Logik für Informatiker: PROLOG
Part 1: Introduction

Andreas Karwath

&

Wolfram Burgard

(original slides by Peter Flach)

- Prolog
  - PROgramming in LOGic
  - PROgramming Language Of God ☺
- Why Prolog ?
  - `completely different programming paradigm`
  - `a declarative programming language (as Haskell)`
    - focus on WHAT instead of HOW
  - `based on first order logic`
  - `used especially in artificial intelligence, natural language processing, search problems, expert systems, databases, …`
  - `very elegant / powerful / compact`

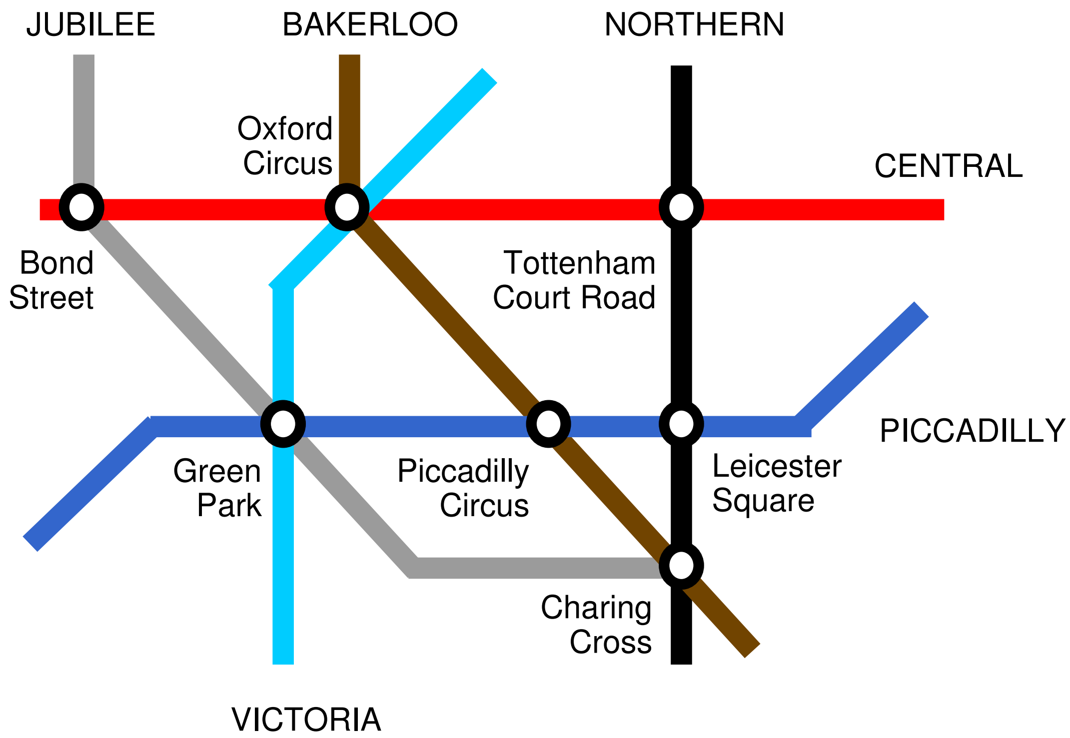
- What we shall do
  - short introduction to logic
  - Prolog as a programming language
  - various programming techniques
  - examples from artificial intelligence
  - a bit of theory / a lot of practice
  - some larger programs

- Materials

  - Simply Logical“ by Peter Flach, Addison-Wiley, 1994. (printed on demand) – 1st few chapters (free download: http://www.cs.bris.ac.uk/~flach/SimplyLogical.html)

  - PROLOG. Programming for Artificial Intelligence, by Ivan Bratko, Addison-Wesley, Third Edition 2001 , next chapters … (Old german version might be availble)

  - Prolog:
    - YAP Prolog (http://sourceforge.net/projects/yap/ or http://www.ncc.up.pt/~vsc/Yap/)
    - SWI Prolog (http://www.swi-prolog.org/)
    - Many more: GNU Prolog, Visual Prolog, …



