There are **five** houses.
The **Englishman** lives in the **red** house.
The **Spaniard** owns the **dog**.
**Coffee** is drunk in the **green** house.
The **Ukrainian** drinks **tea.**
The **green house** is immediately to the right of the **ivory house.**
The **Old Gold** smoker owns **snails.**
**Kools** are smoked in the **yellow** house.
**Milk** is drunk in the **middle** house.
The **Norwegian** lives in the **first** house.
The man who smokes **Chesterfields** lives in the house next to the man with the **fox.**
**Kools** are smoked in the house next to the house where the **horse** is kept.
The **Lucky Strike** smoker drinks **orange juice.**
The Japanese smokes **Parliaments.**
The **Norwegian** lives next to the **blue house.**

# Who owns the zebra and who drinks water?

# Programming in Logic

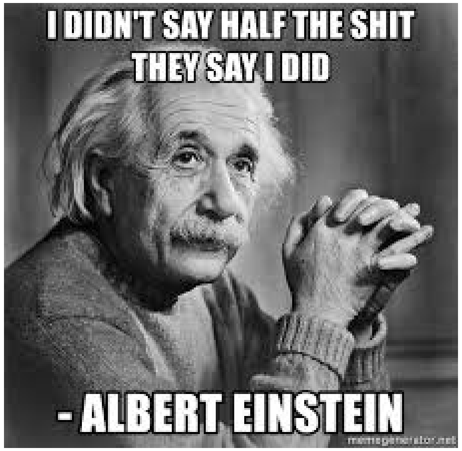
WOSSAT, Thursday 18th July 2019

Simon Merrick

Github @iokiwi · Twitter @iokiwi

# Einstein's Riddle

(alledgedly)


I DIDN'T SAY HALF THE SHIT
THEY SAY I DID
- ALBERT EINSTEIN
memegenerator.net

# There are five houses.

- There are **five** houses.
- The **Englishman** lives in the **red** house.
- The **Spaniard** owns the **dog**.
- **Coffee** is drunk in the **green** house.
- The **Ukrainian** drinks **tea.**
- The **green house** is immediately to the right of the **ivory house.**
- The **Old Gold** smoker owns **snails.**
- **Kools** are smoked in the **yellow** house.

- **Milk** is drunk in the **middle** house.
- The **Norwegian** lives in the **first** house.
- The man who smokes **Chesterfields** lives in the house next to the man with the **fox.**
- **Kools** are smoked in the house next to the house where the **horse** is kept.
- The **Lucky Strike** smoker drinks **orange juice.**
- The Japanese smokes **Parliaments.**
- The **Norwegian** lives next to the **blue house.**

Who owns the **zebra** and who drinks **water**?

# Prolog

*Programmation en logique*

*"The offspring of a successful marriage between natural language processing and automated theorem-proving."*

# 1971

The result of french research into machine natural language processing.

*"The idea of using a natural language like French to reason and communicate directly with a computer seemed like a crazy idea"*

# Formal logic

## The mathematical discipline of formal logic in 4 easy steps

1. Distill problem to notation
2. Apply rules of inference
3. ???
4. ~~Profit~~ Proof!

C+

*If it is raining, then it's cloudy.*

$$P \implies Q$$

*It is raining.*

$$P$$

*Therefore, it's cloudy.*

$$\therefore Q$$

Intuitively, we undertand this **argument** is **valid**

The mathematical **dicipline** of formal logic

1. Distill problem to notation
2. Apply rules of inference
3. ???
4. ~~Profit~~ Proof!

**Predicate logic** introduces a few more important concepts

# Universal Quantification

*For all* $x$

$$\forall x$$

# Existential Quantification

*There exists some $x$*

$$\exists x$$

# Predicates

$x$ is $C$ool

$$Cx$$

$x$ is $A$djacent to $y$

$$Axy$$

Notice the prefix notation

Using these constructs of predicate logic we can start to model the real world in logic

The englishman Lives in the red house

Ler

Finally, we can also combine **quantifiers** and **predicates**

*There exists some $x$ such that $x$ $L$ives in the $r$ed house*

$$\exists x L x r$$

We are ready to write our first prolog program!

