Inductive Logic Programming for "Seek Whence"

Richard Evans

Deep Mind / Imperial College London

September 17, 2017

Introducing the "Seek Whence" Domain

Hofstadter introduced the Seek Whence domain in *"Fluid Concepts and Liquid Analogies"*:

The player is given a sequence of symbols

- a, a, b, b, c, c, d, d, ...

The player needs to guess the next symbol(s)

The only background knowledge is the successor relation:

- succ(a, b), succ(b, c), succ(c, d) ...

Examples Sequences in ‘Seek Whence’

- a, b, c, d, ...
- a, a, b, b, c, c, d, d, ...
- a, k, b, k, k, c, k, k, k, d, k, ...
- a, b, b, c, c, c, ...
- a, k, k, k, k, k, b, k, k, k, k, c, k, k, k, ...
- b, a, b, b, b, b, b, c, b, b, d, b, b, e, b, ...

Examples Sequences in ‘Seek Whence’

- a, b, c, d , e
- a, a, b, b, c, c, d, d , e, e
- a, k, b, k, k, c, k, k, k, d, k , k, k, k
- a, b, b, c, c, c , d, d, d, d
- a, k, k, k, k, k, b, k, k, k, k, c, k, k, k , d, k, k
- b, a, b, b, b, b, b, c, b, b, d, b, b, e, b , b, f, b

Ambiguous Sequences

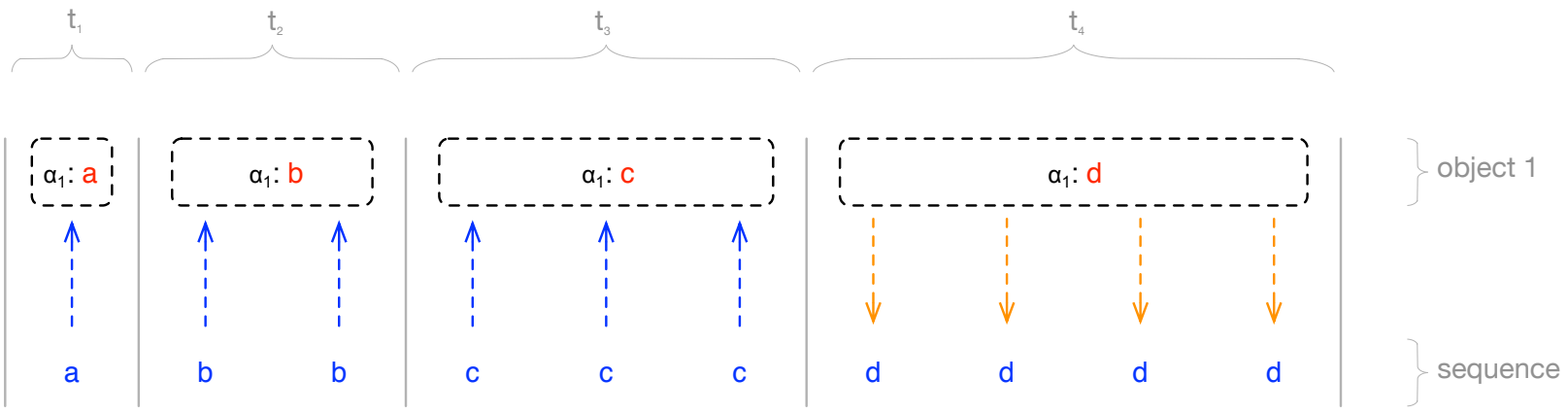
There are always many different ways to continue a series.

Consider a, b, b, c, c, ...