UNDERGROUND

LRT Registered User No. 94/1954

London Underground example


```
connected(bond_street,oxford_circus,central).
connected(oxford_circus,tottenham_court_road,central).
connected(bond_street,green_park,jubilee).
connected(green_park,charing_cross,jubilee).
connected(green_park,piccadilly_circus,piccadilly).
connected(piccadilly_circus,leicester_square,piccadilly).
connected(green_park,oxford_circus,victoria).
connected(oxford_circus,piccadilly_circus,bakerloo).
connected(piccadilly_circus,charing_cross,bakerloo).
connected(tottenham_court_road,leicester_square,northern).
connected(leicester_square,charing_cross,northern).
```


London Underground in Prolog (1)

Two stations are nearby if they are on the same line with at most one other station in between:

```
nearby(bond_street,oxford_circus).
nearby(oxford_circus,tottenham_court_road).
nearby(bond_street,tottenham_court_road).
nearby(bond_street,green_park).
nearby(green_park,charing_cross).
nearby(bond_street,charing_cross).
nearby(green_park,piccadilly_circus).
```

or better

```
nearby(X,Y):-connected(X,Y,L).
nearby(X,Y):-connected(X,Z,L),connected(Z,Y,L).
```

Facts: unconditional truths
Rules/Clauses: conditional truths

Both definitions are equivalent.

London Underground in Prolog (2)

- Query:
  which station is nearby Tottenham Court Road?

```
?- nearby(tottenham_court_road, W).
```

- Prefix `?-` means it‘s a query and not a fact.

- Answer to query is:

  ```
  {W -> leicester_square}
  ```

  a so-called *substitution*.

- When nearby defined by facts, substitution found by simple *matching*.

Answering queries (1)

Exercise 1.1:

Define a predicate `not_too_far`, which is true, if two stations are one the same or a different line, with at most one station in between.

Exercise 1.1

Remember `nearby`:

```
nearby(X,Y):-connected(X,Y,L).
nearby(X,Y):-connected(X,Z,L),connected(Z,Y,L).
```

Then `not_too_far` is:

```
not_too_far(X,Y):-connected(X,Y,L).
not_too_far(X,Y):-connected(X,Z,L1),connected(Z,Y,L2).
```

This can be rewritten with don't cares:

```
not_too_far(X,Y):-connected(X,Y,_).
not_too_far(X,Y):-connected(X,Z,_),connected(Z,Y,_).
```


Exercise 1.1

- If clauses are involved, then answering a query can take several steps.