# Prolog

Programs consist of

**facts** and **rules**

Generically, these are referred to as **clauses**

# Our first **fact**

```
human(simon).
```

$s$imon is $H$uman

$$H s$$

# Our first **query**

- **queries** start with `?-`
- evaluated as **True**, or **False**

```
?-human(simon).
True
```

# Using **variables** in our **queries**

```
?-human(X).
```

*Does there exist some $x$ such that $x$ is $H$uman?*

$$\exists x Hx$$

Prolog look through **facts** it knows about for one that makes the query **True**

Yes!

```
?-human(X).
X=simon
```

$s$imon is $H$uman

$$H s$$

# Family trees

*The FizBuzz of Prolog*

```
father(jamie, tommen).
father(jamie, myrcella).
father(jamie, joffrey).
```

```
mother(cersei, tommen).
mother(cersei, myrcella).
mother(cersei, joffrey).
```

```
?-father(X, tommen)
X=Jamie
```

```
?-father(jamie, Y)
Y=tommen
Y=myrcella
Y=joffrey
```

```
?-father(X, Y)
X=jamie, Y=tommen
X=jamie, Y=myrcella
X=jamie, Y=joffrey
```

# We could go further and define some sibling facts

```
sibling(tommen, myrcella).
sibling(tommen, joffrey).
sibling(joffrey, myrcella).
```

Neither elegant nor scalable.

$$R = (\frac{n^2}{2}) - n$$

There has to be a better way...

# Rules

Specify relationships between facts

*$X$ and $Y$ are siblings if $X$ and $Y$ share a mother or a father*

```
sibling(X, Y) :-
    mother(Z, X),
    mother(Z, Y),
    X \== Y.

sibling(X, Y) :-
    father(Z, X),
    father(Z, Y),
    X \== Y.
```

```
?- sibling(X, Y).
X = tommen, Y = myrcella
X = tommen, Y = joffrey
X = myrcella, Y = tommen
```

# More relations...

```
uncle_or_aunt(X, Y) :-
    mother(M, Y),
    sibling(M, X).

uncle_or_aunt(X, Y) :-
    father(M, Y),
    sibling(X, M).
```

```
father(tywin, jamie).
father(tywin, cersei).
father(tywin, tyrion).
```

```
?- uncle_or_aunt(X ,Y).
X = jamie, Y = tommen
X = tyrion, Y = tommen
X = jamie, Y = myrcella
X = tyrion, Y = myrcella
X = jamie, Y = joffrey
X = tyrion, Y = joffrey
X = cersei, Y = tommen
X = tyrion, Y = tommen
X = cersei, Y = myrcella
X = tyrion, Y = myrcella
X = cersei, Y = joffrey
X = tyrion, Y = joffrey
```

# Lists and some operations

```
[1, 2, 3]
[one, two , three]
```

# Referencing items in lists

[ F | R ]

*The first part of the list, and the rest of the list*

`[a, b, c]`

Then `[ F | R ]` equates to

F=a, R=`[b, c]`

# The append clause

Prolog has a usefull clause for appending to a list.

```
append(A, B, C)
```

```
?-append([1], [2, 3], C)
C=[1, 2, 3]
```

append is nothing more than a clause

*Succeeds if C is the result of appending B to A*

Prolog is working out the value(s) for C which make the append() clause **True**

But, because this is prolog,

we can do this

```
?-append([1], B, [1, 2, 3]).
B=[2,3]
```

# and this

```
?-append(A, B, [1, 2, 3]).
A = [],         B = [1, 2, 3]
A = [1],        B = [2, 3]
A = [1, 2],     B = [3]
A = [1, 2, 3], B = []
```

# The "Don't care" variable

_

*Used like a variable but it tells prolog we **don't care** what it's value is.*

`[a, b, c]`

Then `[F|_]` equates to

`F = a`, we **don't care** about the rest

# Solving Einstein's Riddle

There are **five** houses.
The **Englishman** lives in the **red** house.
The **Spaniard** owns the **dog**.
**Coffee** is drunk in the **green** house.
The **Ukrainian** drinks **tea.**
The **green house** is immediately to the right of the **ivory house.**
The **Old Gold** smoker owns **snails.**
**Kools** are smoked in the **yellow** house.
**Milk** is drunk in the **middle** house.
The **Norwegian** lives in the **first** house.
The man who smokes **Chesterfields** lives in the house next to the man with the **fox.**
**Kools** are smoked in the house next to the house where the **horse** is kept.
The **Lucky Strike** smoker drinks **orange juice.**
The Japanese smokes **Parliaments.**
The **Norwegian** lives next to the **blue house.**

Who owns the zebra and who drinks water?

