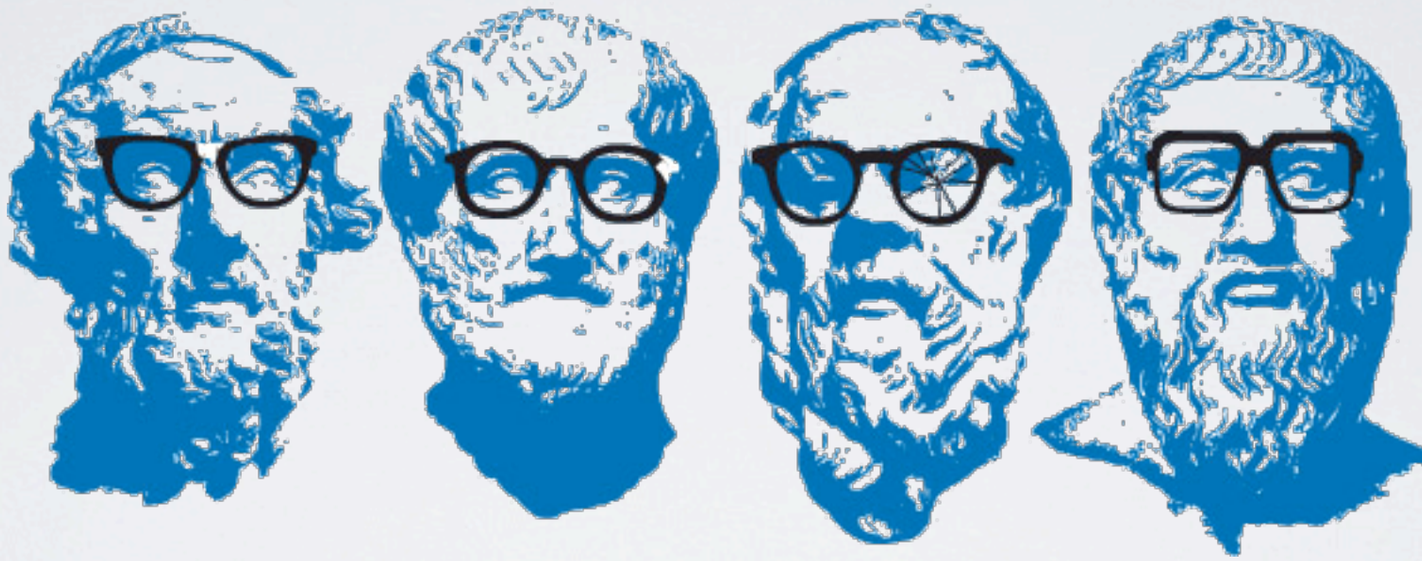
CORE.LOGIC.INTRO

Edmund Jackson
Cambridge Data Science

edmund@data-science.co.uk



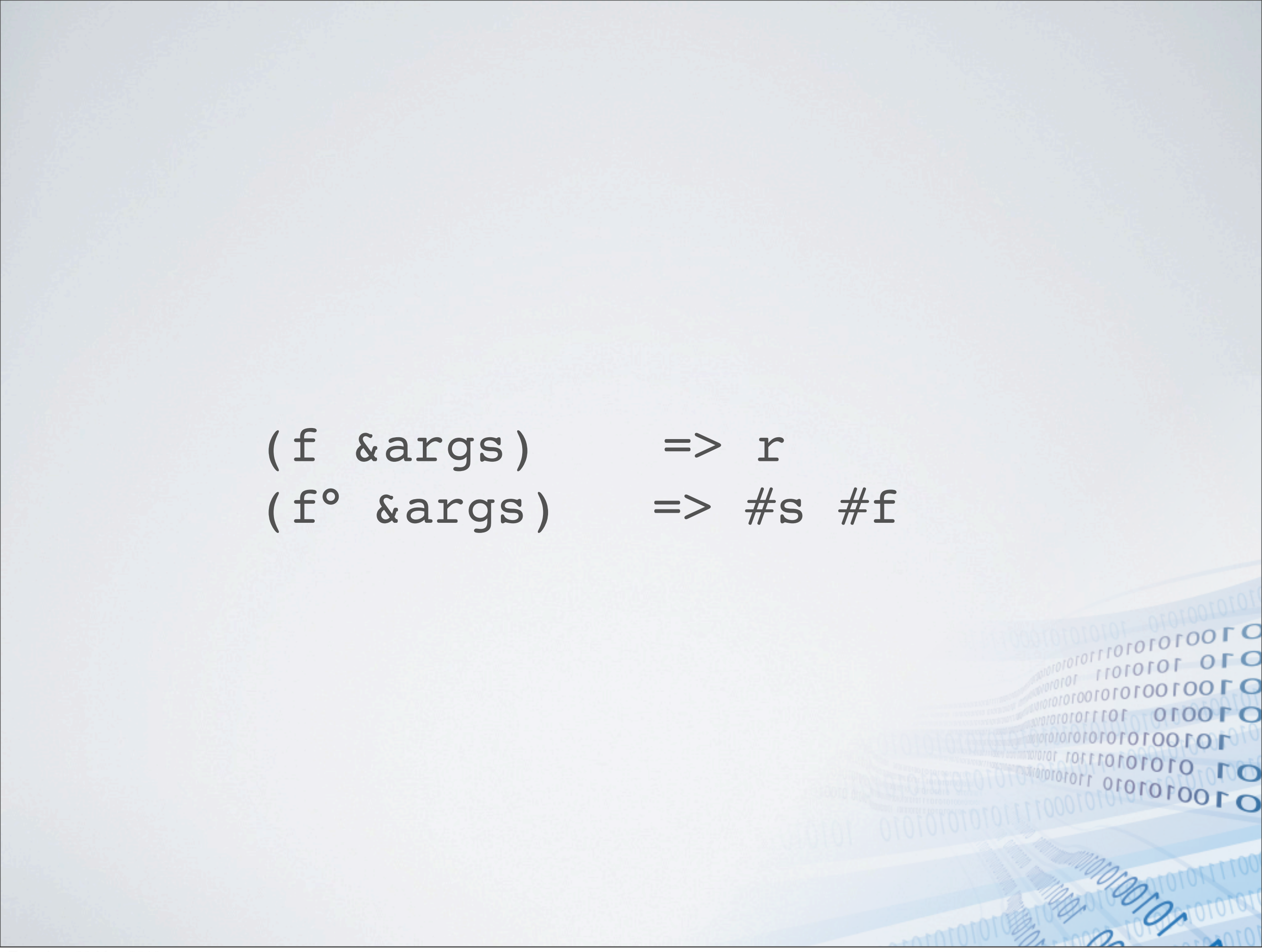
Goals

```
(cons :a [])
;; (:a)
```

```
(cons :b [:a])
;; (:b :a)
```

```
(f &args)      => r
```

```
(f &args)      => r
(f° &args)     => #s #f
```



```
(f &args)      => r
(f° &args)     => #s #f

(f° &args r) => #s #f
```