Possible answers include:

- a, b, b, c, c , c
- a, b, b, c, c , d
- a, b, b, c, c , b
- a, b, b, c, c , q

Example 1

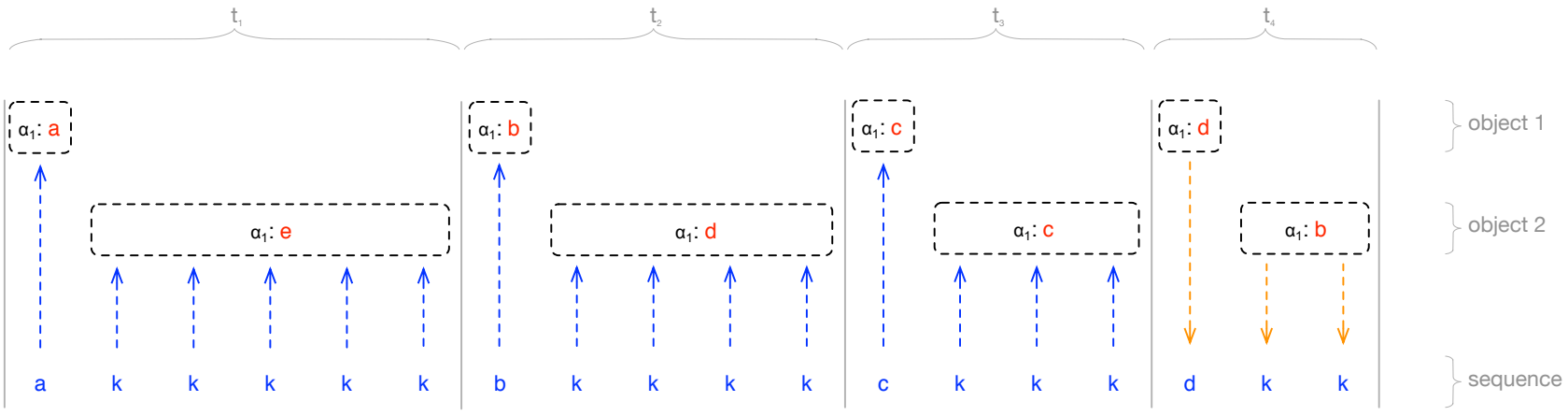


Object 1

```
mark(obj1, A1, I, M) :-
    M = A1.
length(obj1, A1, N) :-
    N = A1.
```

```
initial(obj1, a1, z).

update(obj1, a1, PrevA1, NewA1) :-
    succ(PrevA1, NewA1).
```

Example 2



Object 1

```
mark(obj1, A1, I, M) :-
        M = A1.
length(obj1, A1, N) :-
        N = a.
initial(obj1, a1, z).

update(obj1, a1, PrevA1, NewA1) :-
       succ(PrevA1, NewA1).
```

Object 2

```
mark(obj2, A1, I, M) :-
        M = k.
length(obj2, A1, N) :-
        N = A1.
initial(obj2, a1, f).

update(obj2, a1, PrevA1, NewA1) :-
       succ(NewA1, PrevA1).
```

Assumptions

Inspired by broadly Kantian prior constraints:

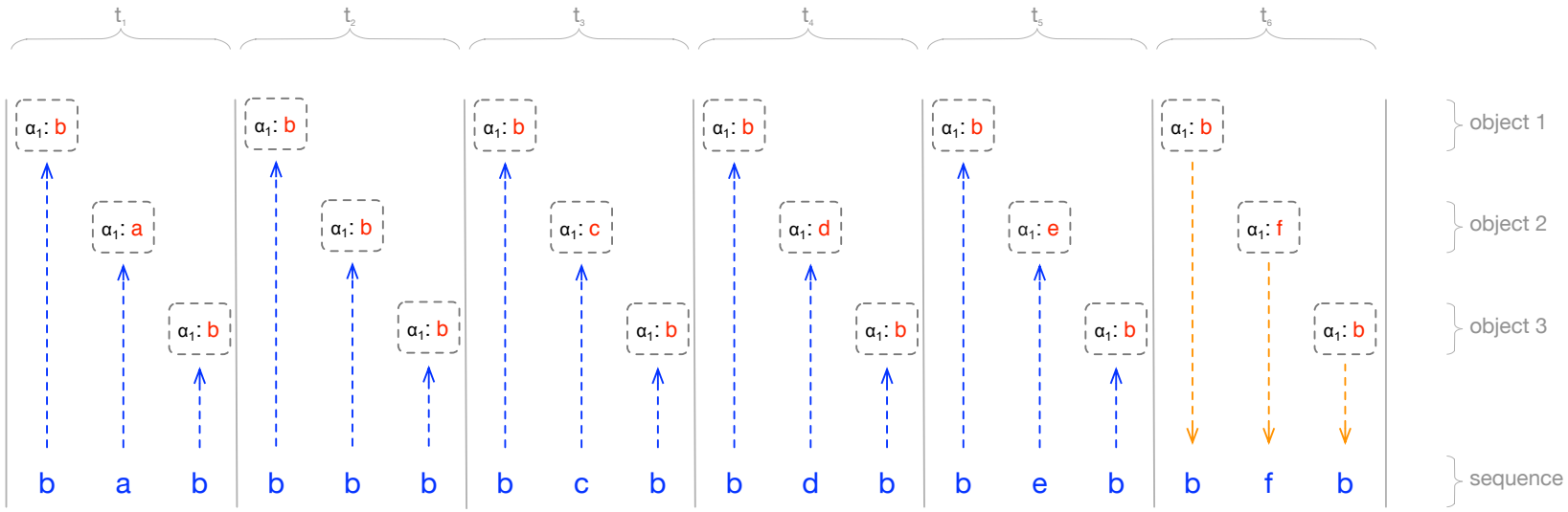
- Our representations must be grouped into various *moments of time*
- Our representations must be grouped into *objects*, persisting over time
- Fluent properties of objects must be explained by *general causal rules*

Perception is rule induction: searching for a set of rules that satisfy the constraints

Example 3

b, a, b, b, b, b, b, c, b, b, d, b, b, e, b, ...

Example 3



Object 1

```
mark(obj1, A1, I, M) :-
        M = A1.
length(obj1, A1, N) :-
        N = a.
initial(obj1, a1, b).

update(obj1, a1, PrevA1, NewA1) :-
          NewA1 = PrevA1.
```

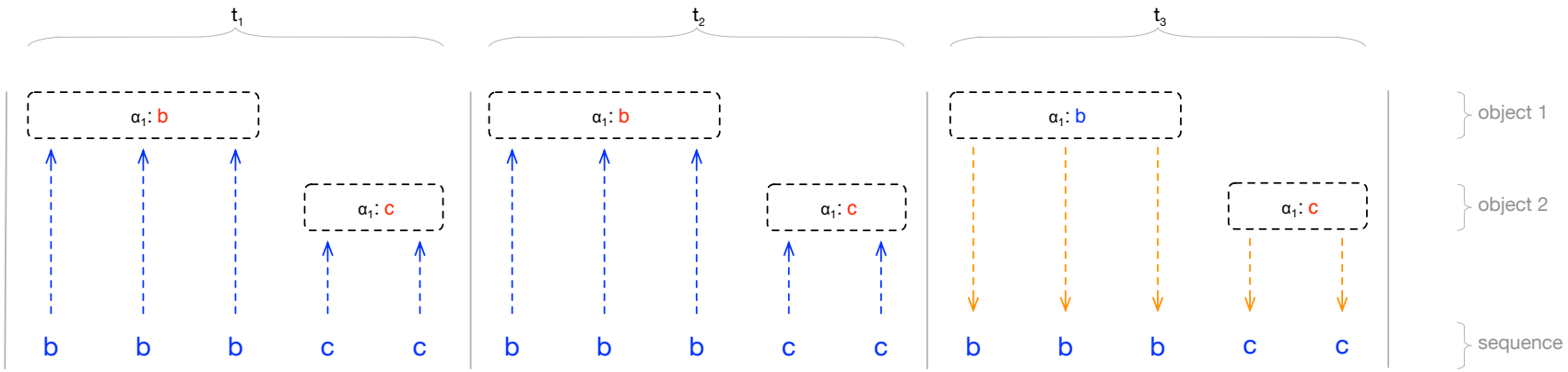
Object 2

```
mark(obj2, A1, I, M) :-
        M = A1.
length(obj2, A1, N) :-
        N = a.
initial(obj2, a1, z).

update(obj2, a1, PrevA1, NewA1) :-
        succ(PrevA1, NewA1).
```

