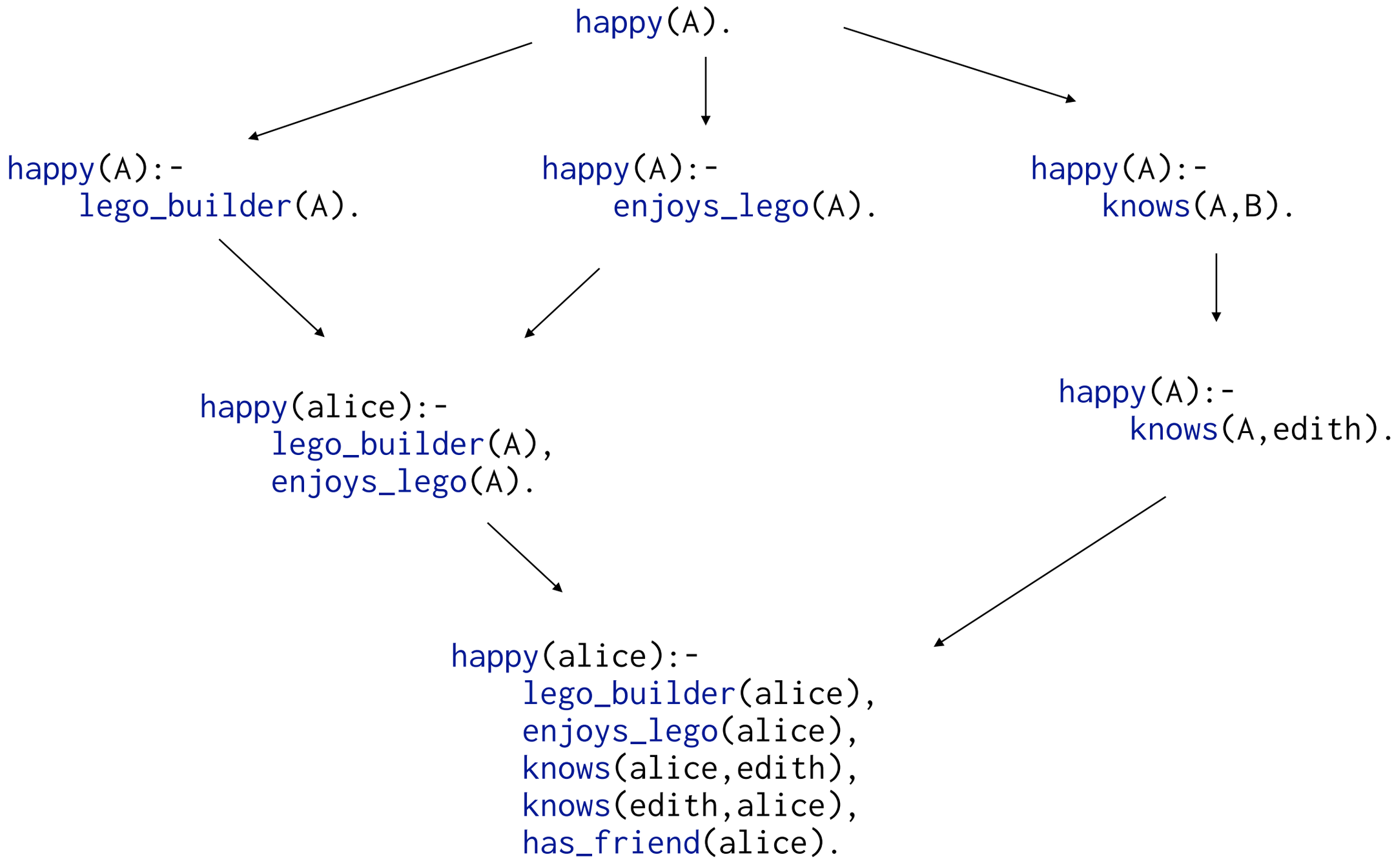
Abstract

Inductive logic programming (ILP) is a form of machine learning. The goal of ILP is to induce a hypothesis (a set of logical rules) that generalises training examples. As ILP turns 30, we provide a new introduction to the field. We introduce the necessary logical notation and the main learning settings; describe the building blocks of an ILP system; compare several systems on several dimensions; describe four systems (Aleph, TILDE, ASPAL, and Metagol); highlight key application areas; and, finally, summarise current limitations and directions for future research.

Subsumption lattice

```
happy(A):- ?
```

Subsumption lattice

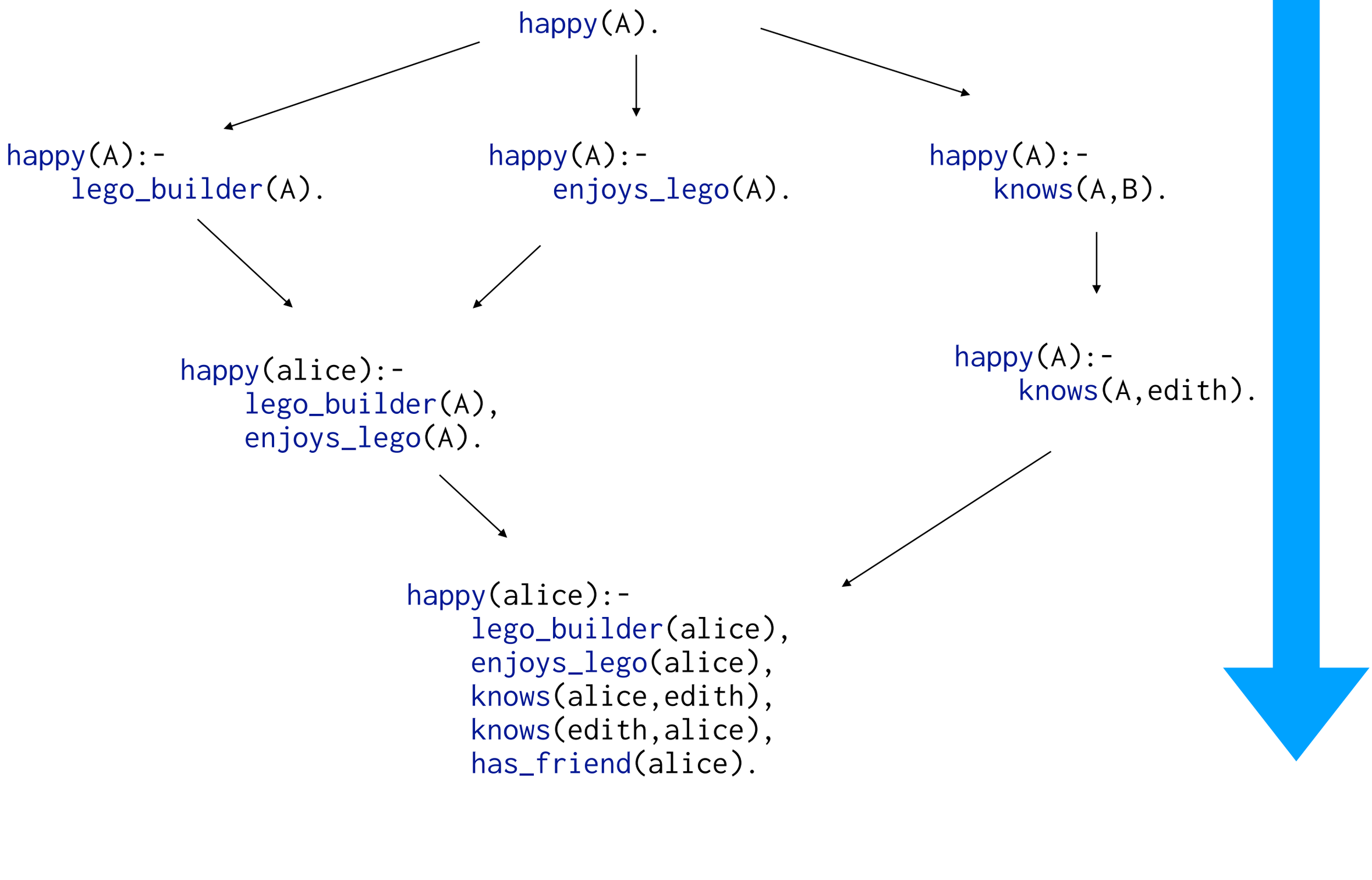


Top-down

Start with a general hypothesis and iteratively specialise it

FOIL, TILDE, HYPER, QuickFOIL, Progol, Aleph**

Top-down

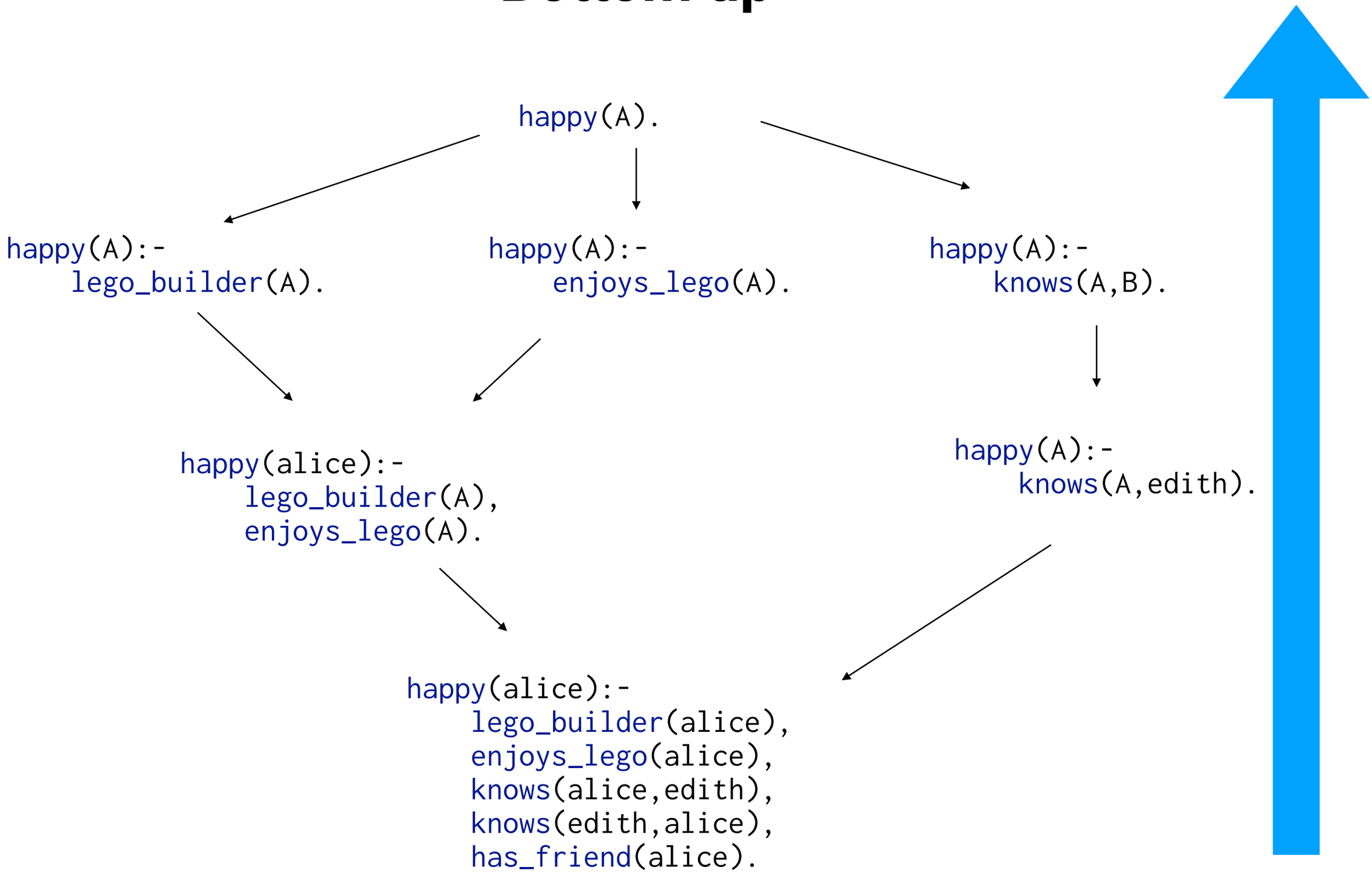

happy(A).
happy(A):-
lego_builder(A).
happy(A):-
enjoys_lego(A).
happy(A):-
knows(A,B).
happy(alice):-
lego_builder(A),
enjoys_lego(A).
happy(A):-
knows(A,edith).
happy(alice):-
lego_builder(alice),
enjoys_lego(alice),
knows(alice,edith),
knows(edith,alice),
has_friend(alice).

Bottom up

Start with a specific hypothesis and iteratively generalise it

CIGOL, GOLEM, XHAIL, Progol, Aleph**

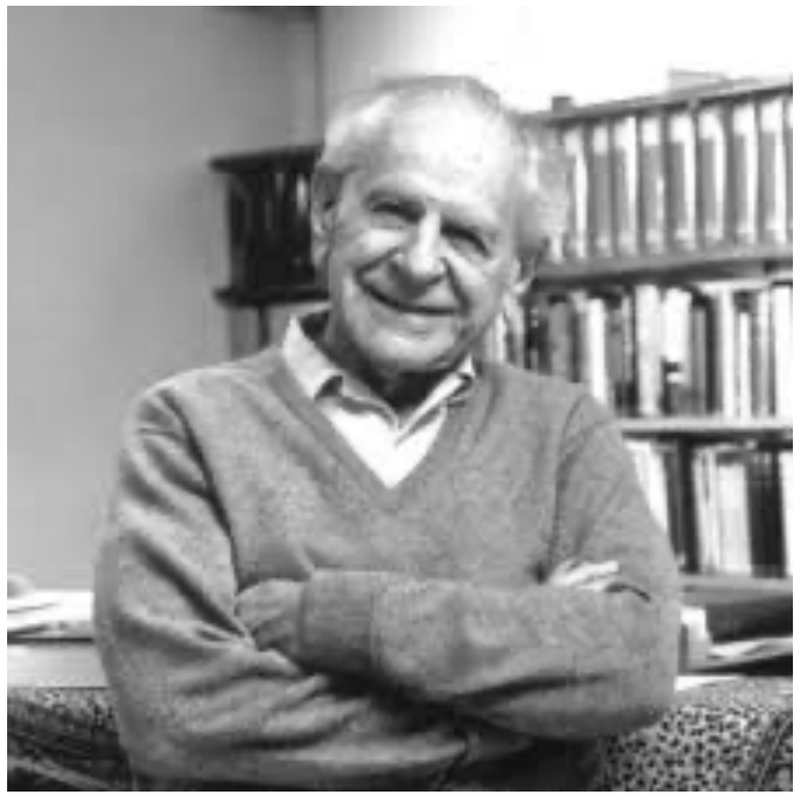
Bottom up


happy(A).
happy(A):-
lego_builder(A).
happy(A):-
enjoys_lego(A).
happy(A):-
knows(A,B).
happy(A):-
knows(A,edith).
happy(alice):-
lego_builder(A),
enjoys_lego(A).
happy(alice):-
lego_builder(alice),
enjoys_lego(alice),
knows(alice,edith),
knows(edith,alice),
has_friend(alice).

How does my group do it?