For each house there are 5 factors to consider

- The nationality of the **Owner**
- The **Pet**
- The **Cigarret** brand
- The **Drink**
- The **Color**

# A **fact** for houses

```
house(Owner, Pet, Cigarette, Drink, Color)
```

# The houses rule

Succeeds when $H$ is a list of 5 **facts** which, collectively, satisfy requirements 2 - 15

```
houses(H) :-
    % There are 5 houses,
    % The Englishman lives in the red house,
    % The Spaniard owns the dog,
```

We can start building up facts about the houses piece by piece

We'll use the **don't care** variable where information is not provided *there are 5 houses*

```
houses(H) :-
    length(H, 5),
    ...
```

Suceeds if $|H| = 5$

*The **Englishman** lives in the **red** house.*

```
houses(H) :-
    ...
    member(house(englishman,_,_,_,red), H),
    ...
```

*The **Spaniard** owns the **dog**.*

```
houses(H) :-
    ...
    member(house(spaniard,dog,_,_,_), H),
    ...
```

***Coffee*** *is drunk in the* ***green*** *house.*

```
houses(H) :-
    ...
    member(house(_,_,_,coffee,green), H),
    ...
```

## *The **Ukrainian** drinks **tea***

```
houses(H) :-
     ...
     member(house(ukrainian,_,_,tea,_), H),
     ...
```

*The **green** house is immediately to the right of the **ivory** house.*

We need a **rule** to determine which houses are next to one another

# The `next(A, B)` clause

Houses $A$ and $B$ are next to each other if

$A$ is next to $B$

```
next(A, B, L) :-

append(_, [A,B|_], L).
```

Or if $B$ is next to $A$

```
next(A, B, L) :-

append(_, [B,A|_], L).
```

*The **green** house is immediately to the right of the **ivory** house.*

```
houses(H) :-
    ...
    next(house(_,_,_,_,ivory),house(_,_,_,_,green), H),
    ...
```

*The **Old Gold** smoker owns **snails**.*

```
houses(H) :-
    ...
    member(house(_,snails,gold,_,_), H),
    ...
```

***Kools*** *are smoked in the* ***yellow*** *house.*

```
houses(H) :-
    ...
    member(house(_,_,kools,_,yellow), H),
    ...
```

***Milk*** *is drunk in the* ***middle house***.

```
houses(H) :-
    ...
    H = [_,_,house(_,_,_,milk,_),_,_],
    ...
```

*The **Norwegian** lives in the **first house**.*

```
houses(H) :-
    ...

    H = [house(norwegian,_,_,_,_)|_],
    ...
```

*The man who smokes **Chesterfields** lives in the house next to the man with the **fox**.*

```
houses(H) :-
    ...

    next(house(_,fox,_,_,_), house(_,_,chesterfield,_,_), H),
    ...
```

***Kools*** *are smoked in the house next to the house where the* ***horse*** *is kept.*

```
houses(H) :-
    ...

    next(house(_,_,kools,_,_), house(_,horse,_,_,_), H),
    ...
```

*The **Lucky Strike** smoker drinks **orange juice**.*

```
houses(H) :-
    ...

    member(house(_,_,lucky,juice,_), H),
    ...
```

*The **Japanese** smokes **Parliaments.***

```
houses(H) :-
    ...
    member(house(japanese,_,parliaments,_,_), H),
    ...
```

*The **norwegian** lives next to the **blue** house*

```
houses(H) :-
    ...
    next(house(norwegian,_,_,_,_), house(_,_,_,_,blue), H).
    ...
```

# The Zebra Owner Rule

Succeeds when some list $H$ meets all of the 15 criteria and, contains a house with a zebra.

```
zebra_owner(O) :-
    houses(H),
    member(house(O,zebra,_,_,_), H).
```

No facts explicitly match **Zebra**

But this rule will also match any facts with no pet value.

# There was only one

```
?-zebra_owner(O).
O=japanese
```

*The **Japanese** man owns the **Zebra***

# The Water Drinker rule

Succeeds when some list $H$ meets all of the 15 criteria and, contains a house where water is drunk.

```
water_drinker(D) :-
    houses(H),
    member(house(D,_,_,water,_), H).
```

Like the Zebra rule, this rule will match any facts with no **Drink** value.

## There was only one

```
?-water_drinker(D).
D=norwegian
```

*The **Norwegian** man drinks the **Water***

# Programming in Logic

WOSSAT, Thursday 18th July 2019

Simon Merrick

Github @iokiwi · Twitter @iokiwi

# Resources

- 4 Programming Paradigms in 40 minutes
  https://youtu.be/cgVVZMfLjEI?t=1185
- The Birth of Prolog
  http://web.archive.org/web/20070703003934/www.l
  mrs.fr/~colmer/ArchivesPublications/HistoireProlog/1

- https://en.wikibooks.org/wiki/Prolog
- http://www.cs.trincoll.edu/~ram/cpsc352/notes/prolo
- http://infolab.stanford.edu/~ullman/focs/ch12.pdf

# Online Compilers

- https://swish.swi-prolog.org/
- https://www.tutorialspoint.com/execute_prolog_online