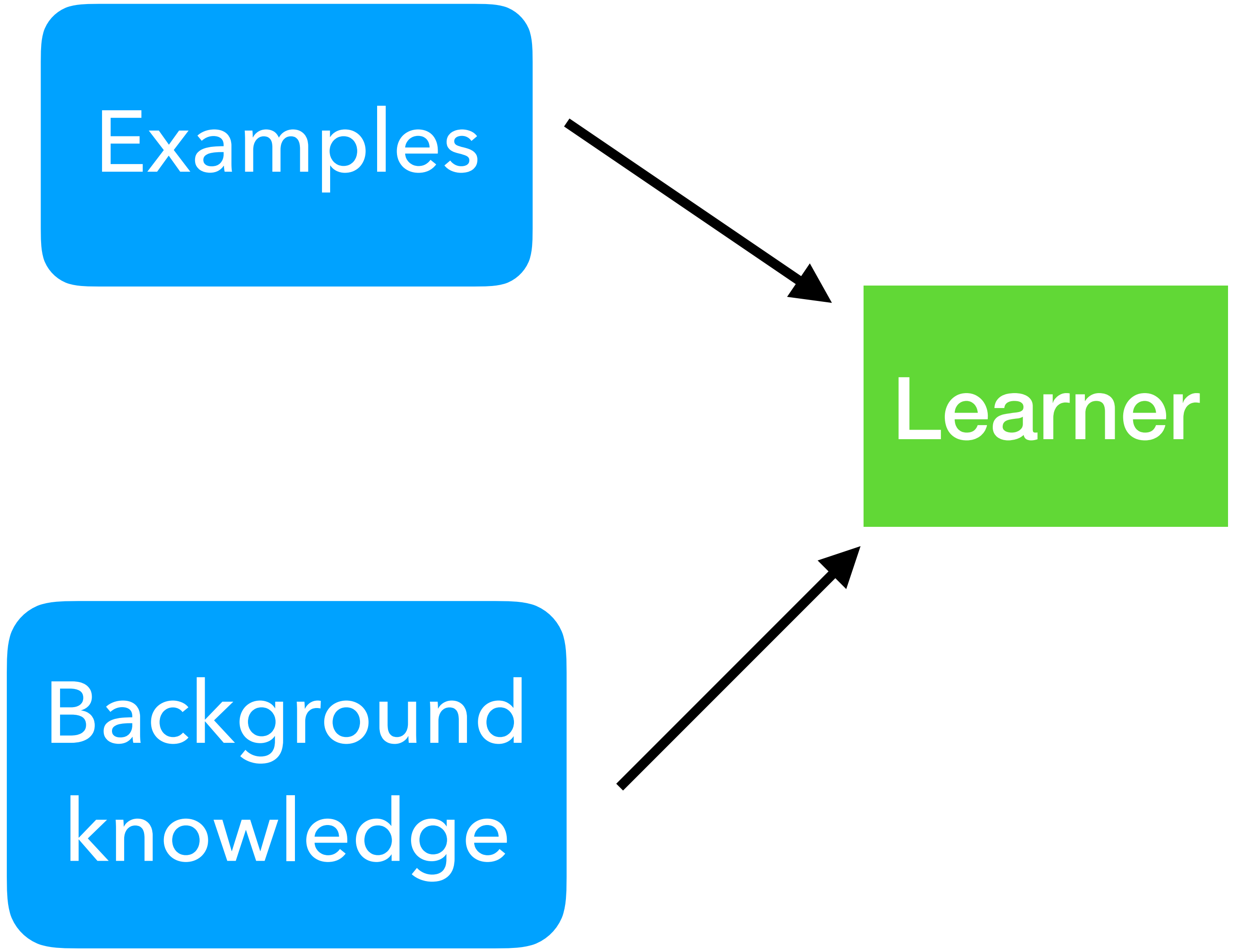