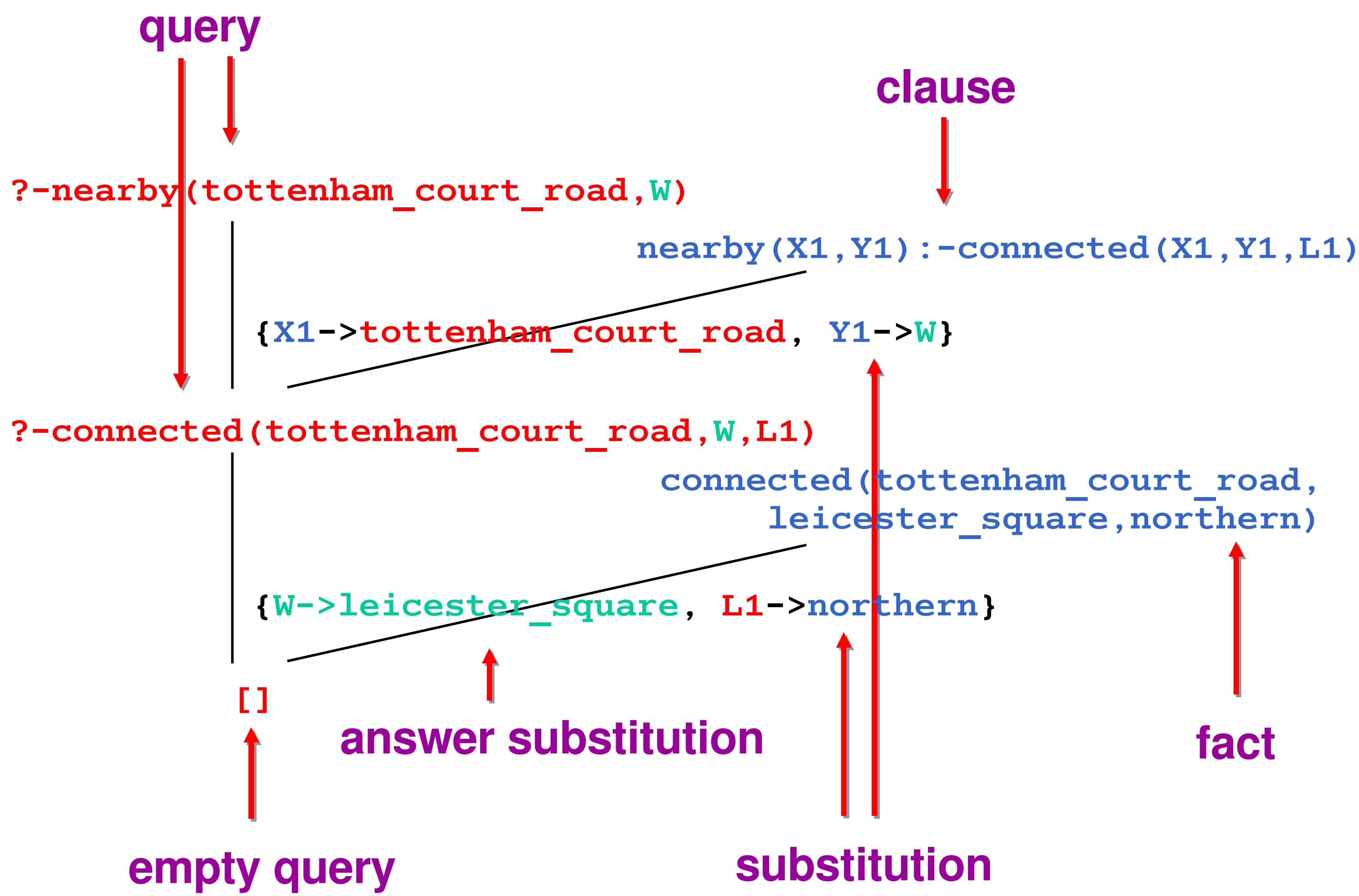
- `?- nearby(tottenham_court_road, W).`
  matches *conclusion* (*head*) of clause
  `nearby(X,Y) :- connected(X,Y,L).`
  with the substitution
  `{X -> tottenham_court_road, Y -> W}`

- Subsequently,
  `?- connected(tottenham_court_road, W, L).`
  is to prove.

- Here, looking up the facts is sufficient for answering the query:
  `{W -> leicester_square, L-> northern}`

- Result:
  `{W -> leicester_square}`

Answering queries (2)


query
clause
?-nearby(tottenham_court_road,W)
nearby(X1,Y1):-connected(X1,Y1,L1)
{X1->tottenham_court_road, Y1->W}
?-connected(tottenham_court_road,W,L1)
connected(tottenham_court_road,
leicester_square,northern)
{W->leicester_square, L1->northern}
[]
answer substitution
fact
empty query
substitution

Proof tree

- To answer a query
  `?- Q1, Q2, ..., Qn.`
  find a clause `A :- B1,..., Bm` such that `A` matches `Q1`, and then answer the query `?- B1,...,Bm,Q2,...,Qn.`

- Adds a procedural interpretation to the declarative interpretation of a logical formula

- Resolution proof: *reductio ad absurdum*; proof by refutation

- Start: clause with empty head (conclusion), e.g.:
  `:- nearby(tottenham_court_road, W).`
  (= negation of `nearby(...)`)

- Contradiction is found, if empty clause is derived. Empty clause: premise (body) is always true, because non-existing.

Resolution

- Up to now: rules (clauses) and facts
- Particular kind of rules; rules that are defined by recurring to themselves: *recursion*
- ```
  IF N = 0 THEN FAC:= 1
           ELSE FAC:= N*FAC(N-1)
  ```
- Recursion is (except for *failure-driven loops*) the only construct for loops in Prolog.
- Example relation `reachable`
  Could be defined by enumeration of facts or by non-recursive rules for routes of length 1, 2, etc.

