Inductive logic programming

Andrew Cropper

Logic-based machine learning

Andrew Cropper

Goals

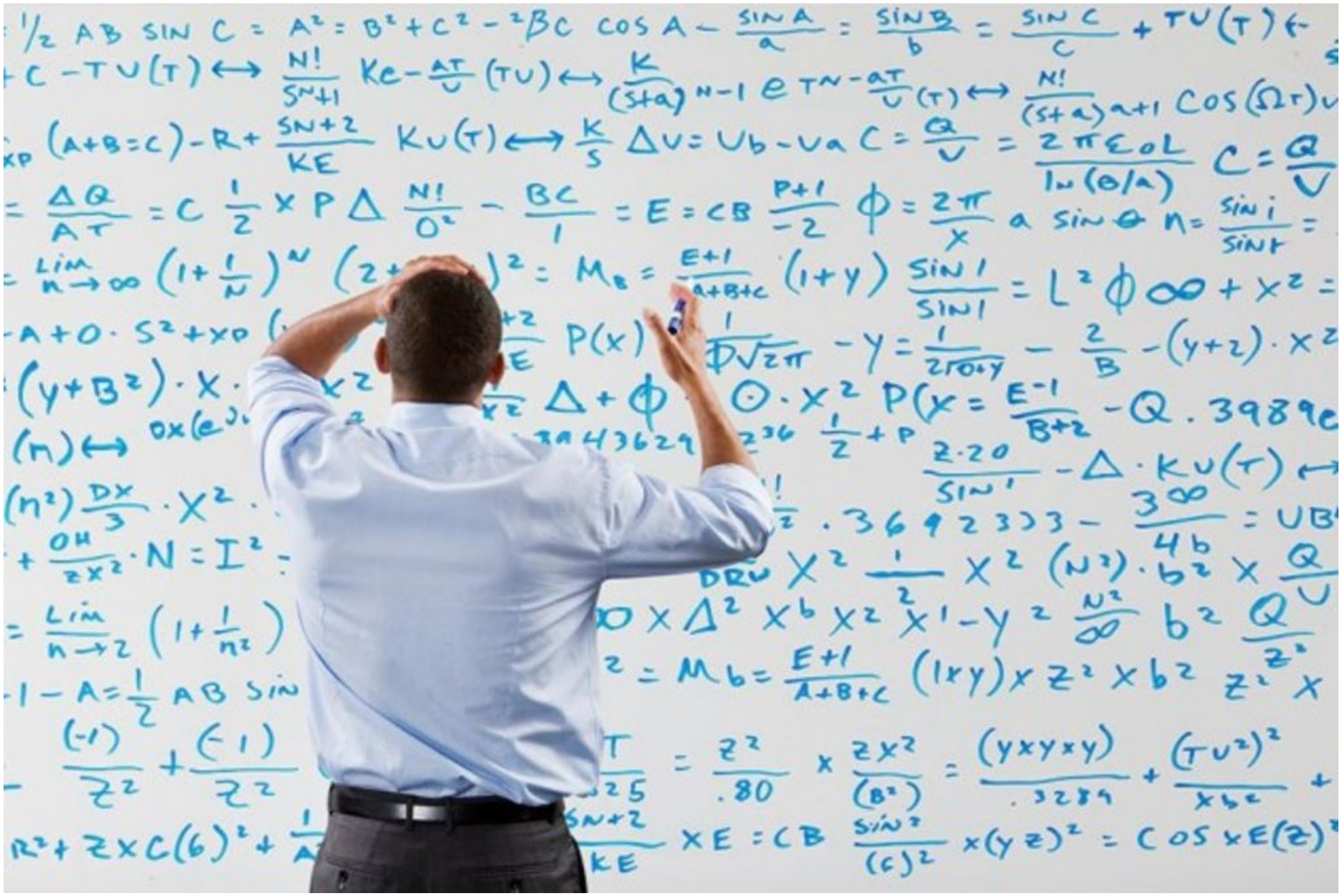
1. A high-level overview of ILP
2. For you to want to read about ILP

Outline

1. **Intro**
2. Logic 101
3. ILP problem
4. ILP techniques
5. ILP features
6. Applications
7. Future directions

Omissions

Technical details



Machine learning limitations

Machine learning limitations

explainability

Machine learning limitations

explainability

data inefficient

Machine learning limitations

explainability

data inefficient

poor knowledge transfer

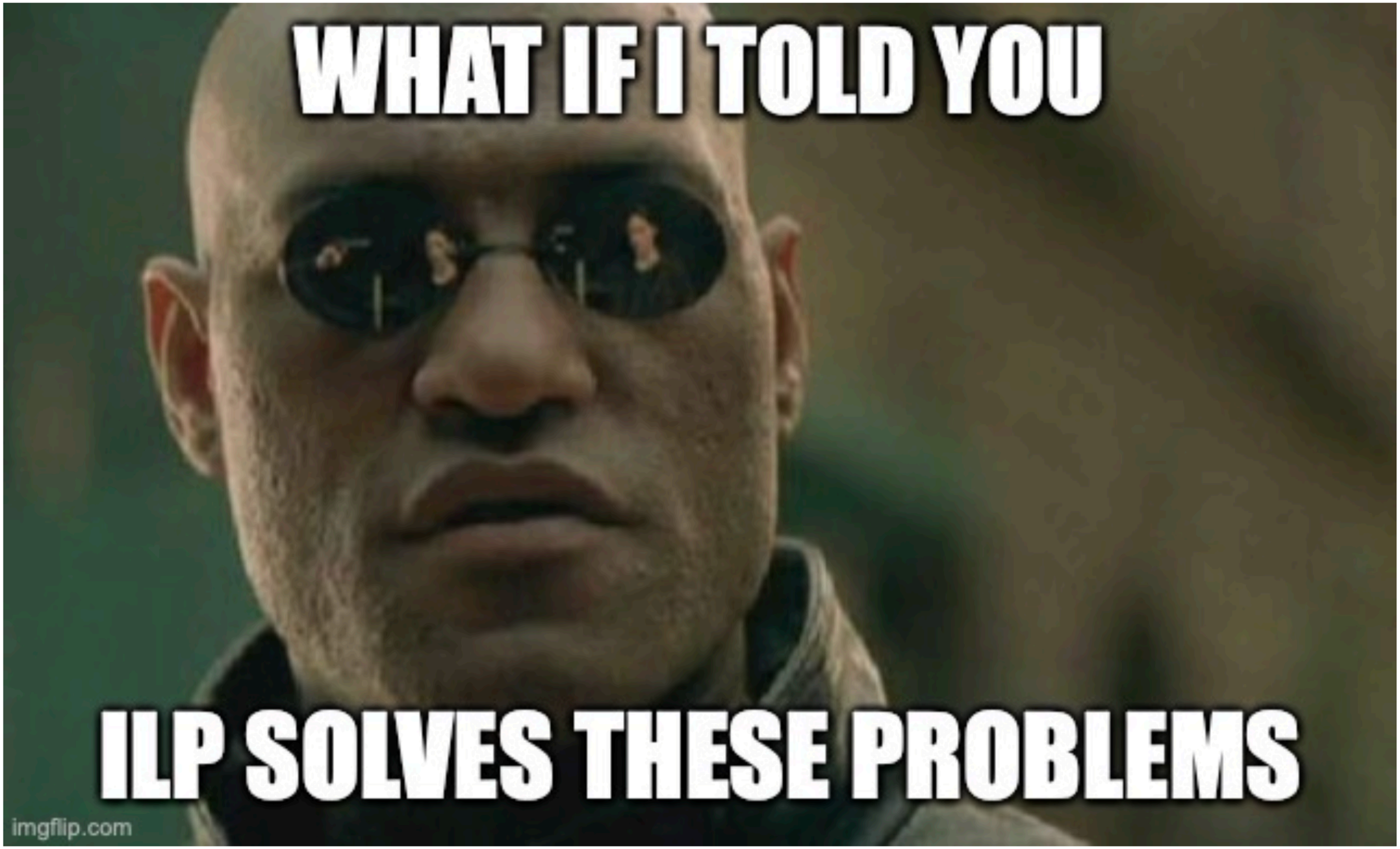
Machine learning limitations

explainability

data inefficient

poor knowledge transfer

power crazy


WHAT IF I TOLD YOU
ILP SOLVES THESE PROBLEMS
imgflip.com

Inductive logic programming

A form of machine learning that uses **logic** to represent data

Outline

1. Intro
2. **Logic 101**
3. ILP problem
4. ILP techniques
5. ILP features
6. Applications
7. Future directions

Socrates is a man.
All men are mortal.

Socrates is a man.
All men are mortal.
———————————————
Therefore, Socrates is mortal.

Socrates is a man.
All men are mortal.
————————————————
Therefore, Socrates is mortal.

fact | atom

man(socrates).
∀**A** man(**A**) → mortal(**A**).

rule | formula

Socrates is a man.
All men are mortal.
———————————————
Therefore, Socrates is mortal.

man(socrates).
∀**A** man(**A**) → mortal(**A**).

if this side is true

then this side is true

Socrates is a man.
All men are mortal.
———————————————
Therefore, Socrates is mortal.

man(socrates).
∀**A** man(**A**) → mortal(**A**).
———————————————
mortal(socrates).

$$\forall \mathbf{A}\ \mathrm{man}(\mathbf{A}) \rightarrow \mathrm{mortal}(\mathbf{A}).$$

$$\forall \mathbf{A}\ \mathrm{man}(\mathbf{A}) \rightarrow \mathrm{mortal}(\mathbf{A}).$$

$$\Downarrow$$

$$\mathrm{man}(\mathbf{A}) \rightarrow \mathrm{mortal}(\mathbf{A}).$$

$$\forall \mathbf{A}\ \mathrm{man}(\mathbf{A}) \rightarrow \mathrm{mortal}(\mathbf{A}).$$

$$\Downarrow$$

$$\mathrm{man}(\mathbf{A}) \rightarrow \mathrm{mortal}(\mathbf{A}).$$

$$\Downarrow$$

$$\mathrm{mortal}(\mathbf{A}) \leftarrow \mathrm{man}(\mathbf{A}).$$

$$\forall \mathbf{A}\ \mathrm{man}(\mathbf{A}) \rightarrow \mathrm{mortal}(\mathbf{A}).$$

$$\Downarrow$$

$$\mathrm{man}(\mathbf{A}) \rightarrow \mathrm{mortal}(\mathbf{A}).$$

$$\Downarrow$$

$$\mathrm{mortal}(\mathbf{A}) \leftarrow \mathrm{man}(\mathbf{A}).$$

$$\Downarrow$$

```
mortal(A):- man(A).
```

valid Prolog / Datalog / ASP rule

$$\forall \mathbf{A}.\forall \mathbf{B}\ \text{knows}(\mathbf{A},\mathbf{B}) \wedge \text{rich}(\mathbf{B}) \wedge \text{famous}(\mathbf{B}) \rightarrow \text{happy}(\mathbf{A}).$$

$$\forall \mathbf{A}.\forall \mathbf{B}\ \text{knows}(\mathbf{A},\mathbf{B}) \wedge \text{rich}(\mathbf{B}) \wedge \text{famous}(\mathbf{B}) \rightarrow \text{happy}(\mathbf{A}).$$

$$\Downarrow$$

$$\text{knows}(\mathbf{A},\mathbf{B}) \wedge \text{rich}(\mathbf{B}) \wedge \text{famous}(\mathbf{B}) \rightarrow \text{happy}(\mathbf{A}).$$

$$\forall \mathbf{A}.\forall \mathbf{B}\ \text{knows}(\mathbf{A},\mathbf{B}) \wedge \text{rich}(\mathbf{B}) \wedge \text{famous}(\mathbf{B}) \rightarrow \text{happy}(\mathbf{A}).$$

$$\Downarrow$$

$$\text{knows}(\mathbf{A},\mathbf{B}) \wedge \text{rich}(\mathbf{B}) \wedge \text{famous}(\mathbf{B}) \rightarrow \text{happy}(\mathbf{A}).$$

$$\Downarrow$$

$$\text{happy}(\mathbf{A}) \leftarrow \text{knows}(\mathbf{A},\mathbf{B}) \wedge \text{rich}(\mathbf{B}) \wedge \text{famous}(\mathbf{B}).$$

$$\forall \mathbf{A}.\forall \mathbf{B}\ \text{knows}(\mathbf{A},\mathbf{B}) \wedge \text{rich}(\mathbf{B}) \wedge \text{famous}(\mathbf{B}) \rightarrow \text{happy}(\mathbf{A}).$$

$$\Downarrow$$

$$\text{knows}(\mathbf{A},\mathbf{B}) \wedge \text{rich}(\mathbf{B}) \wedge \text{famous}(\mathbf{B}) \rightarrow \text{happy}(\mathbf{A}).$$

