Learning higher-order logic programs

Andrew Cropper, Rolf Morel, and Stephen Muggleton

Inductive logic programming

Inductive logic programming

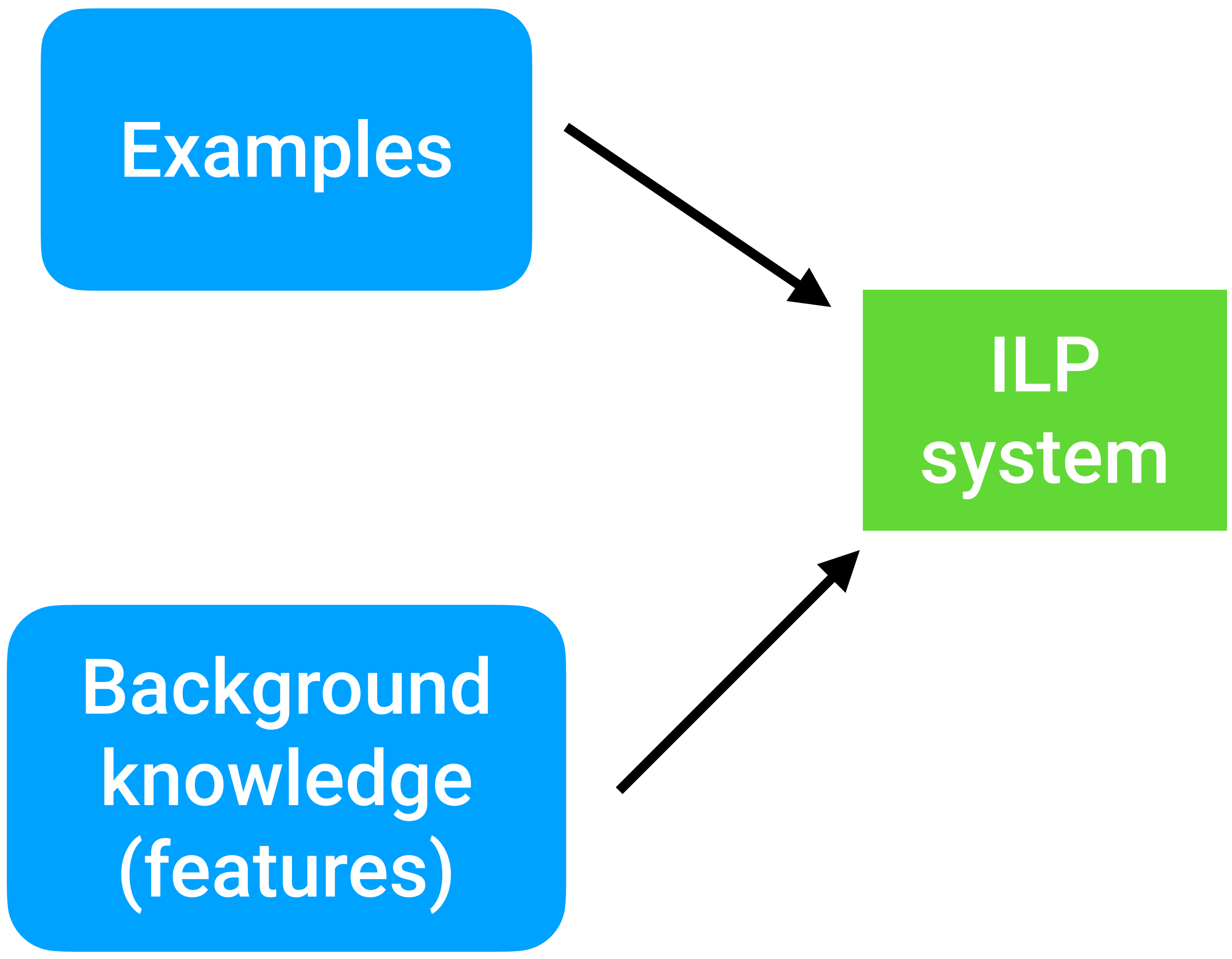
Examples

Inductive logic programming

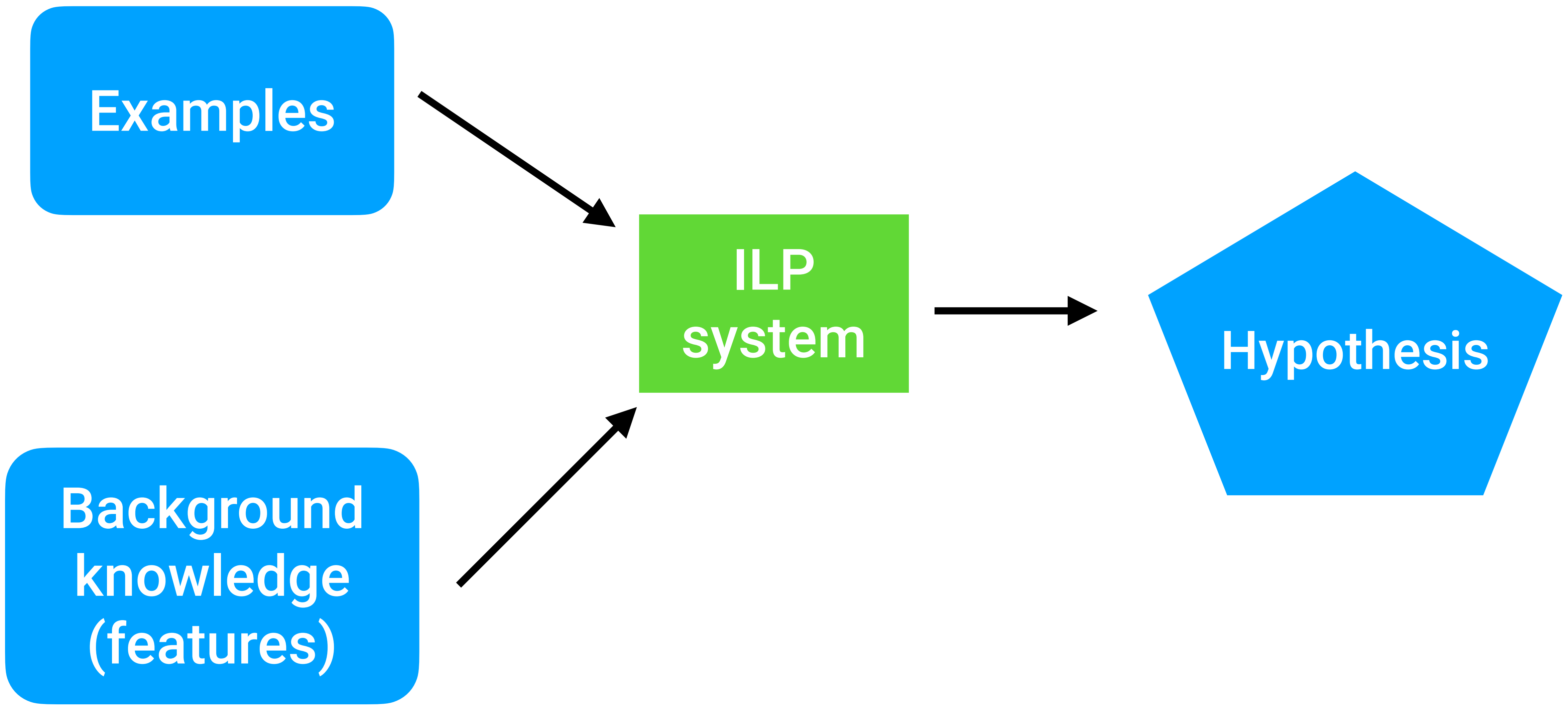
Examples

Background knowledge (features)