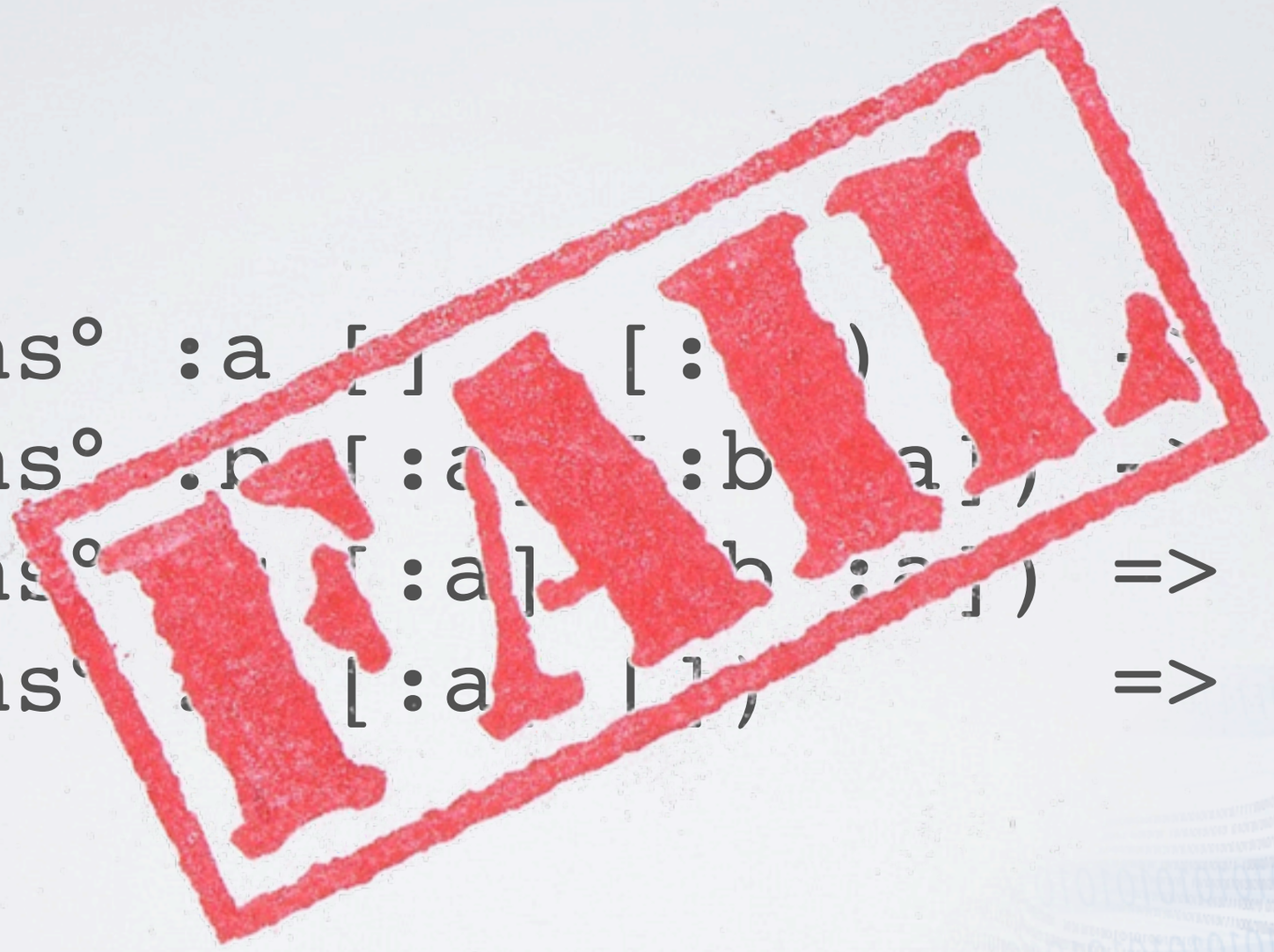
```
(cons :a []) => [:a]

(cons° :a []     [:a])     => #s
```

```
(conso :a []    [:a])     => #s
(conso :b [:a] [:b :a]) => #s
```

```
(conso :a []    [:a])     => #s
(conso :b [:a] [:b :a]) => #s
(conso :c [:a] [:b :a]) => #f
```

```
(conso :a []    [:a])     => #s
(conso :b [:a] [:b :a]) => #s
(conso :c [:a] [:b :a]) => #f
(conso :c [:a] [])        => #f
```