Popper

Automates Karl Popper’s idea of falsification:



Popper

Automate Karl Popper’s falsifiability:

Popper

Automate Karl Popper’s falsifiability:

1. Form a hypothesis

Popper

Automate Karl Popper’s falsifiability:

1. Form a hypothesis
2. Empirically try to refute it

Popper

Automate Karl Popper’s falsifiability:

1. Form a hypothesis
2. Empirically try to refute it
3. If the hypothesis fails, **determine why**

Popper

Automate Karl Popper’s falsifiability:

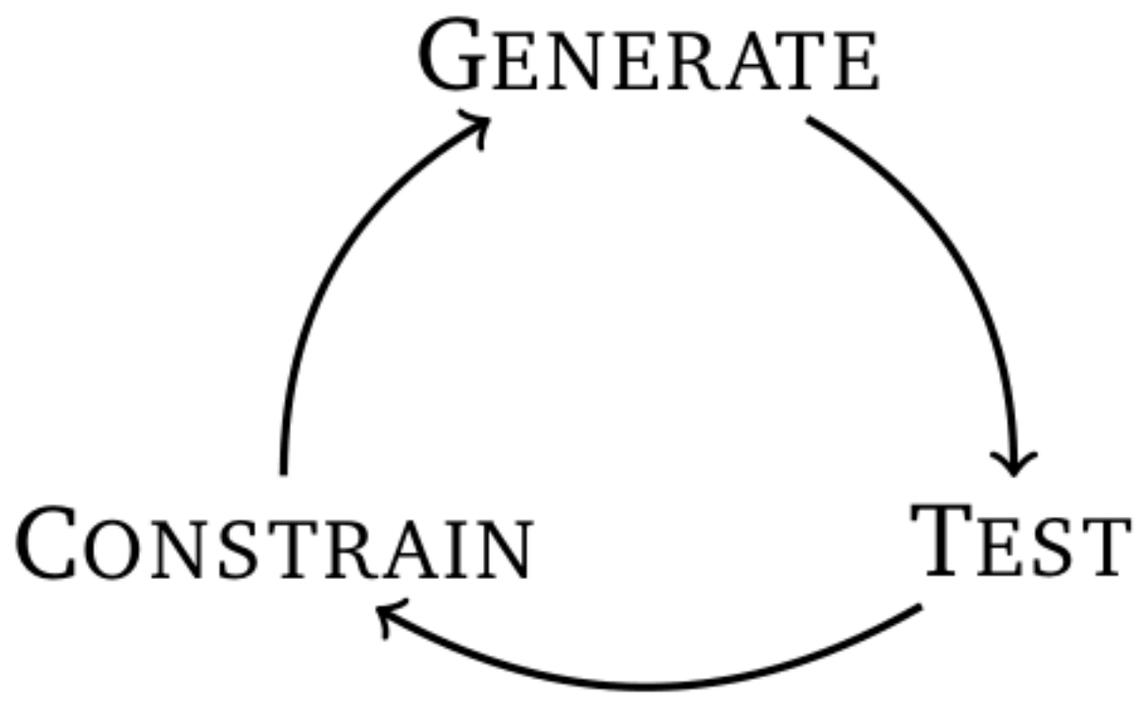
1. Form a hypothesis
2. Empirically try to refute it
3. If the hypothesis fails, **determine why**
4. Use the **explanation** to rule out other hypotheses

Popper

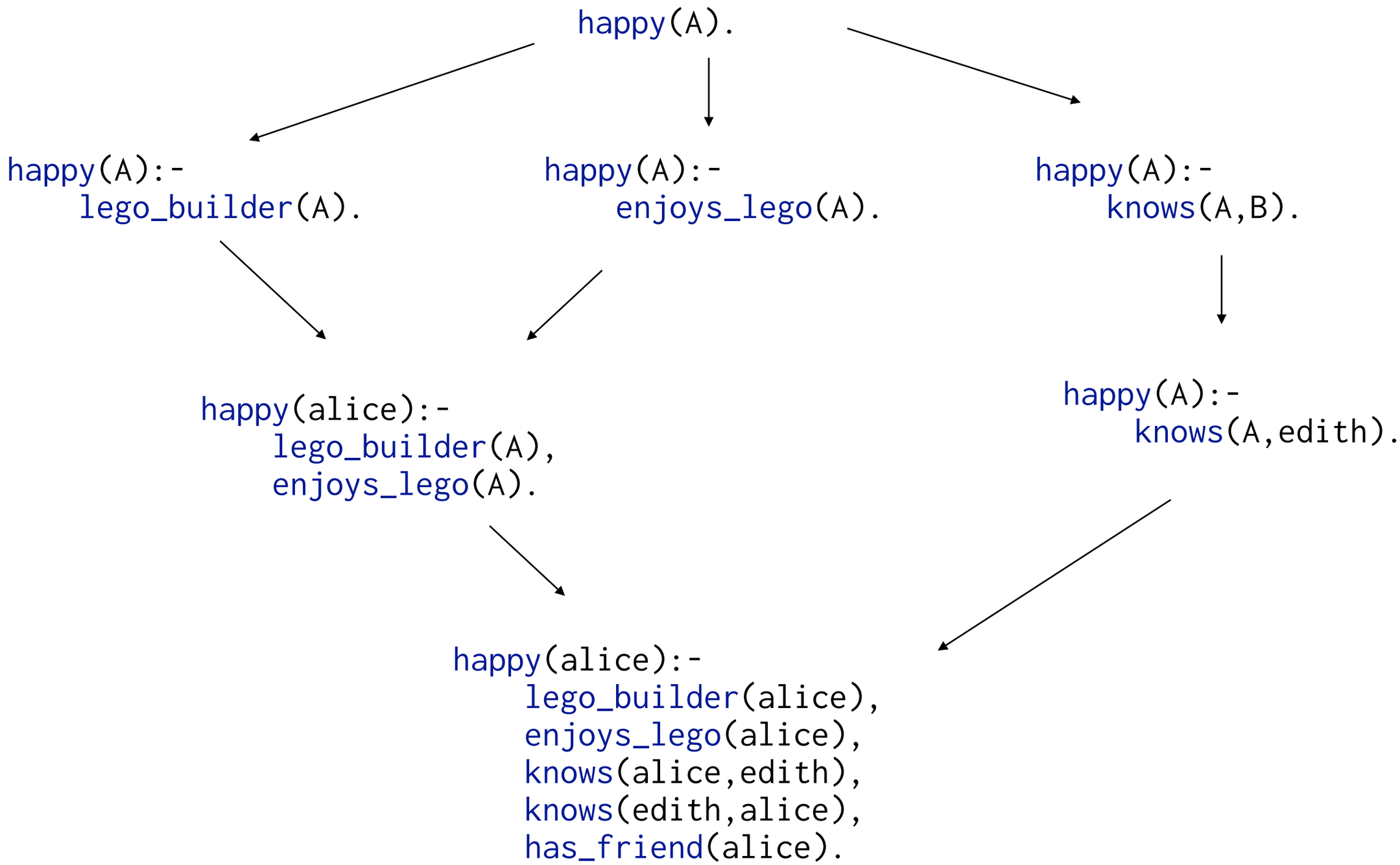
Automate Karl Popper’s falsifiability:

1. Form a hypothesis
2. Empirically try to refute it
3. If the hypothesis fails, **determine why**
4. Use the **explanation** to rule out other hypotheses
5. Go to 1

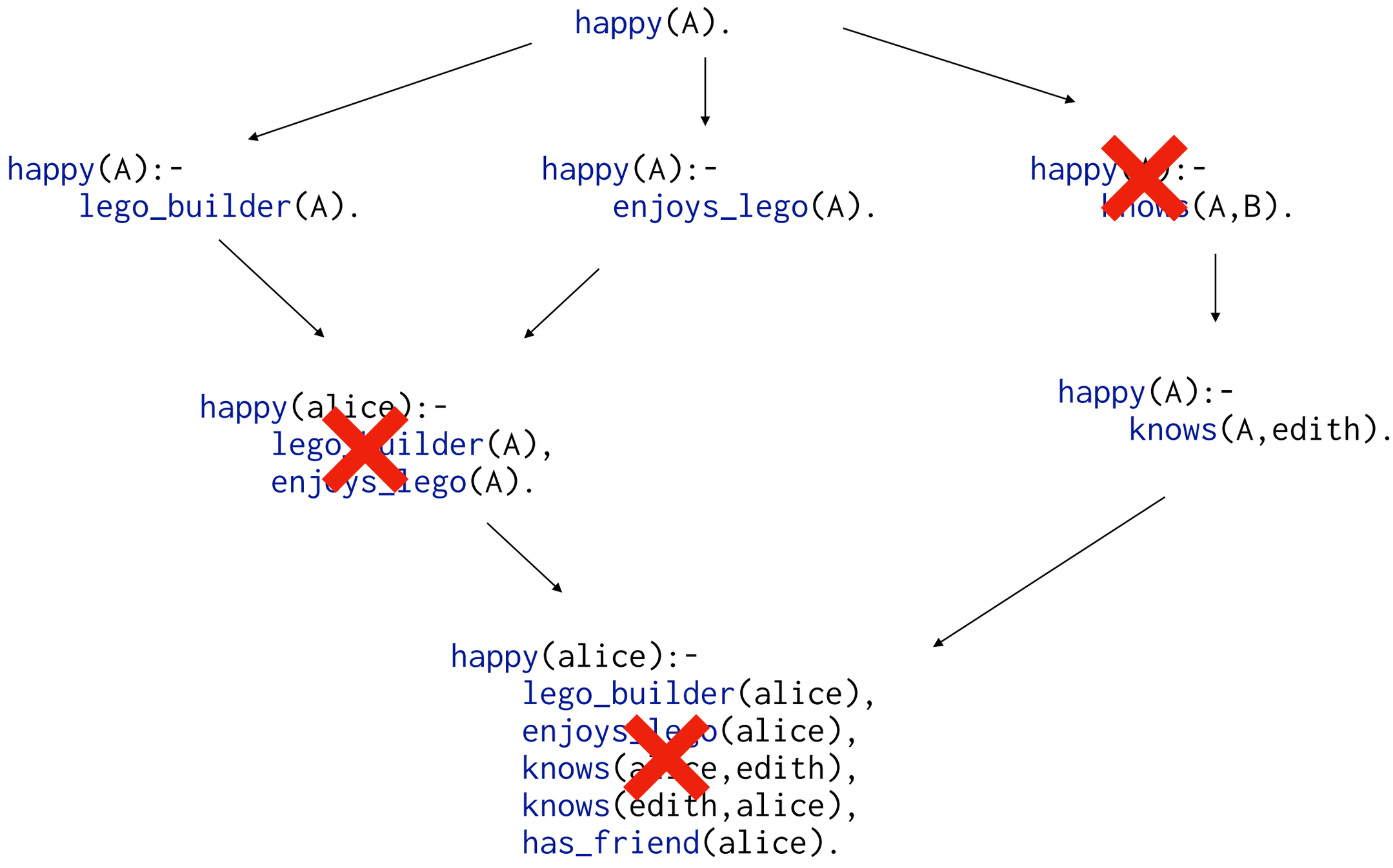
Popper



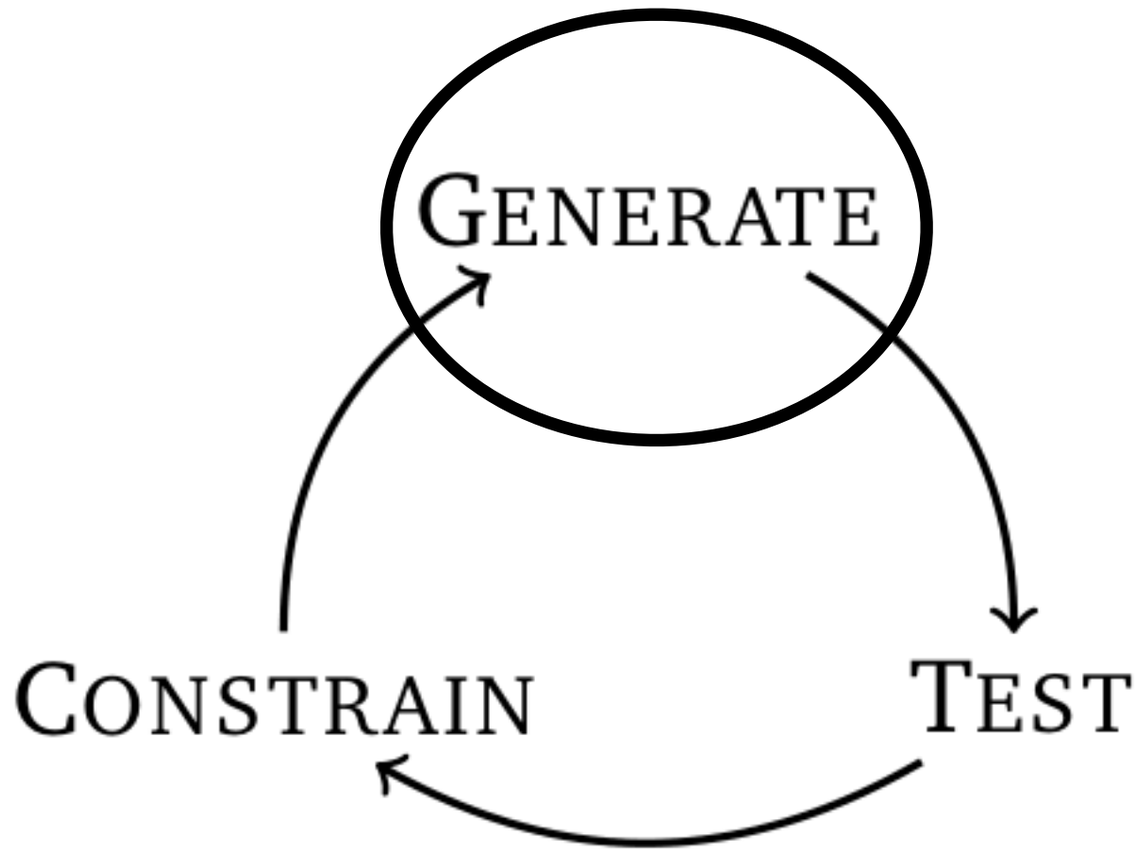
Popper


happy(A).
happy(A):-
lego_builder(A).
happy(A):-
enjoys_lego(A).
happy(A):-
knows(A,B).
happy(A):-
knows(A,edith).
happy(alice):-
lego_builder(A),
enjoys_lego(A).
happy(alice):-
lego_builder(alice),
enjoys_lego(alice),
knows(alice,edith),
knows(edith,alice),
has_friend(alice).

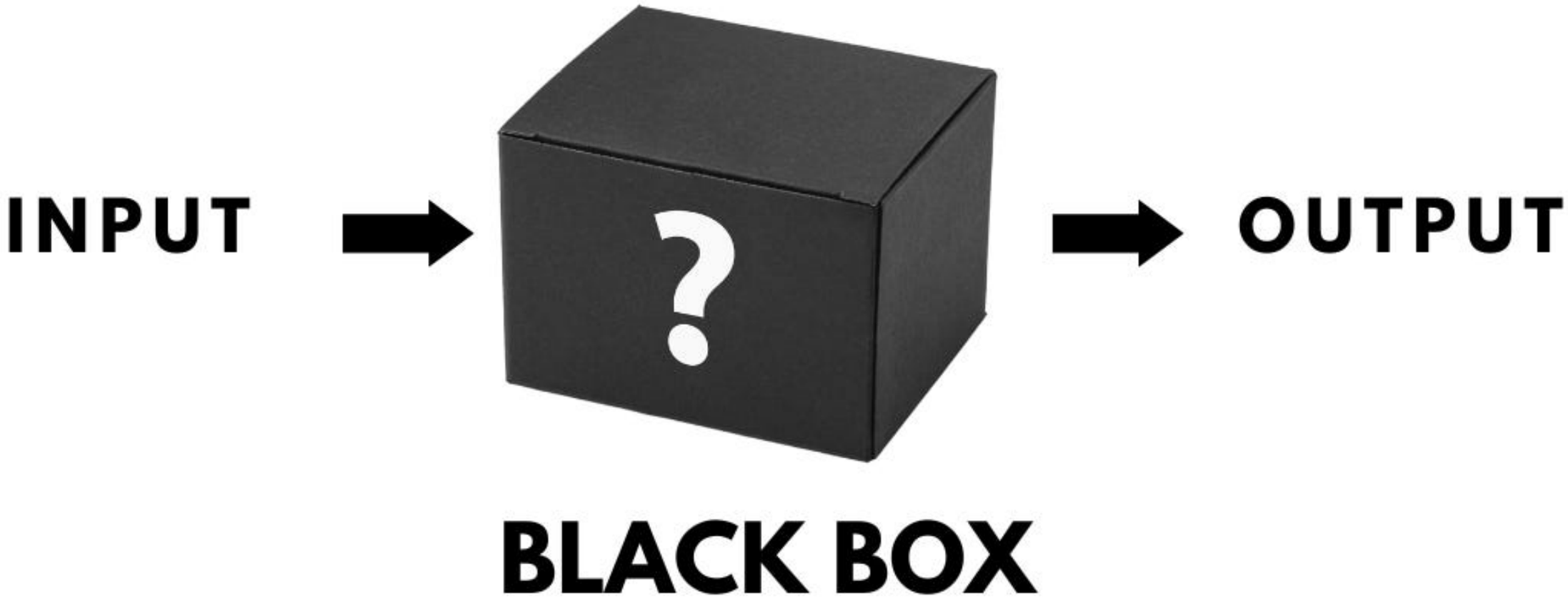
Popper


happy(A).
happy(A):-
lego_builder(A).
happy(A):-
enjoys_lego(A).
happy(A):-
knows(A,B).
happy(A):-
knows(A,edith).
happy(alice):-
lego_builder(A),
enjoys_lego(A).
happy(alice):-
lego_builder(alice),
enjoys_lego(alice),
knows(alice,edith),
knows(edith,alice),
has_friend(alice).

Popper



How to generate a hypothesis?