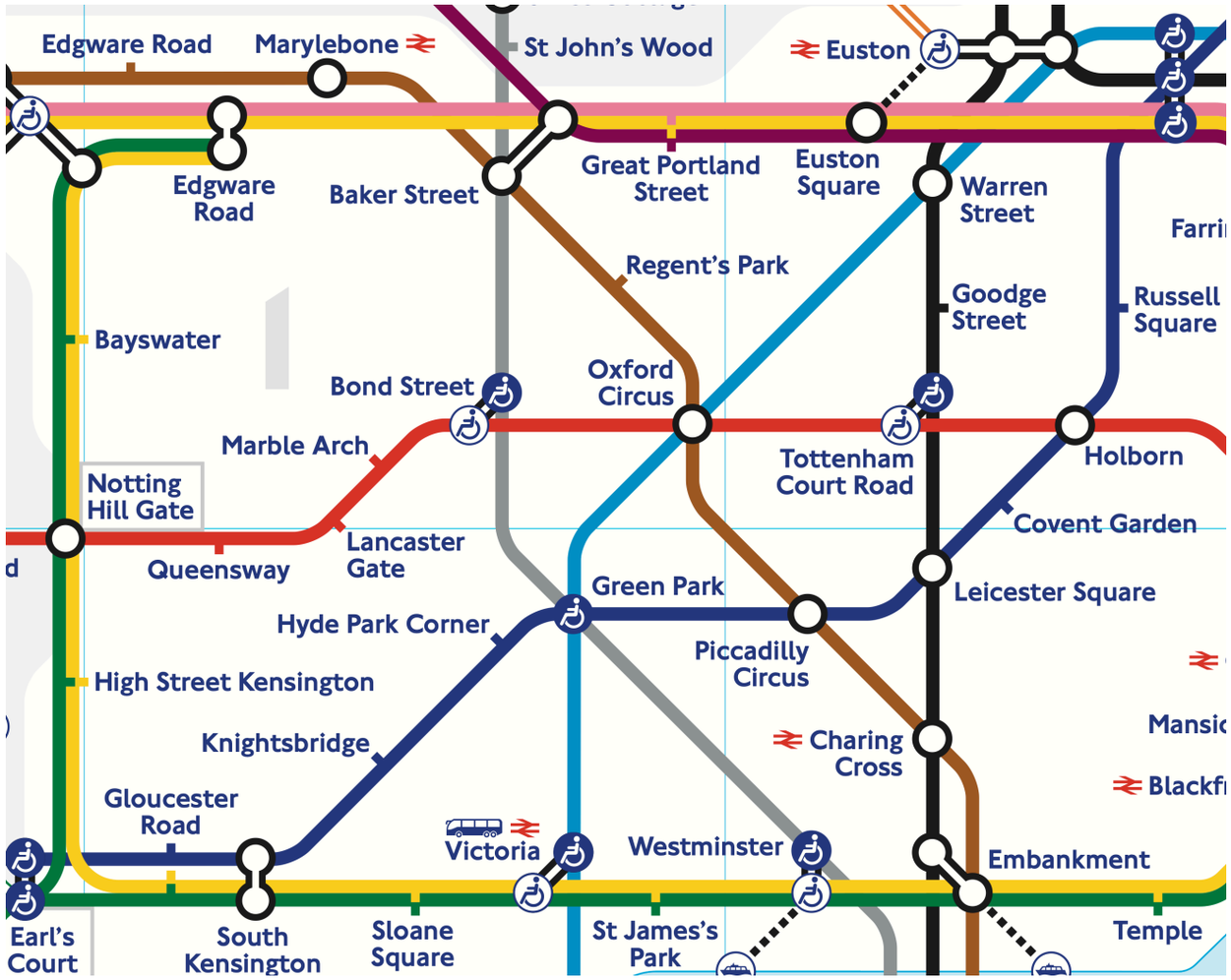
$$\Downarrow$$

$$\text{happy}(\mathbf{A}) \leftarrow \text{knows}(\mathbf{A},\mathbf{B}) \wedge \text{rich}(\mathbf{B}) \wedge \text{famous}(\mathbf{B}).$$

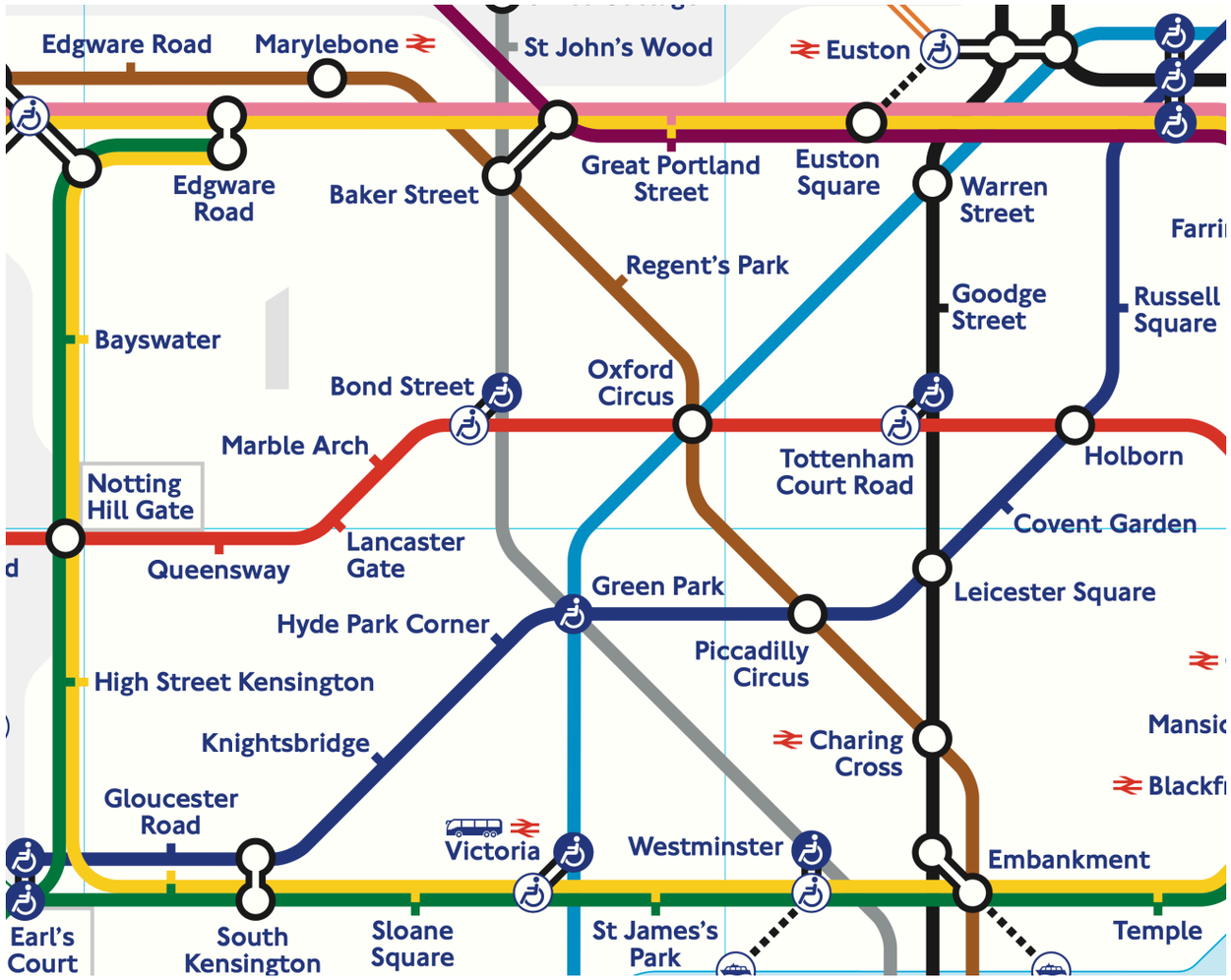
$$\Downarrow$$

```
happy(A):- knows(A,B), rich(B), famous(B).
```

Relational data

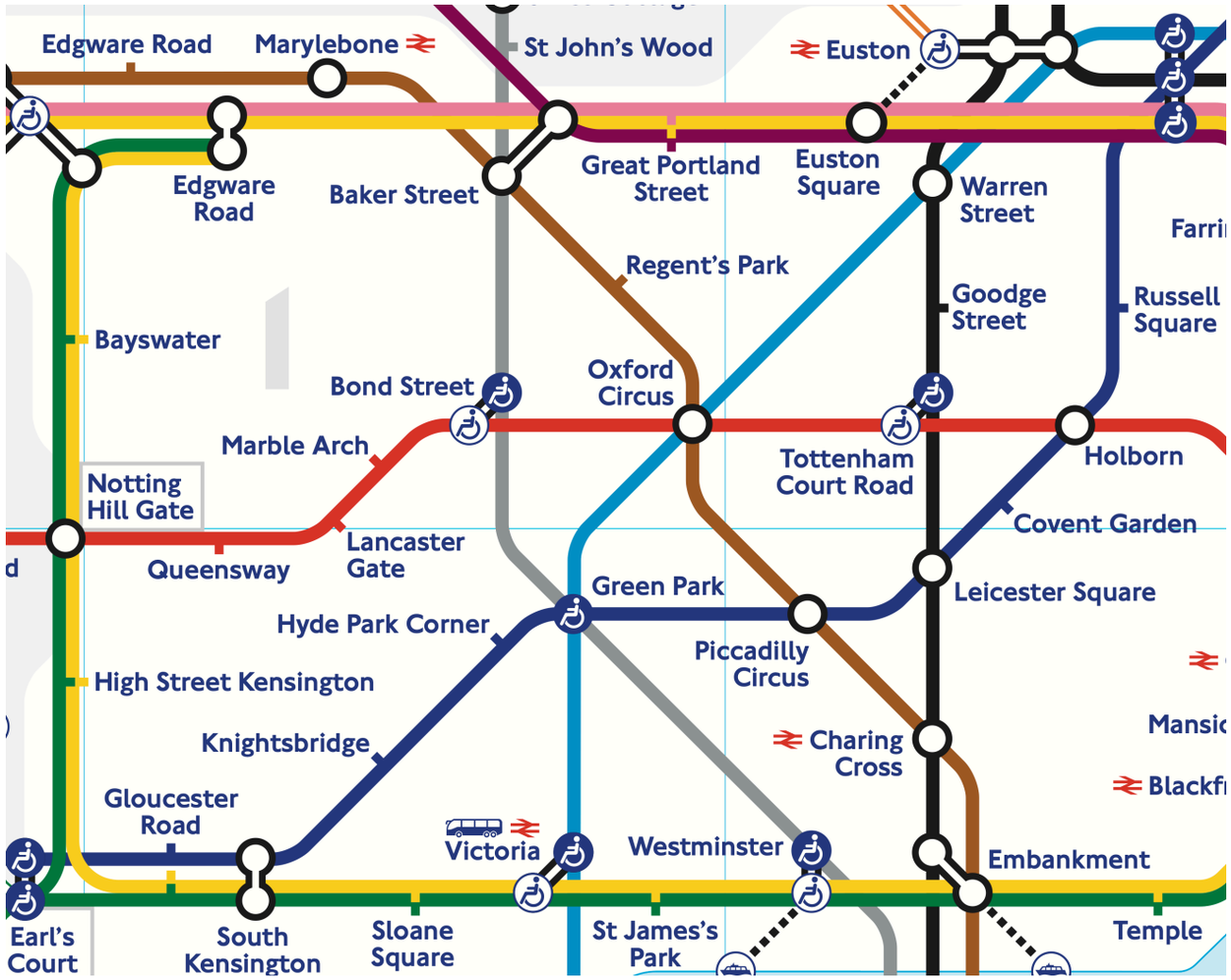


Relational data




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

Relational data




```
connected(S1,S2):- edge(S1,S2).
connected(S1,S2):- edge(S1,S3), connected(S3,S2).
```

What does this have to do with programming?