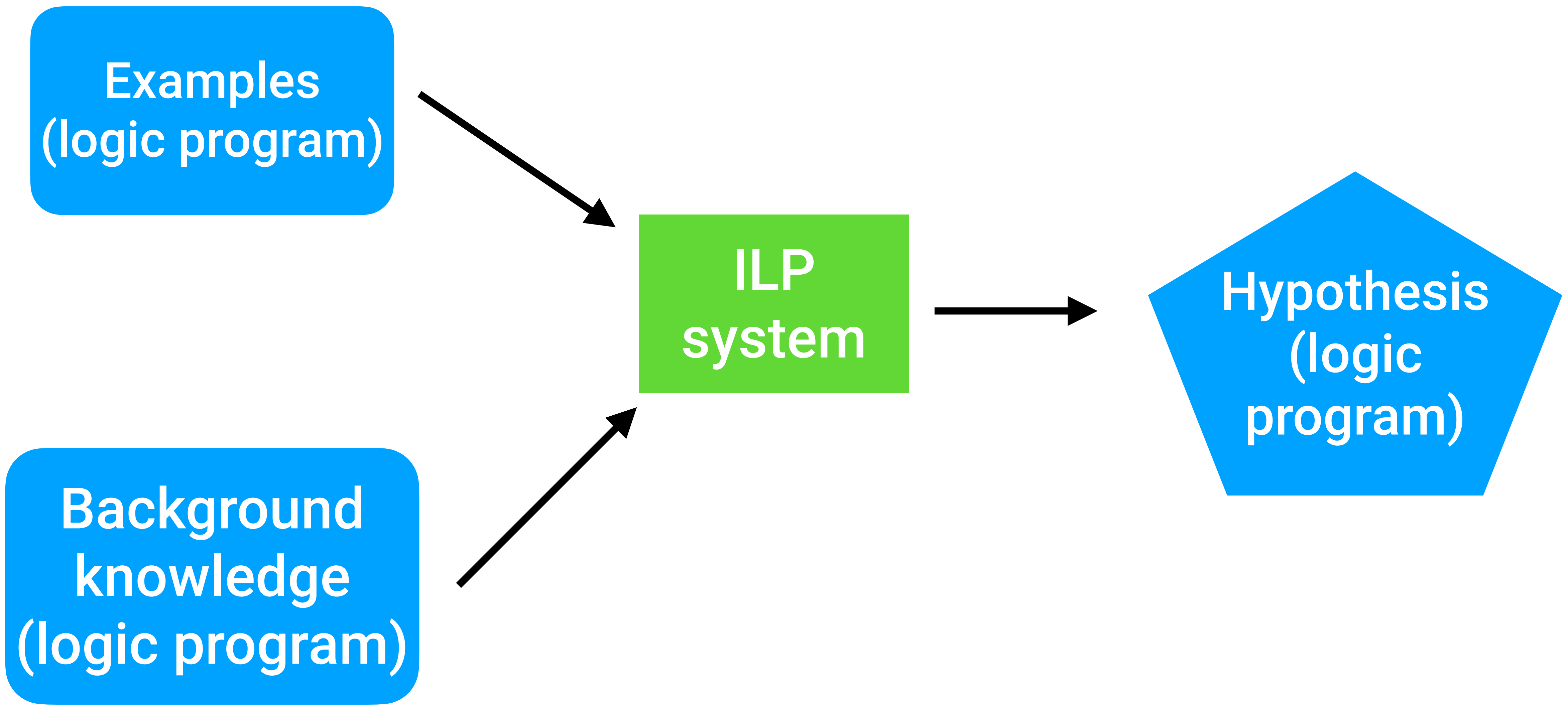
Inductive logic programming



Inductive logic programming



Inductive logic programming



Data are programs rather than tables/vectors

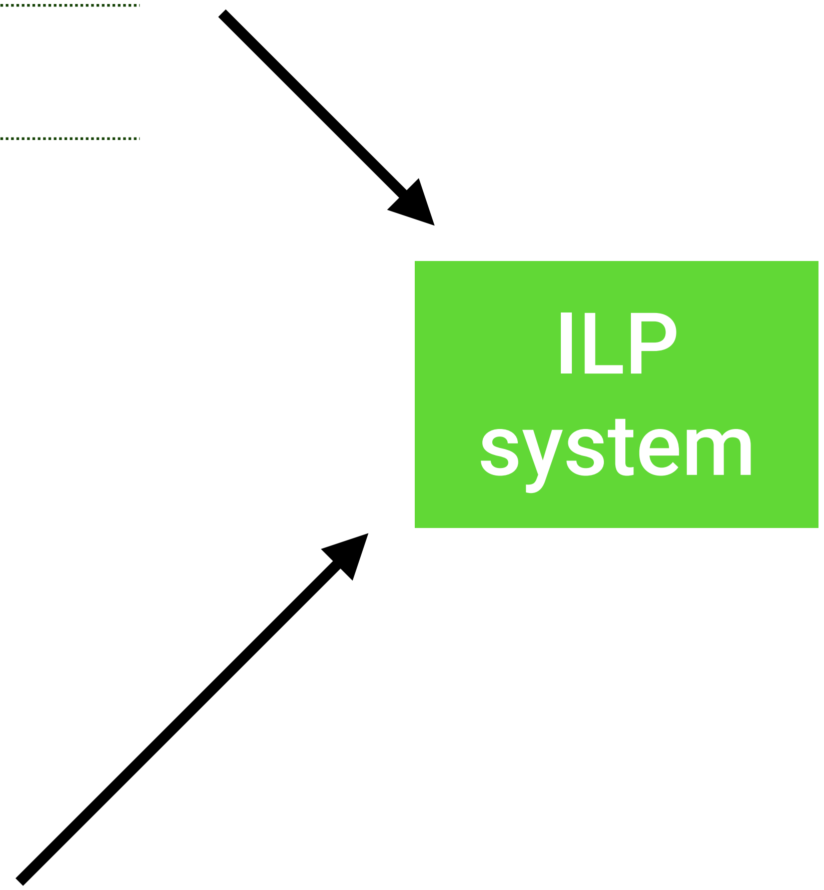
Examples
f(wooldridge,e)
f(calinescu,u)
f(worrell,l)