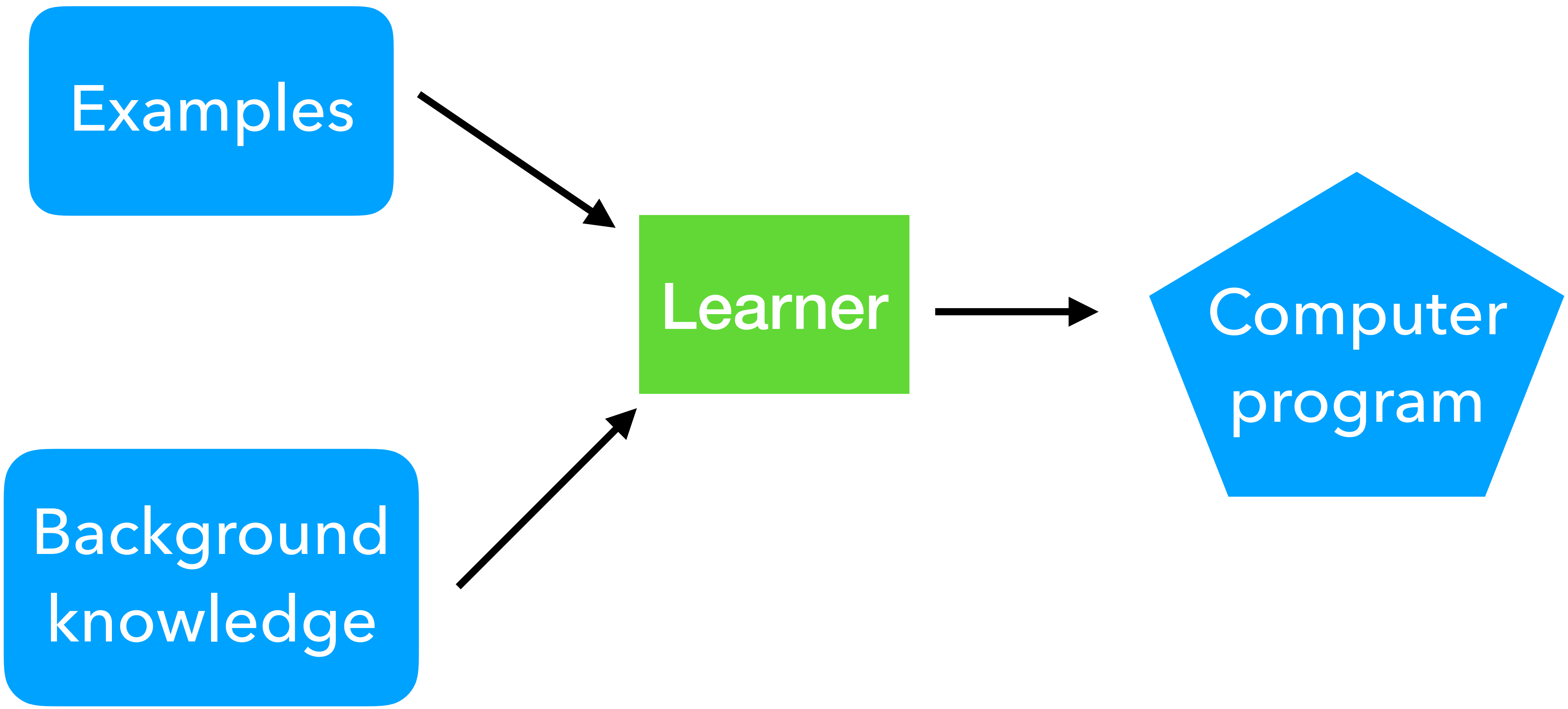
Learning higher-order logic programs

