Programmation en Clogique

Lars Hupel
:clojureD
2020-02-29

INNOQ

```
(defne presentation [orateur]
  ([:lars]
    (fresh [questions]
      (blague :drole) #_(rire)
      (introduction :corelogic)
      (fonctions :cool)
      (audience questions)
      (repondre questions))))
```

Programmation en Clogique

Lars Hupel
:clojureD
2020-02-29

INNOQ

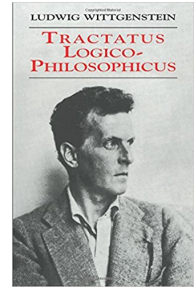
```
(defne presentation [speaker]
  ([:lars]
    (fresh [questions]
      (joke :funny) #_(laugh)
      (introduction :corelogic)
      (features :cool)
      (audience questions)
      (answer questions))))
```

```
(defne presentation [speaker]
  ([:lars]
    (fresh [questions]
      (joke :funny) #_(laugh)
      (introduction :corelogic)
      (features :cool)
      (audience questions)
      (answer questions))))
```

Who invented logic programming?

> *1. The world is everything that is the case.*
> *1.1 The world is the totality of facts, not of things.*
> *1.11 The world is determined by the facts, and by these being all the facts.*

Who invented logic programming?



> *1. The world is everything that is the case.*
> *1.1 The world is the totality of facts, not of things.*
> *1.11 The world is determined by the facts, and by these being all the facts.*

– Ludwig Wittgenstein, 1918

```
(defne presentation [speaker]
  ([:lars]
    (fresh [questions]
      (joke :funny) #_(laugh)
      (introduction :corelogic)
      (features :cool)
      (audience questions)
      (answer questions))))
```

Who invented Prolog?

- appeared in the early 70s in France
- original developers: Alain Colmerauer and Philippe Roussel
- used the `.pl` extension before Perl
- radically different programming paradigm
- this talk: using core.logic syntax

A brief primer on Prolog

1. Prolog programs are sequences of **rules** (or **clauses**).

A brief primer on Prolog

1. Prolog programs are sequences of **rules** (or **clauses**).
2. Rules can have arguments.

A brief primer on Prolog

1. Prolog programs are sequences of **rules** (or **clauses**).
2. Rules can have arguments.
3. Rules can have conditions.

A brief primer on Prolog

1. Prolog programs are sequences of **rules** (or **clauses**).
2. Rules can have arguments.
3. Rules can have conditions.
4. Programs can be queried.

A brief primer on Prolog

1. Prolog programs are sequences of **rules** (or **clauses**).
2. Rules can have arguments.
3. Rules can have conditions.
4. Programs can be queried.
5. Anything that is not in the program is not true.

Erlang, inspired by Prolog

> *The first interpreter was a simple Prolog meta interpreter which added the notion of a suspendable process to Prolog … [it] was rapidly modified (and re-written) …*
>
> – Armstrong, Virding, Williams: Use of Prolog for developing a new programming language

Hello World!

Program

```
(defn hi [])
```

Hello World!

Program

```
(defn hi [])
```

Interpreter

```
=> (run* (hi))
(_0)
```

Hello World!

Program

```
(defn hi [])


(defn hello [x]
  (== x :world))
```

Interpreter

```
=> (run* (hi))
(_0)
```

Hello World!

Program

```
(defn hi [])


(defn hello [x]
  (== x :world))
```

Interpreter

```
=> (run* (hi))
(_0)

=> (run* [_] (hello :world))
(_0)
```

Hello World!

Program

```
(defn hi [])


(defn hello [x]
  (== x :world))
```

Interpreter

```
=> (run* (hi))
(_0)

=> (run* [_] (hello :world))
(_0)

=> (run* [_] (hello :coworld))
()
```

Hello World!

Program

```
(defn hi [])


(defn hello [x]
  (== x :world))
```

Interpreter

```
=> (run* (hi))
(_0)

=> (run* [_] (hello :world))
(_0)

=> (run* [_] (hello :coworld))
()
```

Hello World!