Examples
f(wooldridge,e)
f(calinescu,u)
f(worrell,l)

BK	
`empty([]).`	
`head([H|_],H).`	← *the first element (e.g.* `head(oxford,o)`*)*
`tail([_|T],T).`	← *everything but the first element (e.g.* `head(oxford,xford)`*)*

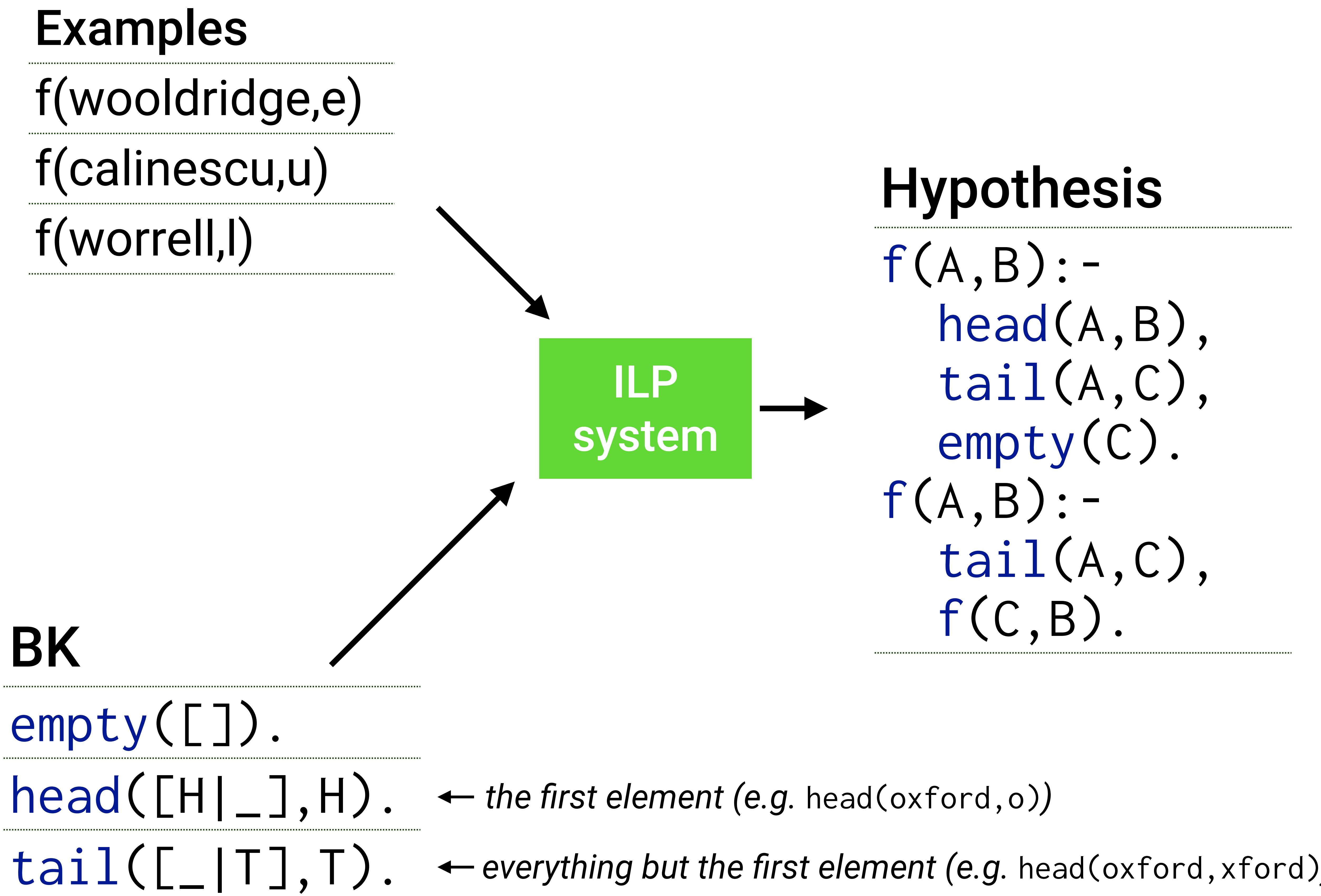
Examples

f(wooldridge,e)
f(calinescu,u)
f(worrell,l)



