Learning algorithms using logic

(inductive logic programming)

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

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

```
def f(a):
    return a[0]
```

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

```
def f(a):
    return head(a)
```

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

$$\forall A. \forall B.\ \mathrm{head}(A,B) \rightarrow f(A,B)$$

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

$$\forall A.\forall B.\ f(A,B) \leftarrow head(A,B)$$

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

```
f(A,B) ← head(A,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**

$$\forall A.\forall B.\forall C\ \mathrm{tail}(A,C) \wedge \mathrm{head}(C,B) \rightarrow f(A,B)$$

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

$$f(A,B) \leftarrow tail(A,C) \wedge head(C,B)$$

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

```
f(A,B) ← tail(A,C), head(C,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**

```python
def f(a):
    return a[-1]
```

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

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

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

```
tail(A,C) ∧ empty(C) ∧ head(A,B) → f(A,B)
tail(A,C) ∧ f(C,B) → f(A,B)
```

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**
dog	g
sheep	p
chicken	**n**

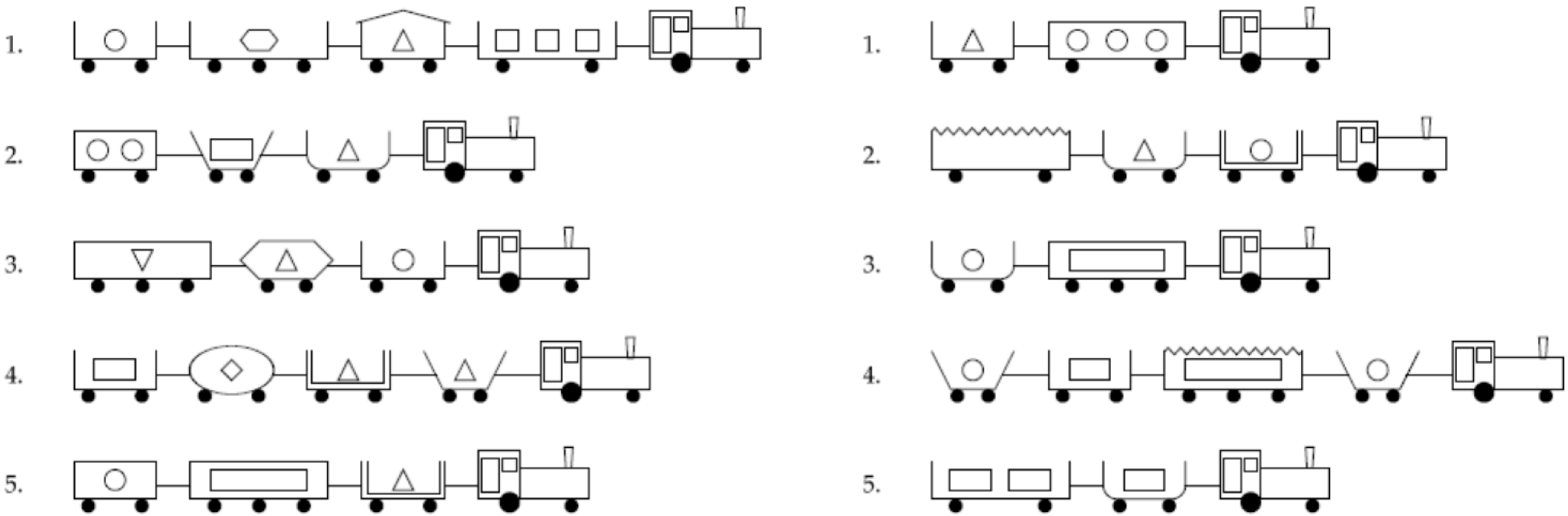
```
f(A,B):- tail(A,C),empty(C),head(A,B).
f(A,B):- tail(A,C),f(C,B).
```