Object 3

```
mark(obj3, A1, I, M) :-
        M = A1.
length(obj3, A1, N) :-
        N = a.
initial(obj3, a1, b).

update(obj3, a1, PrevA1, NewA1) :-
          NewA1 = PrevA1.
```

How to Interpret Sequences



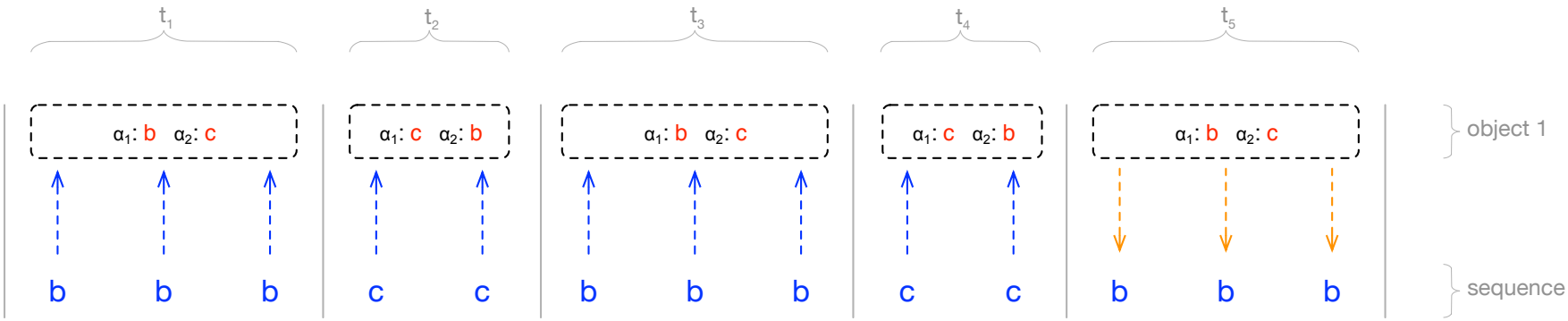
Object 1

```
mark(obj1, A1, I, M) :-
        M = A1.

length(obj1, A1, N) :-
        N = c.

initial(obj1, a1, b).

update(obj1, a1, PrevA1, NewA1) :-
         NewA1 = PrevA1.
```

Object 2

```
mark(obj1, A1, I, M) :-
        M = A1.

length(obj1, A1, N) :-
        N = b.

initial(obj1, a1, c).

update(obj1, a1, PrevA1, NewA1) :-
         NewA1 = PrevA1.
```

How to Interpret Sequences



Object 1

```
mark(obj1, A1, A2, I, M) :-
          M = A1.

length(obj1, A1, A2, N) :-
          N = A2.

initial(obj1, a1, c).

update(obj1, a1, PrevA1, PrevA2, NewA1) :-
              NewA1 = PrevA2.

initial(obj1, a2, b).

update(obj1, a2, PrevA1, PrevA2, NewA2) :-
              NewA2 = PrevA1.
```

Experimental Results

Our program gets 86% correct on three test-sets.

This compares with 25% for Meredith’s implementation.