Black box

SAT problem: a formula where every model represents a different hypothesis

Black box

```
happy(A):-???
```

Black box

L1 = lego_builder(A)

L2 = lego_builder(B)

L3 = enjoys_lego(A)

L4 = enjoys_lego(B)

L5 = knows(A,B)

L6 = knows(B,A)

…

```
happy(A):-???
```

Black box

L1 = lego_builder(A)
L2 = lego_builder(B)
L3 = enjoys_lego(A)
L4 = enjoys_lego(B)
L5 = knows(A,B)
L6 = knows(B,A)
…

F = L1 v L2 v L3 v L4 v L5 v L6

happy(A):-???

Black box

L1 = lego_builder(A)
L2 = lego_builder(B)
L3 = enjoys_lego(A)
L4 = enjoys_lego(B)
L5 = knows(A,B)
L6 = knows(B,A)
…

F = L1 v L2 v L3 v L4 v L5 v L6

{L1}

```
happy(A):- lego_builder(A)
```

Black box

L1 = lego_builder(A)
L2 = lego_builder(B)
L3 = enjoys_lego(A)
L4 = enjoys_lego(B)
L5 = knows(A,B)
L6 = knows(B,A)
…

F = L1 v L2 v L3 v L4 v L5 v L6

{L2}

```
happy(A):- lego_builder(B)
```

Black box

L1 = lego_builder(A)
L2 = lego_builder(B)
L3 = enjoys_lego(A)
L4 = enjoys_lego(B)
L5 = knows(A,B)
L6 = knows(B,A)

…

F = L1 v L2 v L3 v L4 v L5 v L6

{L2}

Black box

L1 = lego_builder(A)
L2 = lego_builder(B)
L3 = enjoys_lego(A)
L4 = enjoys_lego(B)
L5 = knows(A,B)
L6 = knows(B,A)
…

F =
(L1 v L2 v L3 v L4 v L5 v L6)
^
(L1 v L3 v L5 v L6)

The variable **A** must be in the body

{L2}

```
happy(A):-lego_builder(B)
```

Black box

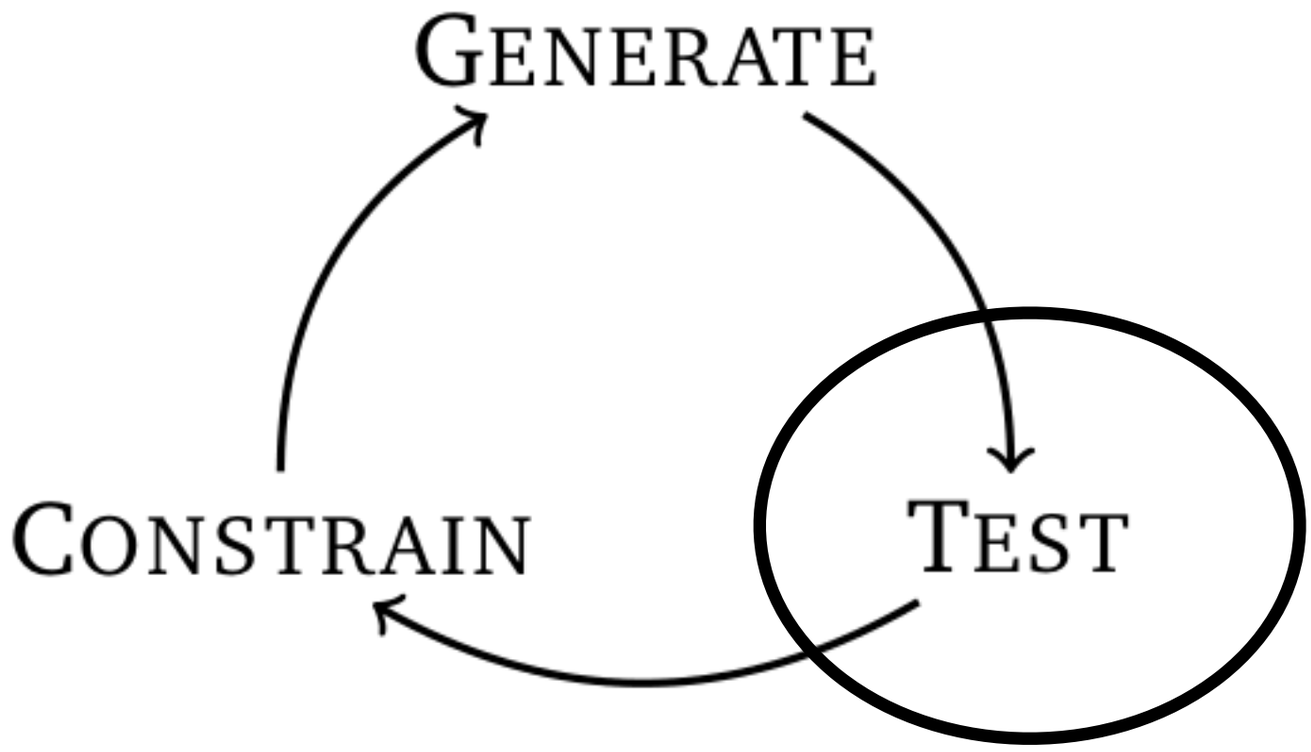
L1 = lego_builder(A)
L2 = lego_builder(B)
L3 = enjoys_lego(A)
L4 = enjoys_lego(B)
L5 = knows(A,B)
L6 = knows(B,A)
…

F =
(L1 v L2 v L3 v L4 v L5 v L6)
^
(L1 v L3 v L5 v L6)

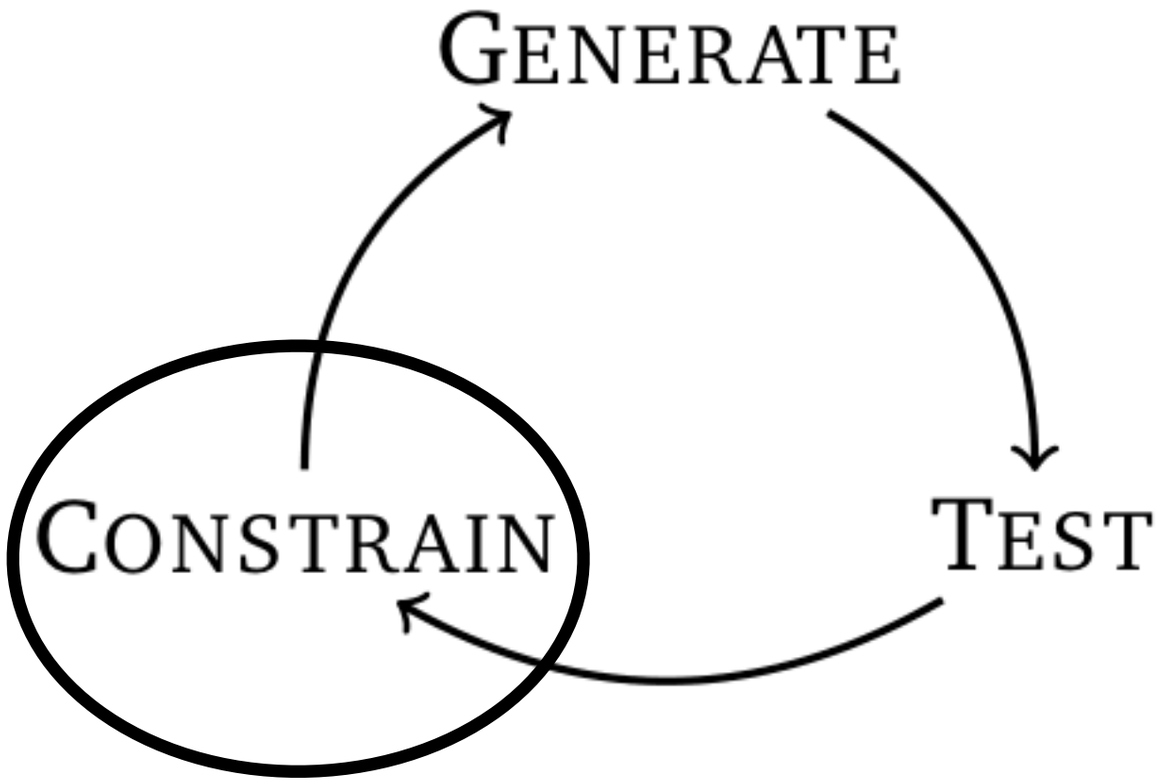
{L1, L5, L4}

```
happy(A):- lego_builder(A), knows(A,B), enjoys_lego(B)
```

Popper



Popper



Constrain

Adding a clause/constraint prunes at least one model/ hypothesis

Failure explanation

If a hypothesis is too specific we prune all hypotheses that syntactically specialise it

Failure explanation

If a hypothesis is too specific we prune all hypotheses that syntactically specialise it

```
{L1, L2, L4}
```

```
happy(A):- lego_builder(A), lego_builder(B), knows(A,B)
```

Failure explanation

If a hypothesis is too specific we prune all hypotheses that syntactically specialise it

{L1, L2, L4}

```
happy(A):- lego_builder(A), lego_builder(B), knows(A,B)
```

not (L1 ^ L2 ^ L4)

Failure explanation

If a hypothesis is too specific we prune all hypotheses that syntactically specialise it

{L1, L2, L4}

```
happy(A):- lego_builder(A), lego_builder(B), knows(A,B)
```

(not L1 v not L2 v not L4)

Failure explanation

If a hypothesis is too specific we prune all hypotheses that syntactically specialise it

F =

(L1 v L2 v L3 v L4 v L5 v L6)

^

(L1 v L3 v L5 v L6)

^

(not L1 v not L2 v not L4)

Failure explanation

If a hypothesis is too general we prune all hypotheses that syntactically generalise it

Failure explanation

```
f(A,B):-
    tail(A,C).
    head(C,B),
    odd(B),
    even(B).
```

Failure explanation

```
f(A,B):-
    tail(A,C).
    head(C,B),
    odd(B),
    even(B).
```

Explanation

```
:-
odd(B),
even(B).
```

Failure explanation

Explanation

```
f(A,B):-                          :-
    tail(A,C).                    odd(B),
    head(C,B),                    even(B).
    odd(B),
    even(B).
```

Prune all hypotheses that contain this pair of literals

MLJ 2023, ECAI 2024

Popper

Recursion [MLJ21]

Predicate invention [IJCAI16, AAAI24]

MDL programs [AAAI24]

Many rules [ECAI23]

Big rules [IJCAI24]

Numerical reasoning [MLJ23, AAAI23]

Popper

Recursion [MLJ21]

Predicate invention [IJCAI16, AAAI24]

MDL programs [AAAI24]

Many rules [ECAI23]

Big rules [IJCAI24]

Numerical reasoning [MLJ23, AAAI23]

Recursion

[MLJ21]


Edgware Road
Marylebone
St John's Wood
Euston
Edgware Road
Baker Street
Great Portland Street
Euston Square
Warren Street
Regent's Park
Goodge Street
Russell Square
Bayswater
Bond Street
Oxford Circus
Marble Arch
Tottenham Court Road
Holborn
Notting Hill Gate
Queensway
Lancaster Gate
Covent Garden
Green Park
Leicester Square
Hyde Park Corner
Piccadilly Circus
High Street Kensington
Knightsbridge
Charing Cross
Gloucester Road
Victoria
Westminster
Embankment
Earl's Court
South Kensington
Sloane Square
St James's Park
Temple


```
edge(oxford_circus, bond_street).
edge(oxford_circus, piccadilly_circus).
edge(south_kensington, gloucester_road).
```

Recursion

```
connected(A,B):-?
```

Recursion

```
connected(A,B):- edge(A,B).
```

Recursion

```
connected(A,B):- edge(A,B).
connected(A,B):- edge(A,C),edge(C,B).
```

Recursion

```
connected(A,B):- edge(A,B).
connected(A,B):- edge(A,C),edge(C,B).
connected(A,B):- edge(A,C),edge(C,D),edge(D,B).
```

Recursion

```
connected(A,B):- edge(A,B).
connected(A,B):- edge(A,C),edge(C,B).
connected(A,B):- edge(A,C),edge(C,D),edge(D,B).
connected(A,B):- edge(A,C),edge(C,D),edge(D,E),edge(E,B).
```

Recursion

```
connected(A,B):- edge(A,B).
connected(A,B):- edge(A,C),edge(C,B).
connected(A,B):- edge(A,C),edge(C,D),edge(D,B).
connected(A,B):- edge(A,C),edge(C,D),edge(D,E),edge(E,B).
```

Cannot generalise to arbitrary depth

Difficult to learn because of its size

Recursion

```
connected(A,B):- edge(A,B).
```

Recursion

```
connected(A,B):- edge(A,B).
connected(A,B):- edge(A,C),connected(C,B).
```

Recursion

```
connected(A,B):- edge(A,B).
connected(A,B):- edge(A,C),connected(C,B).
```

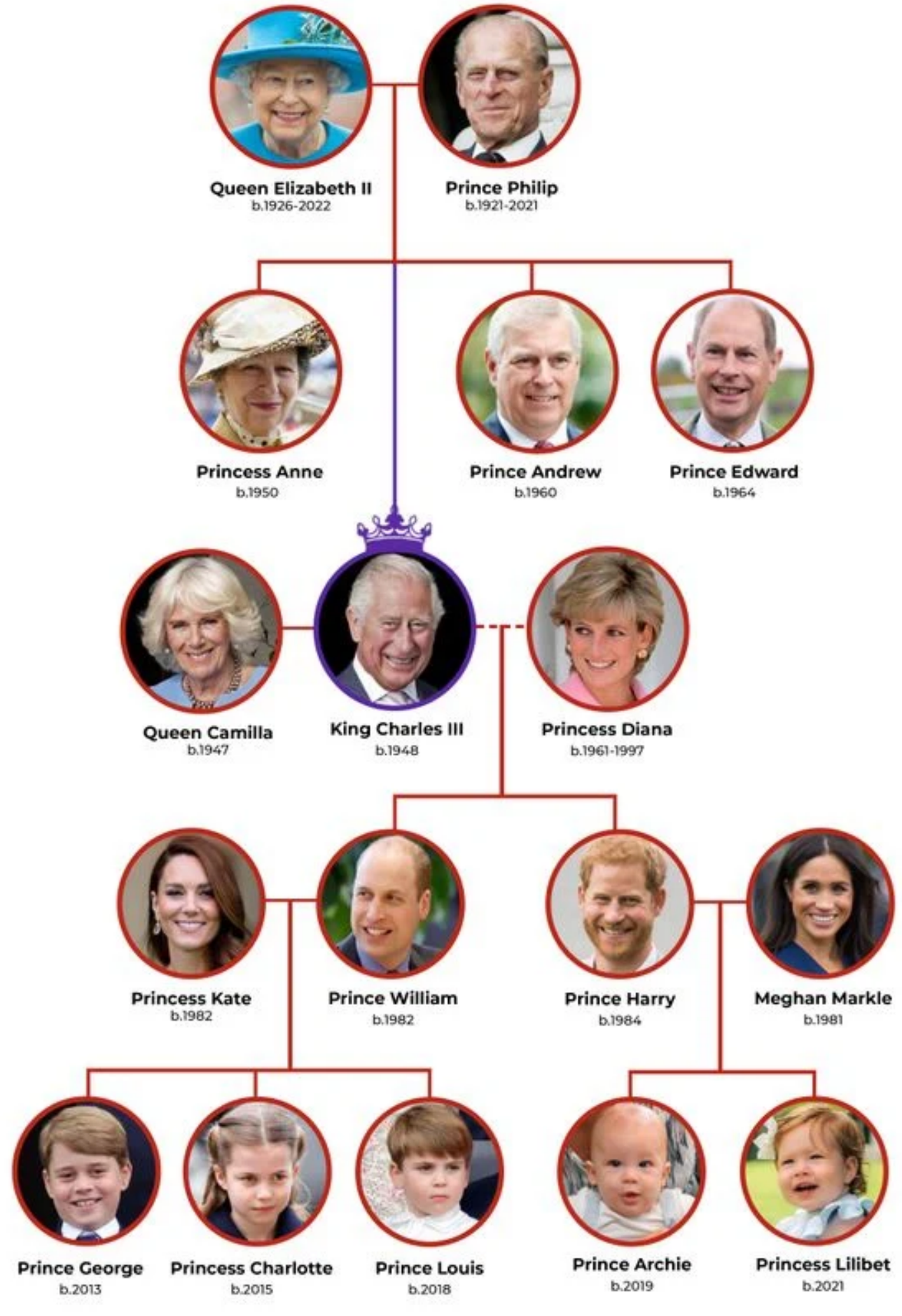
Easier to learn because of its size

Need fewer examples

Predicate invention

Automatically invent new symbols

[IJCAI16, MLJ20, AAAI24, IJCAI24]