Logic programs

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

Logic programs

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

```
?- head([h,e,l,l,o],X).
X = h.

?- tail([h,e,l,l,o],X).
X = [e, l, l, o].
```

Logic programs

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

```
?- tail([h,e,l,l,o],[c,a,t]).
false.
```

Logic programs

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

```
?- tail(X,[c,a,t]).
X = [_9930, c, a, t].
```

Logic programs

```
length([],0).
length([H|T],N2):-
    length(T,N1),
    N2 is N1+1.
```

Logic programs

```
length([],0).
length([H|T],N2):-
    length(T,N1),
    N2 is N1+1.
```

```
?- length([c,a,t],X).
X = 3.
```

Logic programs

```
length([],0).
length([H|T],N2):-
    length(T,N1),
    N2 is N1+1.
```

```
?- length(X,4).
X = [_6240, _6246, _6252, _6258].
```

Please ask me to clarify anything on logic

Outline

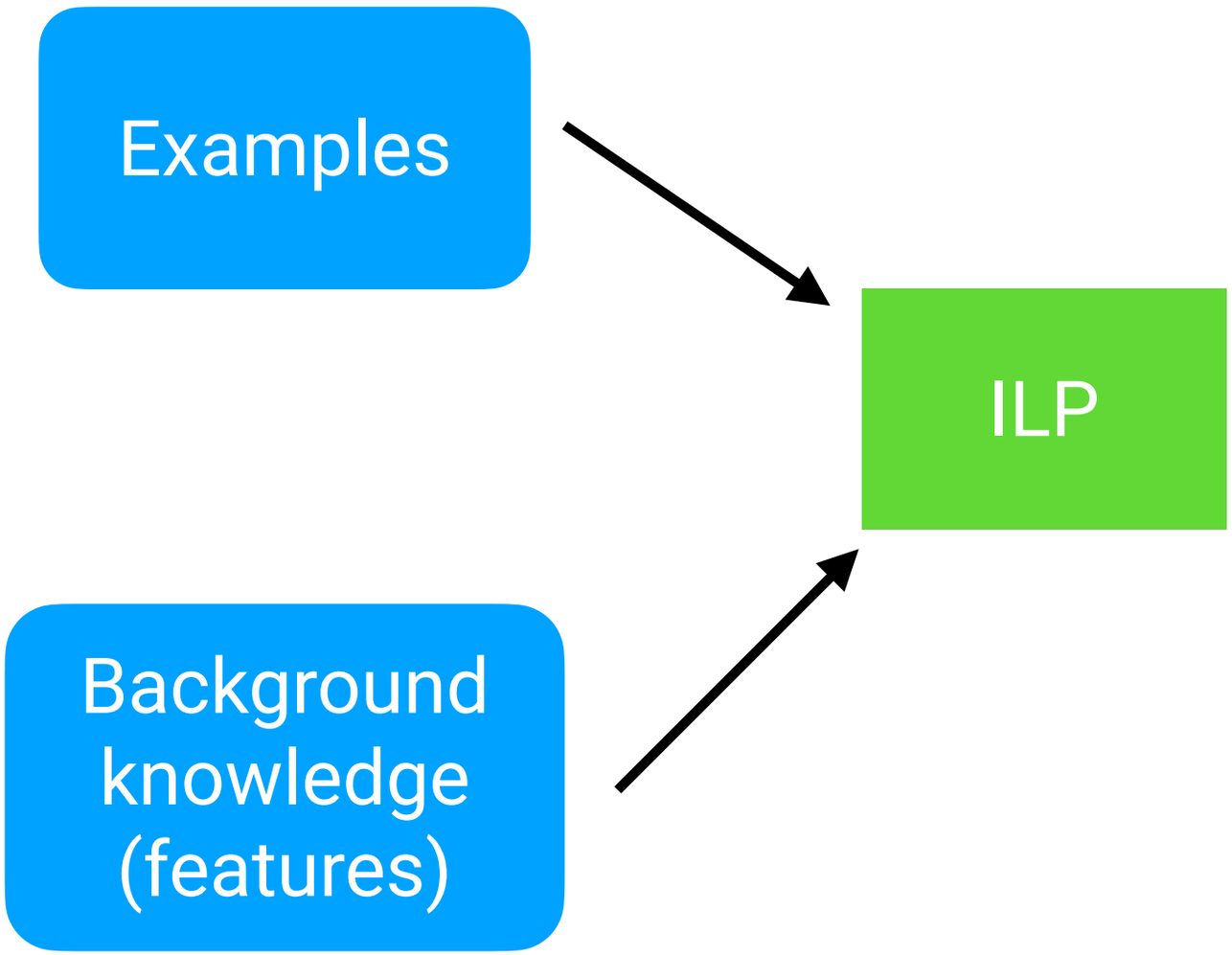
1. Intro
2. Logic 101
3. **ILP problem**
4. ILP techniques
5. ILP features
6. Applications
7. Future directions

Inductive logic programming

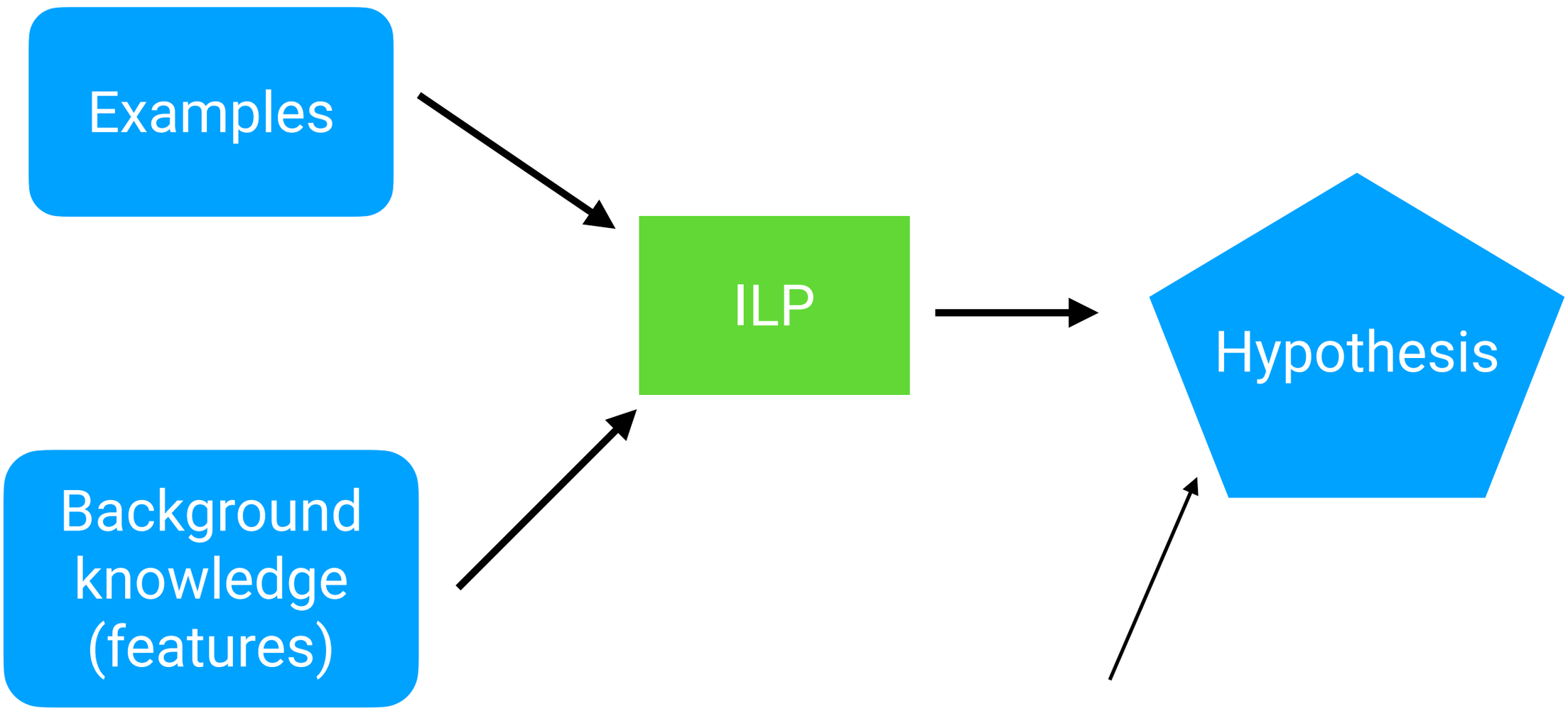
Examples

Background knowledge (features)

Inductive logic programming

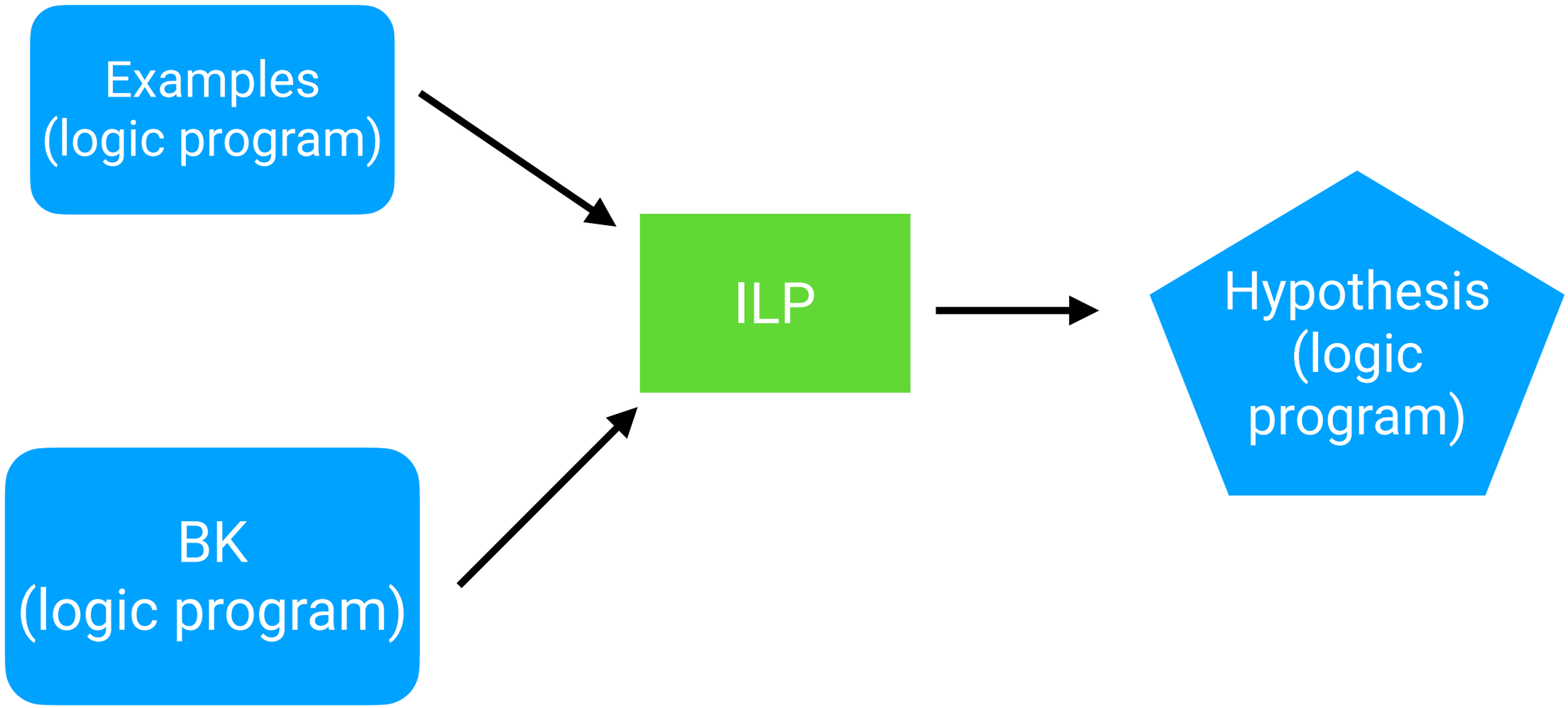


Inductive logic programming



A hypothesis / theory / program / ~~model~~

Inductive logic programming



ILP problem

Given:

- background knowledge **B**
- positive examples **E+**
- negative examples **E-**

ILP problem

Given:

- background knowledge **B**
- positive examples **E+**
- negative examples **E-**

Find: a hypothesis **H** such that:

- H ∪ B *entails* E+
- H ∪ B *does not entail* E-

Boring example

Classical ML task

Name	Lego builder	Estate agent	Golfer	Enjoys Lego	Knows Alice	Knows Bob	Knows Claire	Knows Dave	Knows Edith	Knows Fred	Happy
Alice	Yes	No	No	Yes	Yes	No	No	No	Yes	Yes	Yes

