Programmation en Logique

Lars Hupel
Lambda Days
2019-02-21

INNOQ


PROLOG
PROLOG
PROLOG
coffee bags: 100kr
PROLOG
PROLOG
PROLOG
PROLOG
PROLOG
PROLOG
PROLOG

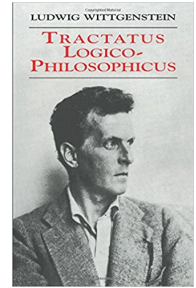
```
talk(lars) :-
  joke(funny), % laugh
  introduction(prolog),
  features(cool),
  audience(Questions),
  answer(Questions).
```

```
talk(lars) :-
  joke(funny), % laugh
  introduction(prolog),
  features(cool),
  audience(Questions),
  answer(Questions).
```

Who invented Prolog?

> *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 Prolog?



> *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

```
talk(lars) :-
  joke(funny), % laugh
  introduction(prolog),
  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

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.

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.
6. Queries may alter the program 🔥🤯😱

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.
6. Queries may alter the program 🔥🤯😱

Just like in SQL!

Hello World!

Program

```
hi.
```

Hello World!

Program

```
hi.
```

Interpreter

```
?- hi.
true.
```

Hello World!

Program

```
hi.


hello(world).
```

Interpreter

```
?- hi.
true.
```

Hello World!

Program

```
hi.


hello(world).
```

Interpreter

```
?- hi.
true.

?- hello(world).
true.
```

Hello World!

Program

```
hi.


hello(world).
```

Interpreter

```
?- hi.
true.

?- hello(world).
true.

?- hello(coworld).
false.
```

Hello World!

Program

```
hi.


hello(world).
```

Interp

```
?- hi.
true.

?- hel
true.

?- hell(coworld).
false.
```

Hello World!

Program

```
hi.


hello(world).
```

Interpreter

```
?- hi.
true.

?- hello(world).
true.

?- hello(coworld).
false.

?- hello(X).
X = world.
```

Hello World!

Program

```
hi.


hello(world).
```

Interpreter

```
?- hi.
true.

?- hello(world).
true.

?- hello(c
false.

?- hello(X).
X = world.
```

Variables: upper-case
Rest: lower-case

A small program

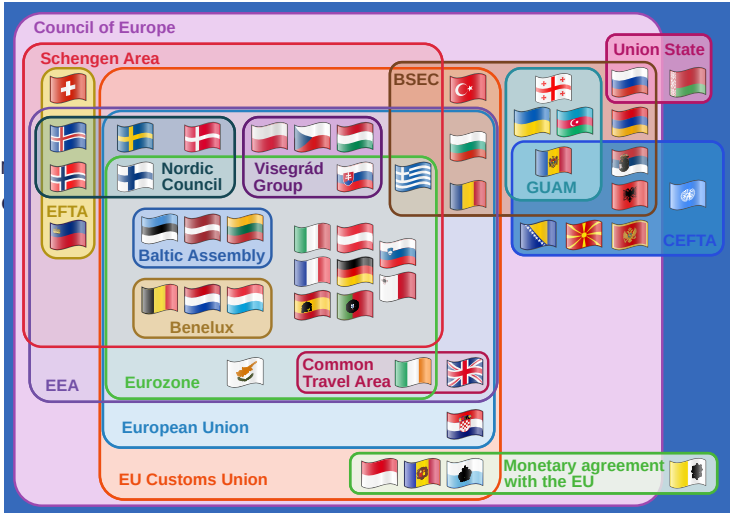
Facts

```
location(munich, germany).
location(augsburg, germany).
location(germany, europe).
location(london, unitedkingdom).
location(unitedkingdom, europe).
```

A small program

Facts

```
location(munich, germany).
location(augsburg, germany).
location(germany, europe).
location(london, unitedkingdo
location(unitedkingdom, europ
```



A small program

Facts

```
location(munich, germany).
location(augsburg, germany).
location(germany, europe).
location(london, unitedkingdom).
location(unitedkingdom, europe).
```

A small program

Facts

```
location(munich, germany).
location(augsburg, germany).
location(germany, europe).
location(london, unitedkingdom).
location(unitedkingdom, europe).
```

Rules

```
neighbour(X, Y) :-
  location(X, Z), location(Y, Z).
```

A small program

Facts