Queen Elizabeth II
b.1926-2022
Prince Philip
b.1921-2021
Princess Anne
b.1950
Prince Andrew
b.1960
Prince Edward
b.1964
Queen Camilla
b.1947
King Charles III
b.1948
Princess Diana
b.1961-1997
Princess Kate
b.1982
Prince William
b.1982
Prince Harry
b.1984
Meghan Markle
b.1981
Prince George
b.2013
Princess Charlotte
b.2015
Prince Louis
b.2018
Prince Archie
b.2019
Princess Lilibet
b.2021

Predicate invention

```
greatgrandparent(A,B):- mother(A,C),mother(C,D),mother(D,B).
```

Predicate invention

```
greatgrandparent(A,B):- mother(A,C),mother(C,D),mother(D,B).
greatgrandparent(A,B):- mother(A,C),mother(C,D),father(D,B).
```

Predicate invention

```
greatgrandparent(A,B):- mother(A,C),mother(C,D),mother(D,B).
greatgrandparent(A,B):- mother(A,C),mother(C,D),father(D,B).
greatgrandparent(A,B):- mother(A,C),father(C,D),mother(D,B).
```

Predicate invention

```
greatgrandparent(A,B):- mother(A,C),mother(C,D),mother(D,B).
greatgrandparent(A,B):- mother(A,C),mother(C,D),father(D,B).
greatgrandparent(A,B):- mother(A,C),father(C,D),mother(D,B).
greatgrandparent(A,B):- mother(A,C),father(C,D),father(D,B).
greatgrandparent(A,B):- father(A,C),father(C,D),father(D,B).
greatgrandparent(A,B):- father(A,C),father(C,D),mother(D,B).
greatgrandparent(A,B):- father(A,C),mother(C,D),father(D,B).
greatgrandparent(A,B):- father(A,C),mother(C,D),mother(D,B).
```

Predicate invention

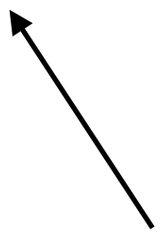
```
greatgrandparent(A,B):- mother(A,C),mother(C,D),mother(D,B).
greatgrandparent(A,B):- mother(A,C),mother(C,D),father(D,B).
greatgrandparent(A,B):- mother(A,C),father(C,D),mother(D,B).
greatgrandparent(A,B):- mother(A,C),father(C,D),father(D,B).
greatgrandparent(A,B):- father(A,C),father(C,D),father(D,B).
greatgrandparent(A,B):- father(A,C),father(C,D),mother(D,B).
greatgrandparent(A,B):- father(A,C),mother(C,D),father(D,B).
greatgrandparent(A,B):- father(A,C),mother(C,D),mother(D,B).
```

Difficult to learn because of its size

Need many examples

Predicate invention

```
greatgrandparent(A,B):- inv(A,C),inv(C,D),inv(D,B).
inv(A,B):- mother(A,B).
inv(A,B):- father(A,B).
```



parent

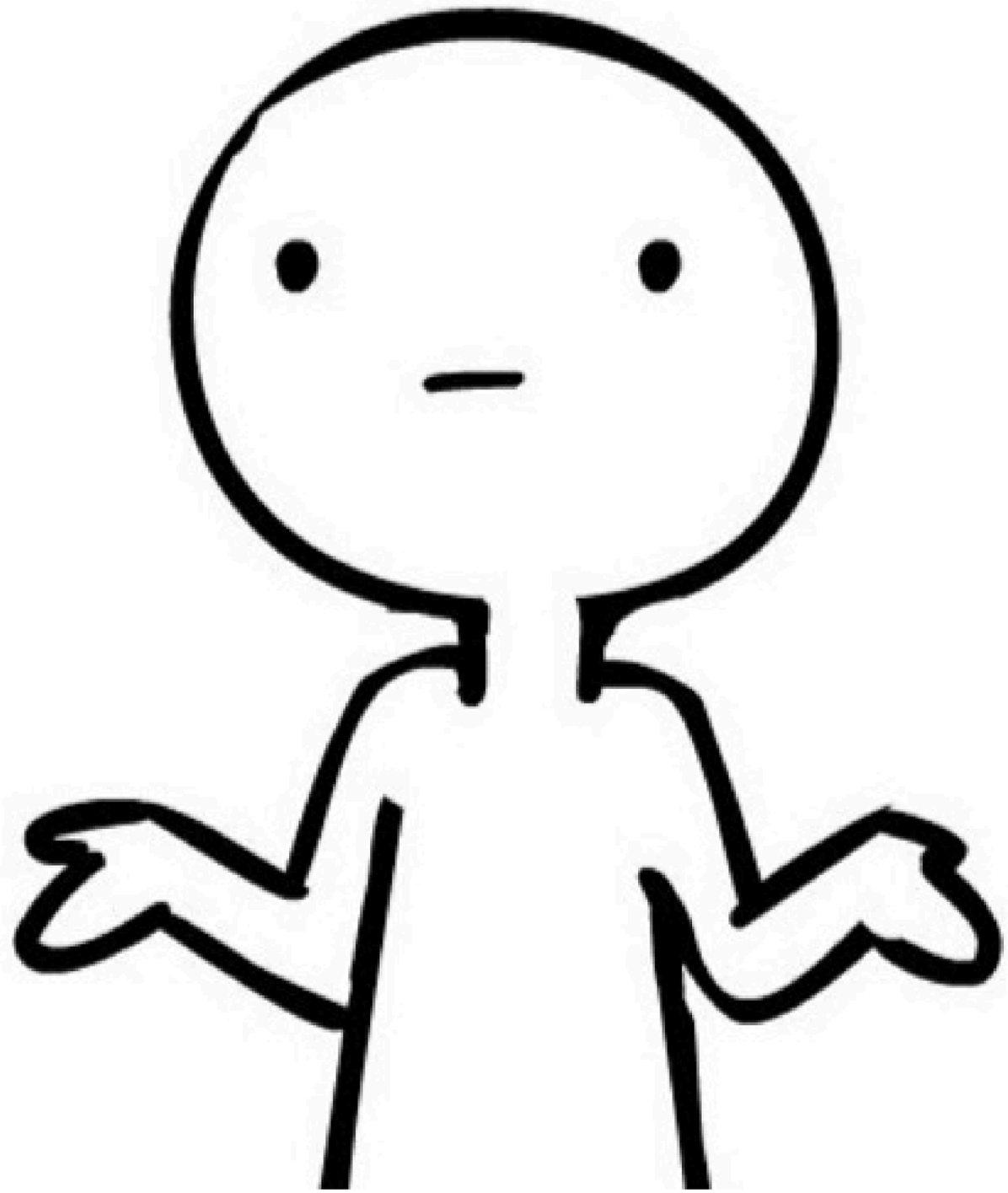
Predicate invention

```
greatgrandparent(A,B):- inv(A,C),inv(C,D),inv(D,B).
inv(A,B):- mother(A,B).
inv(A,B):- father(A,B).
```

Easier to learn because of its size

Need fewer examples

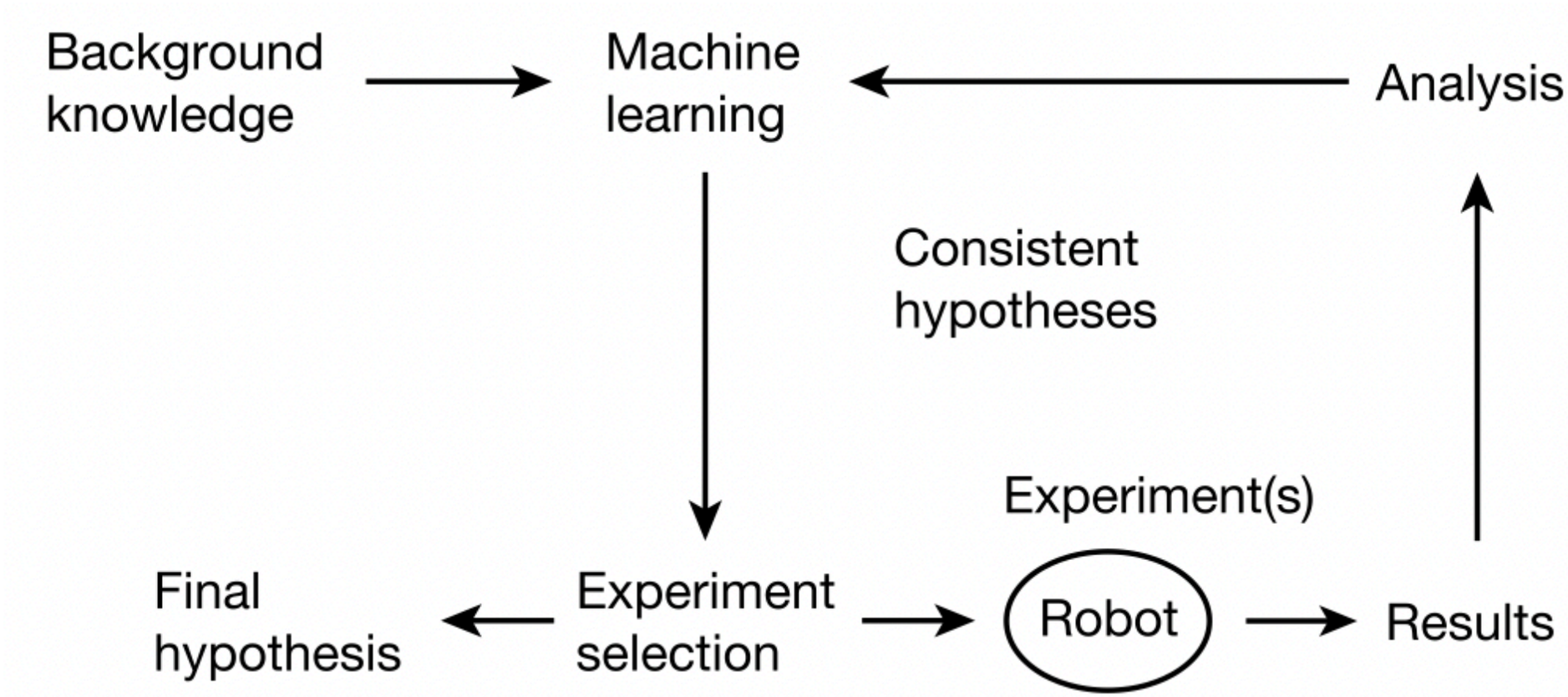
So what?



Robot scientist

King et al. Nature, 2004

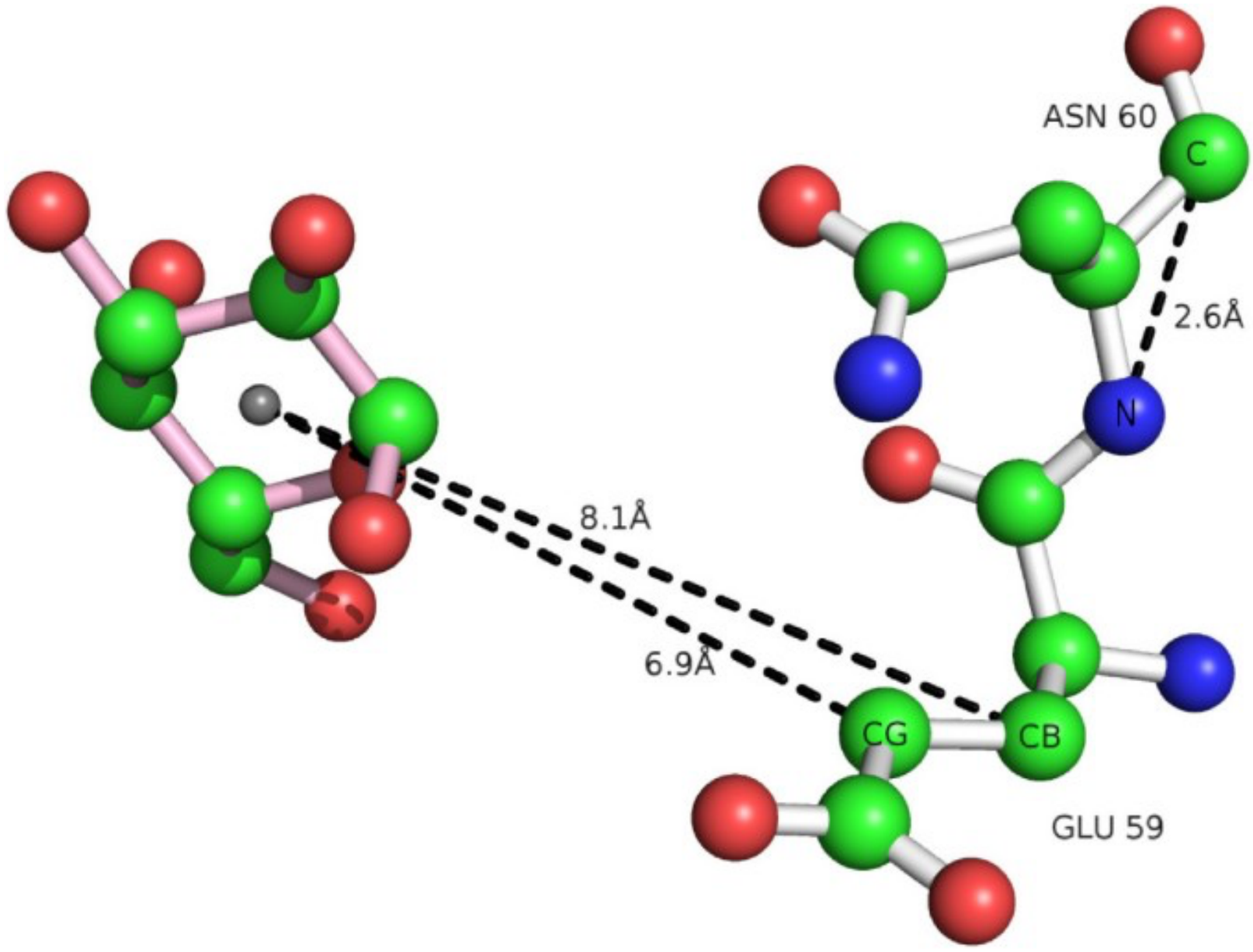
Robot scientist



Robot scientist

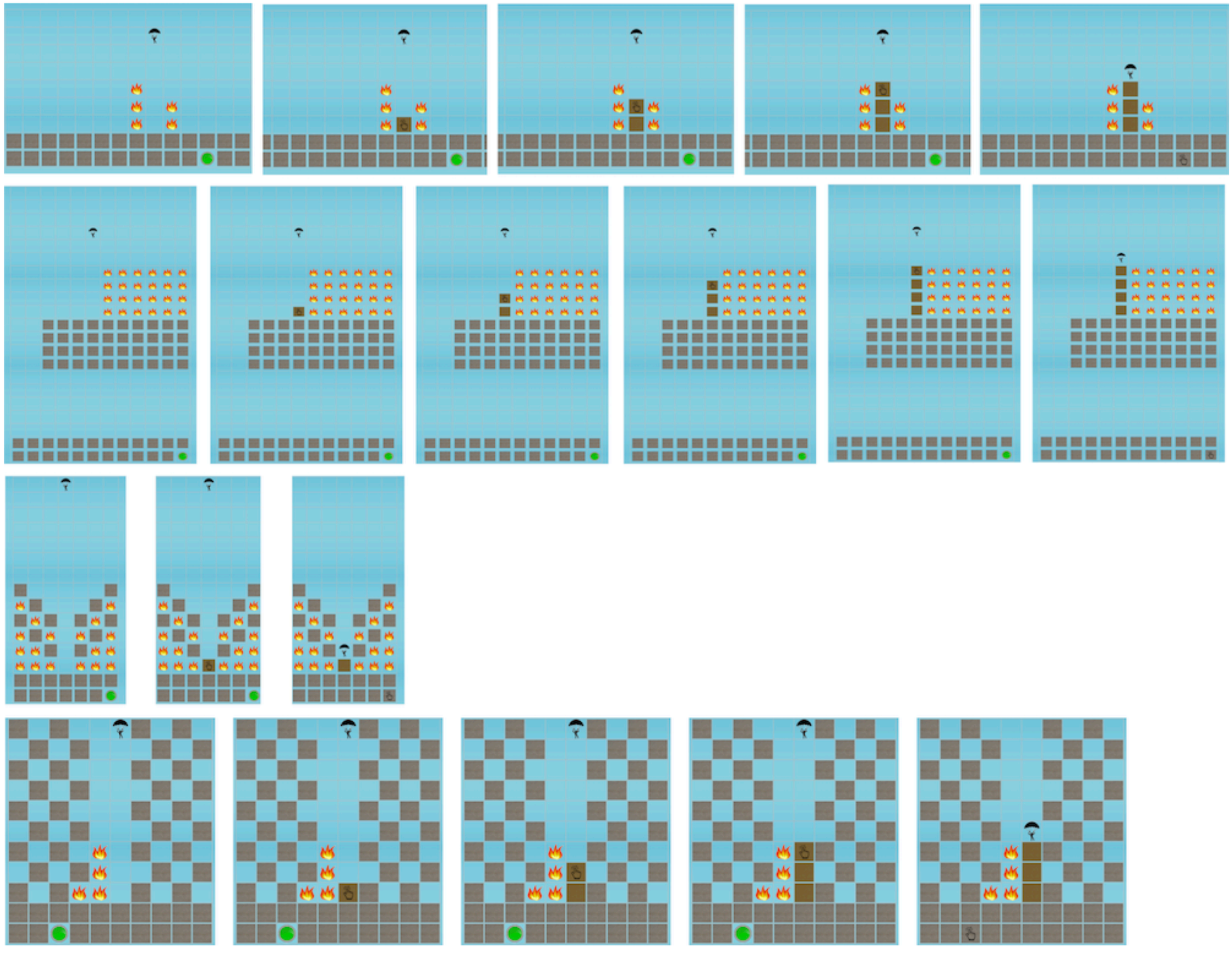
The first machine to discover new scientific knowledge independently of its human creators