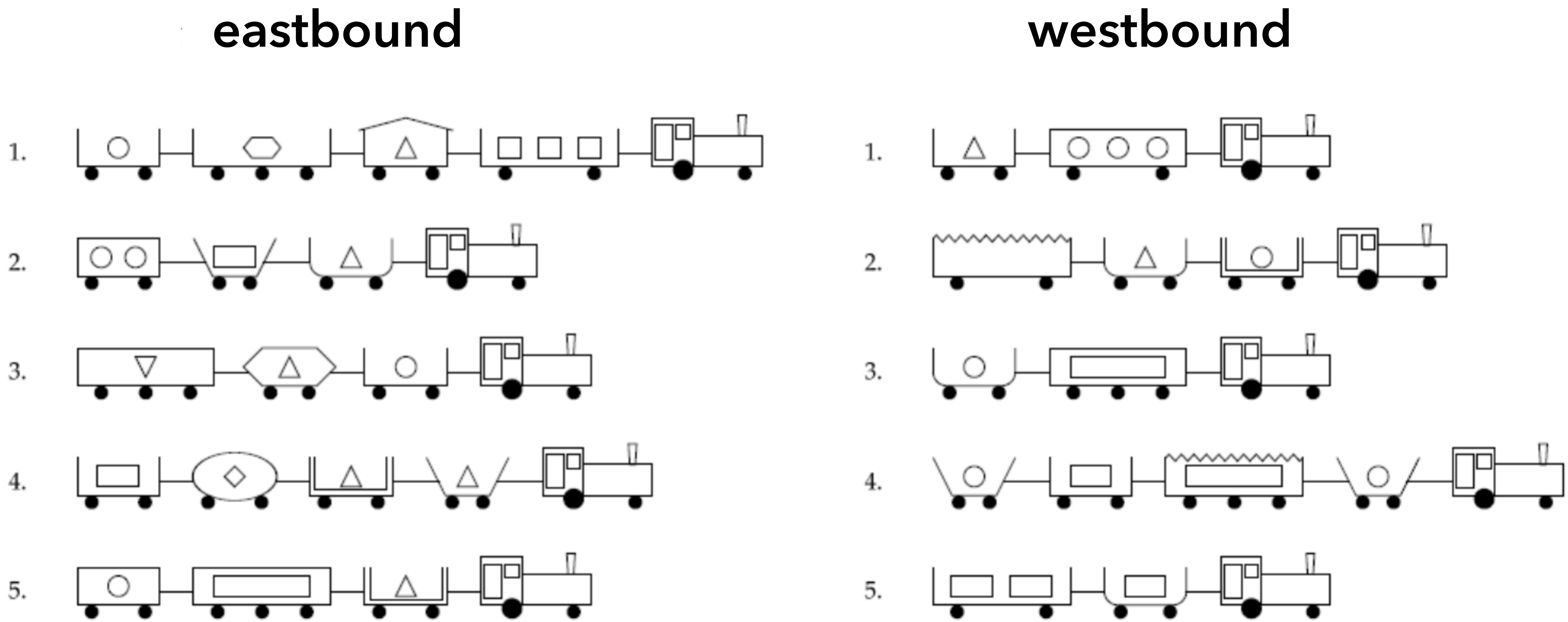
input	**output**
ecv	cat
fqi	dog
iqqug	**?**

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

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

eastbound westbound






```
eastbound(A):-
    has_car(A,B),
    short(B),
    closed(B).
```