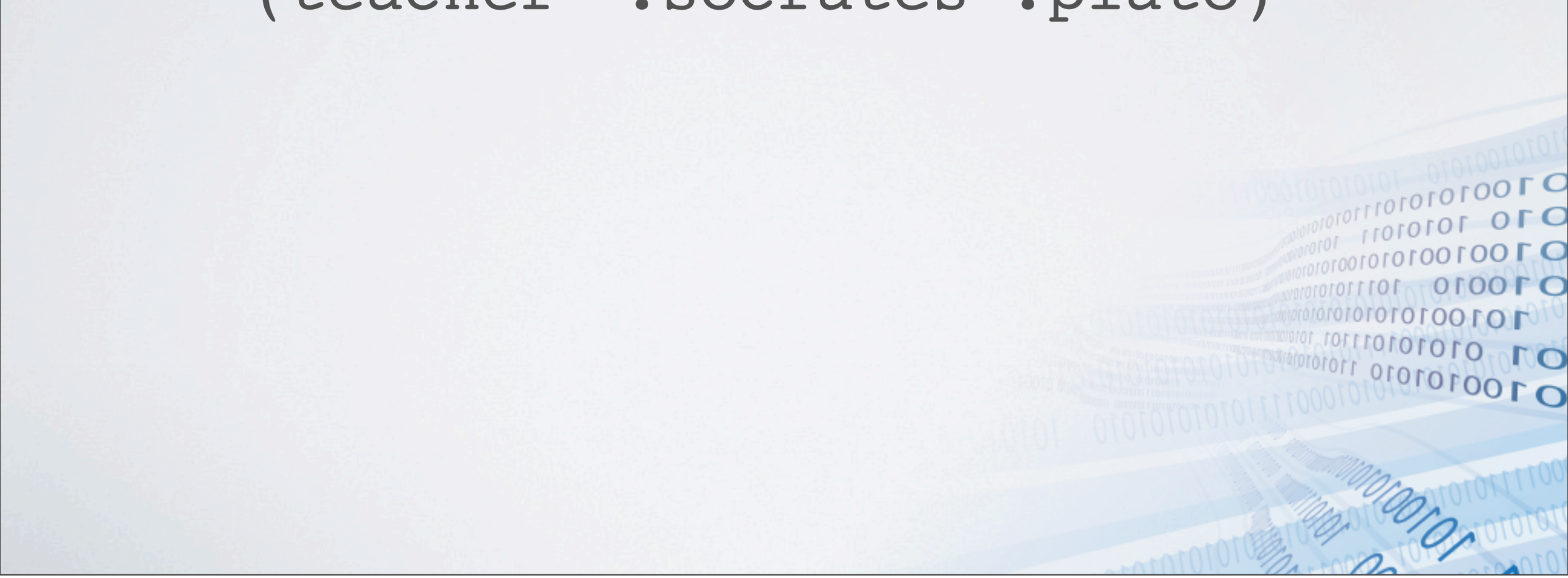
```
(cons° :a [] [:b]) => #s
(cons° :p [:a :b a]) => #s
(cons° [:a] [:b :a]) => #f
(cons° [:a] []) => #f
```