```
location(munich, germany).
location(augsburg, germany).
location(germany, europe).
location(london, unitedkingdom).
location(unitedkingdom, europe).
```

Rules

```
is_in(X, Y) :- location(X, Y).
is_in(X, Y) :- location(X, Z), is_in(Z, Y).
```

Prolog syntax

What's with the weird syntax?

Prolog syntax

What's with the weird syntax?

Is it stolen from Erlang?

Prolog syntax

What's with the weird syntax?

Is it stolen from Erlang?

Prolog syntax

What's with the weird syntax?

Is it stolen from Erlang?

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

```
talk(lars) :-
  joke(funny), % laugh
  introduction(prolog),
  features(cool),
  audience(Questions),
  answer(Questions).
```

Backtracking

```
best_boy(X) :-
  dog(good, X),
  colour(dark_brown, X),
  behind(X, Y),
  colour(light_brown, Y).
```

Backtracking

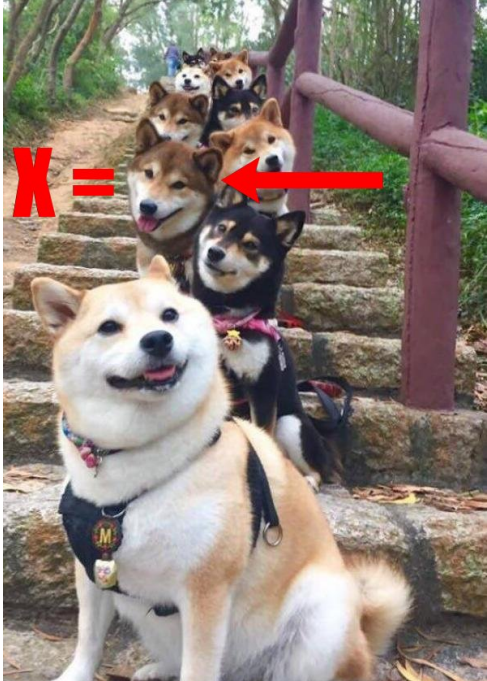
```
best_boy(X) :-
  dog(good, X),
  colour(dark_brown, X),
  behind(X, Y),
  colour(light_brown, Y).
```

Backtracking

```
best_boy(X) :-
  dog(good, X),
  colour(dark_brown, X),
  behind(X, Y),
  colour(light_brown, Y).
```

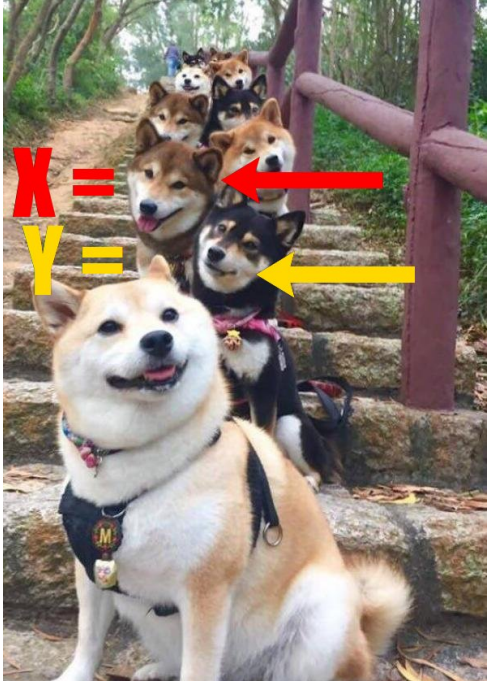
Backtracking

```
best_boy(X) :-
  dog(good, X),
  colour(dark_brown, X),
  behind(X, Y),
  colour(light_brown, Y).
```



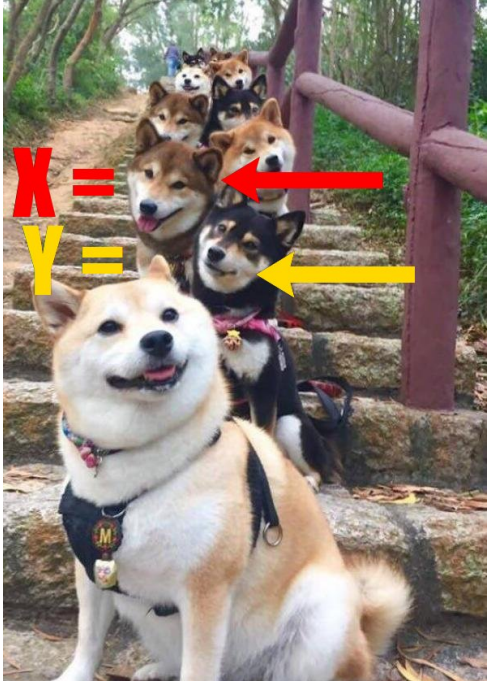
Backtracking