Scientific discovery



Santos et al. BMC Bioinformatics, 2012

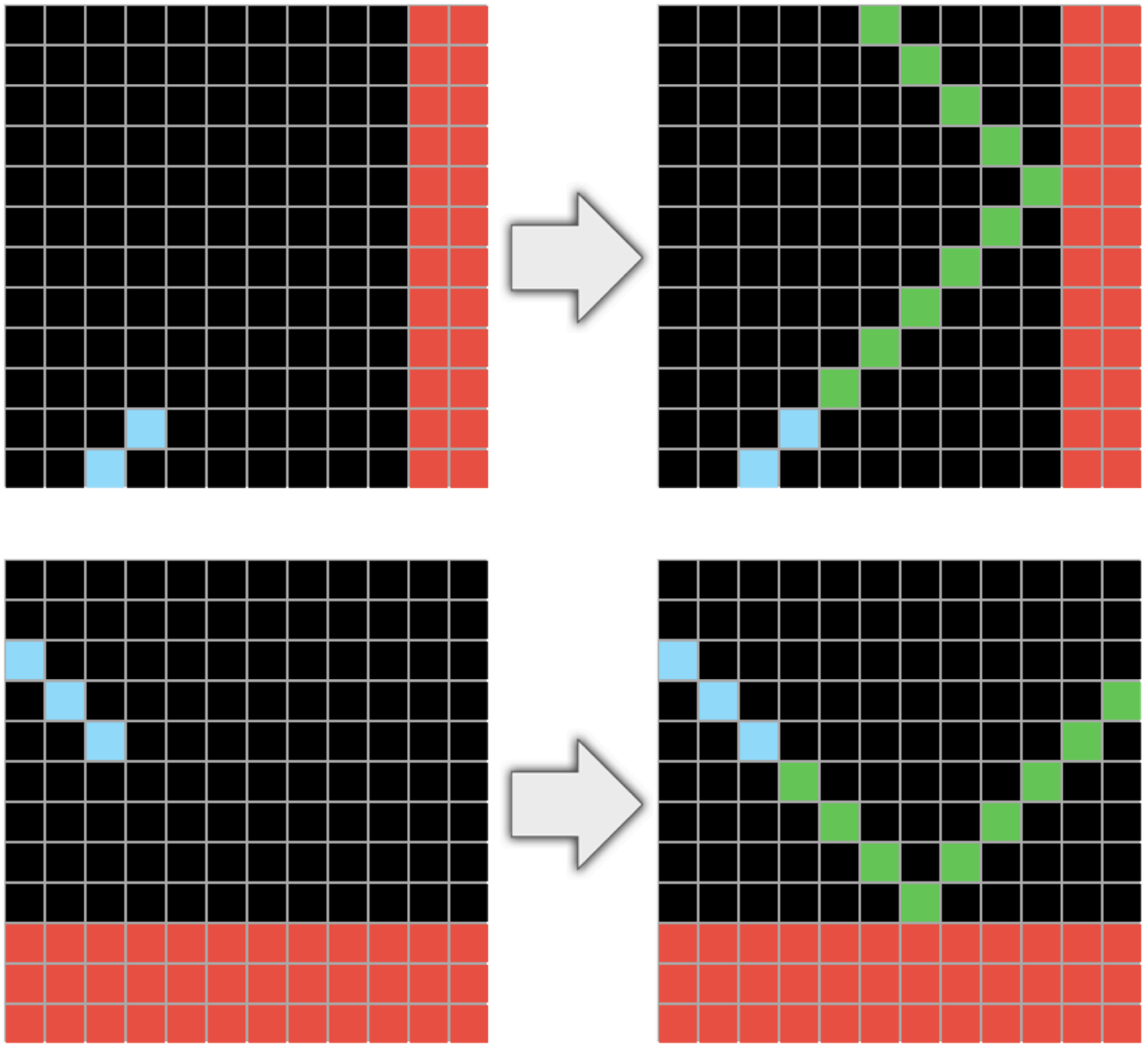
Game playing



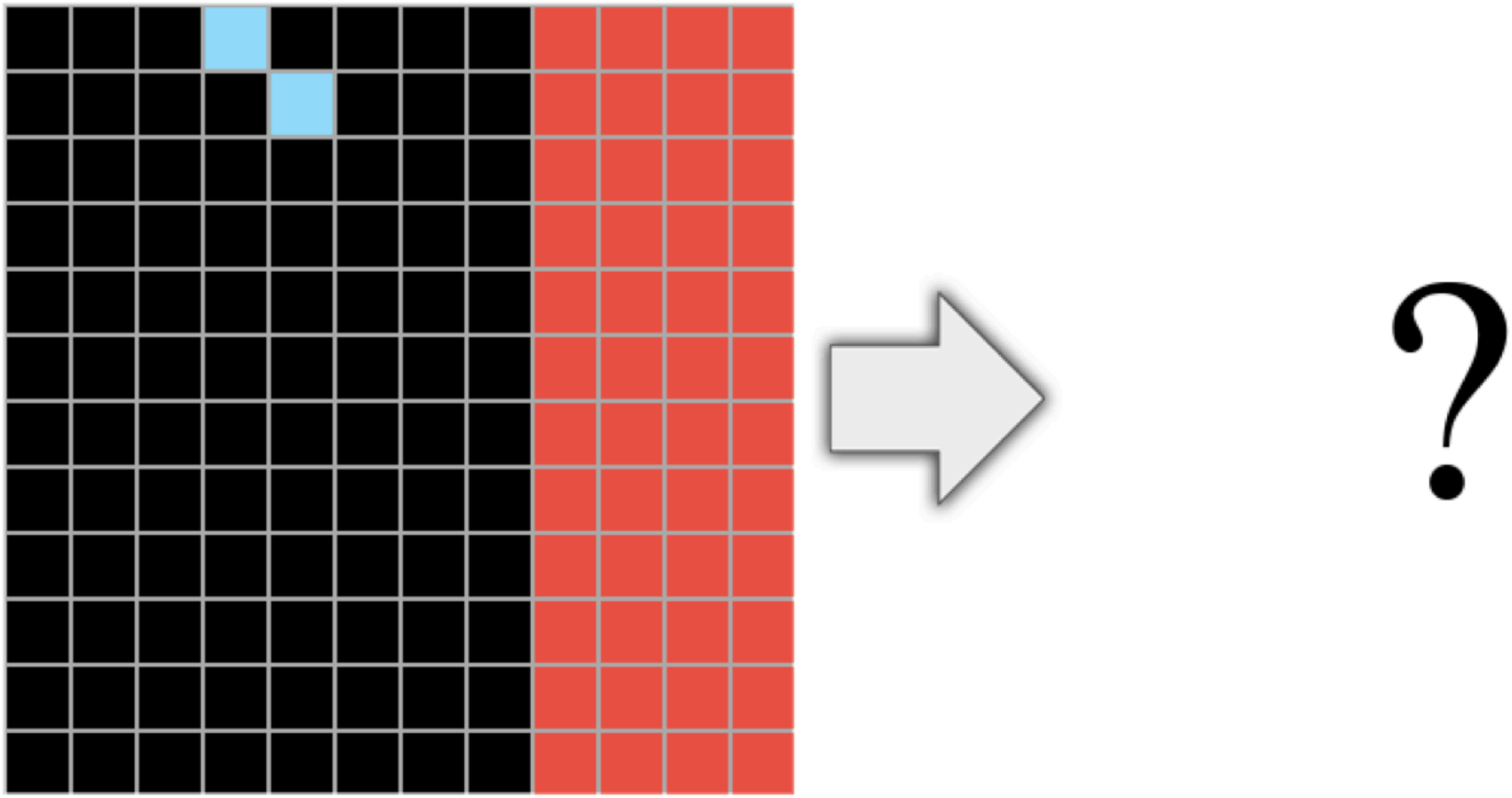
Silver et al. AAAI20

Abstraction and reasoning corpus

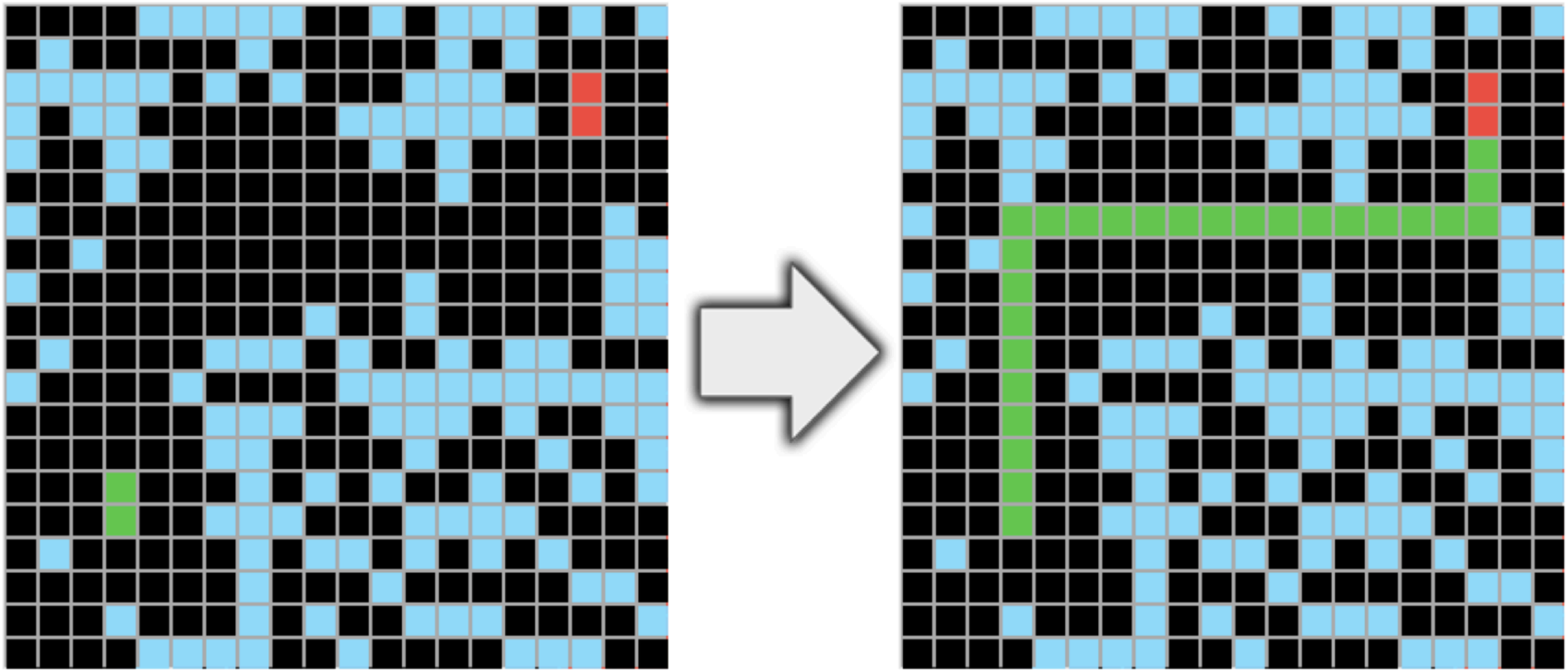
Abstraction and reasoning corpus



Abstraction and reasoning corpus



Abstraction and reasoning corpus



Abstraction and reasoning corpus

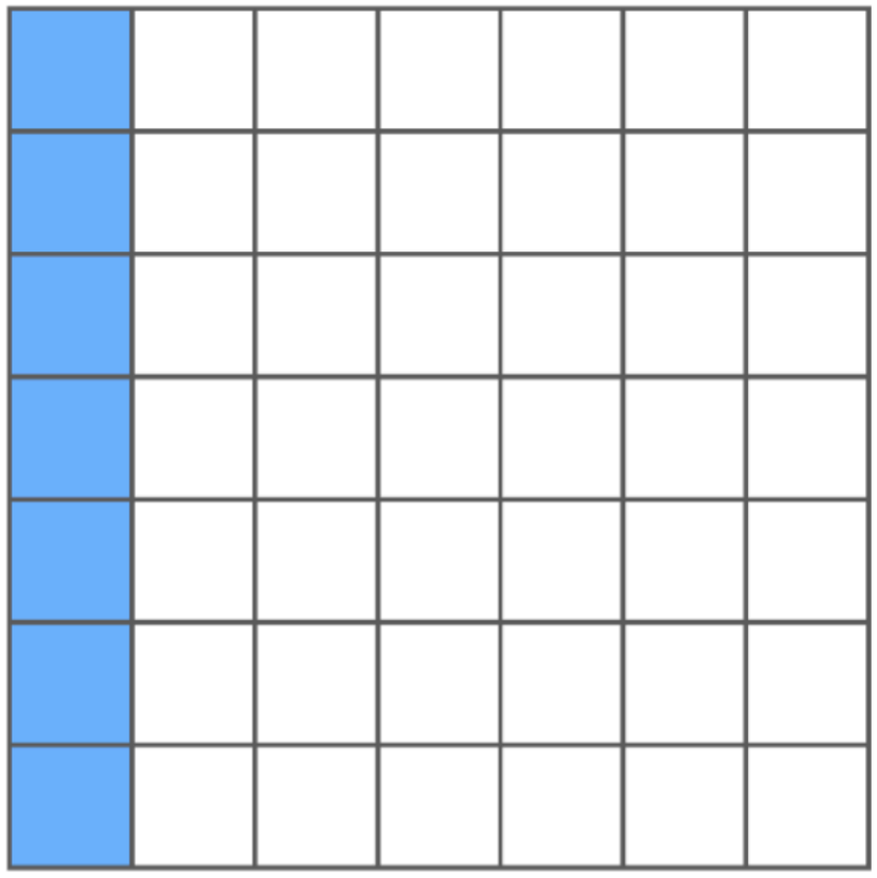
Only a few examples of every task

Solutions are programs

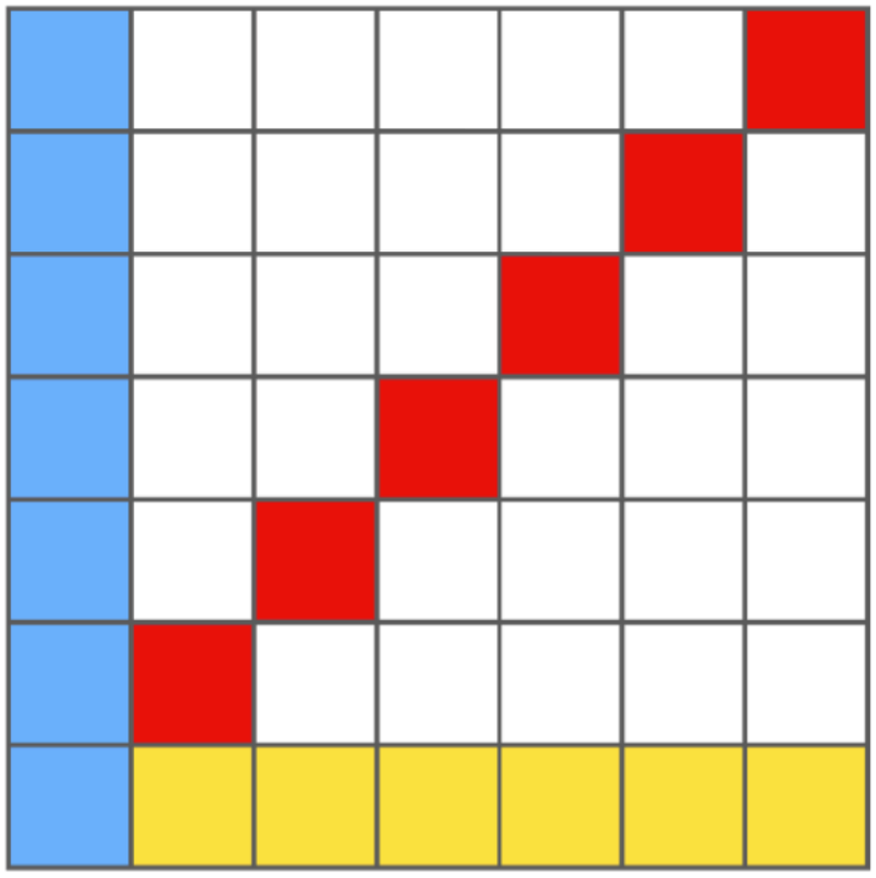
Need recursive concepts (draw line)