A goal succeeds
if it the
implied relation holds

```
(rest°   [:a :b :c] [:b :c])     => #s
(first°  [:a :b :c] :a)          => #s
(member° :a                [:a :b])     => #s
(perm°   [:a :b :c] [:b :c :a]) => #s
```

```
(teacher° :socrates :plato)
```

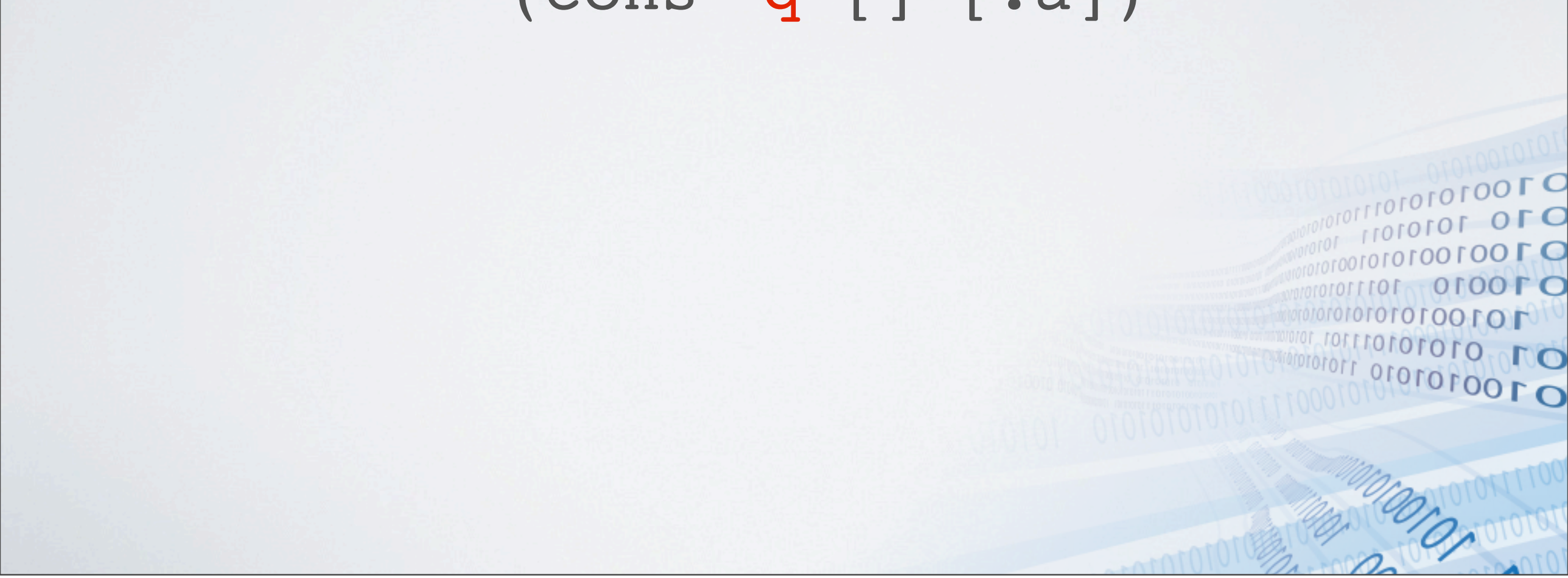


Logic Variables

q

¿q

```
(conso q [] [:a])
```



```
(run* [q]
      (cons° q [] [:a]))
```

```
(run* [q]
      (cons° q [] [:a]))
```

```
q -> :a
```

```
(run* [q]
      (cons° :a q [:a]))
```

```
q -> []
```

```
(run* [q]
      (cons° :a [] q))
```

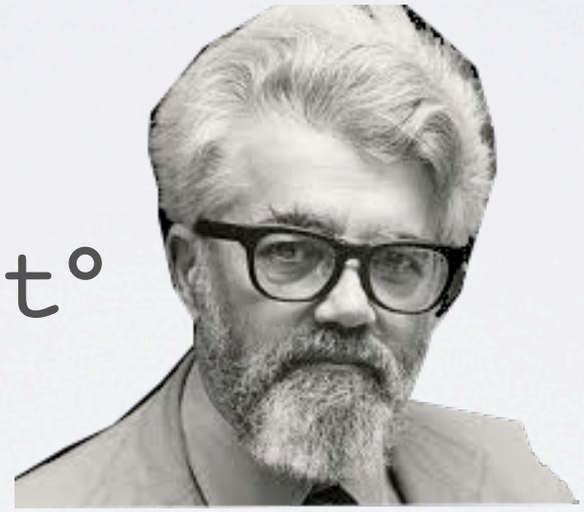
```
q -> [:a]
```

```
(run* [q]
      (cons° :a [] [q]))
```

```
q -> :a
```

(ancient° q )

(ancient°



)

Composition

MULTIPLE POSSIBILITIES

```
(run* [q]
      (membero q [:a :b :c]))
```

```
=> [:a :b :c]
```

ALL POSSIBILITIES

```
(run* [q]
      (membero :b [:a :b :c]))
```

```
=> [_]
```