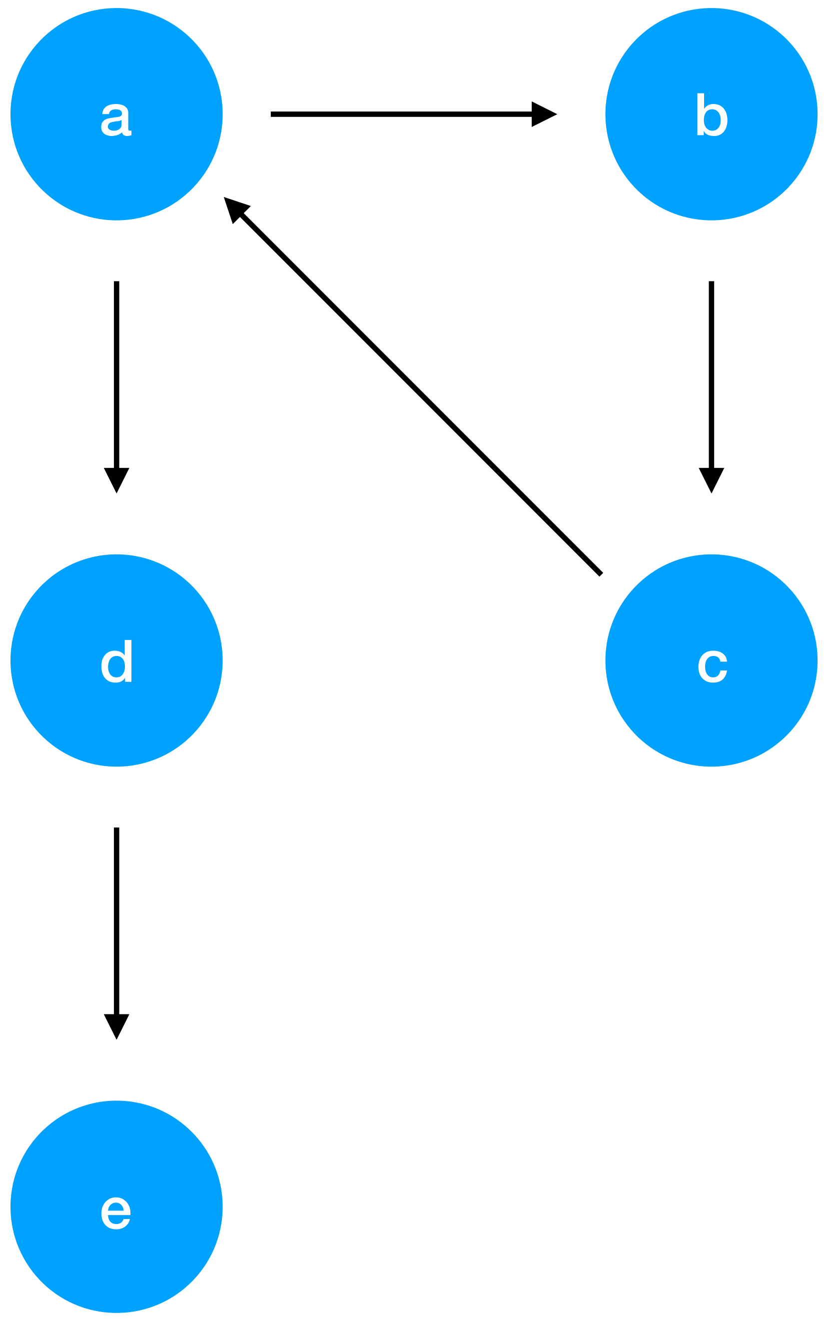
ILP learning from entailment setting

Input:

- Sets of atoms E^+ and E^-
- Logic program BK

Output:

- logic program H s.t
- $BK \cup H \models E^+$
- $BK \cup H \; ! \models E^-$




```
% bk
edge(a,b).
edge(b,c).
edge(c,a).
edge(a,d).
edge(d,e).

% examples
pos(reachable(a,c)).
pos(reachable(b,e)).
neg(reachable(d,a)).
```

```prolog
reachable(A,B):- edge(A,B).
reachable(A,B):- edge(A,C),reachable(C,B).
```

ILP approaches

Set covering

- generalise a specific clause (Progol, Aleph)
- specialise a general clause (FOIL)

Generate and test

- Answer set programming (HEXMIL, ILASP, INSPIRE)
- PL systems

Neural-ILP (DILP and now about 10^6 other systems)

Proof search (Metagol)

Metagol

- Prolog meta-interpreter
- 50 lines of code
- Proof search
- Uses metarules to guide the search
- Supports:
  - Recursion
  - Predicate invention
  - Higher-order programs

Meta-interpreter 1

```
prove(Atom):-
    call(Atom).
```

Meta-interpreter 2

```
prove(true).

prove(Atom):-
    clause(Atom,Body),
    prove(Body).

prove((Atom,Atoms)):-
    prove(Atom),
    prove(Atoms).
```

Meta-interpreter 3

```
prove([]).

prove([Atom|Atoms]):-
    clause(Atom,Body),
    body_as_list(Body,BList),
    prove(BList).
```

Metagol 1

```
prove([]).

prove([Atom|Atoms]):-
    prove_aux(Atom),
    prove(Atoms).

prove_aux(Atom):-
    call(Atom).

prove_aux(Atom):-
    metarule(Atom,Body),
    prove(Body).
```