Abstraction and reasoning corpus

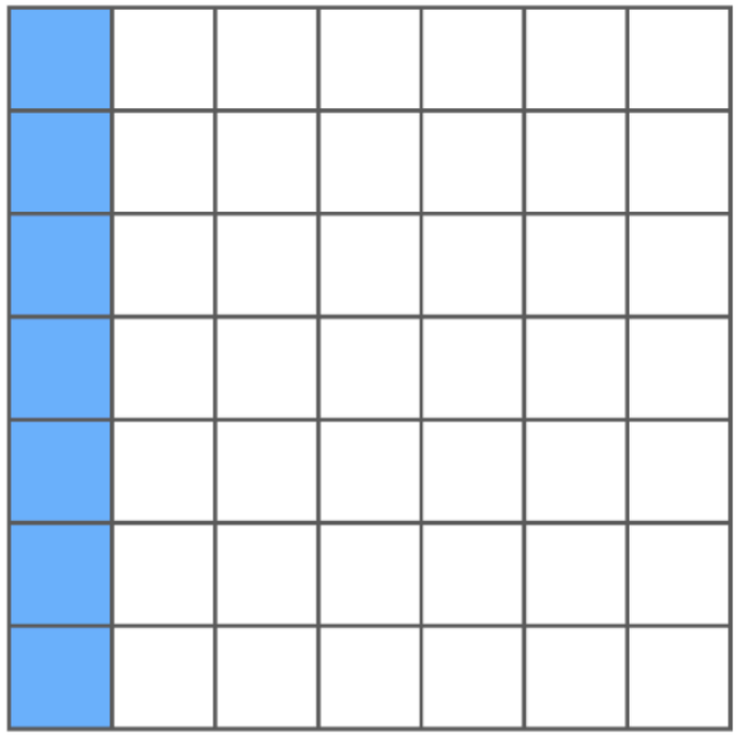
Input



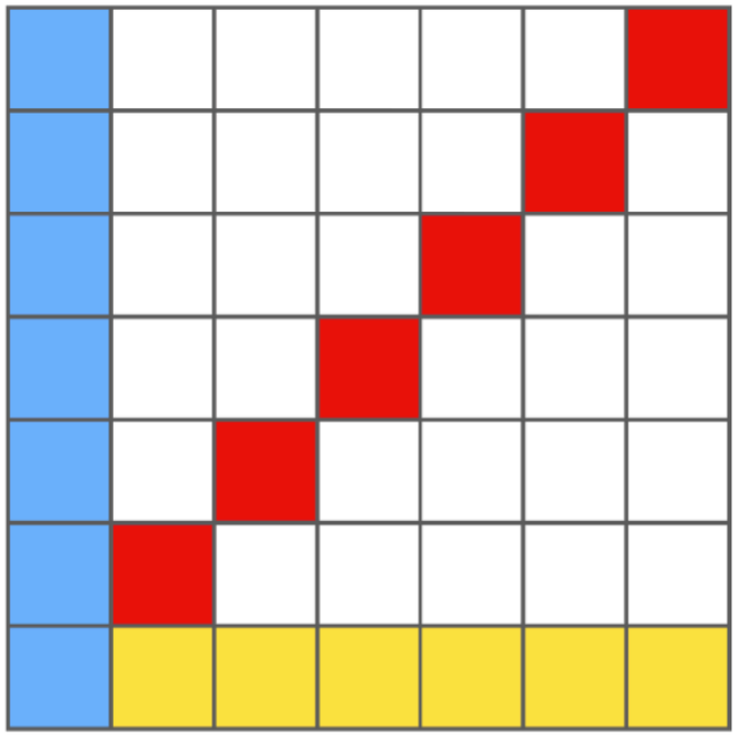
Output



Input

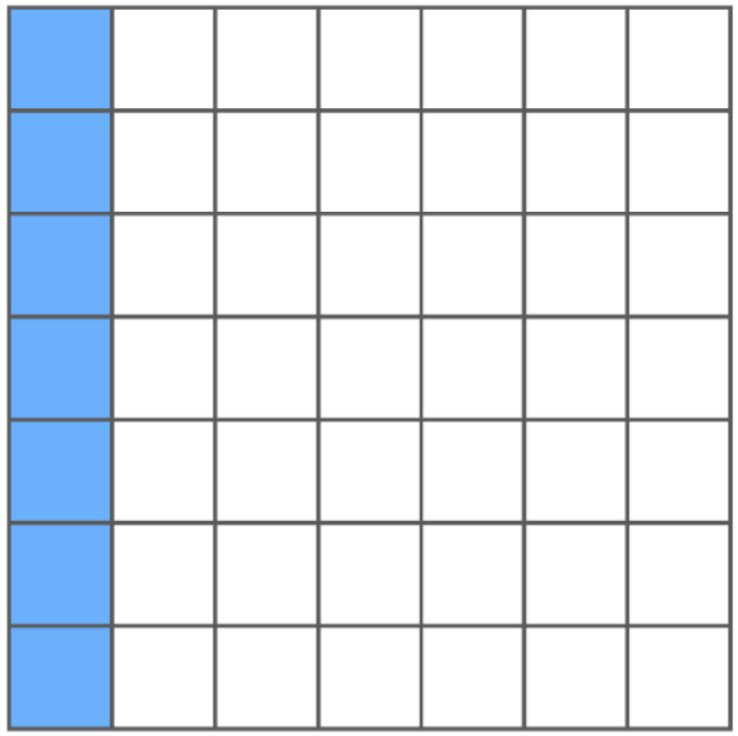


Output

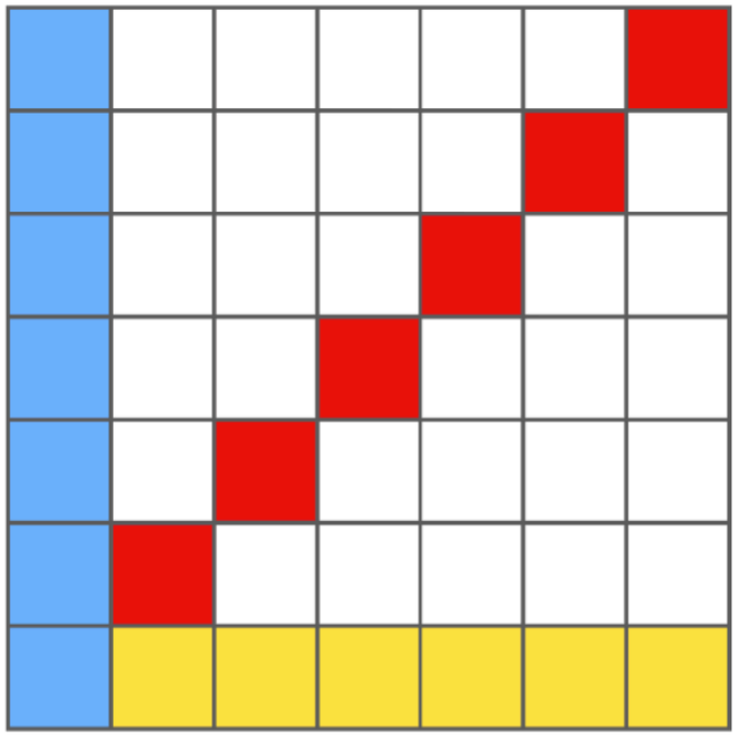


```
out(X,Y,C) :- in(X,Y,C).
out(X,Y,yellow) :- empty(X,Y), height(X).
out(X,Y,red) :- empty(X,Y), height(X+Y-1).
```

Input



Output

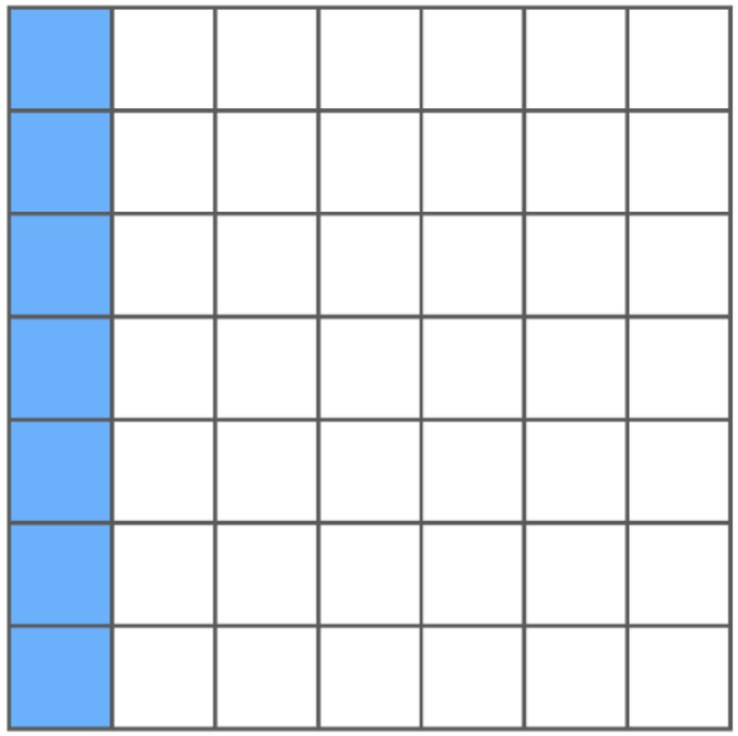


An output pixel is colour C if it is colour C in the input.

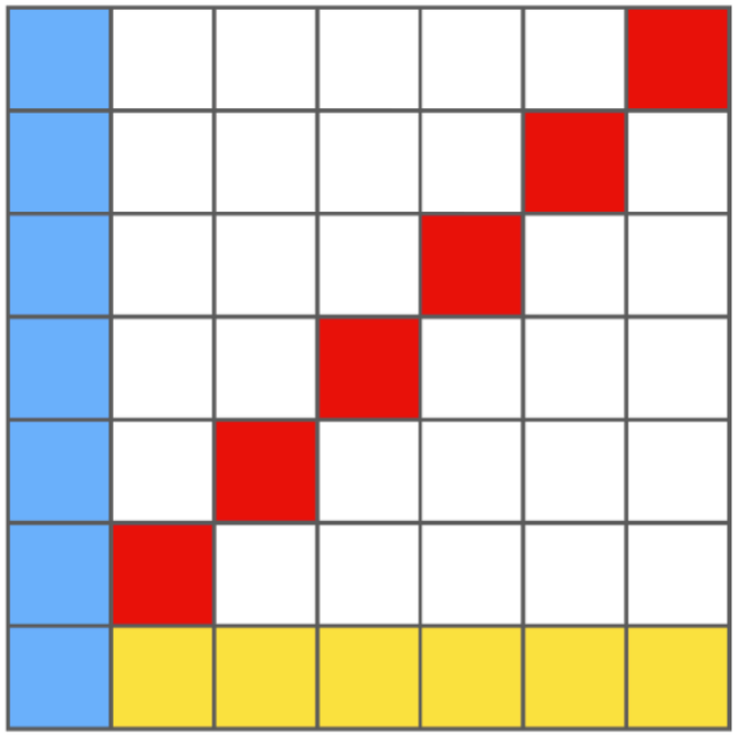
An output pixel is yellow if it is in the bottom row and empty in the input.

An output pixel is red if it is empty in the input and the sum of its coordinates X and Y equals H + 1, i.e. it is on the diagonal.

Input



Output

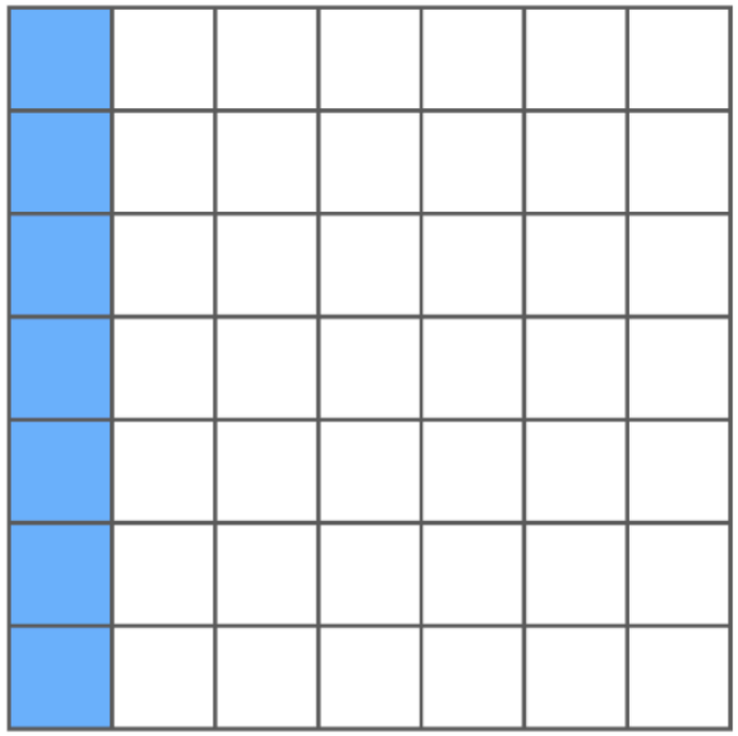


An output pixel is colour C if it is colour C in the input.

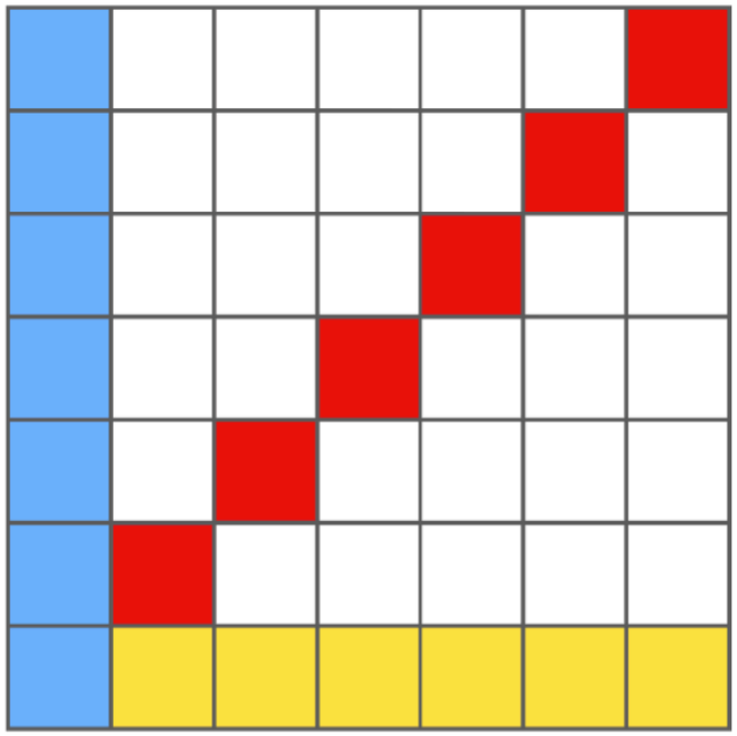
An output pixel is yellow if it is in the bottom row and empty in the input.

An output pixel is red if it is empty in the input and the sum of its coordinates X and Y equals H + 1, i.e. it is on the diagonal.