Program

```
(defn hi [])


(defn hello [x]
  (== x :world))
```

Interpreter

```
=> (run* (hi))
(_0)

=> (run* [_] (hello :world))
(_0)

=> (run* [_] (hello :coworld))
()

=> (run* [x] (hello x))
(:world)
```

A small program

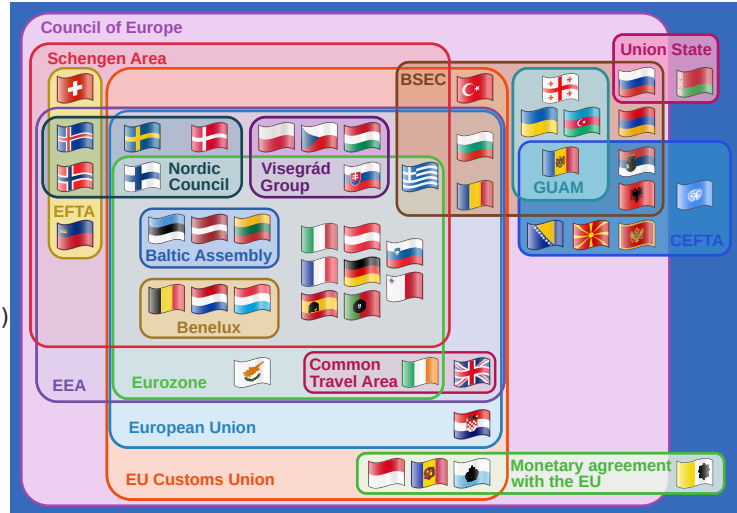
Facts

```
(defne location [place in]
  ([:munich :germany])
  ([:augsburg :germany])
  ([:germany :europe])
  ([:london :unitedkingdom])
  ([:unitedkingdom :europe]))
```

A small program

Facts

```
(defne location [place in]
  ([:munich :germany])
  ([:augsburg :germany])
  ([:germany :europe])
  ([:london :unitedkingdom])
  ([:unitedkingdom :europe]))
```



A small program

Facts

```
(defne location [place in]
  ([:munich :germany])
  ([:augsburg :germany])
  ([:germany :europe])
  ([:london :unitedkingdom])
  ([:unitedkingdom :europe]))
```

A small program

Facts

```
(defne location [place in]
  ([:munich :germany])
  ([:augsburg :germany])
  ([:germany :europe])
  ([:london :unitedkingdom])
  ([:unitedkingdom :europe]))
```

Rules

```
(defn neighbour [x y]
  (fresh [z]
    (location x z)
    (location y z)))
```

A small program

Facts

```
(defne location [place in]
  ([:munich :germany])
  ([:augsburg :germany])
  ([:germany :europe])
  ([:london :unitedkingdom])
  ([:unitedkingdom :europe]))
```

Rules

```
(defn is-in [x y]
  (conde
    [(location x y)]
    [(fresh (z)
        (location x z)
        (is-in z y))]))
```

```
(defne presentation [speaker]
  ([:lars]
    (fresh [questions]
      (joke :funny) #_(laugh)
      (introduction :corelogic)
      (features :cool)
      (audience questions)
      (answer questions))))
```

Backtracking

```
(defn best-boy [x]
  (fresh [y]
    (dog :good x)
    (colour :dark_brown x)
    (behind x y)
    (colour :light_brown y)))
```

Backtracking

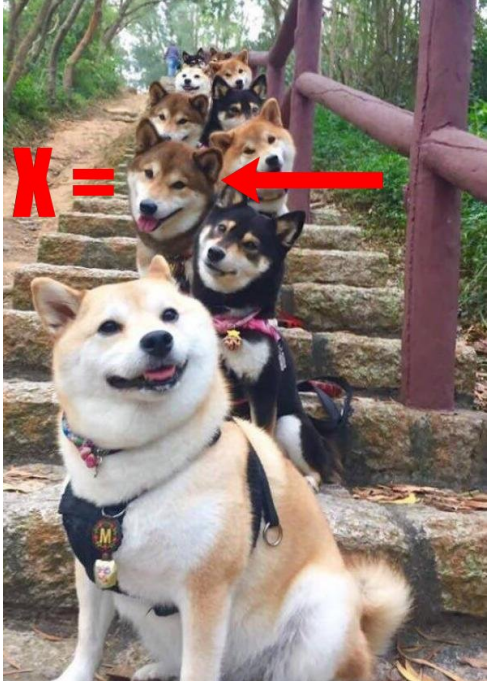
```
(defn best-boy [x]
  (fresh [y]
    (dog :good x)
    (colour :dark_brown x)
    (behind x y)
    (colour :light_brown y)))
```