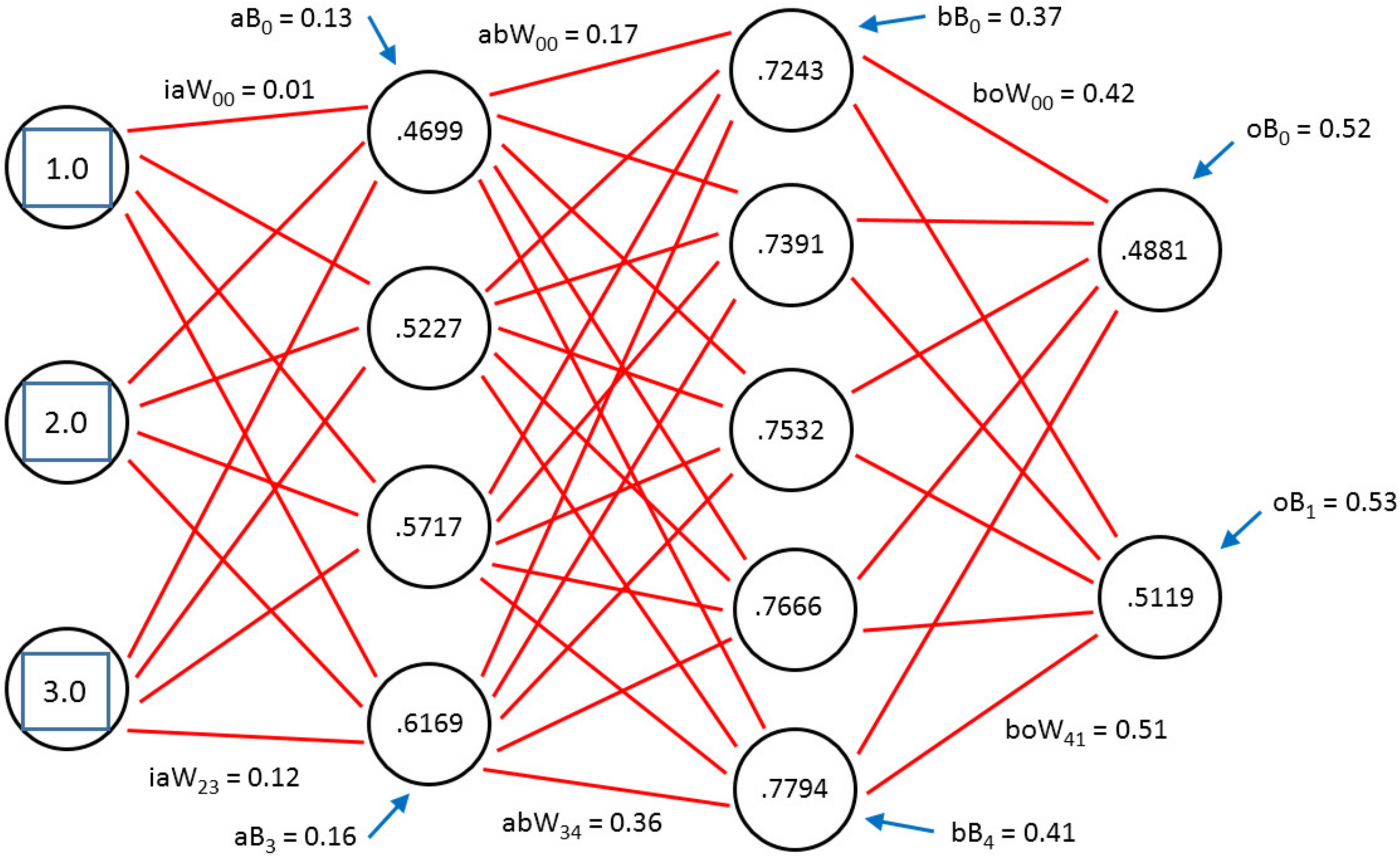
Classical ML task

Name	Lego builder	Estate agent	Golfer	Enjoys Lego	Knows Alice	Knows Bob	Knows Claire	Knows Dave	Knows Edith	Knows Fred	Happy
Alice	Yes	No	No	Yes	Yes	No	No	No	Yes	Yes	Yes
Bob	Yes	No	No	No	No	Yes	Yes	Yes	Yes	No	No
Claire	No	Yes	No	Yes	No	Yes	Yes	Yes	No	No	No
Dave	No	Yes	No	No	No	Yes	Yes	Yes	No	No	No
Edith	No	No	Yes	No	No	Yes	No	No	Yes	No	No
Fred	No	No	Yes	No	Yes	No	No	No	No	Yes	Yes

Standard ML representation

Name	Lego builder	Estate agent	Golfer	Enjoys Lego	Knows Alice	Knows Bob	Knows Claire	Knows Dave	Knows Edith	Knows Fred	Happy
Alice	1	0	0	1	1	0	0	0	1	1	1
Bob	1	0	0	0	0	1	1	1	1	0	0
Claire	0	1	0	1	0	1	1	1	0	0	0
Dave	0	1	0	0	0	1	1	1	0	0	0
Edith	0	0	1	0	0	1	0	0	1	0	0
Fred	0	0	1	0	1	0	0	0	0	1	1

Table of numbers



Estimate parameters

Name	Lego builder	Estate agent	Golfer	Enjoys Lego	Knows Alice	Knows Bob	Knows Claire	Knows Dave	Knows Edith	Knows Fred	Happy
Alice	Yes	No	No	Yes	Yes	No	No	No	Yes	Yes	Yes
Bob	Yes	No	No	No	No	Yes	Yes	Yes	Yes	No	No
Claire	No	Yes	No	Yes	No	Yes	Yes	Yes	No	No	No
Dave	No	Yes	No	No	No	Yes	Yes	Yes	No	No	No
Edith	No	No	Yes	No	No	Yes	No	No	Yes	No	No
Fred	No	No	Yes	No	Yes	No	No	No	No	Yes	Yes

Name	Lego builder	Estate agent	Golfer	Enjoys Lego	Knows Alice	Knows Bob	Knows Claire	Knows Dave	Knows Edith	Knows Fred	Happy
Alice	Yes	No	No	Yes	Yes	No	No	No	Yes	Yes	Yes
Bob	Yes	No	No	No	No	Yes	Yes	Yes	Yes	No	No
Claire	No	Yes	No	Yes	No	Yes	Yes	Yes	No	No	No
Dave	No	Yes	No	No	No	Yes	Yes	Yes	No	No	No
Edith	No	No	Yes	No	No	Yes	No	No	Yes	No	No
Fred	No	No	Yes	No	Yes	No	No	No	No	Yes	Yes

```
%% B
lego_builder(alice).
lego_builder(bob).
```

Name	Lego builder	Estate agent	Golfer	Enjoys Lego	Knows Alice	Knows Bob	Knows Claire	Knows Dave	Knows Edith	Knows Fred	Happy
Alice	Yes	No	No	Yes	Yes	No	No	No	Yes	Yes	Yes
Bob	Yes	No	No	No	No	Yes	Yes	Yes	Yes	No	No
Claire	No	Yes	No	Yes	No	Yes	Yes	Yes	No	No	No
Dave	No	Yes	No	No	No	Yes	Yes	Yes	No	No	No
Edith	No	No	Yes	No	No	Yes	No	No	Yes	No	No
Fred	No	No	Yes	No	Yes	No	No	No	No	Yes	Yes

```
%% B
lego_builder(alice).
lego_builder(bob).
estate_agent(claire).
estate_agent(dave).
```

Name	Lego builder	Estate agent	Golfer	Enjoys Lego	Knows Alice	Knows Bob	Knows Claire	Knows Dave	Knows Edith	Knows Fred	Happy
Alice	Yes	No	No	Yes	Yes	No	No	No	Yes	Yes	Yes
Bob	Yes	No	No	No	No	Yes	Yes	Yes	Yes	No	No
Claire	No	Yes	No	Yes	No	Yes	Yes	Yes	No	No	No
Dave	No	Yes	No	No	No	Yes	Yes	Yes	No	No	No
Edith	No	No	Yes	No	No	Yes	No	No	Yes	No	No
Fred	No	No	Yes	No	Yes	No	No	No	No	Yes	Yes

```
%% B
lego_builder(alice).
lego_builder(bob).
estate_agent(claire).
estate_agent(dave).
golfer(edith).
golfer(fred).
```

Name	Lego builder	Estate agent	Golfer	Enjoys Lego	Knows Alice	Knows Bob	Knows Claire	Knows Dave	Knows Edith	Knows Fred	Happy
Alice	Yes	No	No	Yes	Yes	No	No	No	Yes	Yes	Yes
Bob	Yes	No	No	No	No	Yes	Yes	Yes	Yes	No	No
Claire	No	Yes	No	Yes	No	Yes	Yes	Yes	No	No	No
Dave	No	Yes	No	No	No	Yes	Yes	Yes	No	No	No
Edith	No	No	Yes	No	No	Yes	No	No	Yes	No	No
Fred	No	No	Yes	No	Yes	No	No	No	No	Yes	Yes

```
%% B
lego_builder(alice).
lego_builder(bob).
estate_agent(claire).
estate_agent(dave).
golfer(edith).
golfer(fred).
enjoys_lego(alice).
enjoys_lego(claire).
```

Name	Lego builder	Estate agent	Golfer	Enjoys Lego	Knows Alice	Knows Bob	Knows Claire	Knows Dave	Knows Edith	Knows Fred	Happy
Alice	Yes	No	No	Yes	Yes	No	No	No	Yes	Yes	Yes
Bob	Yes	No	No	No	No	Yes	Yes	Yes	Yes	No	No
Claire	No	Yes	No	Yes	No	Yes	Yes	Yes	No	No	No
Dave	No	Yes	No	No	No	Yes	Yes	Yes	No	No	No
Edith	No	No	Yes	No	No	Yes	No	No	Yes	No	No
Fred	No	No	Yes	No	Yes	No	No	No	No	Yes	Yes

```
%% B
lego_builder(alice).
lego_builder(bob).
estate_agent(claire).
estate_agent(dave).
golfer(edith).
golfer(fred).
enjoys_lego(alice).
enjoys_lego(claire).
knows(fred,alice).
```

Name	Lego builder	Estate agent	Golfer	Enjoys Lego	Knows Alice	Knows Bob	Knows Claire	Knows Dave	Knows Edith	Knows Fred	Happy
Alice	Yes	No	No	Yes	Yes	No	No	No	Yes	Yes	Yes
Bob	Yes	No	No	No	No	Yes	Yes	Yes	Yes	No	No
Claire	No	Yes	No	Yes	No	Yes	Yes	Yes	No	No	No
Dave	No	Yes	No	No	No	Yes	Yes	Yes	No	No	No
Edith	No	No	Yes	No	No	Yes	No	No	Yes	No	No
Fred	No	No	Yes	No	Yes	No	No	No	No	Yes	Yes

```
%% B
lego_builder(alice).
lego_builder(bob).
estate_agent(claire).
estate_agent(dave).
golfer(edith).
golfer(fred).
enjoys_lego(alice).
enjoys_lego(claire).
knows(fred,alice).
knows(A,A).
```

Name	Lego builder	Estate agent	Golfer	Enjoys Lego	Knows Alice	Knows Bob	Knows Claire	Knows Dave	Knows Edith	Knows Fred	Happy
Alice	Yes	No	No	Yes	Yes	No	No	No	Yes	Yes	Yes
Bob	Yes	No	No	No	No	Yes	Yes	Yes	Yes	No	No
Claire	No	Yes	No	Yes	No	Yes	Yes	Yes	No	No	No
Dave	No	Yes	No	No	No	Yes	Yes	Yes	No	No	No
Edith	No	No	Yes	No	No	Yes	No	No	Yes	No	No
Fred	No	No	Yes	No	Yes	No	No	No	No	Yes	Yes

```
%% B
lego_builder(alice).
lego_builder(bob).
estate_agent(claire).
estate_agent(dave).
golfer(edith).
golfer(fred).
enjoys_lego(alice).
enjoys_lego(claire).
```

```
knows(fred,alice).
knows(A,A).
knows(A,B):- knows(B,A).
```