Andrew Cropper, Rolf Morel, and Stephen Muggleton

Program induction/synthesis



Program induction/synthesis



Examples

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

Examples

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

Background knowledge

head

tail

empty

Examples

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

Background knowledge

head
tail
empty

```
def f(a):
    t = tail(a)
    if empty(t):
        return head(a)
    return f(t)
```

Examples

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

Background knowledge

head

tail

empty

```
def f(a):
    t = tail(a)
    if empty(t):
        return head(a)
    return f(t)
```

Examples

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

Background knowledge

head

tail

empty

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

input	output
dbu	cat
eph	dog
hpptf	?

input	**output**
dbu	cat
eph	dog
hpptf	**goose**

base case

```
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).
```

inductive case

```
f(A,B):-
    empty(A),
    empty(B).
f(A,B):-
    head(A,C),
    f1(C,F),
    head(B,F),
    tail(A,G),
    tail(B,H),
    f(G,H).
```

list manipulation

```
f1(A,B):-
    char_to_int(A,C),
    prec(C,D),
    int_to_char(D,B).
```

cool stuff

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


```
    tail(A,G),
    tail(B,H),
    f(G,H).
```

```
f1(A,B):-
    char_to_int(A,C),
    prec(C,D),
    int_to_char(D,B).
```

Idea

Learn higher-order programs

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

```
f(A,B):-
    map(A,B,f1).
```

```
f1(A,B):-
    char_to_int(A,C),
    prec(C,D),
    int_to_char(D,B).
```

```
f(A,B):-
    map(A,B,f1).
```

```
f1(A,B):-
    char_to_int(A,C),
    prec(C,D),
    int_to_char(D,B).
```

From 12 to 6 literals

Why?

Search complexity is $\mathbf{b^n}$

b is the number of background relations
n is the size of the program

Idea: increase branching to reduce depth

Fragment	Complexity
First-order	$6^{12} = 2,176,782,336$

Fragment	**Complexity**
First-order	6^{12} = 2,176,782,336
Higher-order	7^{6} = 117,649

+1 because of map

Fragment	Complexity
First-order	$6^{12} = 2,176,782,336$
Higher-order	$7^{6} = 117,649$
Higher-order*	$4^{6} = 4,096$

If we do not give head, tail, empty

How?

Extend Metagol

[Cropper and Muggleton, 2016]

Metagol

Proves examples using a Prolog **meta-interpreter**

Extracts a **logic program** from the proof

Uses **metarules** to guide the search

Metarule

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

P, **Q**, and **R** are second-order variables

A, **B**, and **C** are first-order variables

Examples

input	output
1	3
2	4
3	?

Examples

input	output
1	3
2	4
3	**?**

Background knowledge

succ/2

Metarule

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

Examples

input	output
1	3
2	4
3	**?**

Background knowledge

succ/2

Metarule

P(A,B) ← **Q**(A,C),**R**(C,B)

P/target, Q/succ, R/succ

target(A,B) ← **succ**(A,C),**succ**(C,B)

Examples

input	output
1	3
2	4
3	**5**

Background knowledge

succ/2

Metarule

P(A,B) ← **Q**(A,C),**R**(C,B)

P/target, Q/succ, R/succ

target(A,B) ← **succ**(A,C),**succ**(C,B)

Examples

input	output
[1,2,3]	[c,d,e]
[2,3,4]	?
[3,4,5]	?

Examples

input	output
[1,2,3]	[c,d,e]
[2,3,4]	**?**
[3,4,5]	**?**

Background knowledge

succ/2

int_to_char/2

map/3

Metarules

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

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

negated example (i.e. a goal)

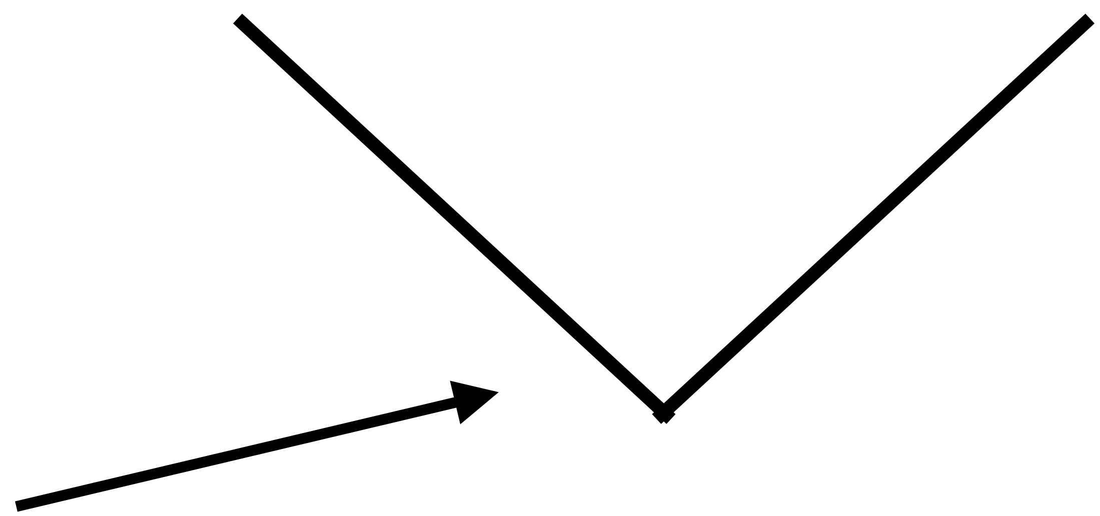
← f([1,2,3],[c,d,e])

metarule

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

← f([1,2,3],[c,d,e])