Backtracking

```
(defn best-boy [x]
  (fresh [y]
    (dog :good x)
    (colour :dark_brown x)
    (behind x y)
    (colour :light_brown y)))
```

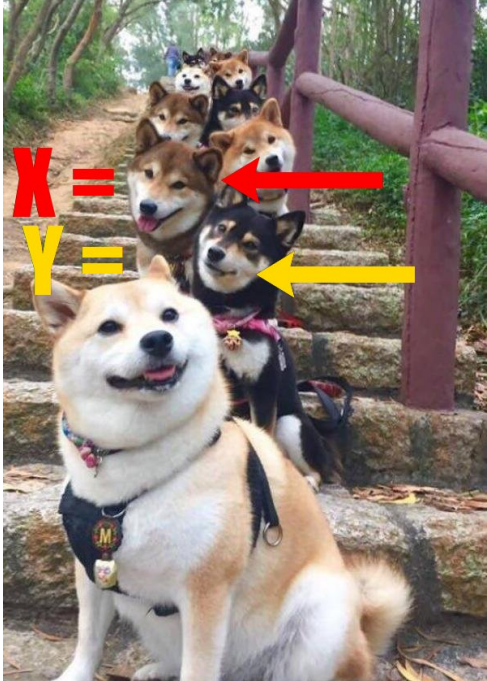
Backtracking

```
(defn best-boy [x]
  (fresh [y]
    (dog :good x)
    (colour :dark_brown x)
    (behind x y)
    (colour :light_brown y)))
```



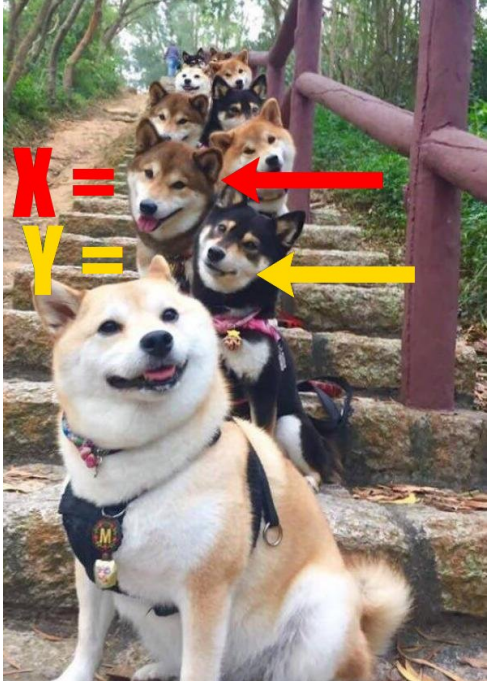
Backtracking

```
(defn best-boy [x]
  (fresh [y]
    (dog :good x)
    (colour :dark_brown x)
    (behind x y)
    (colour :light_brown y)))
```



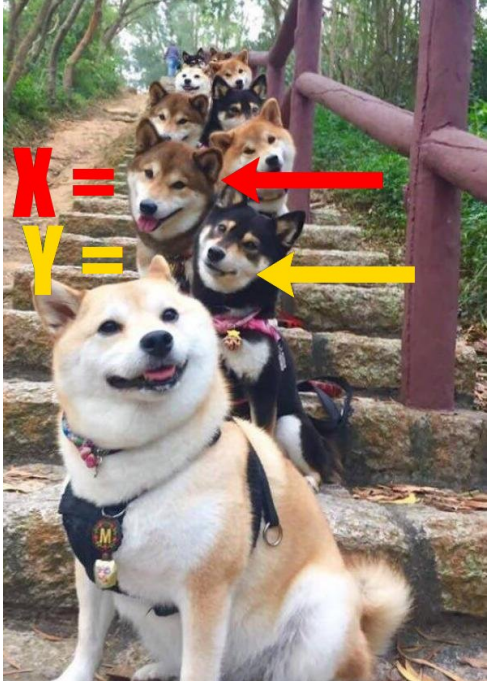
Backtracking

```
(defn best-boy [x]
  (fresh [y]
    (dog :good x)
    (colour :dark_brown x)
    (behind x y)
    (colour :light_brown y)))
```