BK

`empty([]).`	
`head([H|_],H).`	← *the first element (e.g.* `head(oxford,o)`*)*
`tail([_|T],T).`	← *everything but the first element (e.g.* `head(oxford,xford)`*)*



Examples
f(wooldridge,e)
f(calinescu,u)
f(worrell,l)
ILP system
Hypothesis
f(A,B):-
head(A,B),
tail(A,C),
empty(C).
f(A,B):-
tail(A,C),
f(C,B).
BK
empty([]).
head([H|_],H).
tail([_|T],T).
the first element (e.g. head(oxford,o))
everything but the first element (e.g. head(oxford,xford))

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

“The last item is the head item if there is only one item”

or

“The last item is the last item in the tail”

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

```
?- f([o,x,f,o,r,d],X).
X = d .
```

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

```
?- f([o,x,f,o,r,d],X).
X = d .
```

Generalisation from small data!

Any high-level questions about ILP?

Novel contribution

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

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

First-order solution

```
f(A,B):-
    empty(A),
    empty(B).
f(A,B):-
    head(A,C),
    char_to_int(C,D),
    prec(D,E),
    int_to_char(E,F),
    head(B,F),
    tail(A,G),
    tail(B,H),
    f(G,H).
```

First-order solution (refactored)

```
f(A,B):-
    empty(A),
    empty(B).
f(A,B):-
    head(A,C),
    aux(C,F),
    head(B,F),
    tail(A,G),
    tail(B,H),
    f(G,H).
```

```
aux(A,B):-
    char_to_int(A,C),
    prec(C,D),
    int_to_char(D,B).
```

Boiler plate code

Idea

Learn higher-order programs

Higher-order solution

predicate symbol is an argument

```
f(A,B):-
    map(A,B,inv1).
inv1(A,B):-
    char_to_int(A,C),
    prec(C,D),
    int_to_char(D,B).
```

Higher-order BK

```
map([],[],_F).
map([A|As],[B|Bs],F):-
    call(F,A,B),
    map(As,Bs,F).
```

Why?

Reduce the size of the program

How?

Extend Metagol

Metagol

% background knowledge
succ/2
int_to_char/2
map/3

% positive example
f([1,2,3],[c,d,e])

% metarules
P(A,B) ← **Q**(A,C),**R**(C,B)
P(A,B) ← **Q**(A,B,**R**)

Metarules

$$\mathbf{P}(A,B) \leftarrow \mathbf{Q}(A,C), \mathbf{R}(C,B)$$

```
metarule(
  chain, % name
  [P,Q,R], % subs
  [P,A,B], % head
  [[Q,A,C],[R,C,B]] % body
).
```

Outer loop

```
learn(Pos,Neg,Prog):-
    prove(Pos,[],Prog),
    \+ prove(Neg,Prog,Prog).
```

Initial empty program

Prove each example (an atom)

```
prove([],Prog,Prog).
prove([Atom|Atoms],Prog1,Prog2):-
    prove_aux(Atom,Prog1,Prog3),
    prove(Atoms,Prog3,Prog2).
```

Prove by calling Prolog

```
prove_aux(Atom,Prog,Prog):-
    call(Atom).
```

Prove using a metarule

```
prove_aux(Atom,Prog1,Prog2):-
    metarule(Name,Subs,Atom,Body),
    bind(Subs),
    Prog3 = [sub(Name,Subs)|Prog1],
    prove(Body,Prog3,Prog2).
```