CONJUNCTION

```
(run* [q]
      (membero q [:a :b :c])
      (membero q [:b :c :d]))
```

```
=> [:b :c]
```

CONJUNCTION

```
(run* [q]
      (membero q  [:a :b :c])
      (membero :a [:b q]))

=> [:a]
```

CONJUNCTION

```
(run* [q]
      (membero q  [:b :c])
      (membero :a [:b q]))

=> [ ]
```

DISJUNCTION

```
(run* [q]
   (conde
      [(membero q  [:b :c])]
      [(membero :a [:b q])]))
```

```
=> [:b :c :a]
```

EQUALITY

(unification, actually)

```
(run* [q]
      (== q :b))
```

```
=> [:b]
```

DISEQUALITY

```
(run* [q]
      (membero q [:b :c])
      (!= q :b))

=> [:c]
```

FRESH

```
(run* [q]
   (fresh [e]
      (member° q [:b e])
      (== e :a)))

=> [:b :a]
```

(enlightenment° q )

Logic & Data

SELECT * FROM GEEKS WHERE GEEK_ID=0;

geed_id	geek_name
0	Archimedes

OOP:

OOP: $\mathcal{Q} \longrightarrow \mathcal{D} \longrightarrow \mathcal{R}$

FP: $f(\mathcal{Q}, \mathcal{D}) \longrightarrow \mathcal{R}$

OOP: $\mathcal{Q} \longrightarrow \mathcal{D} \longrightarrow \mathcal{R}$

FP: $f(\mathcal{Q}, \mathcal{D}) \longrightarrow \mathcal{R}$

LP: $f^{\circ}(\mathcal{Q}, \mathcal{D})$

$$f^{\circ}(\mathcal{Q}, \mathcal{D})$$

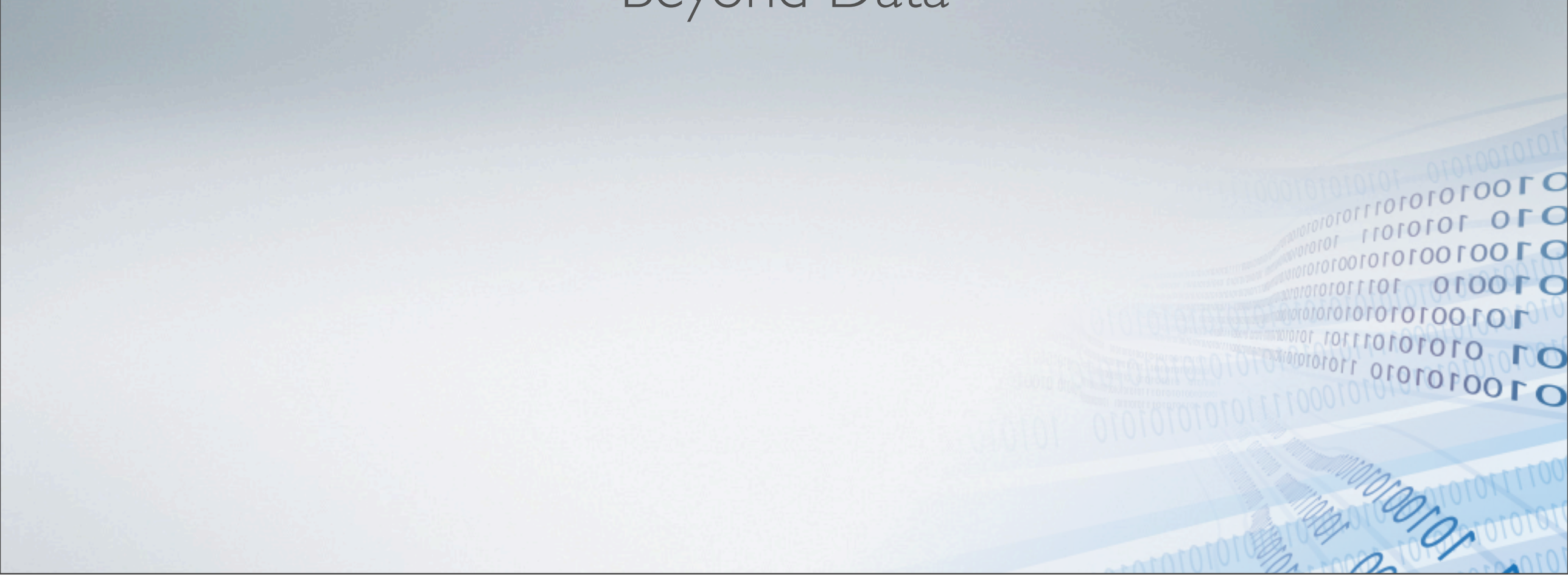
```
(member° q D)
```

$$f^{\circ}(Q, D)$$

```
(member° q D)
```

```
(run* [q]
   (member° q D)
   (== q {:geek_id 0 :geek_name _}))
```