Backtracking

```
(defn best-boy [x]
  (fresh [y]
    (dog :good x)
    (colour :dark_brown x)
    (behind x y)
    (colour :light_brown y)))
```



Backtracking

```
(defn best-boy [x]
  (fresh [y]
    (dog :good x)
    (colour :dark_brown x)
    (behind x y)
    (colour :light_brown y)))
```

Backtracking

```
(defn best-boy [x]
  (fresh [y]
    (dog :good x)
    (colour :dark_brown x)
    (behind x y)
    (colour :light_brown y)))
```

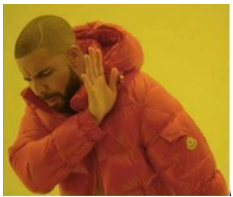
Backtracking

```
(defn best-boy [x]
  (fresh [y]
    (dog :good x)
    (colour :dark_brown x)
    (behind x y)
    (colour :light_brown y)))
```

Bi-directional computing

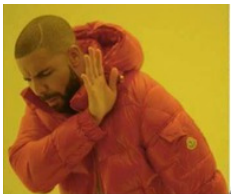
$$f : I \rightarrow O$$

Bi-directional computing



$$f : I \rightarrow O$$

Bi-directional computing



$$f : I \rightarrow O$$

$$R : (I \times O) \rightarrow \{0, 1\}$$

Bi-directional computing

Clojure

```
(concat xs ys)
```

Bi-directional computing

Clojure

```
(concat xs ys)
```

core.logic

```
(appendo xs ys zs)
```

Not a silver bullet ...

How to reverse `flatten`?

What about numbers?

```
=> (run* [x y] (membero x [1 2 3]) (== y 2) (> x y))
```

What about numbers?

```
=> (run* [x y] (membero x [1 2 3]) (== y 2) (> x y))
```

What about numbers?

```
=> (run* [x y] (membero x [1 2 3]) (== y 2) (> x y))
```

```
=> (require '[clojure.core.logic.fd :as fd])
=> (run* [x y] (membero x [1 2 3]) (== y 2) (fd/> x y))
([3 2])
```

Fact databases

How to model facts evolving over time?

Fact databases

How to model facts evolving over time?

Self-modifying code? 🔥🤯😱

Fact databases

How to model facts evolving over time?

Self-modifying code? 🔥🤯😱

Just like in SQL!

PLDB

```
(pldb/db-rel location p q)

(def facts
  (pldb/db
    [location :munich :germany]
    [location :augsburg :germany]
    [location :germany :europe]
    [location :london :unitedkingdom]
    [location :unitedkingdom :europe]))
```

PLDB as real database

- data model is similar to SQL: sets of tuples (“relations”)
- PLDB is static & immutable

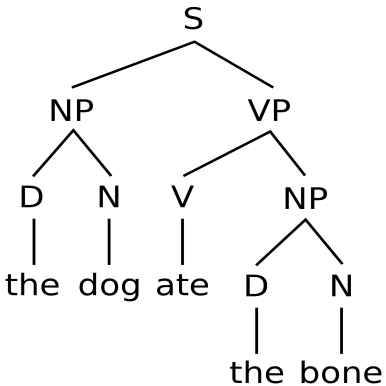
Constraint solving

Puzzle

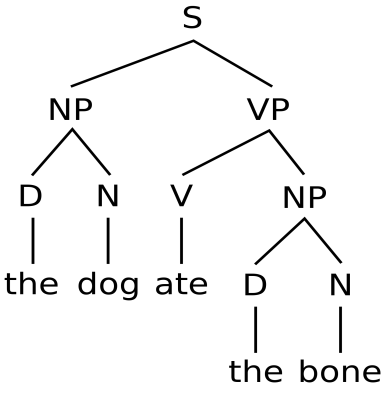
There are five houses.