Metagol 2

```
prove([],P,P).

prove([Atom|Atoms],P1,P2):-
    prove_aux(Atom,P1,P3),
    prove(Atoms,P3,P2).

prove_aux(Atom,P,P):-
    call(Atom).

prove_aux(Atom,P1,P2):-
    metarule(Atom,Body,Subs),
    save(Subs,P1,P3),
    prove(Body,P3,P2).
```

Metarules

$$P(A,B) \leftarrow Q(A,B)$$

$$P(A,B) \leftarrow Q(B,A)$$

$$P(A,B) \leftarrow Q(A),R(A,B)$$

$$P(A,B) \leftarrow Q(A,B),R(B)$$

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

Logical reduction of metarules [ILP14, ILP18]

$P(A,B) \leftarrow Q(A,B)$

$P(A,B) \leftarrow Q(B,A)$

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

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

$P(A,B) \leftarrow Q(B,A),R(A,B)$

$P(A,B) \leftarrow Q(B,A),R(B,A)$ $\longrightarrow$?

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

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

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

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

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

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

Logical reduction of metarules [ILP14, ILP18]

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

$\longrightarrow$

$P(A,B) \leftarrow Q(B,A)$

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

Learning game rules

```
% examples
fizz(4,4).
fizz(3,fizz).
fizz(10,buzz).
fizz(11,11).
fizz(30,fizzbuzz).
```

```prolog
% examples
fizz(4,4).
fizz(3,fizz).
fizz(10,buzz).
fizz(11,11).
fizz(30,fizzbuzz).
```

```prolog
% hypothesis
fizzbuzz(N,fizz):-
    divisible(N,3),
    not(divisible(N,5)).
fizzbuzz(N,buzz):-
    not(divisible(N,3)),
    divisible(N,5).
fizzbuzz(N,fizzbuzz):-
    divisible(N,15).
fizzbuzz(N,N):-
    not(divisible(N,3)),
    not(divisible(N,5)).
```

Game	R	L	D	P
Minimal Decay	2	6	0	1
Minimal Even	8	19	0	1
Rainbow	10	48	0	1
Rock Paper Scissors	12	36	0	1
GT Chicken	16	78	0	2
GT Attrition	16	60	0	2
Coins	16	45	0	1
Buttons and Lights	16	44	1	1
Leafy	17	80	2	2
GT Prisoner	17	75	0	2
Eight Puzzle	17	60	2	1
Lightboard	18	69	2	2
Knights Tour	18	46	2	1
Sukoshi	19	49	1	2
Walkabout	22	66	2	2
Horseshoe	22	59	2	2
GT Ultimatum	22	67	0	2
Tron	23	76	2	2
9x Buttons and Lights	24	77	2	1
Hunter	24	69	2	1
GT Centipede	24	69	0	2
Fizz Buzz	25	74	0	1
Untwisty Corridor	27	68	0	1
Don’t Touch	29	84	2	2
Tiger vs Dogs	30	88	2	2

Game	R	L	D	P
Sheep and Wolf	30	89	2	2
Duikoshi	31	76	2	2
TicTacToe	32	92	2	2
HexForThree	35	130	2	3
Connect 4	36	124	2	4
Breakthrough	36	126	2	2
Centipede	37	134	2	1
Forager	40	106	2	1
Sudoku	41	101	2	1
Sokoban	41	172	2	1
9x TicTacToe	42	149	2	2
Switches	44	183	2	1
Battle of Numbers	44	134	2	2
Free For All	46	130	2	2
Alquerque	49	134	2	2
Kono	50	134	2	2
Checkers	52	167	2	2
Pentago	53	188	2	2
Platform Jumpers	62	168	2	2
Pilgrimage	80	240	2	2
Firesheep	85	290	2	2
Farming Quandries	88	451	2	2
TTCC4	94	301	2	2
Frogs and Toads	97	431	2	2
Asylum	101	273	2	2

Learning higher-order programs

[IJCAI16]

Input	Output
[[i,j,c,a,i],[2,0,1,6]]	[[i,j,c,a]]
[[1,1],[a,a],[x,x]]	[[1],[a]]
[[1,2,3,4,5],[1,2,3,4,5]]	[[1,2,3,4]]
[[1,2],[1,2,3],[1,2,3,4],[1,2,3,4,5]]	[[1],[1,2],[1,2,3]]