Name	Lego builder	Estate agent	Golfer	Enjoys Lego	Knows Alice	Knows Bob	Knows Claire	Knows Dave	Knows Edith	Knows Fred	Happy
Alice	Yes	No	No	Yes	Yes	No	No	No	Yes	Yes	Yes
Bob	Yes	No	No	No	No	Yes	Yes	Yes	Yes	No	No
Claire	No	Yes	No	Yes	No	Yes	Yes	Yes	No	No	No
Dave	No	Yes	No	No	No	Yes	Yes	Yes	No	No	No
Edith	No	No	Yes	No	No	Yes	No	No	Yes	No	No
Fred	No	No	Yes	No	Yes	No	No	No	No	Yes	Yes

```
%% B
lego_builder(alice).
lego_builder(bob).
estate_agent(claire).
estate_agent(dave).
golfer(edith).
golfer(fred).
enjoys_lego(alice).
enjoys_lego(claire).
knows(fred,alice).
knows(claire,bob).
knows(dave,bob).
knows(edith,bob).
knows(dave,claire).
knows(alice,edith).
knows(bob,edith).
knows(A,A).
knows(A,B):-  knows(B,A).
```

Name	Lego builder	Estate agent	Golfer	Enjoys Lego	Knows Alice	Knows Bob	Knows Claire	Knows Dave	Knows Edith	Knows Fred	Happy
Alice	Yes	No	No	Yes	Yes	No	No	No	Yes	Yes	Yes
Bob	Yes	No	No	No	No	Yes	Yes	Yes	Yes	No	No
Claire	No	Yes	No	Yes	No	Yes	Yes	Yes	No	No	No
Dave	No	Yes	No	No	No	Yes	Yes	Yes	No	No	No
Edith	No	No	Yes	No	No	Yes	No	No	Yes	No	No
Fred	No	No	Yes	No	Yes	No	No	No	No	Yes	Yes

```
%% E+
happy(alice).
happy(fred).
```

Name	Lego builder	Estate agent	Golfer	Enjoys Lego	Knows Alice	Knows Bob	Knows Claire	Knows Dave	Knows Edith	Knows Fred	Happy
Alice	Yes	No	No	Yes	Yes	No	No	No	Yes	Yes	Yes
Bob	Yes	No	No	No	No	Yes	Yes	Yes	Yes	No	No
Claire	No	Yes	No	Yes	No	Yes	Yes	Yes	No	No	No
Dave	No	Yes	No	No	No	Yes	Yes	Yes	No	No	No
Edith	No	No	Yes	No	No	Yes	No	No	Yes	No	No
Fred	No	No	Yes	No	Yes	No	No	No	No	Yes	Yes

```
%% E+
happy(alice).
happy(fred).
```

```
%% E-
happy(bob).
happy(claire).
happy(dave).
happy(edith).
```

```
%% B
lego_builder(alice).
lego_builder(bob).
estate_agent(claire).
estate_agent(dave).
golfer(edith).
enjoys_lego(alice).
enjoys_lego(claire).
knows(fred,alice).
knows(claire,bob).
knows(dave,bob).
knows(edith,bob).
knows(dave,claire).
knows(alice,edith).
knows(bob,edith).
knows(A,A).
knows(A,B):-  knows(B,A).
```

```
%% E+
happy(alice).
happy(fred).

%% E-
happy(bob).
happy(claire).
happy(dave).
happy(edith).
```

```
%% H
happy(A):-
  lego_builder(A).
```

```
%% B
lego_builder(alice).
lego_builder(bob).
estate_agent(claire).
estate_agent(dave).
golfer(edith).
enjoys_lego(alice).
enjoys_lego(claire).
knows(fred,alice).
knows(claire,bob).
knows(dave,bob).
knows(edith,bob).
knows(dave,claire).
knows(alice,edith).
knows(bob,edith).
knows(A,A).
knows(A,B):-  knows(B,A).
```

```
%% E+
happy(alice).
happy(fred).

%% E-
happy(bob).
happy(claire).
happy(dave).
happy(edith).
```

```
%% H
happy(A):-
  lego_builder(A).
```

```
%% B
lego_builder(alice).
lego_builder(bob).
estate_agent(claire).
estate_agent(dave).
golfer(edith).
enjoys_lego(alice).
enjoys_lego(claire).
knows(fred,alice).
knows(claire,bob).
knows(dave,bob).
knows(edith,bob).
knows(dave,claire).
knows(alice,edith).
knows(bob,edith).
knows(A,A).
knows(A,B):-  knows(B,A).
```

```
%% E+
happy(alice).
happy(fred).
```

```
%% E-
happy(bob).
happy(claire).
happy(dave).
happy(edith).
```

```
%% H
happy(A):-
  lego_builder(A).
```

```
%% B
lego_builder(alice).
lego_builder(bob).
estate_agent(claire).
estate_agent(dave).
golfer(edith).
enjoys_lego(alice).
enjoys_lego(claire).
knows(fred,alice).
knows(claire,bob).
knows(dave,bob).
knows(edith,bob).
knows(dave,claire).
knows(alice,edith).
knows(bob,edith).
knows(A,A).
knows(A,B):-  knows(B,A).
```

```
%% E+
happy(alice).
happy(fred).

%% E-
happy(bob).
happy(claire).
happy(dave).
happy(edith).
```

```
%% H
happy(A):-
  lego_builder(A).
  enjoys_lego(A).
```

```
%% B
lego_builder(alice).
lego_builder(bob).
estate_agent(claire).
estate_agent(dave).
golfer(edith).
enjoys_lego(alice).
enjoys_lego(claire).
knows(fred,alice).
knows(claire,bob).
knows(dave,bob).
knows(edith,bob).
knows(dave,claire).
knows(alice,edith).
knows(bob,edith).
knows(A,A).
knows(A,B):-  knows(B,A).
```

```
%% E+
happy(alice).
happy(fred).

%% E-
happy(bob).
happy(claire).
happy(dave).
happy(edith).
```

```
%% H
happy(A):-
   lego_builder(A),
   enjoys_lego(A).
```

```
%% B
lego_builder(alice).
lego_builder(bob).
estate_agent(claire).
estate_agent(dave).
golfer(edith).
enjoys_lego(alice).
enjoys_lego(claire).
knows(fred,alice).
knows(claire,bob).
knows(dave,bob).
knows(edith,bob).
knows(dave,claire).
knows(alice,edith).
knows(bob,edith).
knows(A,A).
knows(A,B):-  knows(B,A).
```

```
%% E+
happy(alice).
happy(fred).
```

```
%% E-
happy(bob).
happy(claire).
happy(dave).
happy(edith).
```

```
%% H
happy(A):-
  lego_builder(A),
  enjoys_lego(A).
```

```
%% B
lego_builder(alice).
lego_builder(bob).
estate_agent(claire).
estate_agent(dave).
golfer(edith).
enjoys_lego(alice).
enjoys_lego(claire).
knows(fred,alice).
knows(claire,bob).
knows(dave,bob).
knows(edith,bob).
knows(dave,claire).
knows(alice,edith).
knows(bob,edith).
knows(A,A).
knows(A,B):-  knows(B,A).
```

```
%% E+
happy(alice).
happy(fred).

%% E-
happy(bob).
happy(claire).
happy(dave).
happy(edith).
```

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

```
%% B
lego_builder(alice).
lego_builder(bob).
estate_agent(claire).
estate_agent(dave).
golfer(edith).
enjoys_lego(alice).
enjoys_lego(claire).
knows(fred,alice).
knows(claire,bob).
knows(dave,bob).
knows(edith,bob).
knows(dave,claire).
knows(alice,edith).
knows(bob,edith).
knows(A,A).
knows(A,B):-  knows(B,A).
```

```
%% E+
happy(alice).
happy(fred).

%% E-
happy(bob).
happy(claire).
happy(dave).
happy(edith).
```

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

```
%% B
lego_builder(alice).
lego_builder(bob).
estate_agent(claire).
estate_agent(dave).
golfer(edith).
enjoys_lego(alice).
enjoys_lego(claire).
knows(fred,alice).
knows(claire,bob).
knows(dave,bob).
knows(edith,bob).
knows(dave,claire).
knows(alice,edith).
knows(bob,edith).
knows(A,A).
knows(A,B):-  knows(B,A).
```

```
%% E+
happy(alice).
happy(fred).

%% E-
happy(bob).
happy(claire).
happy(dave).
happy(edith).
```

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

```
%% B
lego_builder(alice).
lego_builder(bob).
estate_agent(claire).
estate_agent(dave).
golfer(edith).
enjoys_lego(alice).
enjoys_lego(claire).
knows(fred,alice).
knows(claire,bob).
knows(dave,bob).
knows(edith,bob).
knows(dave,claire).
knows(alice,edith).
knows(bob,edith).
knows(A,A).
knows(A,B):-  knows(B,A).
```

```
%% E+
happy(alice).
happy(fred).
```

```
%% E-
happy(bob).
happy(claire).
happy(dave).
happy(edith).
```

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

```
%% B
lego_builder(alice).
lego_builder(bob).
estate_agent(claire).
estate_agent(dave).
golfer(edith).
enjoys_lego(alice).
enjoys_lego(claire).
knows(fred,alice).
knows(claire,bob).
knows(dave,bob).
knows(edith,bob).
knows(dave,claire).
knows(alice,edith).
knows(bob,edith).
knows(A,A).
knows(A,B):-  knows(B,A).
```

```
%% E+
happy(alice).  ✓
happy(fred).   ✓

%% E-
happy(bob).    ✓
happy(claire). ✓
happy(dave).   ✓
happy(edith).  ✓
```

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

From learning rules to learning programs

input	output
cat	c
dog	d
bear	**?**

input	output
cat	c
dog	d
bear	**b**

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

input	output
cat	c
dog	d
bear	**b**

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

input	output
cat	a
dog	o
bear	**?**

input	output
cat	a
dog	o
bear	**e**

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

input	output
cat	a
dog	o
bear	**e**

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

input	output
dog	g
sheep	p
chicken	**?**

input	output
dog	g
sheep	p
chicken	**n**

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

input	output
dog	g
sheep	p
chicken	**n**

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

input	output
ecv	cat
fqi	dog
iqqug	**?**

input	output
ecv	cat
fqi	dog
iqqug	**goose**

input	output
ecv	cat
fqi	dog
iqqug	**goose**

```
f(A,B):-
    map(f1,A,B).
f1(A,B):-
    char_code(A,C),
    succ(D,C),
    succ(E,D),
    char_code(B,E).
```

input	output
ecv	cat
fqi	dog
iqqug	**goose**

Higher-order abstraction

Invented symbol

```
f(A,B):-
    map(f1,A,B).
f1(A,B):-
    char_code(A,C),
    succ(D,C),
    succ(E,D),
    char_code(B,E).
```

Cropper et al. MLJ2020

Outline

1. Intro
2. Logic 101
3. ILP problem
4. **ILP techniques**
5. ILP features
6. Applications
7. Future directions

Building an ILP system

Building an ILP system

1. Learning setting
   - Examples: atoms, proofs, interpretations, …

Building an ILP system

1. Learning setting
   - Examples: atoms, proofs, interpretations, …
   - Hypotheses: Datalog, definite, ASP, …

Building an ILP system

1. Learning setting
   - Examples: atoms, proofs, interpretations, …
   - Hypotheses: Datalog, definite, ASP, …

2. Language bias*, i.e. defining legal hypotheses

*Please ignore nonsensical papers claiming to learn without a language bias

Building an ILP system

1. Learning setting
   - Examples: atoms, proofs, interpretations, …
   - Hypotheses: Datalog, definite, ASP, …

2. Language bias*, i.e. defining legal hypotheses

3. How to search the hypothesis space

Building an ILP system

1. Learning setting
   - Examples: atoms, proofs, interpretations, …
   - Hypotheses: Datalog, definite, ASP, …

2. Language bias*, i.e. defining legal hypotheses

3. How to search the hypothesis space

Top-down

Specialise a general program

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).
```

Top-down

Advantages	Disadvantages
• Recursion	• Inefficient

FOIL, TILDE, HYPER, Progol, Aleph**

Bottom-up

Generalise the examples