P(A,B) ← **Q**(A,B,**R**)



resolution

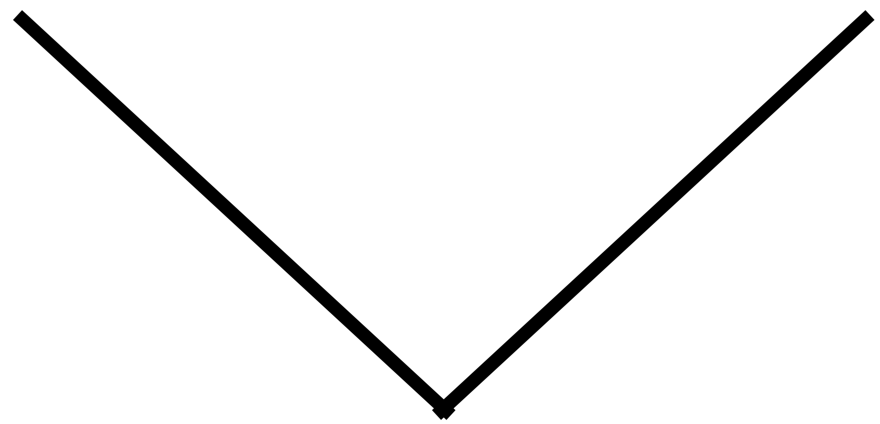
{P/f}

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

← **Q**([1,2,3],[c,d,e],**R**)

new goal

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

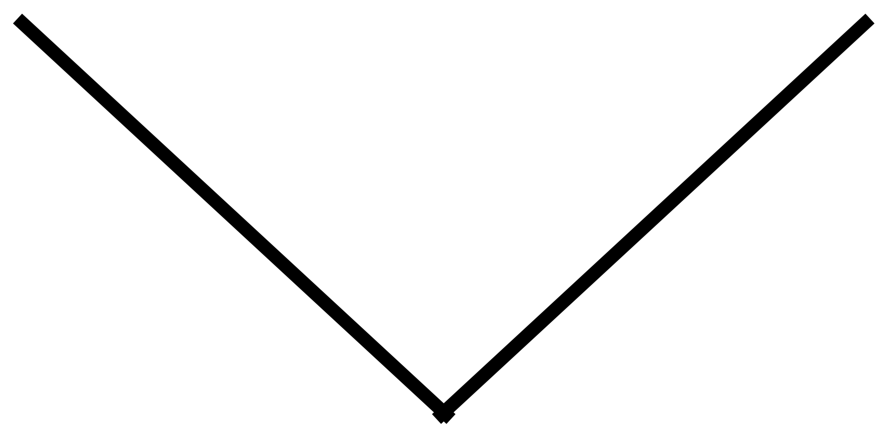


← **Q**([1,2,3],[c,d,e],**R**)