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

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

Lifelong learning

[ECAI14]

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

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

```
f(A,B):-
    f1(A,C),
    skip1(C,D),
    space(D,E),
    f1(E,F),
    skiprest(F,B).
f1(A,B):-
    uppercase(A,C),
    copyword(C,B).
```

10 seconds

task	input	output
g	tony	Tony

task	input	output
g	tony	Tony

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

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

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

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):-f1(A,C),f3(C,B).
f1(A,B):-f3(A,C),skip1(C,B).
f2(A,B):-g(A,C),skiprest(C,B).
f3(A,B):-g(A,C),space(C,B).
```

2 seconds

Learning efficient programs

[IJCAI15, MLJ18]

input	output
[s,h,e,e,p]	e
[a,l,p,a,c,a]	a
[c,h,i,c,k,e,n]	**?**

input	output
[s,h,e,e,p]	e
[a,l,p,a,c,a]	a
[c,h,i,c,k,e,n]	**c**

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

input	output
[s,h,e,e,p]	e
[a,l,p,a,c,a]	a
[c,h,i,c,k,e,n]	**c**

```
f(A,B):-mergesort(A,C),f1(C,B).
f1(A,B):-head(A,B),tail(A,C),head(C,B).
f1(A,B):-tail(A,C),f1(C,B).
```

input	output
My name is John.	John
My name is Bill.	Bill
My name is Josh.	Josh
My name is Albert.	Albert
My name is Richard.	Richard

```
f(A,B):-
    tail(A,C),
    dropLast(C,D),
    dropWhile(D,B,not_uppercase).
```

```
f(A,B):-
    tail(A,C),
    dropLast(C,D),
    dropWhile(D,B,not_uppercase).
```

1

n

4n

```
% learning f/2
% clauses: 1
% clauses: 2
% clauses: 3
% is better: 67
% is better: 57
% clauses: 4
% is better: 55
% clauses: 5
% is better: 53
% is better: 51
% is better: 49
% is better: 46
% clauses: 6
% is better: 41
% is better: 36
% is better: 31
f(A,B):-tail(A,C),f_1(C,B).
f_1(A,B):-f_2(A,C),dropLast(C,B).
f_2(A,B):-f_3(A,C),f_3(C,B).
f_3(A,B):-tail(A,C),f_4(C,B).
f_4(A,B):-f_5(A,C),f_5(C,B).
f_5(A,B):-tail(A,C),tail(C,B).
```

```
f(A,B):-
    tail(A,C),
    tail(C,D),
    tail(D,E),
    tail(E,F),
    tail(F,G),
    tail(G,H),
    tail(H,I),
    tail(I,J),
    tail(J,K),
    tail(K,L),
    tail(L,M),
    dropLast(M,B).
```

```
f(A,B):-
    tail(A,C),
    tail(C,D),
    tail(D,E),
    tail(E,F),
    tail(F,G),
    tail(G,H),
    tail(H,I),
    tail(I,J),
    tail(J,K),
    tail(K,L),
    tail(L,M),
    dropLast(M,B).
```

does this last

The good

- Generalisation
- Abstraction
- Data efficient
- Readable hypotheses
- Include prior knowledge
- Reason about the learning

The bad

- Tricky on messy problems
- Tricky on big problems
- Need to know what you are doing

- S. Tourret and A. Cropper. SLD-resolution reduction of second-order horn fragments.. JELIA 2019.
- Andrew Cropper, Stephen H. Muggleton: Learning efficient logic programs. Machine learning 2018.
- A. Cropper and S. Tourret. Derivation reduction of metarules in meta-interpretive learning. ILP 2018.
- Andrew Cropper, Stephen H. Muggleton: Learning Higher-prder logic programs through abstraction and invention. IJCAI 2016.
- Andrew Cropper, Stephen H. Muggleton: Learning Efficient Logical Robot Strategies Involving Composable Objects. IJCAI 2015.
- Stephen H. Muggleton, Dianhuan Lin, Alireza Tamaddoni-Nezhad: Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited. Machine Learning 2015.

https://github.com/metagol/metagol