```
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

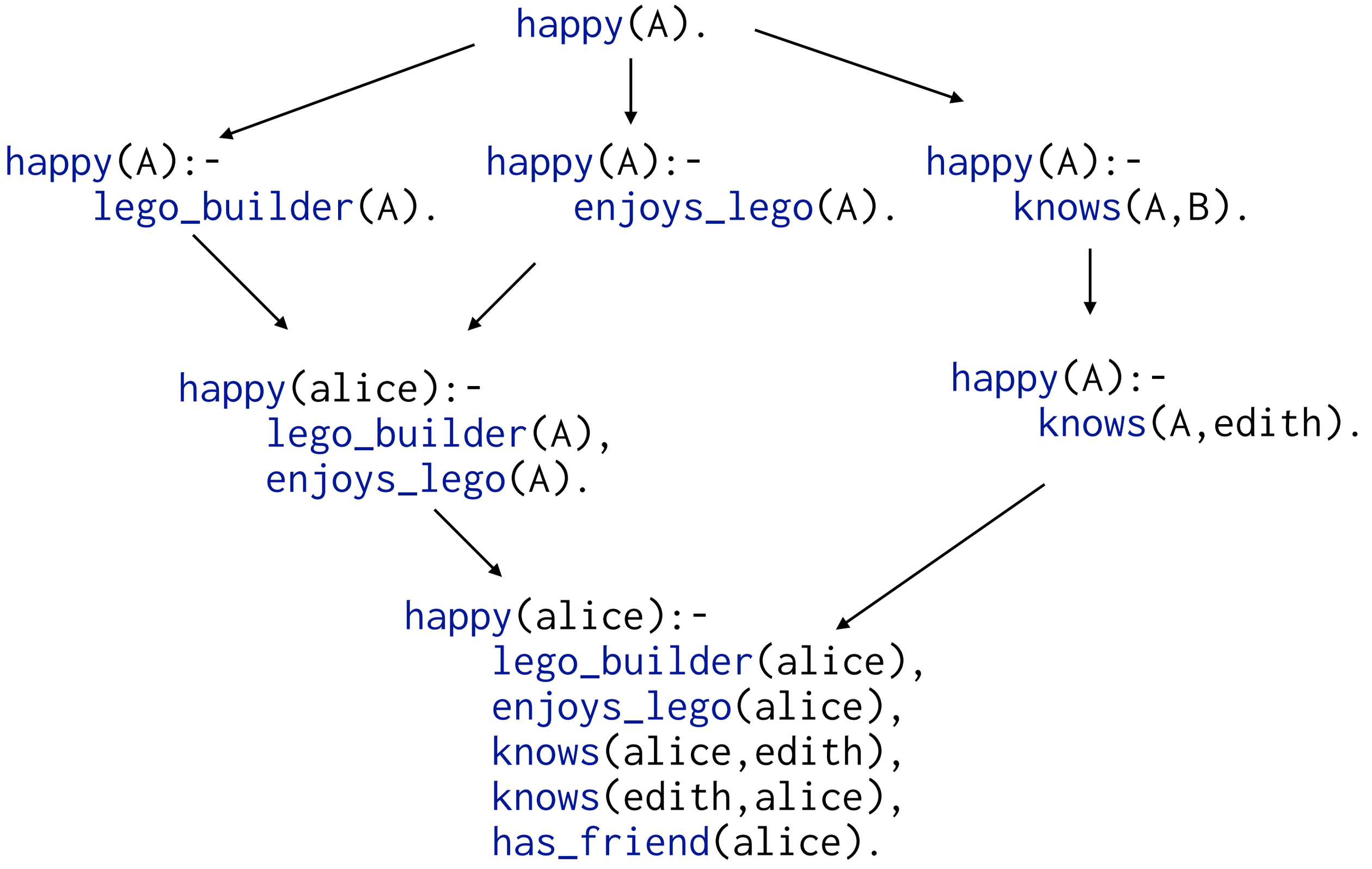
Bottom-up

Advantages	Disadvantages
• Fast • Constants	• Optimality • Recursion

CIGOL, GOLEM, LFIT, Progol, Aleph**

Meta-level

Delegate the search to something else


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).

```
happy(A).
```

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

```
happy(A):-
    enjoys_lego(A).
```

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

```
happy(A):-
    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).
```

Meta-level

Meta-level

Advantages	Disadvantages
• Recursion • Completeness • Optimality	• Small domains

ASPAL, Metagol, ILASP, HEXMIL, DILP, Apperception, Popper

Missing details

Lots of clever techniques to make the three approaches work

Outline

1. Intro
2. Logic 101
3. ILP problem
4. ILP techniques
5. **ILP features**
6. Applications
7. Future directions

Recursion

Old ILP

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

Old ILP

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

Old ILP

```
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).
```

Old ILP

```
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).
```

Old ILP

```
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).
```

Key point 1: difficult to learn because of its size
Key point 2: difficult to learn from little training data

New ILP

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

New ILP

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

New ILP

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

Key point 1: easier to learn because of its size
Key point 2: only need a couple of examples

Predicate invention

Automatically invent new symbols

Russel (2019) thinks it is the most important step needed for human-level AI

Old ILP

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

Old ILP

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

Old ILP

```
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).
```

Old ILP

```
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).
```

Old ILP

```
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).
```

Old ILP

```
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).
```

Key point 1: difficult to learn because of its size
Key point 2: difficult to learn from from little training data

New ILP

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

New ILP

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

Key point 1: easier to learn because of its size
Key point 2: only need a couple of examples

Higher-order invention

Input	Output
[alic**e**,bo**b**,charli**e**]	[alic,bo,charli]
[inductiv**e**,logi**c**,programmin**g**]	[inductiv,logi,programmin]
[ferrar**a**,orlean**s**,londo**n**,kyot**o**]	[ferrar,orlean,londo,kyot]

Higher-order invention

```
f(A,B):-map(A,B,inv1).
inv1(A,B):-inv2(A,C),tail(C,D),inv2(D,B).
inv2(A,B):-reduceback(A,B,concat).
```

Higher-order invention

reuses inv2

invents droplast

```
f(A,B):-map(A,B,inv1).
inv1(A,B):-inv2(A,C),tail(C,D),inv2(D,B).
inv2(A,B):-reduceback(A,B,concat).
```

invents reverse

Cropper et al. MLJ2020

Double droplasts

Input	Output
[alic**e**,bo**b**,**charlie**]	[alic,bo]
[inductiv**e**,logi**c**,**programming**]	[inductiv,logi]
[ferrar**a**,orlean**s**,londo**n**,**kyoto**]	[ferrar,orlean,londo]

Higher-order invention

invents droplast

reuses inv1

```
f(A,B):-map(A,C,inv1),inv1(C,B).
inv1(A,B):-inv2(A,C),tail(C,D),inv2(D,B).
inv2(A,B):-reduceback(A,B,concat).
```

Cropper et al. MLJ2020

Optimality

Which hypothesis?

- Shortest?
- Maximum coverage?
- Minimal description length?

Efficient programs

input	output
sheep	e
alaca	a
chicken	?

Efficient programs

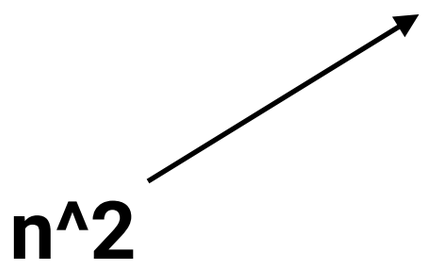
input	output
sheep	e
alaca	a
chicken	**c**

Efficient programs

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

Efficient programs

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



Cropper and Muggleton, IJCAI2015, MLJ2019

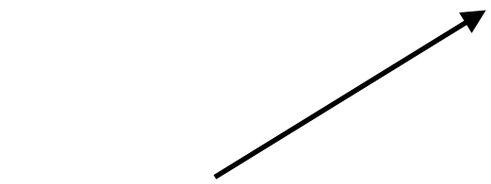
Efficient programs

```
f(A,B):- mergesort(A,C),inv1(C,B).
inv1(A,B):- head(A,B),tail(A,C),head(C,B).
inv1(A,B):- tail(A,C),inv1(C,B).
```

Cropper and Muggleton, IJCAI2015, MLJ2019

Efficient programs

```
f(A,B):- mergesort(A,C),inv1(C,B).
inv1(A,B):- head(A,B),tail(A,C),head(C,B).
inv1(A,B):- tail(A,C),inv1(C,B).
```



n log n

Cropper and Muggleton, IJCAI2015, MLJ2019

Knowledge transfer

task	input	output
f	philip.larkin@sj.ox.ac.uk	Philip Larkin

Knowledge transfer

task	input	output
f	philip.larkin@sj.ox.ac.uk	Philip Larkin

```
f(A,B):-
  inv1(A,C),skip1(C,D),space(D,E),
  inv1(E,F),skiprest(F,B).
inv1(A,B):-
  uppercase(A,C),copyword(C,B).
```

Knowledge transfer

task	input	output
f	philip.larkin@sj.ox.ac.uk	Philip Larkin

```
f(A,B):-
  inv1(A,C),skip1(C,D),space(D,E),
  inv1(E,F),skiprest(F,B).
inv1(A,B):-
  uppercase(A,C),copyword(C,B).
```

10 seconds*

*5 years ago

Knowledge transfer

task	input	output
g	tony	Tony

Knowledge transfer

task	input	output
g	tony	Tony

```
g(A,B):-uppercase(A,C),copyword(C,B).
```

Knowledge transfer

task	input	output
g	tony	Tony
f	philip.larkin@sj.ox.ac.uk	Philip Larkin

```
g(A,B):-uppercase(A,C),copyword(C,B).
```

Knowledge transfer

task	input	output
g	tony	Tony
f	philip.larkin@sj.ox.ac.uk	Philip Larkin

```
g(A,B):-uppercase(A,C),copyword(C,B).

f(A,B):-g(A,C),skip1(C,D),space(D,E),
     g(E,F),skiprest(F,B).
```

Knowledge transfer

task	input	output
g	tony	Tony
f	philip.larkin@sj.ox.ac.uk	Philip Larkin

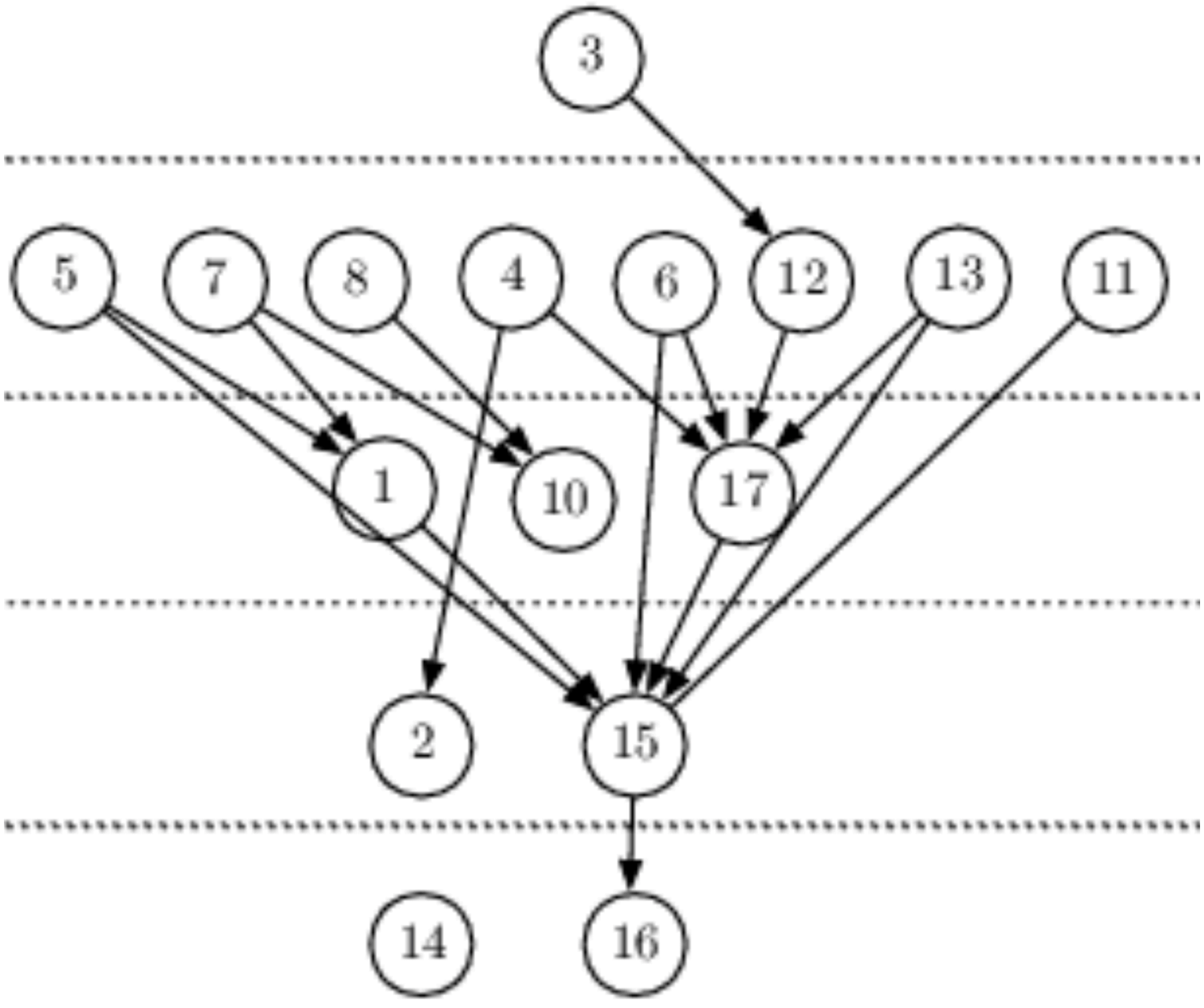
```
g(A,B):-uppercase(A,C),copyword(C,B).

f(A,B):-g(A,C),skip1(C,D),space(D,E),
     g(E,F),skiprest(F,B).
```