succ/2

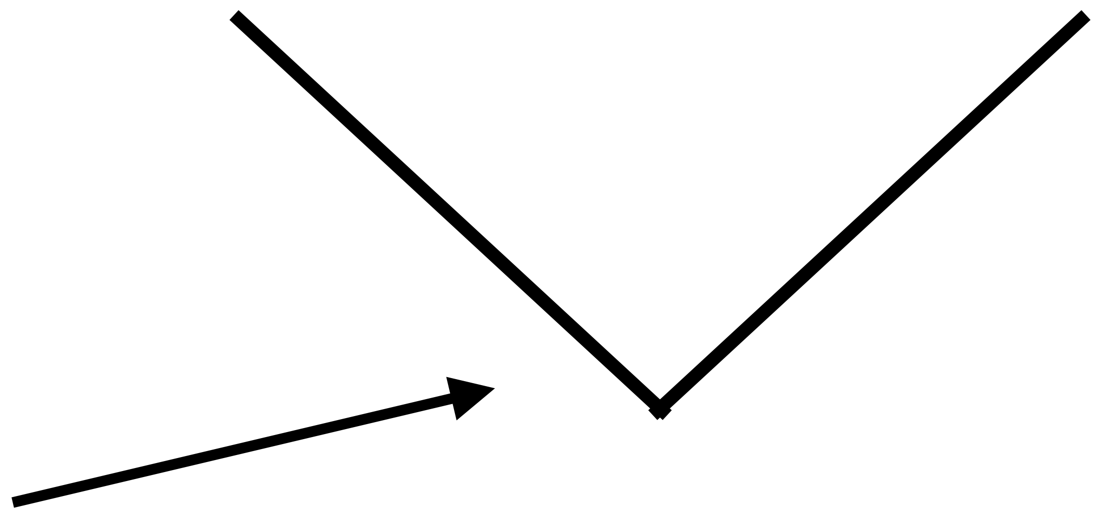
int_to_char/2

map/3



← **Q**([1,2,3],[c,d,e],**R**)

map/3



resolution

{Q/map}

← **Q**([1,2,3],[c,d,e],**R**)

map/3

← map([1,2,3],[c,d,e],**R**)

← map([1,2,3],[c,d,e],**R**)

succ/2
int_to_char/2
map/3

← map([1,2,3],[c,d,e],**R**)

succ/2
int_to_char/2
map/3

← 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 unfolded solution

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

Metagol$_{HO}$

Allows **interpreted** background knowledge

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

Examples

input	output
[1,2,3]	[c,d,e]
[2,3,4]	**?**
[3,4,5]	**?**

BK

succ/2

int_to_char/2

Interpreted BK

map/3

Metarules

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

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

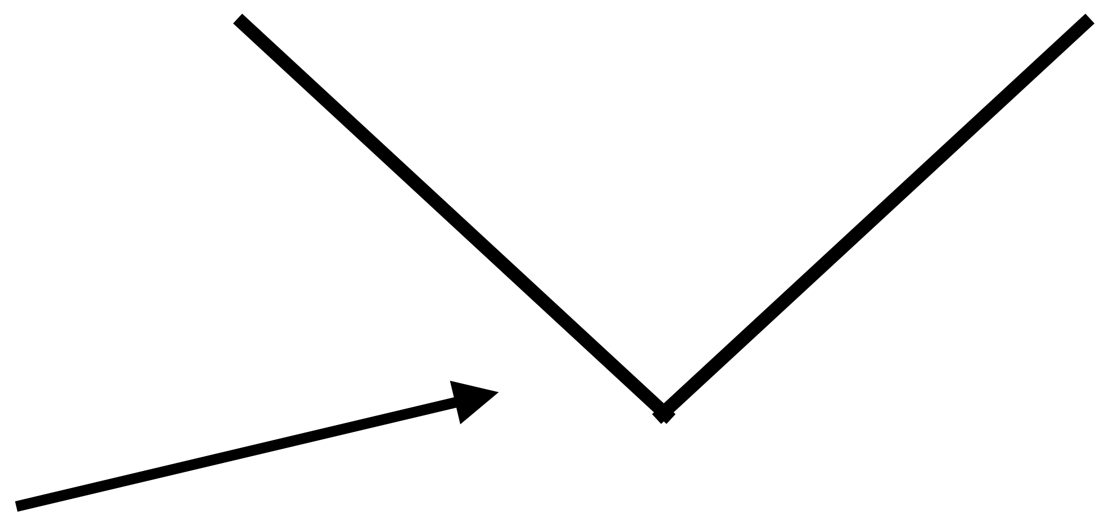
negated example (i.e. a goal)