Beyond Data



$\mathcal{D}$

```
(run* [q] (fooº q D))
```

$\#\{d_0, d_1, \ldots, d_n\}$

```
(run* [q] (fooº q D))
```

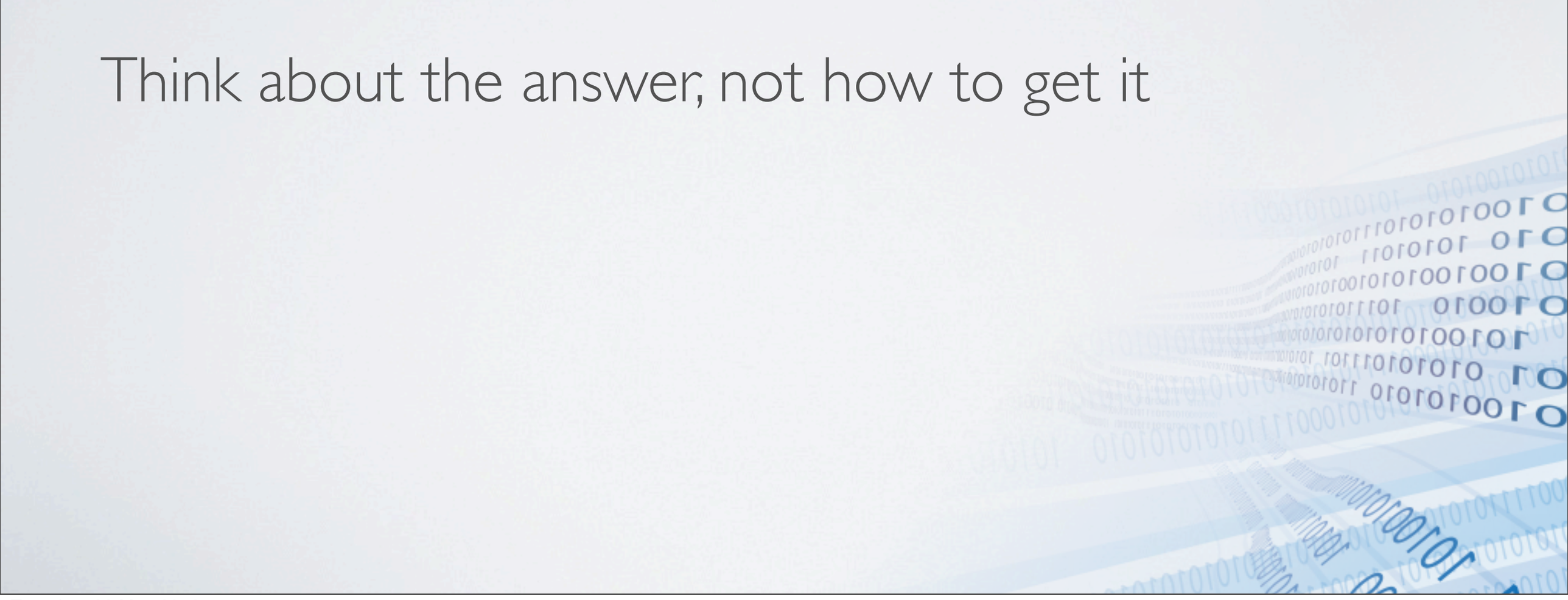
```
(run* [D] (fooº q D))
```

TIMETABLING

WHY

Separate the search algo from the logical construction

Think about the answer, not how to get it