2 seconds*

Lin et al. ECAI2014, Cropper IJCAI2019, Cropper AAAI2020

Knowledge hierarchy



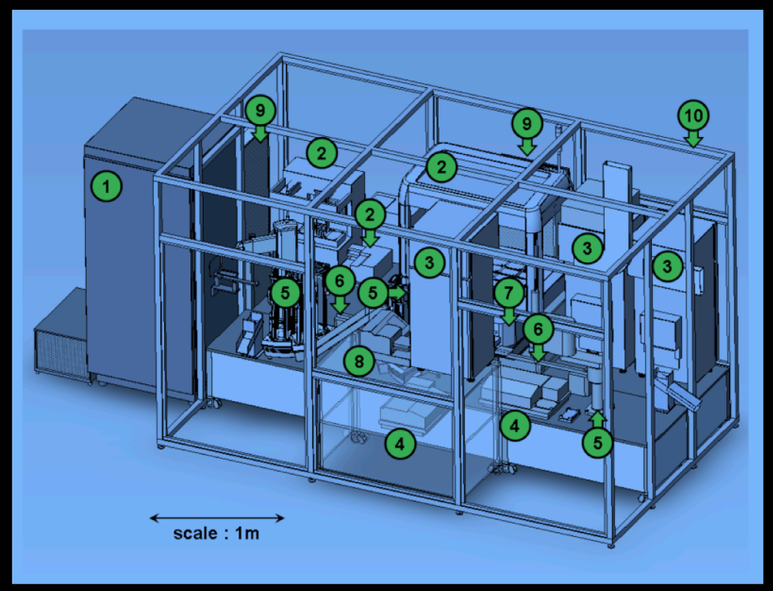
Lin et al. ECAI2014

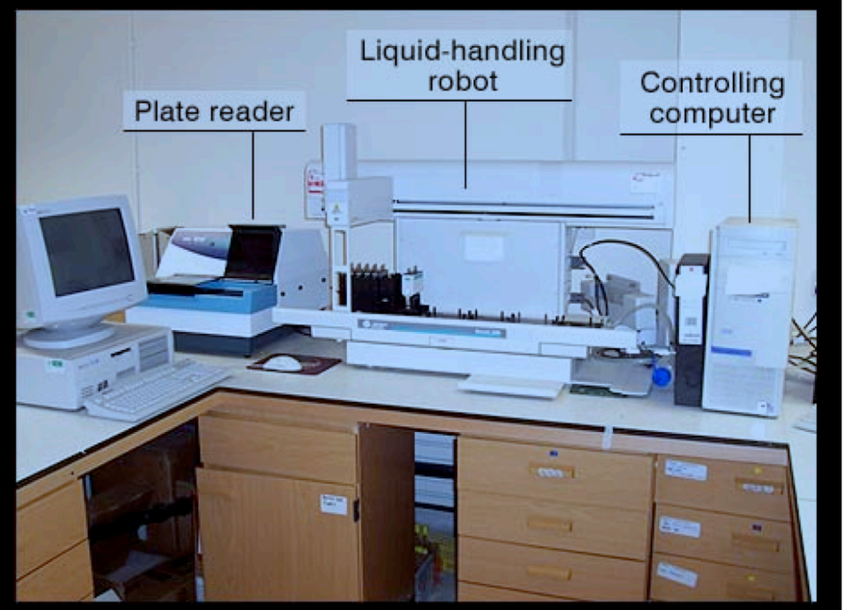
Outline

1. Intro
2. Logic 101
3. ILP problem
4. ILP techniques
5. ILP features
6. **Applications**
7. Future directions

Scientific discovery

Case 2: The Robot Scientist





REPORTS

The Automation of Science

Ross D. King,[1*] Jem Rowland,[1] Stephen G. Oliver,[2] Michael Young,[3] Wayne Aubrey,[1] Emma Byrne,[1] Maria Liakata,[1] Magdalena Markham,[1] Pinar Pir,[2] Larisa N. Soldatova,[1] Andrew Sparkes,[1] Kenneth E. Whelan,[1] Amanda Clare[1]

Science 2009

Functional genomic hypothesis generation and experimentation by a robot scientist

Ross D. King[1], Kenneth E. Whelan[1], Ffion M. Jones[1], Philip G. K. Reiser[1], Christopher H. Bryant[2], Stephen H. Muggleton[3], Douglas B. Kell[4] & Stephen G. Oliver[5]

Nature 2004

Slide taken from Luc De Raedt's slides from IJCAI2009

Visual reasoning

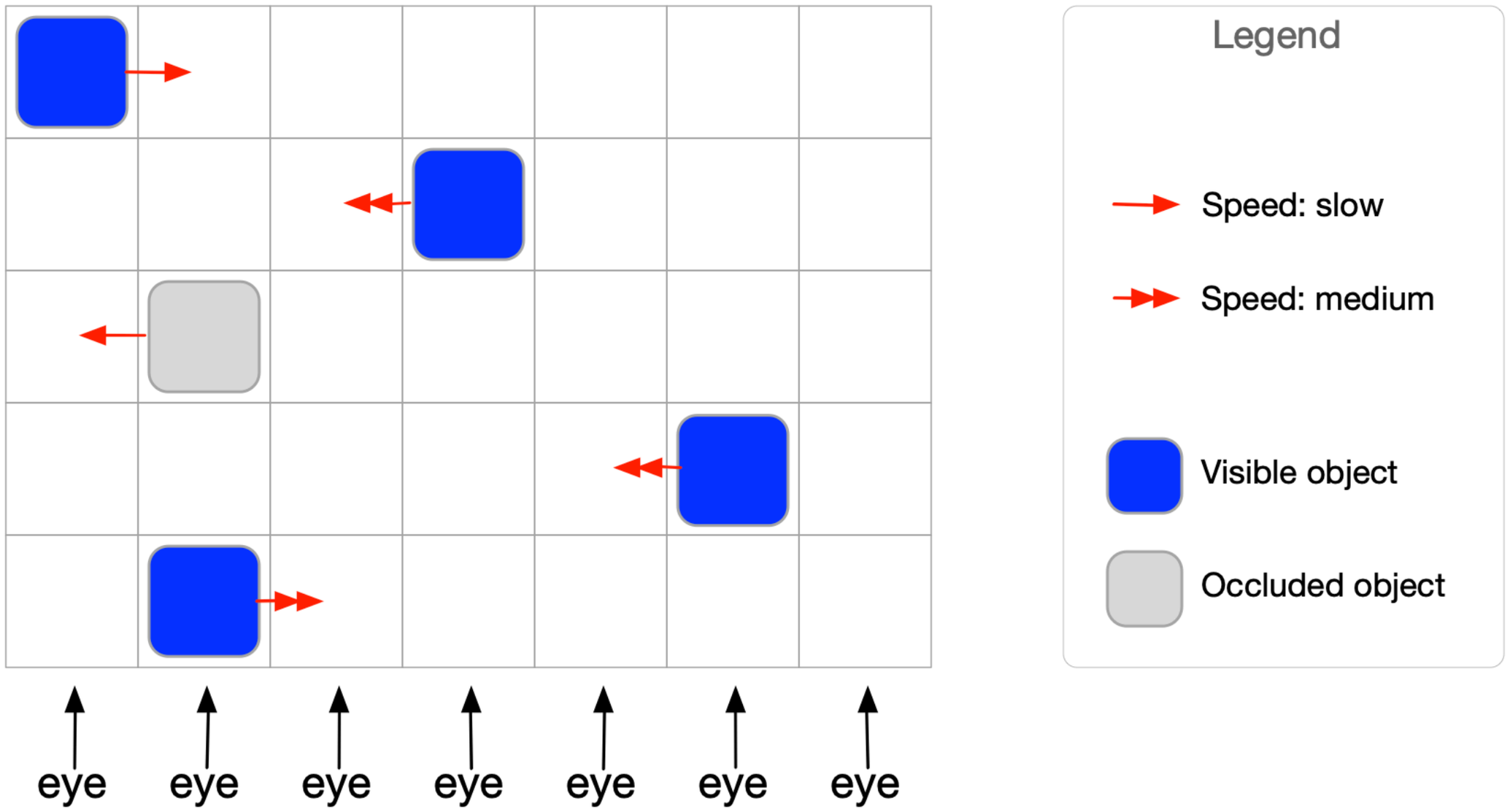
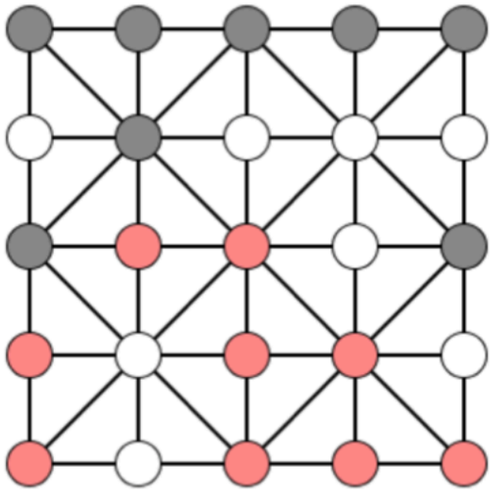
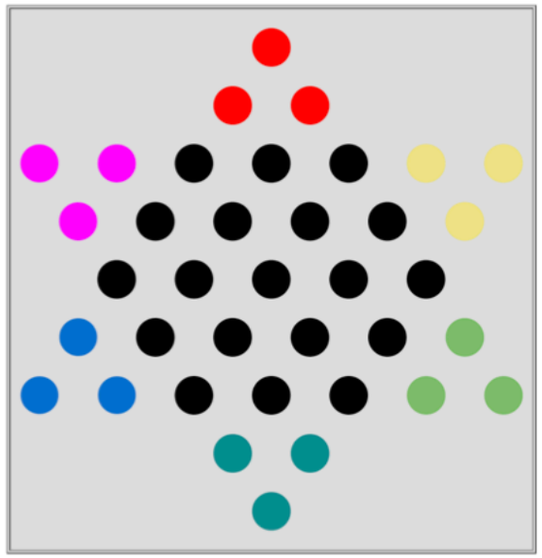