← f([1,2,3],[c,d,e])

metarule

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

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



resolution

{P/f}

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

$\leftarrow$ **Q**([1,2,3],[c,d,e],**R**)

new goal

← **Q**([1,2,3],[c,d,e],**R**)

interpreted BK

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

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

resolution

{Q/map}

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

←**R**(1,c), **R**(2,d), **R**(3,e)

map ***decomposes*** *goal into subgoals*

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

metarule

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

resolution
{R/S}

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

←**T**(1,C1),**U**(C1,c),
T(2,C2),**U**(C2,d),
T(3,C3),**U**(C3,e)

decomposes *problem again*

←**R**(1,c), **R**(2,d), **R**(3,e)

S(A,B) ← **T**(A,C),**U**(C,B)

←**T**(1,C1),**U**(C1,c),
T(2,C2),**U**(C2,d),
T(3,C3),**U**(C3,e)

and the proof continues …

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$_{HO}$ unfolded solution

invented

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

Decryption example

input	output
dbu	cat
eph	dog
hpptf	**?**

Metagol

```
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).
```

7 clauses and 21 literals

Metagol_{HO}

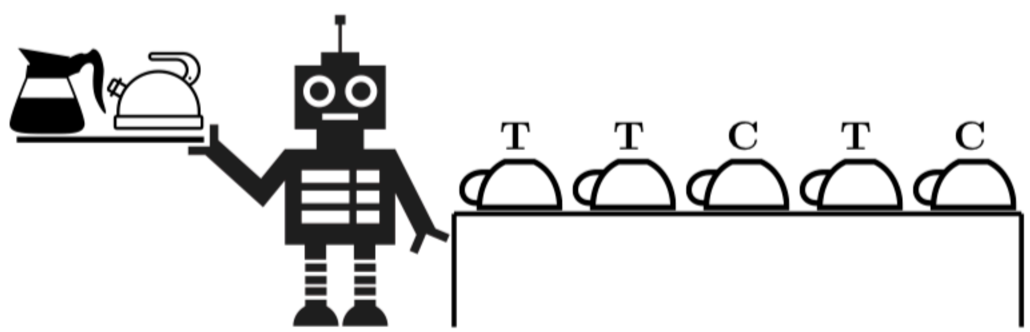
```
f(A,B):-map(A,B,f1).
f1(A,B):-char_to_int(A,C),f2(C,B).
f2(A,B):-prec(A,C),int_to_char(C,B).
```

3 clauses and 8 literals

Does it help in practice?