Find substitutions for the variables

```
% background knowledge
succ/2
int_to_char/2
map/3

% positive example
f([1,2,3],[c,d,e])

% metarules
```

$$\mathbf{P}(A,B) \leftarrow \mathbf{Q}(A,C),\mathbf{R}(C,B)$$

$$\mathbf{P}(A,B) \leftarrow \mathbf{Q}(A,B,\mathbf{R})$$

$\leftarrow f([1,2,3],[c,d,e])$

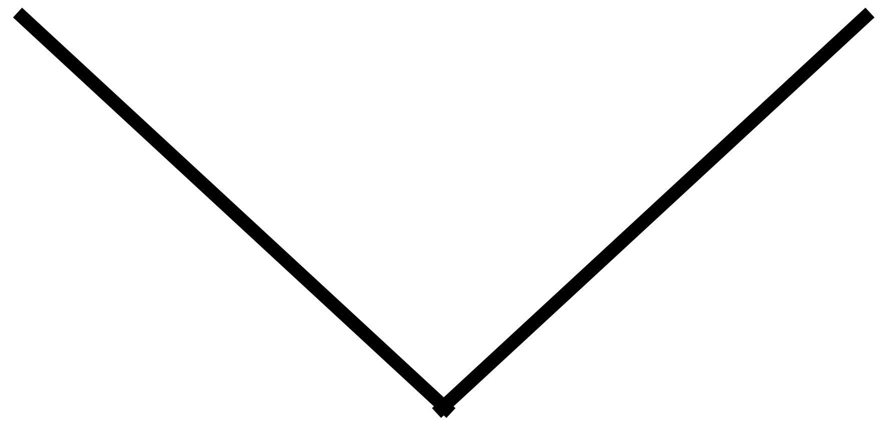
$\leftarrow f([1,2,3],[c,d,e])$ $\mathbf{P}(A,B) \leftarrow \mathbf{Q}(A,B,\mathbf{R})$

← f([1,2,3],[c,d,e])   **P**(A,B) ← **Q**(A,B,**R**)

← **Q**([1,2,3],[c,d,e],**R**)

`{P/f,A/[1,2,3],B/[c,d,e]}`

← **Q**([1,2,3],[c,d,e],**R**)



← **Q**([1,2,3],[c,d,e],**R**)

% proof fails

map([1,2,3],[c,d,e],succ) ✗

map([1,2,3],[c,d,e],int_to_char) ✗

Metagol solution

```
f(A,B):-f1(A,C),f3(C,B)
f1(A,B):-f2(A,C),f2(C,B).
f2(A,B):-map(A,B,succ).
f3(A,B):-map(A,B,int_to_char).
```

Metagol solution (refactored)

```
f(A,B):-
    map(A,C,succ).
    map(C,D,succ).
    map(D,B,int_to_char).
```

Higher-order definitions

```
ibk(
    [map,[],[],_F], % head
    [] % body
).

ibk(
    [map,[A|As],[B|Bs],F], % head
    [[F,A,B],[map,As,Bs,F]] % body
).
```

Metagol$_{HO}$

```
prove_aux(Atom,Prog1,Prog2):-
    ibk(Atom,Body),
    prove(Body,Prog1,Prog2).
```

% background
succ/2, int_to_char/2

% ibk
map/3

% example
f([1,2,3],[c,d,e])

% metarule
P(A,B) ← **Q**(A,C),**R**(C,B)
P(A,B) ← **Q**(A,B,**R**)

← f([1,2,3],[c,d,e])

$\leftarrow f([1,2,3],[c,d,e])$

$\mathbf{P}(A,B) \leftarrow \mathbf{Q}(A,B,\mathbf{R})$

← f([1,2,3],[c,d,e]) **P**(A,B) ← **Q**(A,B,**R**)

← **Q**([1,2,3],[c,d,e],**R**)

`{P/f,A/[1,2,3],B/[c,d,e]}`

← **Q**([1,2,3],[c,d,e],**R**)

$\leftarrow \mathbf{Q}([1,2,3],[c,d,e],\mathbf{R})$ map([A|As],[B|Bs],**R**) ← …

← **Q**([1,2,3],[c,d,e],**R**)   map([A|As],[B|Bs],**R**) ← …

←**R**(1,c), **R**(2,d), **R**(3,e)

`{Q/map,A/1,AS/[2,3],B/c,…}`

$\leftarrow \mathbf{R}(1,c), \mathbf{R}(2,d), \mathbf{R}(3,e)$

$\leftarrow \mathbf{R}(1,c),\ \mathbf{R}(2,d),\ \mathbf{R}(3,e)$ $\qquad$ $\mathbf{S}(A,B) \leftarrow \mathbf{T}(A,C), \mathbf{U}(C,B)$

← **R**(1,c), **R**(2,d), **R**(3,e)　　　**S**(A,B) ← **T**(A,C),**U**(C,B)

← **T**(1,C),**U**(C,c),**R**(2,d),**R**(3,e)

Metagol$_{HO}$ solution

```
f(A,B):-map(A,B,f1).
f1(A,B):-succ(A,C),f2(C,B).
f2(A,B):-succ(A,C),int_to_char(C,B).
```

Metagol$_{HO}$ solution (refactored)

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

Any questions about the approach?