Input



Output

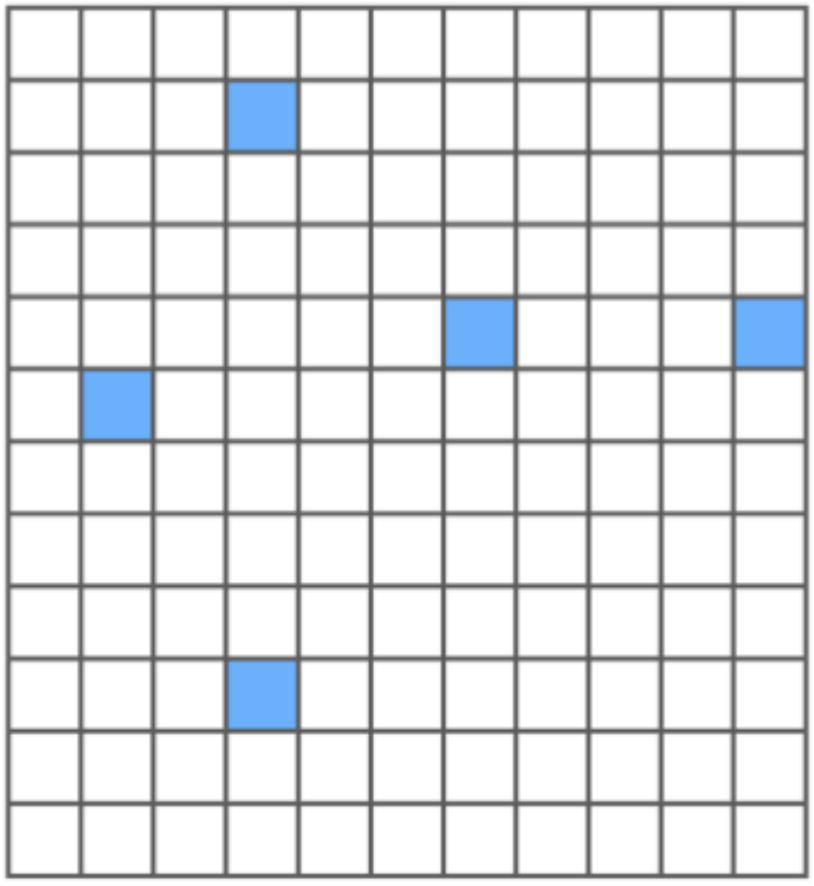


An output pixel is colour C if it is colour C in the input.

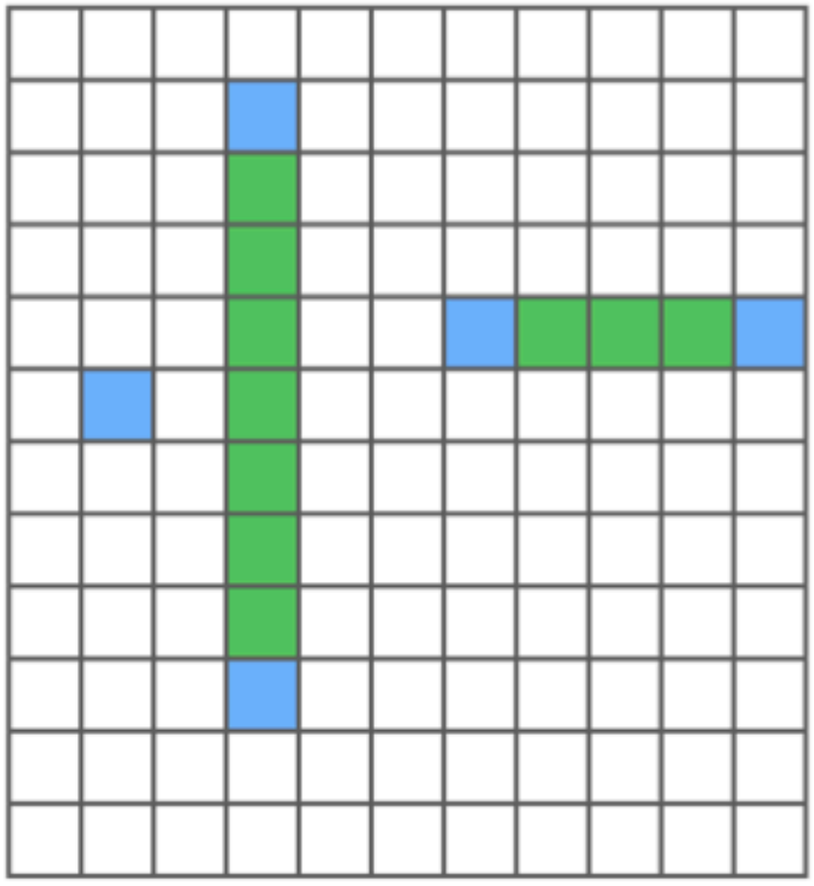
An output pixel is yellow if it is in the bottom row and empty in the input.

An output pixel is red if it is empty in the input and the sum of its coordinates X and Y equals H + 1, i.e. it is on the diagonal.

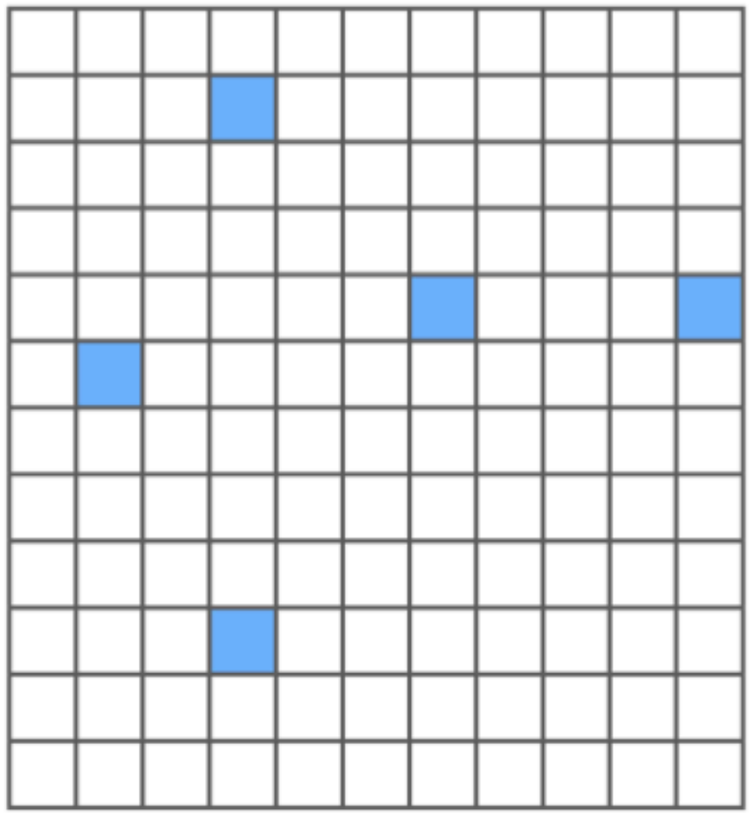
Input



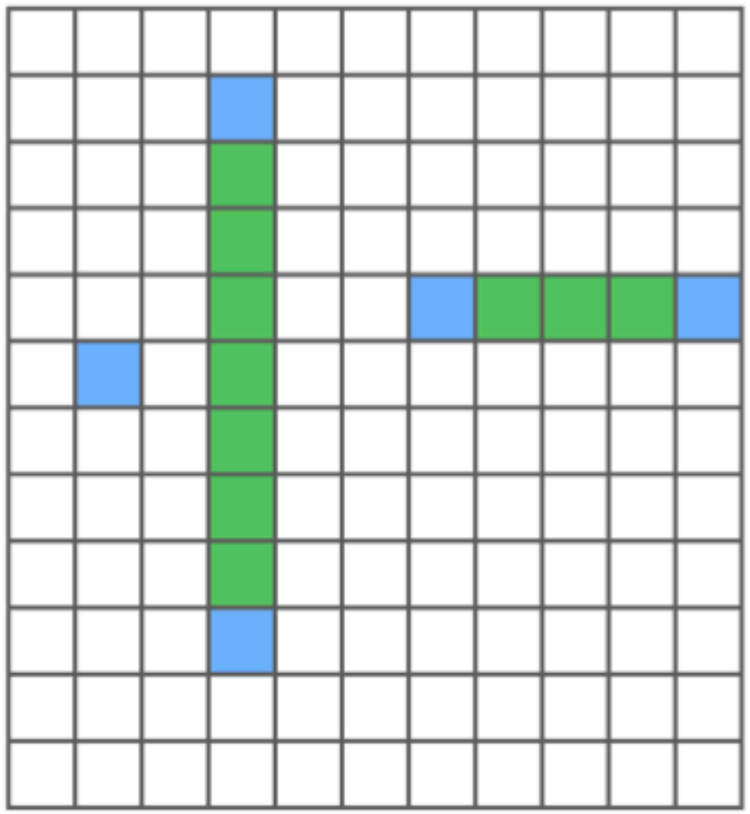
Output



Input



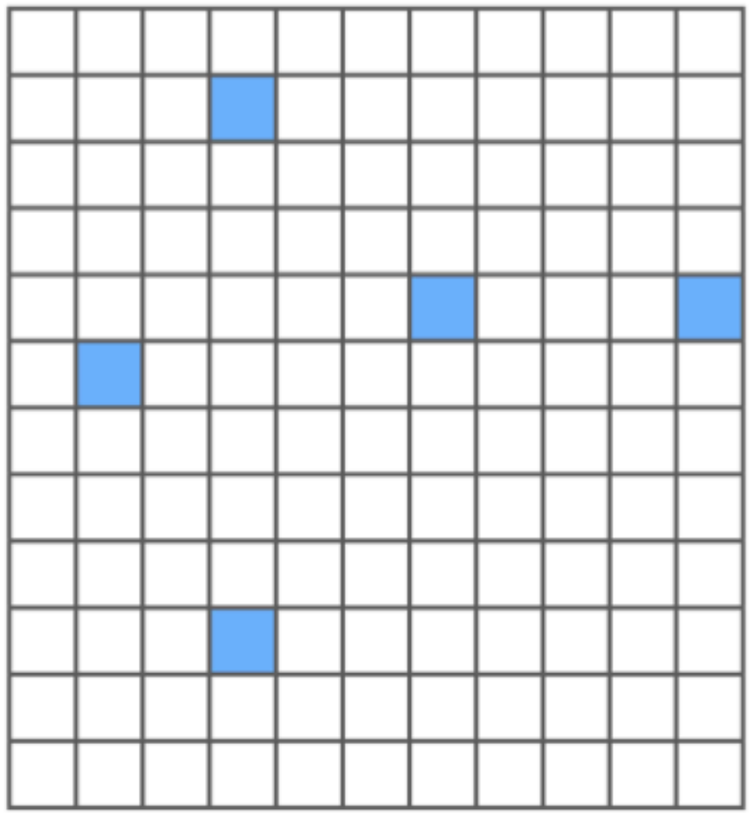
Output



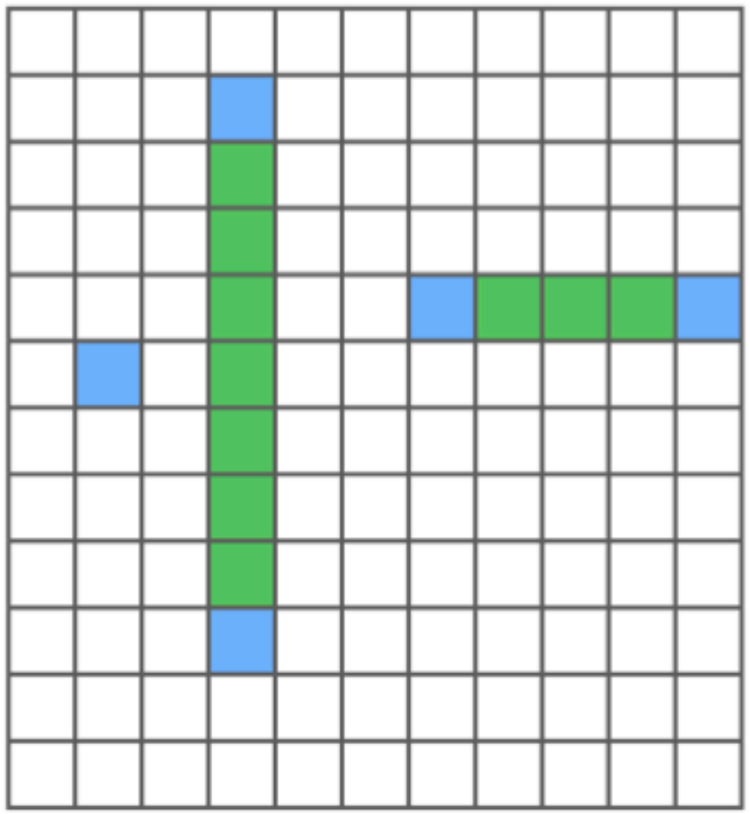
An out pixel has colour C if it is colour C in the input.

An out pixel is red if it is empty in the input and there is a pixel in the same row (X) and a pixel in the same column (Y) with the same colour (C) in the input.

Input



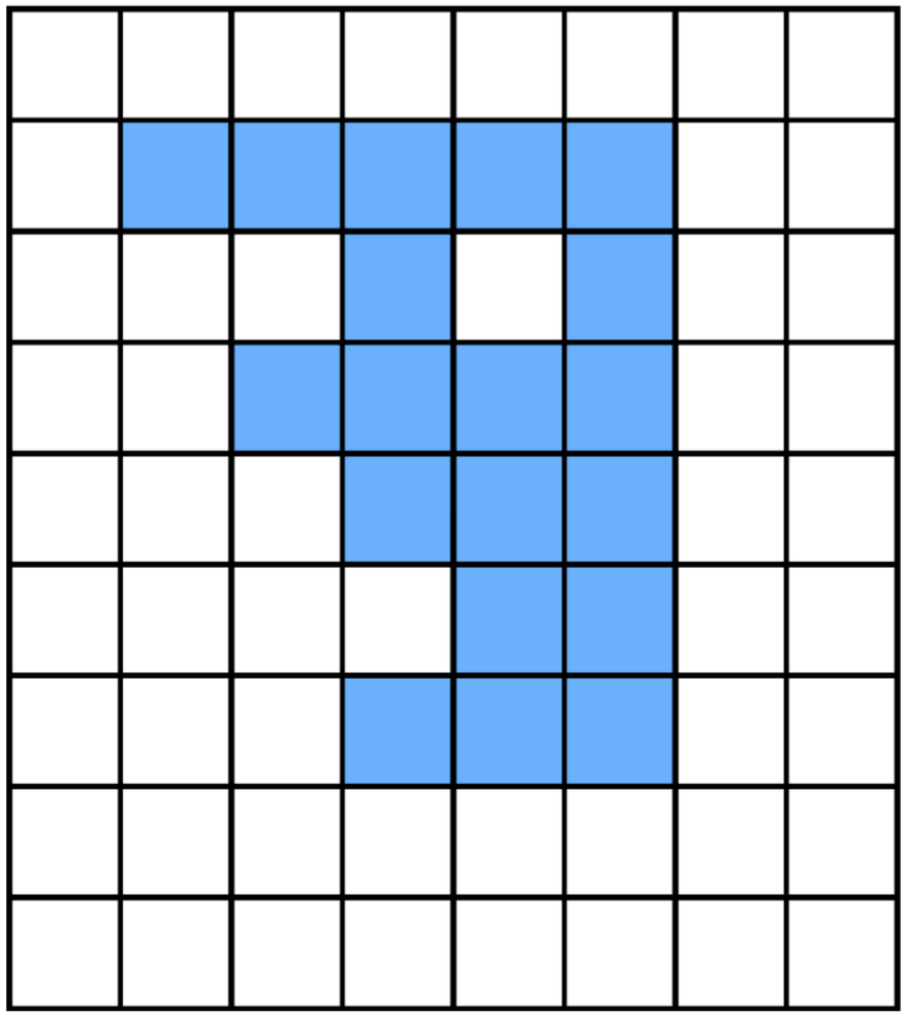
Output



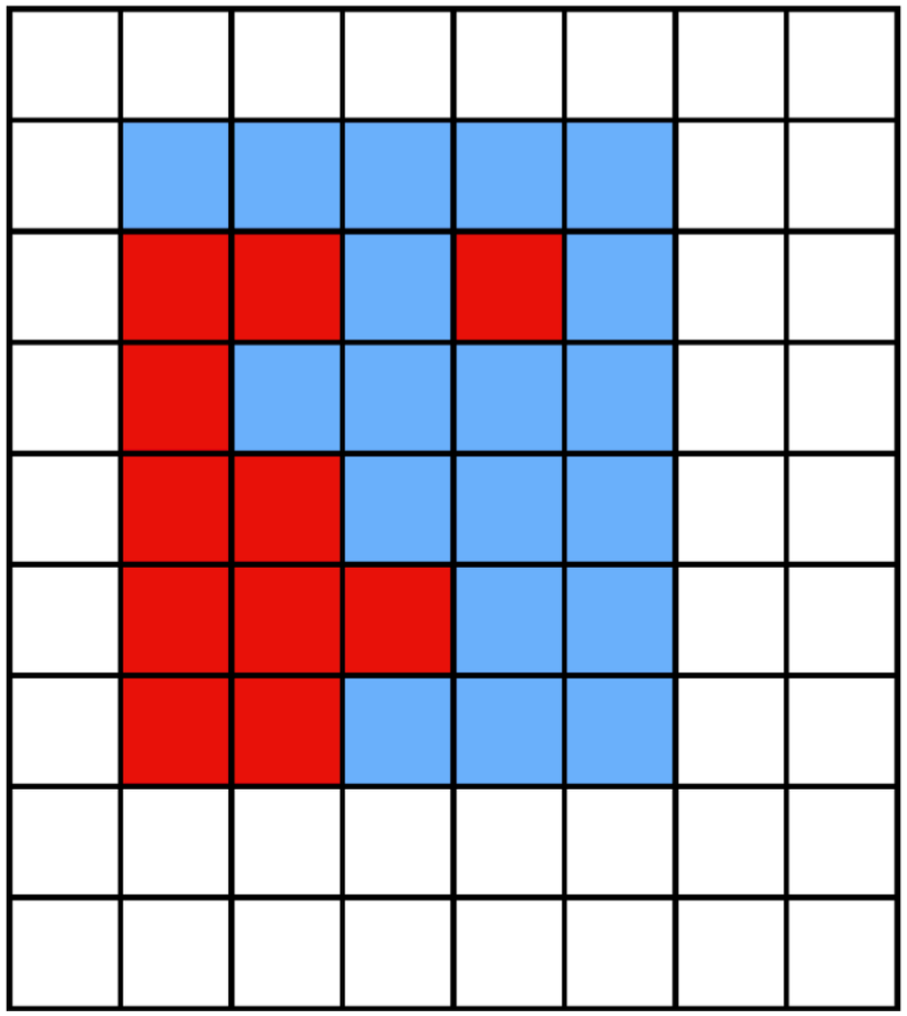
An out pixel has colour C if it is colour C in the input.