Q. Can learning higher-order programs improve learning performance?

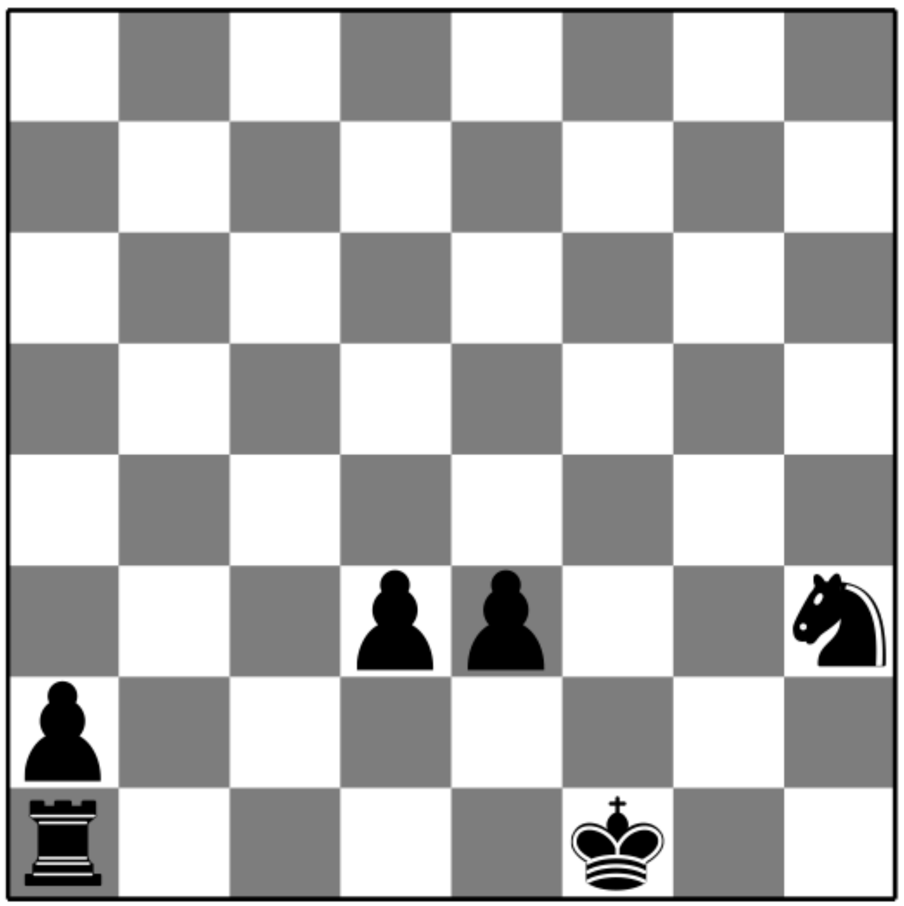
Robot waiter



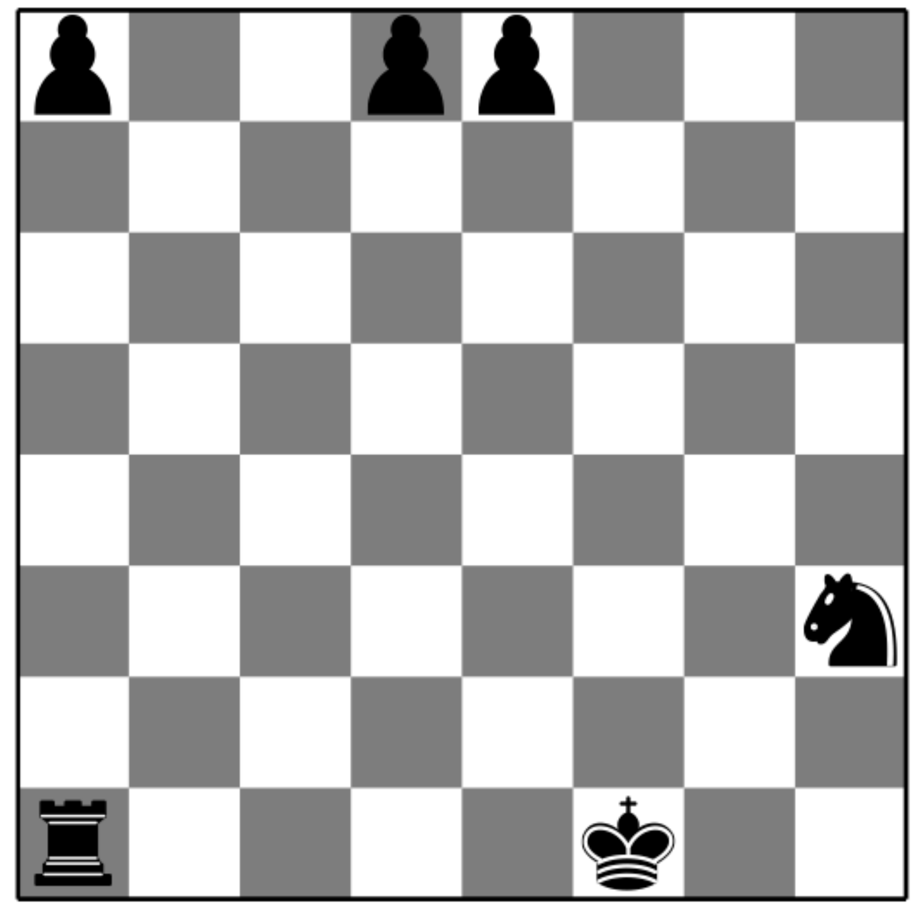
(a) Initial state

(b) Final state

Chess



(a) Initial state



(b) Final state

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$_{HO}$ unfolded solution

invented droplast

```
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).
```

invented reverse

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$_{HO}$ unfolded solution

uses f1 as a predicate symbol

uses f1 as a term

```
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

Inducing higher-order programs can reduce program size and sample complexity and improve learning performance

Can **decompose** problems through **predicate invention**

Limitations

Inefficient search

Which metarules?

Which higher-order definitions?

Thank you

Cropper, A., Morel, R., and Muggleton, S. Learning higher-order logic programs. Machine Learning. 2019.

Metagol system. https://github.com/metagol/metagol