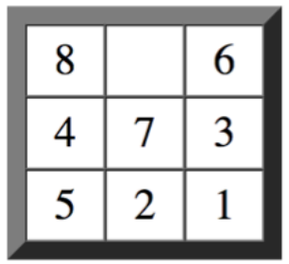
Figure 6: An occlusion task

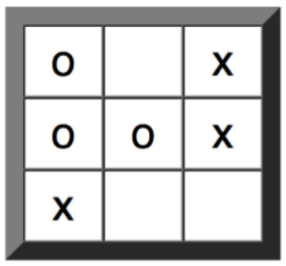
Evens et al. AIJ2021

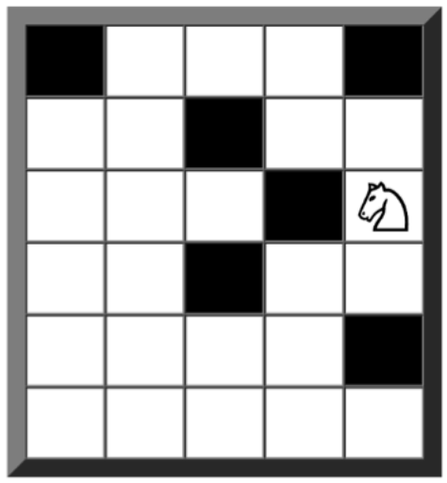
Games

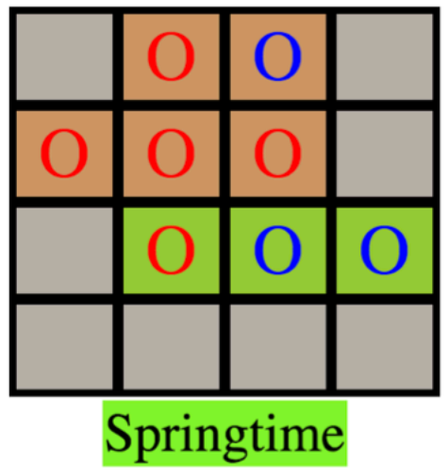












Cropper et al. MLJ2020

Outline

1. Intro
2. Logic 101
3. ILP problem
4. ILP techniques
5. ILP features
6. Applications
7. **Future directions**

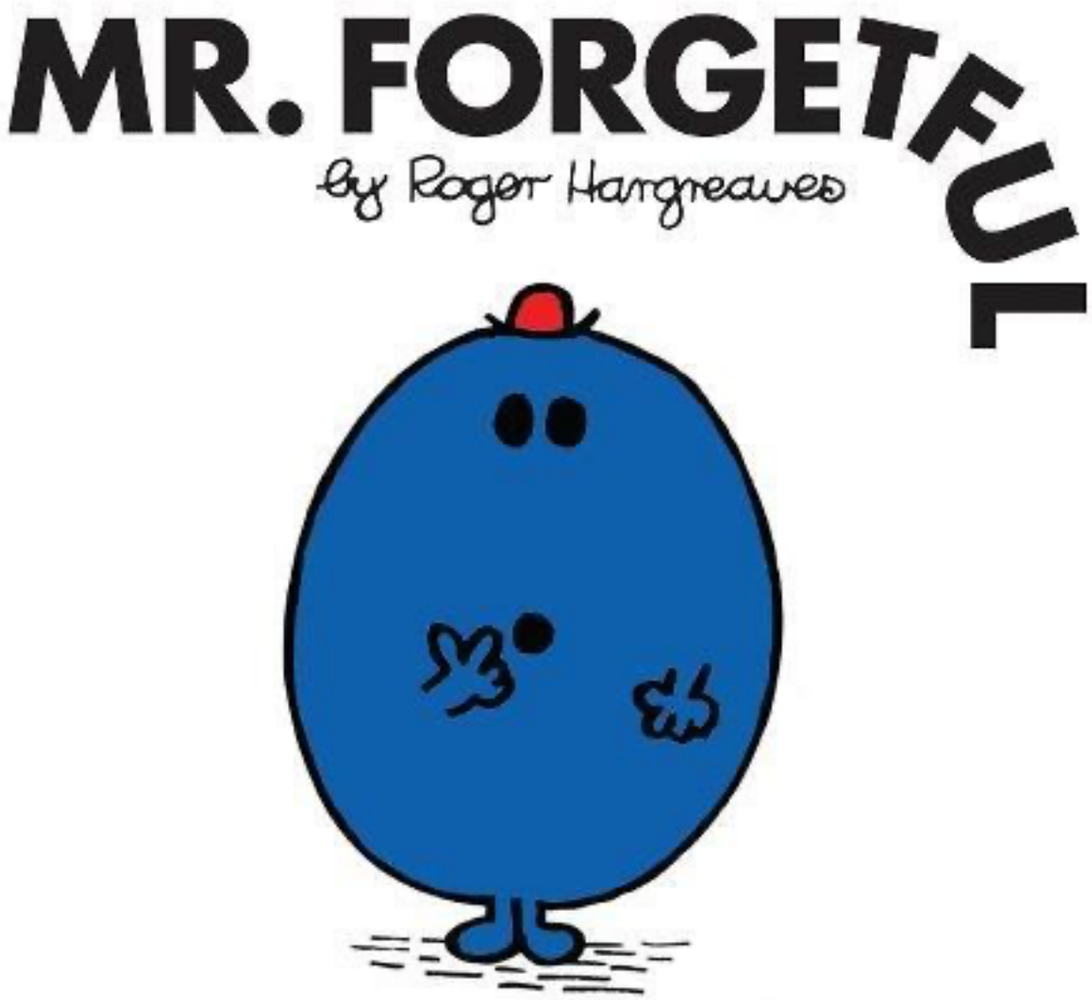
CogSci for ILP

How can we use the knowledge we have more efficiently?

Playing

Cropper IJCAI2019

Forgetting

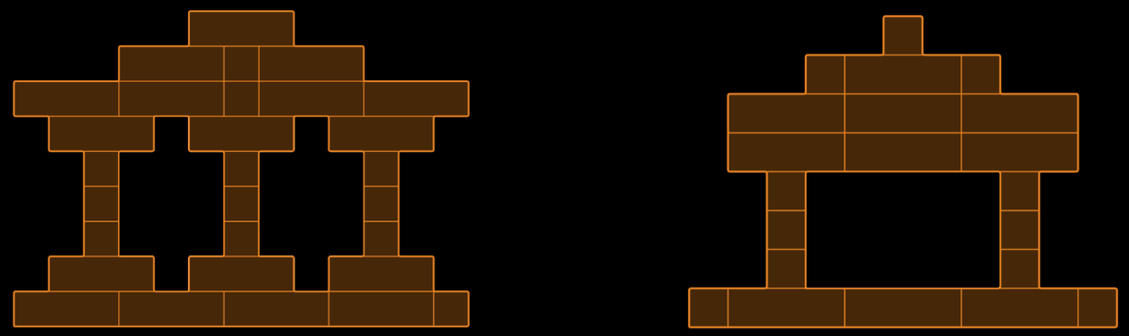


Cropper AAAI2020

Refactoring

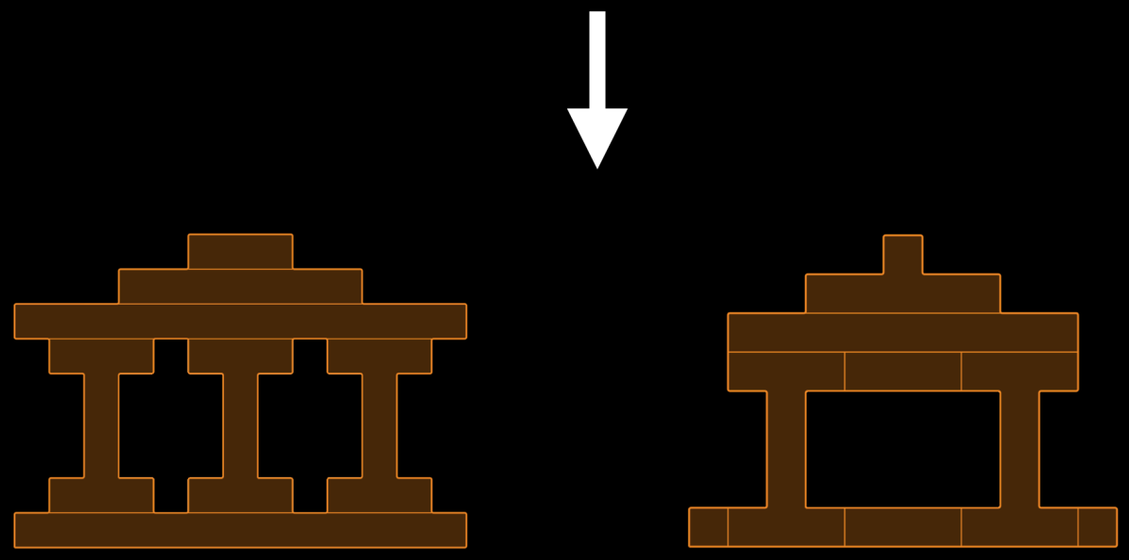
The knowledge refactoring problem

Agent's knowledge:



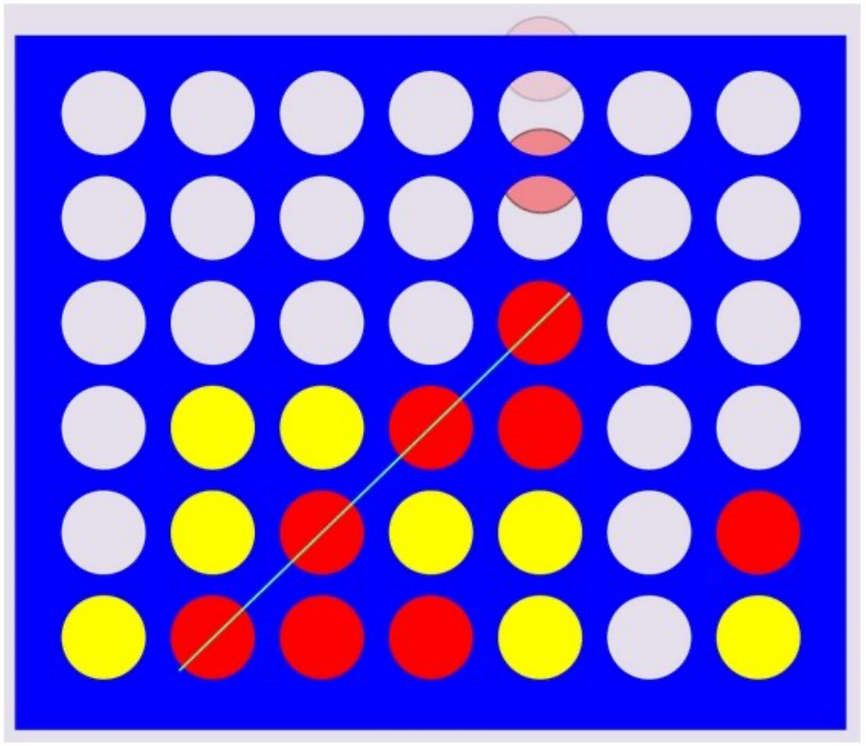
Knowledge refactoring:

How can we achieve the same goal but more efficiently?



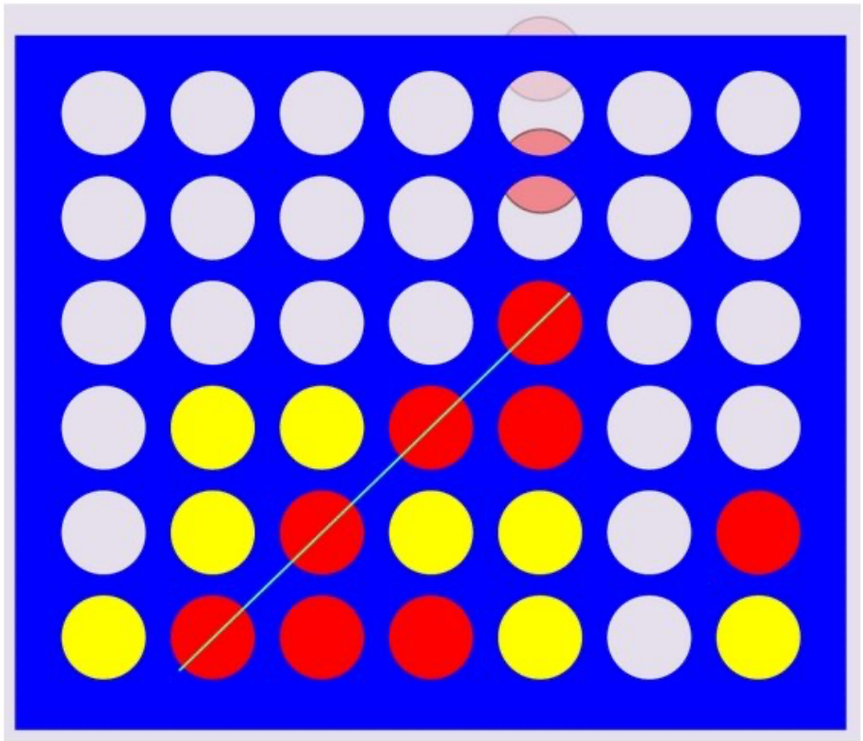
Dumancic et al. AAAI2021

Abstractions and invention



Cropper and Muggleton IJCAI2016, Cropper IJCAI2019, Cropper AAAI2020, Cropper et al. MLJ2020

Abstractions and invention



Cropper and Muggleton IJCAI2016, Cropper IJCAI2019, Cropper AAAI2020, Cropper et al. MLJ2020

ILP for CogSci

?

Summary

- ILP is a form of ML that uses logic to represent knowledge
- Allows humans to leverage their prior knowledge to learn **human-readable** programs from **small** data
- Potential for CogSci to advance ILP (and vice-versa?)

References

Inductive logic programming at 30: a new introduction. A. Cropper and S. Dumančić. arxiv

Turning 30: new ideas in inductive logic programming. A. Cropper, S. Dumančić, and S.H. Muggleton. IJCAI 2020

Inductive Logic Programming. S. Muggleton. New Gener. Comput. 1991.

Logical and relational learning. Luc De Raedt. Springer 2008.