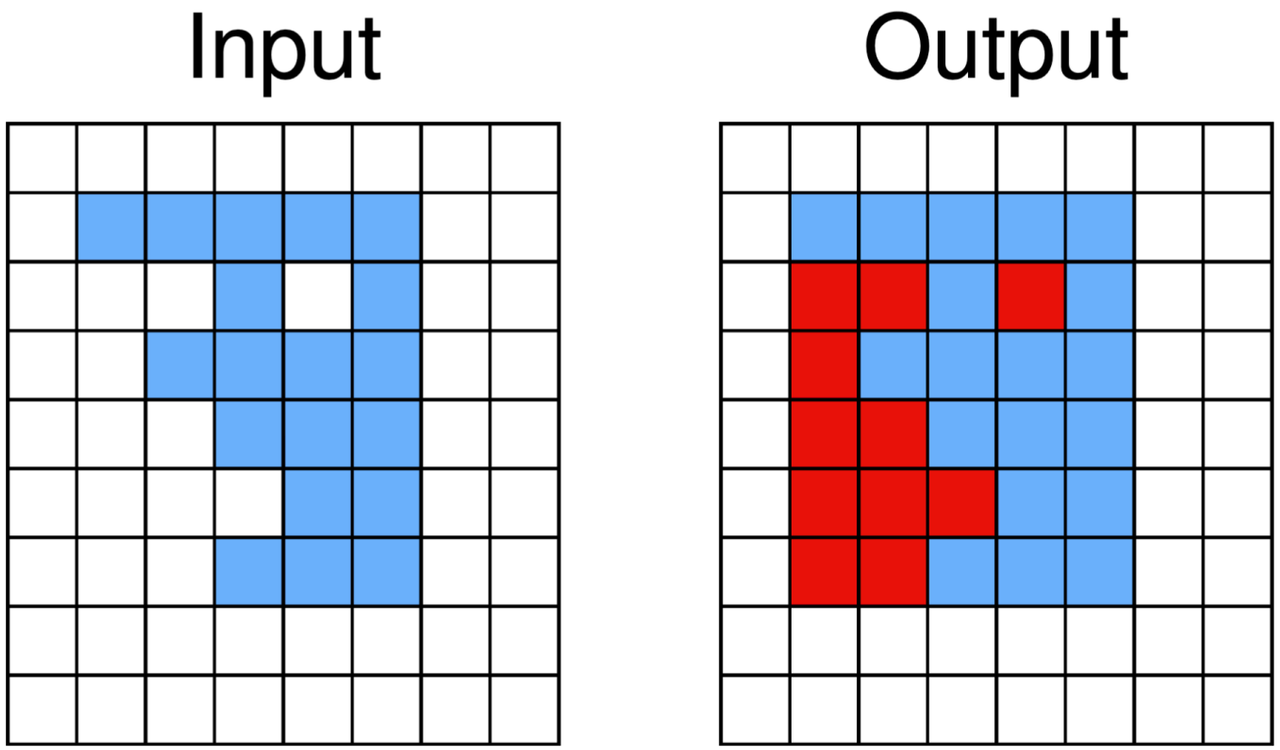
An out pixel is red if it is empty in the input and there is a pixel in the same row (X) and a pixel in the same column (Y) with the same colour (C) in the input.

Input



Output

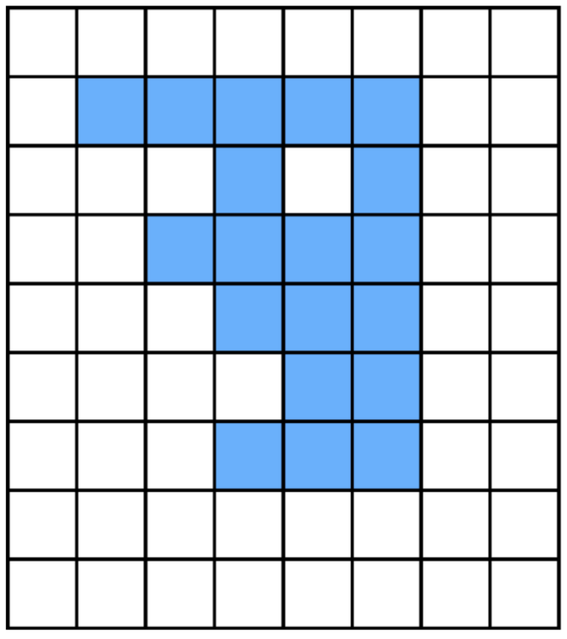




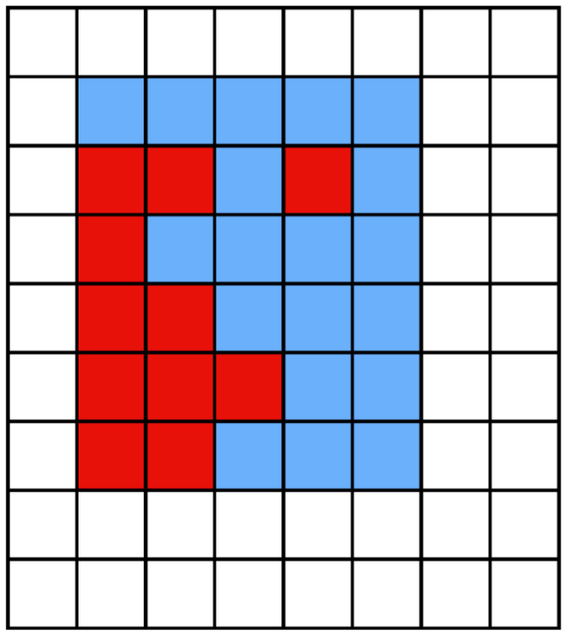
An output pixel is colour C if it is colour C in the input.

An output pixel is red if it is empty in the input and an input pixel in the row X has colour C, and an input pixel in the column Y has colour C.

Input



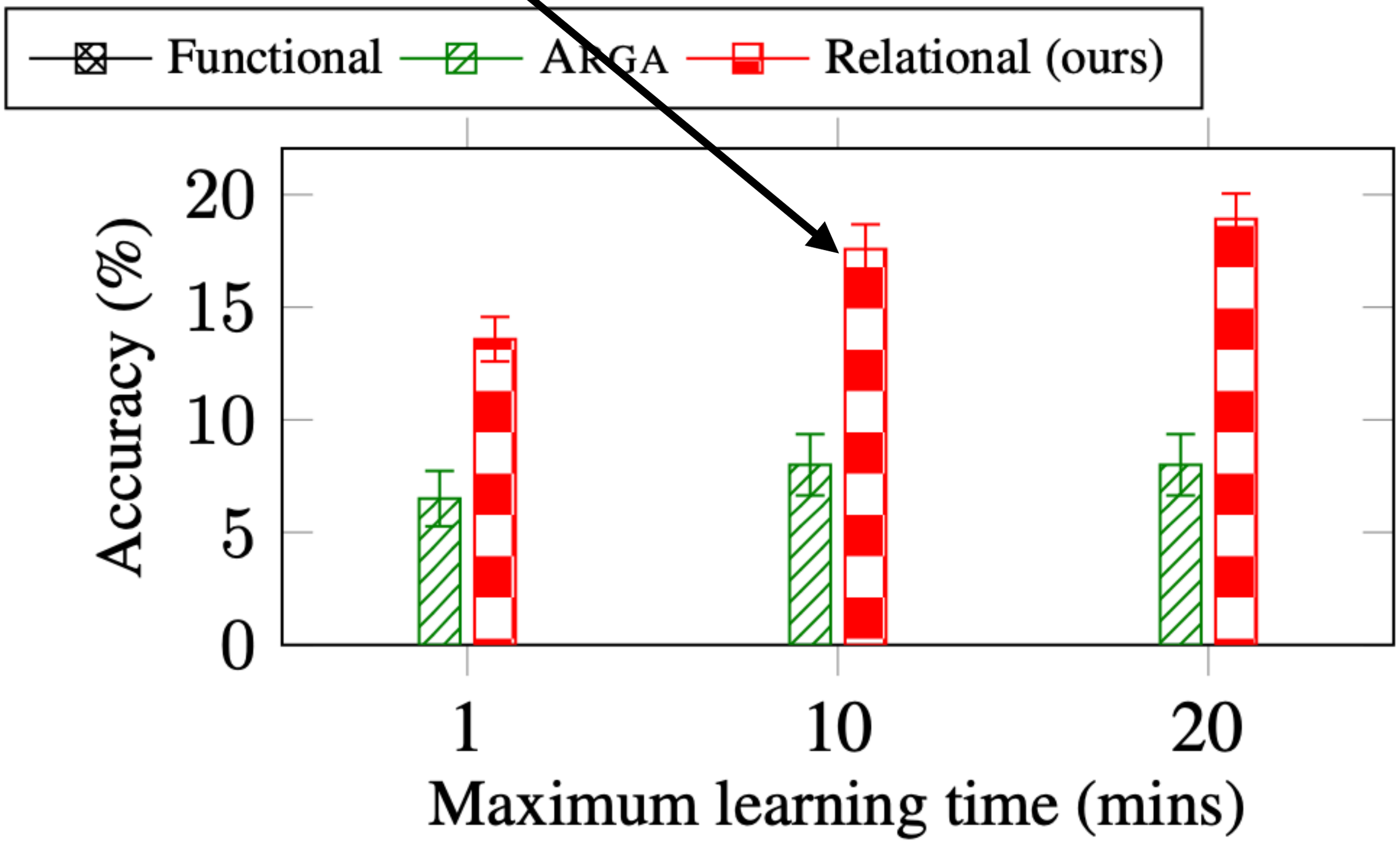
Output



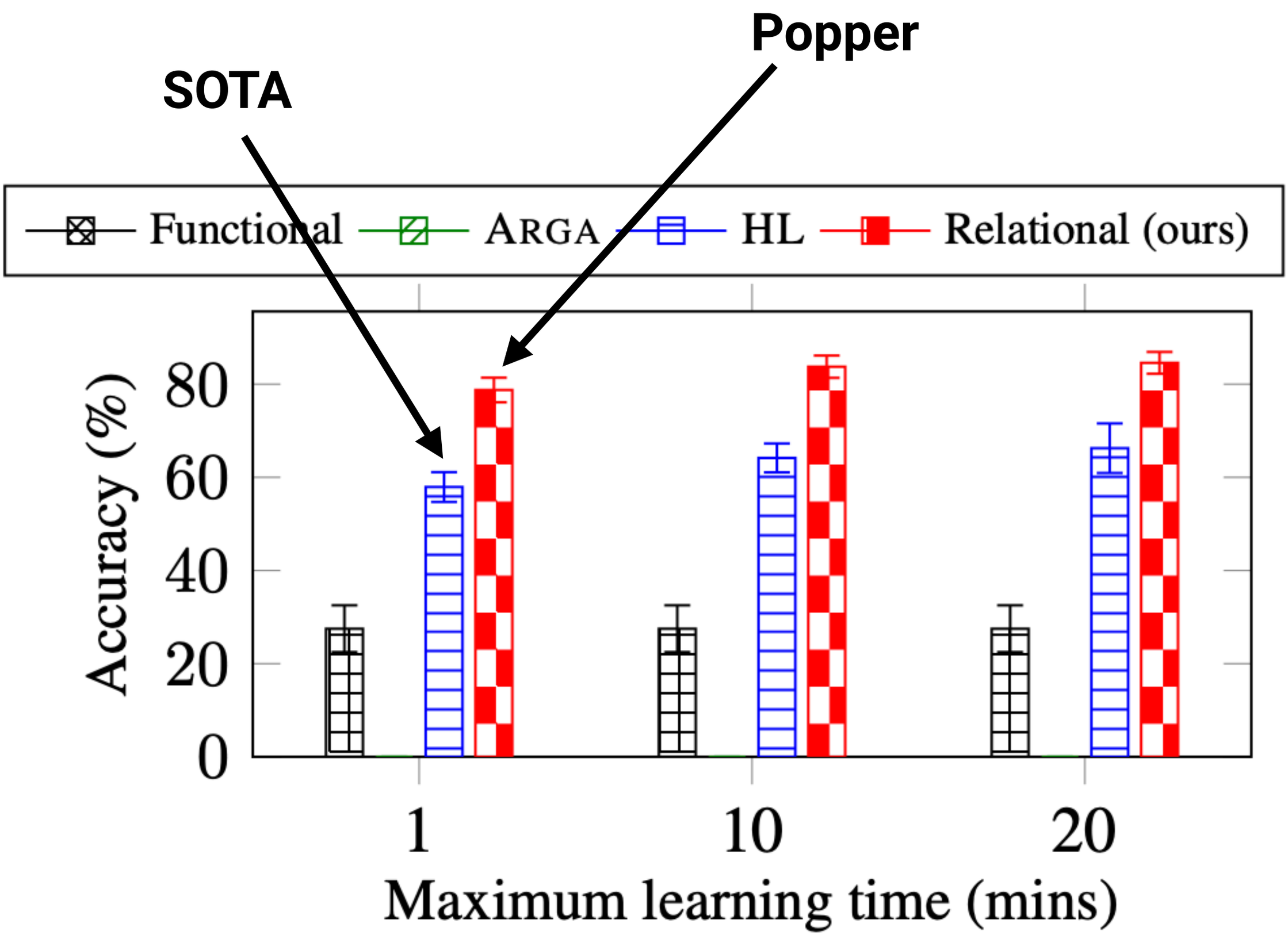
An output pixel is colour C if it is colour C in the input.

An output pixel is red if it is empty in the input and an input pixel in the row X has colour C, and an input pixel in the column Y has colour C.

Popper


Functional
ARGA
Relational (ours)
Accuracy (%)
20
15
10
5
0
1
10
20
Maximum learning time (mins)

$$[6,9,2,8,0,5] \xrightarrow{\mathcal{F}} [6,6,9,9,2,2,8,8,0,0,5,5]$$


SOTA
Popper
Functional
ARGA
HL
Relational (ours)
Accuracy (%)
0
20
40
60
80
1
10
20
Maximum learning time (mins)

Why is ILP not taking over the world?

It is not very good when there is no clear rule

It is not very good when there is no clear rule

It is not very good when there is no clear rule

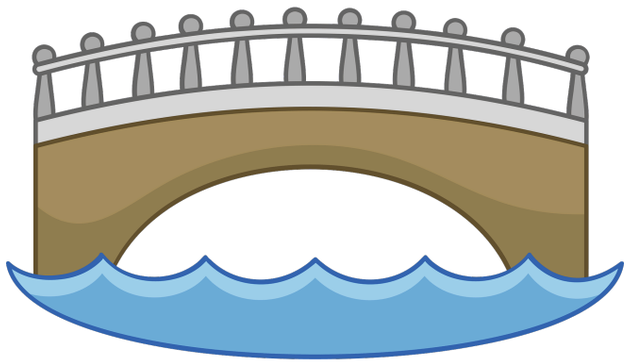
The tools are a nightmare to use

You need a PhD in ILP to use ILP

JAIR22

Load of opportunities

ML



CP

KR

PL

ILP

Summary

ILP is a form of logic-based machine learning

Tremendous progress in the past 10 years

Nice applications in algorithm/scientific discovery

Many opportunities to bridge AI communities

Questions?

Inductive logic programming at 30: a new introduction.
Cropper and Dumančić. JAIR 2022

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

Why logic programs?

- Compactness
- Declarative nature
- Similarity to natural language
- Easy to reuse techniques from CP/SAT/ASP