```
best_boy(X) :-
  dog(good, X),
  colour(dark_brown, X),
  behind(X, Y),
  colour(light_brown, Y).
```



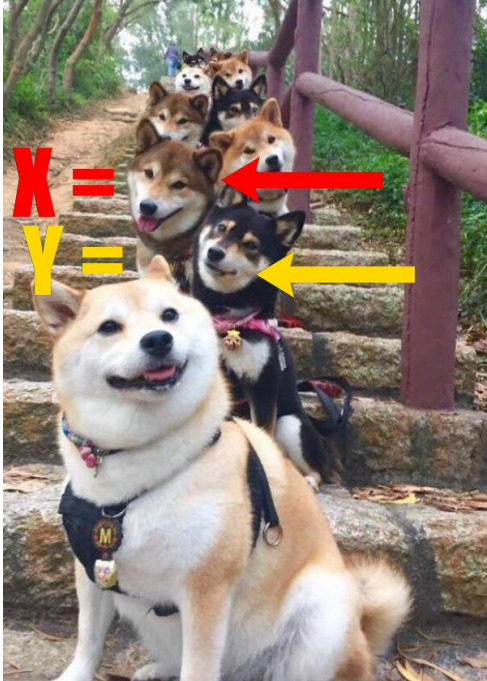
Backtracking

```
best_boy(X) :-
  dog(good, X),
  colour(dark_brown, X),
  behind(X, Y),
  colour(light_brown, Y).
```



Backtracking

```
best_boy(X) :-
  dog(good, X),
  colour(dark_brown, X),
  behind(X, Y),
  colour(light_brown, Y).
```



Backtracking

```
best_boy(X) :-
  dog(good, X),
  colour(dark_brown, X),
  behind(X, Y),
  colour(light_brown, Y).
```

Backtracking

```
best_boy(X) :-
  dog(good, X),
  colour(dark_brown, X),
  behind(X, Y),
  colour(light_brown, Y).
```

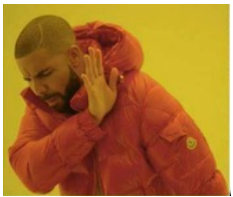
Backtracking

```
best_boy(X) :-
  dog(good, X),
  colour(dark_brown, X),
  behind(X, Y),
  colour(light_brown, Y).
```

Bi-directional computing

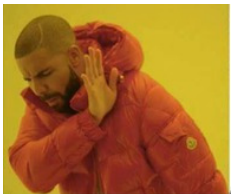
$$f : I \rightarrow O$$

Bi-directional computing



$$f : I \rightarrow O$$

Bi-directional computing



$$f : I \rightarrow O$$

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

Bi-directional computing

Scala

```
def append[A](xs: List[A], ys: List[A]): List[A]
```

Bi-directional computing

Scala

```
def append[A](xs: List[A], ys: List[A]): List[A]
```

Prolog

```
append(?List1, ?List2, ?List1AndList2)
```

Bi-directional computing

Scala

```
def appendAll[A](xss: List[List[A]]): List[A]
```

Bi-directional computing

Scala

```
def appendAll[A](xss: List[List[A]]): List[A]
```

Prolog

```
append_all(+ListOfLists, ?List)
```

Mode signatures

++	Argument must be ground, i.e., the argument may not contain a variable anywhere.
+	Argument must be fully instantiated to a term that satisfies the type. This is not necessarily *ground*, e.g., the term `[_]` is a *list*, although its only member is unbound.
-	Argument is an *output* argument. Unless specified otherwise, output arguments need not to be unbound. For example, the goal `findall(X, Goal, [T])` is good style and equivalent to `findall(X, Goal, Xs), Xs = [T]`[45] Note that the *determinism* specification, e.g., ``det'' only applies if this argument is unbound.
--	Argument must be unbound. Typically used by predicates that create `something' and return a handle to the created object, such as open/3 which creates a *stream*.
?	Argument must be bound to a *partial term* of the indicated type. Note that a variable is a partial term for any type. Think of the argument as either *input* or *output* or *both* input and output. For example, in `stream_property(S, reposition(Bool))`, the `reposition` part of the term is input and the uninstantiated *Bool* is output.
:	Argument is a meta-argument. Implies +. See chapter 6 for more information on module handling.
@	Argument is not further instantiated. Typically used for type tests.
!	Argument contains a mutable structure that may be modified using setarg/3 or nb_setarg/3.

Not a silver bullet ...