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

Metagol solution

```
f(A,B):-f1(A,B),f5(A,B).
f1(A,B):-head(A,C),f2(C,B).
f2(A,B):-head(B,C),f3(A,C).
f3(A,B):-char_to_int(A,C),f4(C,B).
f4(A,B):-prec(A,C),int_to_char(C,B),
f5(A,B):-tail(A,C),f6(C,B).
f6(A,B):-tail(B,C),f(A,C).
```

Metagol solution

```
f(A,B):-f1(A,B),f5(A,B).
f1(A,B):-head[illegible],C),f[illegible]C,B).
f2(A,B):-head([illegible]C)[illegible](A,C).
f3(A,B):-char_to[illegible]t(A,C),f4(C,B).
f4(A,B):-prec([illegible]C)[illegible]t_to_char(C,B),
f5(A,B):-tail[illegible],C),f[illegible]C,B).
f6(A,B):-tail(B,C),f(A,C).
```

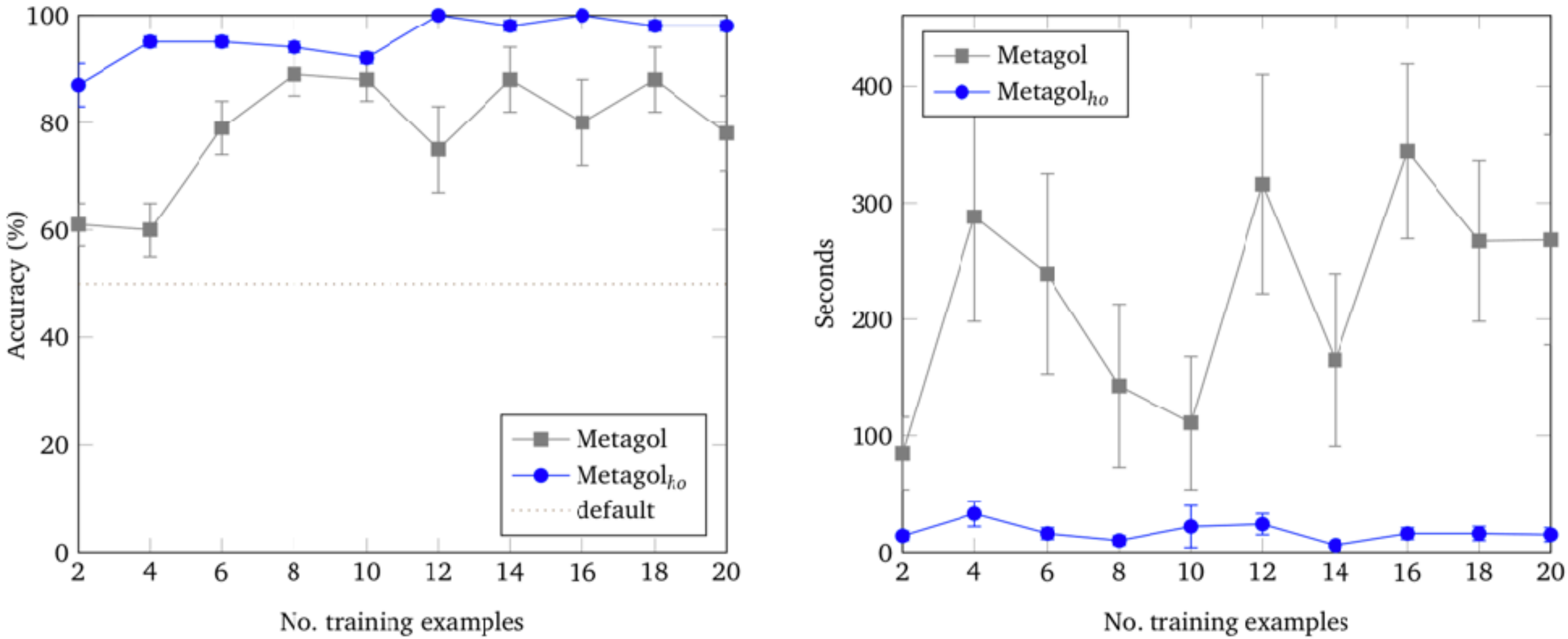
Metagol$_{HO}$

```
f(A,B):-map(A,B,f1).
f1(A,B):-char_to_int(A,C),f2(C,B).
f2(A,B):-prec(A,C),int_to_char(C,B).
```

Does it help in practice?