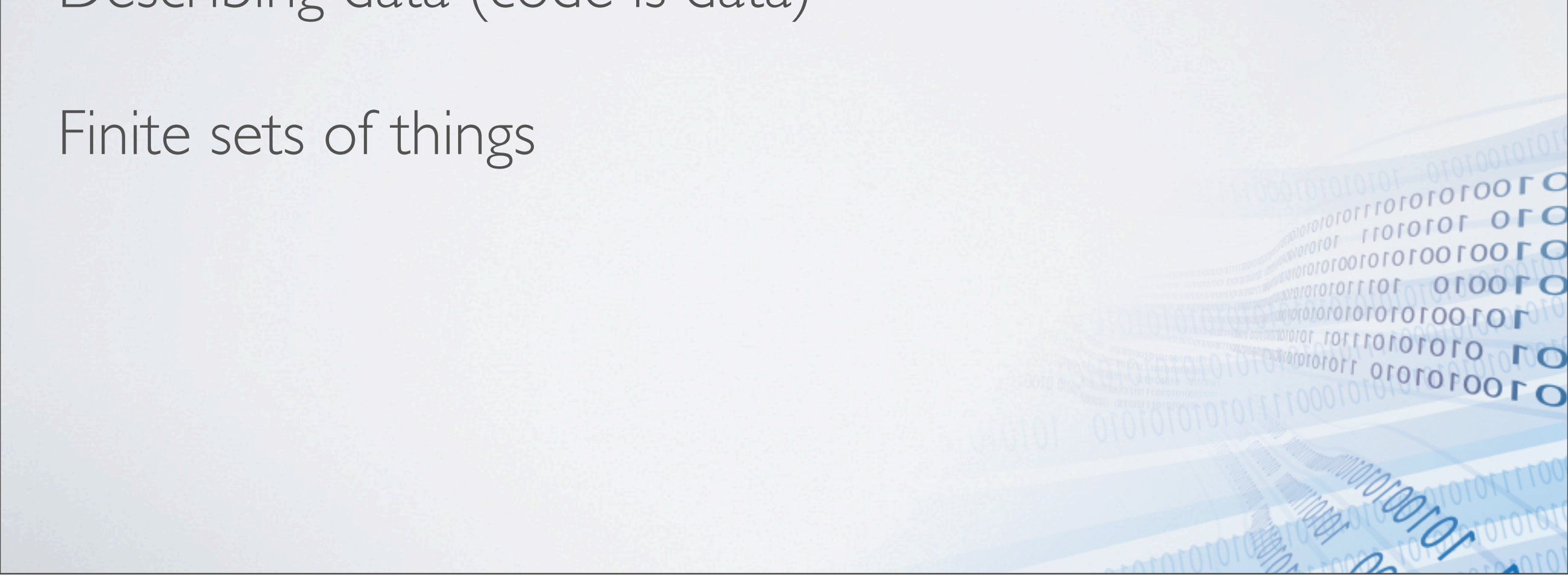


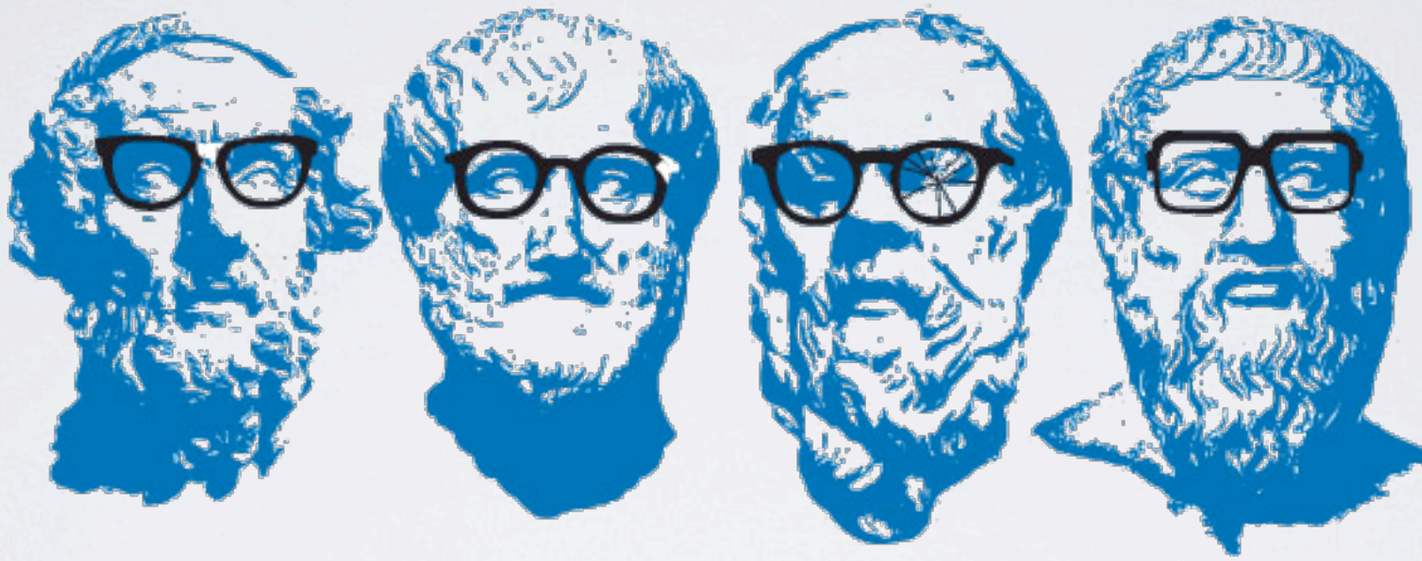
WHEN

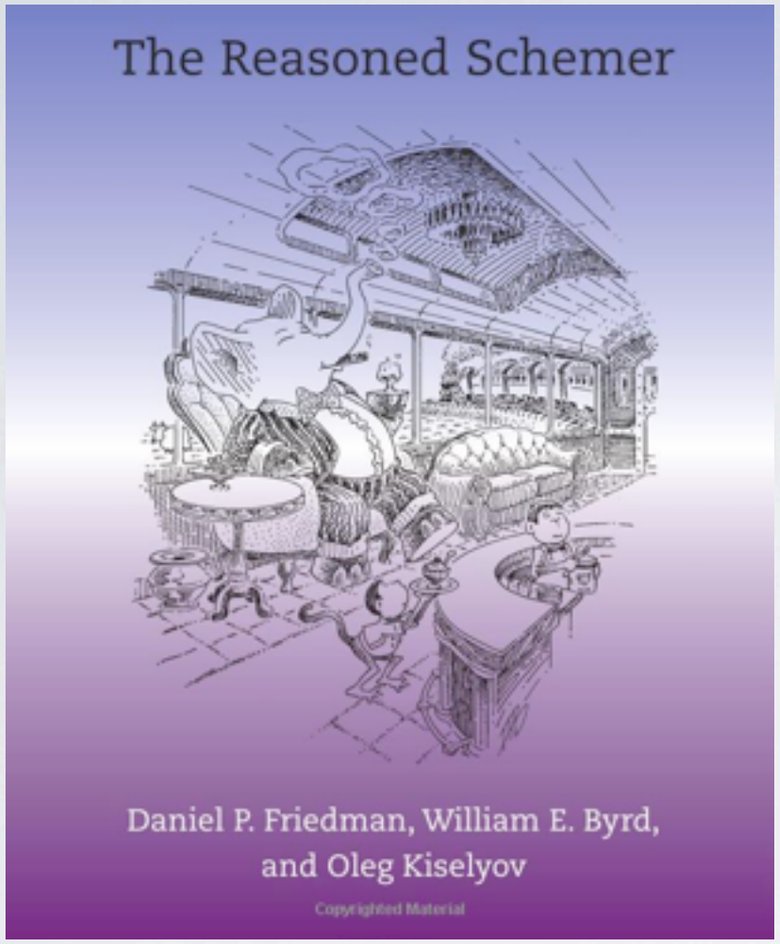
Relations

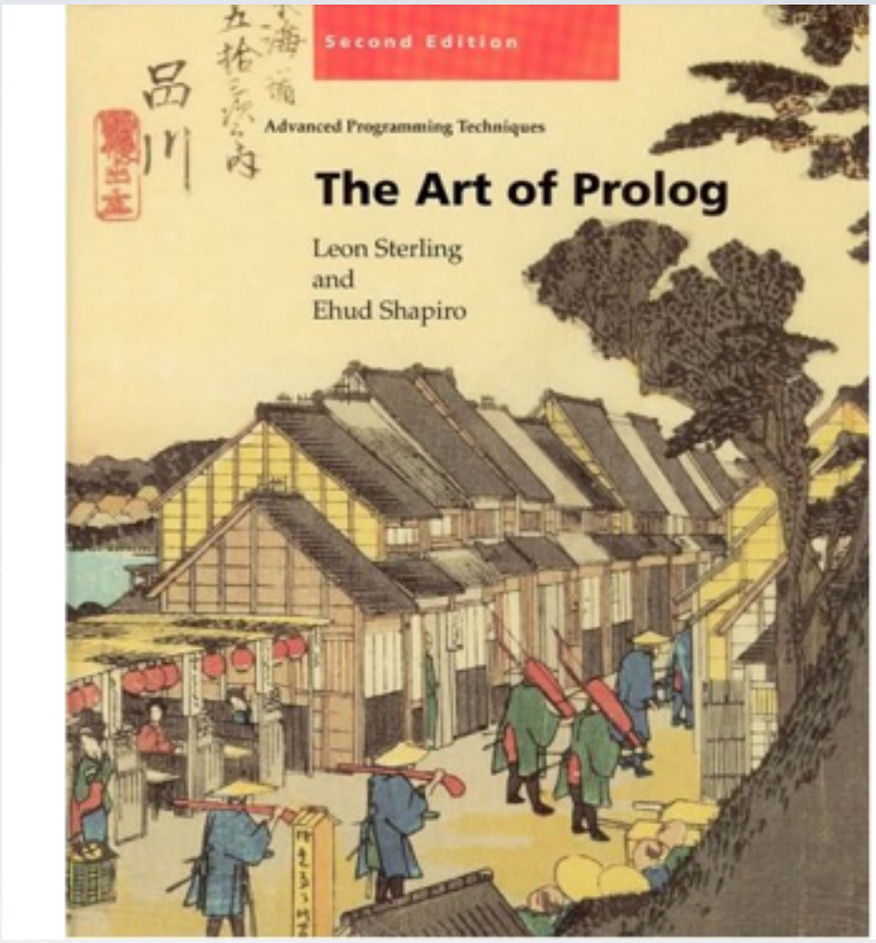
Describing data (code is data)

Finite sets of things









http://blip.tv/clojure/ambrose-bonnaire-sergeant-introduction-to-logic-programming-with-clojure-5936196

http://blip.tv/clojure/dan-friedman-and-william-byrd-minikanren-5936333

https://github.com/clojure/core.logic

https://github.com/clojure/core.logic/wiki/A-Core.logic-Primer

Thank You

Rich Hickey
Jim Duey
David Nolen
Christophe Grand

Geeks, Ancient and Modern