Results: the "Blackburn Dozen"

Sequence	Human	MAP
a,a,b,a,b,c,a,b,c,d, ...	a	a
a,b,c,d, ...	e	e
b,a,b,b,b,b,b,c,b,b,d,b,b, ...	e	e
a,b,b,c,c,c,d,d,d,d, ...	e	e
a,h,e,h,a,h,e,h, ...	a	a
b,a,b,b,b,c,b,d,b,e,b, ...	f	f
b,c,a,b,c,b,b,b,c,c,c,c,b,c,d,d,d,d, ...	b	b
a,b,b,c,c,d,d,e, ...	e	e
a,b,c,c,d,d,e,e,e,f,f,f, ...	g	g
i,a,i,b,i,c,i,d, ...	i	i
a,h,a,b,a,h,a,b,c,b,a,h,a,b,c,d,c,b,a,h,a,b,c,d, ...	e	e
a,h,e,e,h,a,a,h,e,e,h,a, ...	a	a

Results: the "Hofstadter Fifteen"

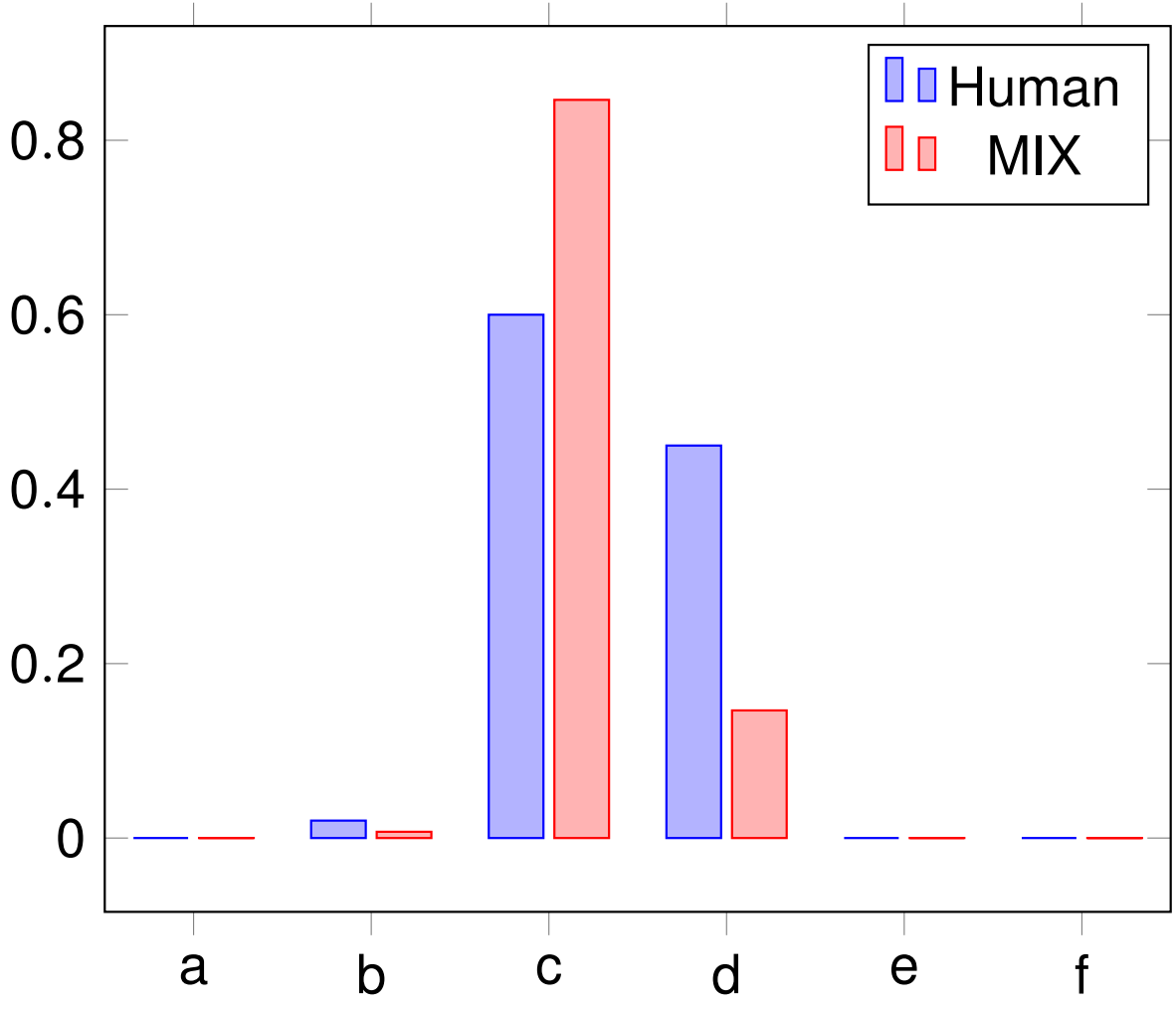
Sequence	Human	MAP
b,b,b,c,c,b,b,b,c,c,b,b,b,c,c, ...	b	b
b,a,a,b,b,b,a,a,a,a,b,b,b,b,b, ...	a	a
b,a,b,e,b,a,a,a,e,b,b,b,b,e,a, ...	e	-
b,c,a,c,a,c,b,d,b,d,b,c,a,c,a, ...	e	-
a,b,b,c,c,d,d,e,e,f,f,g,g, ...	h	h
a,a,b,a,b,c,a,b,c,d,a,b,c,d,e, ...	a	a
b,a,c,a,b,d,a,b,c,e,a,b,c,d,f, ...	a	a
a,b,a,c,b,a,d,c,b,a,e,d,c,b, ...	g	g
c,b,a,b,c,b,a,b,c,b,a,b,c, ...	b	b
a,a,a,b,b,c,e,f,f,g,g,g,h,h,i, ...	s	-
a,a,b,a,a,b,c,b,a,a,b,c,d,c, ...	a	a
a,a,b,c,a,b,b,c,a,b,c,c,a,a, ...	a	a
a,a,b,c,a,b,b,c,a,b,c,c,a,b, ...	a	a
a,b,b,c,c,a,a,b,c,c,a,a,b,b, ...	a	a
a,b,b,c,c,a,b,b,c,a,b,c,c,a, ...	a	-