```
?- member(X, [1, 2, 3]), Y = 2, X > Y.
X = 3, Y = 2.
```

Not a silver bullet ...

```
?- member(X, [1, 2, 3]), Y = 2, X > Y.
X = 3, Y = 2.


?- X > Y, member(X, [1, 2, 3]), Y = 2.
```

Not a silver bullet ...

```
?- member(X, [1, 2, 3]), Y = 2, X > Y.
X = 3, Y = 2.
```

```
?- X > Y, member(X, [1, 2, 3]), Y = 2.
```

Not a silver bullet ...

```
?- member(X, [1, 2, 3]), Y = 2, X > Y.
X = 3, Y = 2.
```

```
?- use_module(library(clpfd)).
?- X #> Y, X in 1..3, Y = 2.
```

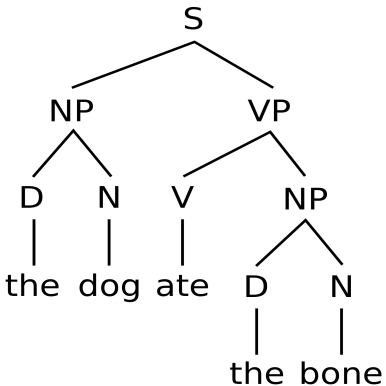
Constraint solving

Puzzle

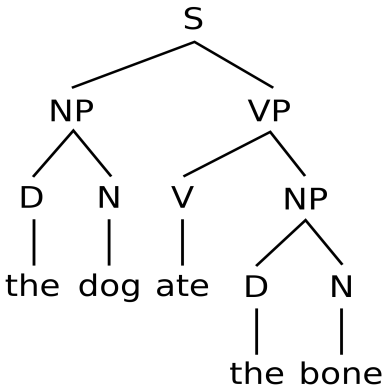
There are five houses.

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. ...

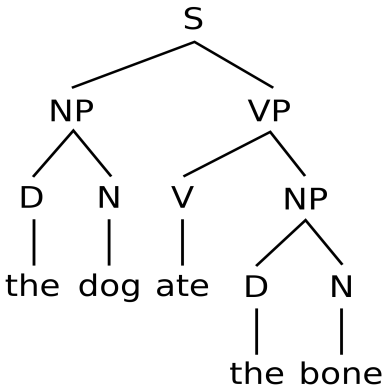
Grammars



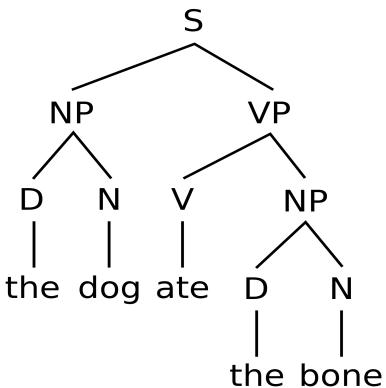
Grammars



Grammars



Grammars




```
s --> np, vp.
np --> d, n.
d --> [the].
d --> [a].
vp --> v, np.
n --> [dog].
n --> [bone].
```

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

Parsing

Scala

```
type Parser[A] = String => List[(A, String)]
```

Parsing

Scala

```
type Parser[A] = String => List[(A, String)]
```

Prolog

```
parse(?A, ?ListIn, ?ListOut)
```

Parsing

Scala

```
type Parser[A] = String => List[(A, String)]
```

+ monad syntax

Prolog

```
parse(?A, ?ListIn, ?ListOut)
```

Parsing

Scala

```
type Parser[A] = String => List[(A, String)]
```

+ monad syntax

Prolog

```
parse(?A, ?ListIn, ?ListOut)
```

+ DCG syntax

```
talk(lars) :-
  joke(funny), % laugh
  introduction(prolog),
  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

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

- Prolog coffee: Marek Kubica
- 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/