1. The English person lives in the red house.
2. The Swedish person owns a dog.
3. The Danish person likes to drink tea.
4. The green house is left to the white house.
5. The owner of the green house drinks coffee.
6. ...

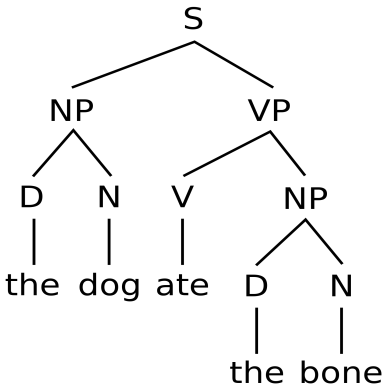
Grammars



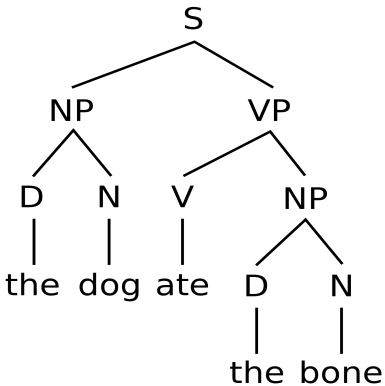
Grammars



Grammars



Grammars




```
(def-->e det [d]
  ([[:d 'the]] '[the])
  ([[:d 'a]] '[a]))

(def-->e noun-phrase [n]
  ([[:np ?d ?n]]
   (det ?d) (noun ?n)))
```

Prolog is for parsing?

> *The programming language … was born of a project aimed not at producing a programming language but at processing natural languages; in this case, French.*
>
> – Colmerauer, Roussel: The Birth of Prolog

Prolog is for parsing?

> *The programming language … was born of a project aimed not at producing a programming language but at processing natural languages; in this case, French.*
>
> – Colmerauer, Roussel: The Birth of Prolog

```
(defne presentation [speaker]
  ([:lars]
    (fresh [questions]
      (joke :funny) #_(laugh)
      (introduction :corelogic)
      (features :cool)
      (audience questions)
      (answer questions))))
```

Q & A

INNOQ
www.innoq.com

Lars Hupel

lars.hupel@innoq.com

@larsr_h

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim a. Rh.
Germany
+49 2173 3366-0

Ohlauer Str. 43
10999 Berlin
Germany

Ludwigstr. 180 E
63067 Offenbach
Germany

Kreuzstr. 16
80331 München
Germany

c/o WeWork
Hermannstrasse 13
20095 Hamburg
Germany

innoQ Schweiz GmbH

Gewerbestr. 11
CH-6330 Cham
Switzerland
+41 41 743 01 11

Albulastr. 55
8048 Zürich
Switzerland

LARS HUPEL

Consultant
innoQ Deutschland GmbH

Lars enjoys programming in a variety of languages, including Scala, Haskell, and Rust. He is known as a frequent conference speaker and one of the founders of the Typelevel initiative which is dedicated to providing principled, type-driven Scala libraries.

Image sources

- Title (map): https://pixabay.com/photos/world-europe-map-connections-1264062/
- Shiba row: https://www.pinterest.de/pin/424112489894679416/
- Shiba with mlem: https://www.reddit.com/r/mlem/comments/6tc1of/shibe_doing_a_mlem/
- Happy dog: https://www.rover.com/blog/is-my-dog-happy/
- Kid with crossed arms: https://www.psychologytoday.com/us/blog/spycatcher/201410/9-truths-exposing-myth-about-body-language
- Noam Chomsky: https://en.wikipedia.org/wiki/File:Noam_Chomsky_Toronto_2011.jpg
- Alain Colmerauer: https://de.wikipedia.org/wiki/Datei:A-Colmerauer_web-800x423.jpg
- Joe Armstrong: Erlang, the Movie
- Signatures: http://www.swi-prolog.org/pldoc/man?section=preddesc
- Zebra puzzle: StackOverflow contributors (https://stackoverflow.com/q/11122814/4776939)
- Owl: https://www.theloop.ca/angry-owl-terrorizes-oregon-joggers/
- Clock: https://pixabay.com/photos/clock-alarm-alarm-clock-dial-time-1031503/
- Files: https://pixabay.com/photos/files-ddr-archive-1633406/