Q. Can learning higher-order programs improve performance?

Robot waiter



(a) Predictive accuracies

(b) Learning times

Robot waiter - Metagol

```
f(A,B):-turn_cup_over(A,C),f1(C,B).
f1(A,B):-move_right(A,B),at_end(B).
f1(A,B):-f2(A,C),f1(C,B).
f2(A,B):-wants_coffee(A),pour_coffee(A,B).
f2(A,B):-move_right(A,C),turn_cup_over(C,B).
f2(A,B):-wants_tea(A),pour_tea(A,B).
```

Robot waiter - Metagol$_{HO}$

```
f(A,B):-until(A,B,at_end,f1).
f1(A,B):-turn_cup_over(A,C),f2(C,B).
f2(A,B):-f3(A,C),move_right(C,B).
f3(A,B):-ite(A,B,wants_coffee,pour_coffee,pour_tea).
```

Droplasts

Input	Output
[alice,bob,charlie]	[alic,bo,charli]
[inductive,logic,programming]	[inductiv,logi,programmin]
[ferrara,orleans,london,kyoto]	[ferrar,orlean,londo,kyot]

Metagol$_{HO}$ solution

```
f(A,B):-map(A,B,f1).
f1(A,B):-f2(A,C),f3(C,B).
f2(A,B):-f3(A,C),tail(C,B).
f3(A,B):-reduceback(A,B,concat).
```

Metagol$_{HO}$ solution

```
f(A,B):-map(A,B,f1).
f1(A,B):-f2(A,C),tail(C,D),f2(D,B).
f2(A,B):-reduceback(A,B,concat).
```

Double droplasts

Input	Output
[alice,bob,charlie]	[alic,bo]
[inductive,logic,programming]	[inductiv,logi]
[ferrara,orleans,london,kyoto]	[ferrar,orlean,londo]

Metagol$_{HO}$ solution

```
f(A,B):-f1(A,C),f2(C,B).
f1(A,B):-map(A,B,f2).
f2(A,B):-f3(A,C),f4(C,B).
f3(A,B):-f4(A,C),tail(C,B).
f4(A,B):-reduceback(A,B,concat).
```

Metagol$_{HO}$ solution (refactored)

```
f(A,B):-map(A,C,f1),f1(C,B).
f1(A,B):-f2(A,C),tail(C,D),f2(D,B).
f2(A,B):-reduceback(A,B,concat).
```

Conclusions

- Learning higher-order programs can improve learning performance
- Approach needs predicate invention

Limitations

- Inefficient search
- Which metarules?
- Which higher-order definitions?

The future?

Machine Learning
https://doi.org/10.1007/s10994-020-05934-z

Learning programs by learning from failures

Andrew Cropper[1] · Rolf Morel[1]

Received: 5 May 2020 / Revised: 22 November 2020 / Accepted: 2 December 2020

Abstract
We describe an inductive logic programming (ILP) approach called *learning from failures*. In this approach, an ILP system (the learner) decomposes the learning problem into three separate stages: *generate*, *test*, and *constrain*. In the generate stage, the learner generates a hypothesis (a logic program) that satisfies a set of *hypothesis constraints* (constraints on the syntactic form of hypotheses). In the test stage, the learner tests the hypothesis against training examples. A hypothesis *fails* when it does not entail all the positive examples or entails a negative example. If a hypothesis fails, then, in the constrain stage, the learner learns constraints from the failed hypothesis to prune the hypothesis space, i.e. to constrain subsequent hypothesis generation. For instance, if a hypothesis is too general (entails a negative example), the constraints prune generalisations of the hypothesis. If a hypothesis is too specific (does not entail all the positive examples), the constraints prune specialisations of the hypothesis. This loop repeats until either (i) the learner finds a hypothesis that entails

References

Learning higher-order logic programs A. Cropper, R. Morel, and S.H. Muggleton Machine learning 2020

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