Results: the C-test [Hernandez-Orallo et al]

Sequence	Human	MAP
a,b,a,b,a,b, ...	a	a
a,c,b,d,c,e, ...	d	d
a,c,z,b,y,a, ...	x	x
a,a,z,z,y,y, ...	x	x
a,a,z,c,y,e,x, ...	g	g
a,a,a,b,b,b,c, ...	c	c
a,z,b,d,c,e,g,f, ...	h	b
a,d,g,j, ...	m	m
c,a,b,d,b,c,c,e,c,d, ...	d	d
a,a,a,a,b,b,b,b,c, ...	c	c
a,a,c,y,a,w,y, ...	s	s
a,b,d,e,g, ...	h	b
a,a,b,b,z,a,b,b, ...	y	y
z,a,y,x,x,u,w, ...	r	r
z,a,x,a,v,a, ...	t	t

Results: Applying a Mixture Model to Ambiguous Sequences

Consider the ambiguous sequence a, b, b, c, c, ...



Using IGOR2 to Solve Number Sequences

“Applying Inductive Program Synthesis to Induction of Number Series – A Case Study with IGOR” - Jacqueline Hofmann, Emanuel Kitzelmann, and Ute Schmid

“The Artificial Jack of All Trades: The Importance of Generality in Approaches to Human-Level Artificial Intelligence” Tarek R. Besold and Ute Schmid

Comparing IGOR2 with Our Approach

Major points of agreement:

- We are not happy with domain-specific solutions...
- We are both looking for a *general* learning system that can be applied "off the shelf" to solve sequence induction tasks

Comparing IGOR2 with Our Approach

Differences:

- We are looking at a verbal reasoning task, rather than a maths task
- We are learning logic programs rather than functional programs
- We use additional (Kant-inspired) priors: priors which are *domain-independent*

Kant’s Priors on Program Synthesis Systems

The key idea of my approach is to **reinterpret Kant’s Principles as a set of domain-independent priors on a rule-induction system**.

- These priors are *domain-independent*
- These are the *only* domain-independent priors