Recursion (1)
  ```

A station is reachable from another if they are on the same line, or with one, two, … changes:

```
reachable(X,Y):-connected(X,Y,L).
reachable(X,Y):-connected(X,Z,L1),connected(Z,Y,L2).
reachable(X,Y):-
connected(X,Z1,L1),connected(Z1,Z2,L2),
                connected(Z2,Y,L3).
…
```

or better

```
reachable(X,Y):-connected(X,Y,L).
reachable(X,Y):-connected(X,Z,L),reachable(Z,Y).
```


Recursion (2)

- Recursive definition:

```
reachable(X, Y) :- connected(X, Y, L).
reachable(X, Y) :- connected(X, Z, L),
                   reachable(Z, Y).
```

- Examples so far have shown:
  Prolog performs *search* in order to answer queries

- *Backtracking:* returning to previous *choice points*, if proof *fails* at some point.

Recursion (3)

```
:-reachable(bond_street,W)                  reachable(X1,Y1):-connected(X1,Z1,L1),
                                                              reachable(Z1,Y1)
                                            {X1->bond_street, Y1->W}

:-connected(bond_street,Z1,L1),             connected(bond_street,
  reachable(Z1,W)                                     oxford_circus,
                                                      central)
                                            {Z1->oxford_circus, L1->central}

:-reachable(oxford_circus,W)                reachable(X2,Y2):-connected(X2,Z2,L2),
                                                              reachable(Z2,Y2)
                                            {X2->oxford_circus, Y2->W}

:-connected(oxford_circus,Z2,L2),           connected(oxford_circus,
  reachable(Z2,W)                                     tottenham_court_road,
                                                      central)
                                            {Z2->tottenham_court_road, L2->central}

:-reachable(tottenham_court_road,W)         reachable(X3,Y3):-connected(X3,Y3,L3)
                                            {X3->tottenham_court_road, Y3->W}

:-connected(tottenham_court_road,W,L3)      connected(tottenham_court_road,
                                                      leicester_square,
                                                      northern)
                                            {W->leicester_square, L3->northern}

[]
```


Recursion (4)

```
reachable0(X,Y):-
        connected(X,Y,L).
reachable1(X,Y,Z):-
        connected(X,Z,L1),
        connected(Z,Y,L2).
reachable2(X,Y,Z1,Z2):-
        connected(X,Z1,L1),
        connected(Z1,Z2,L2),
        connected(Z2,Y,L3).
```

One clause for each route of length *n*.

Solution: *functors*

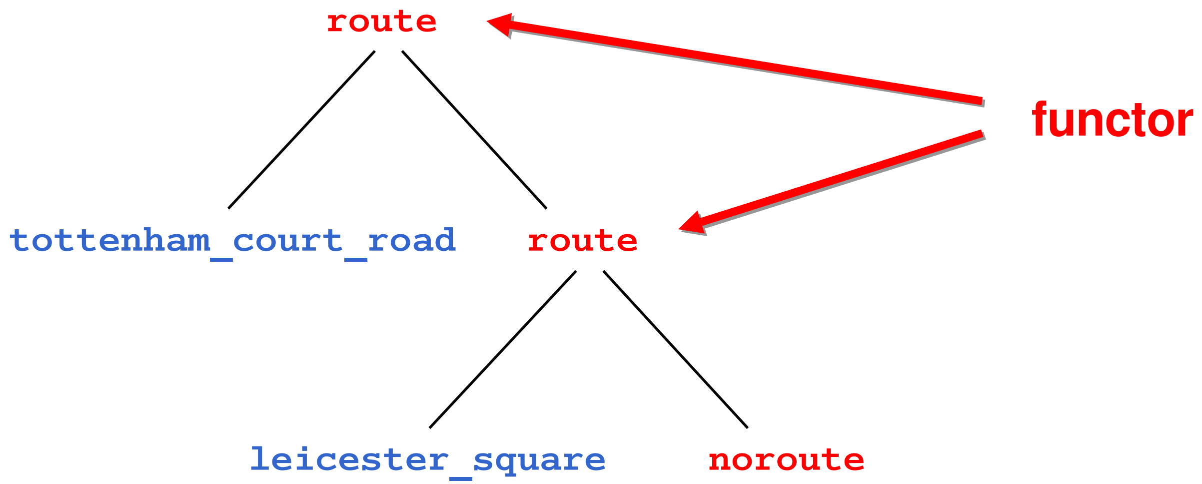
Are used to construct complex objects out of simpler ones.

e.g., `route(oxford_circus, tottenham_court_road)`

Structured terms (1)

```
reachable(X,Y,noroute):-connected(X,Y,L).
reachable(X,Y,route(Z,R)):-connected(X,Z,L),
                            reachable(Z,Y,R).

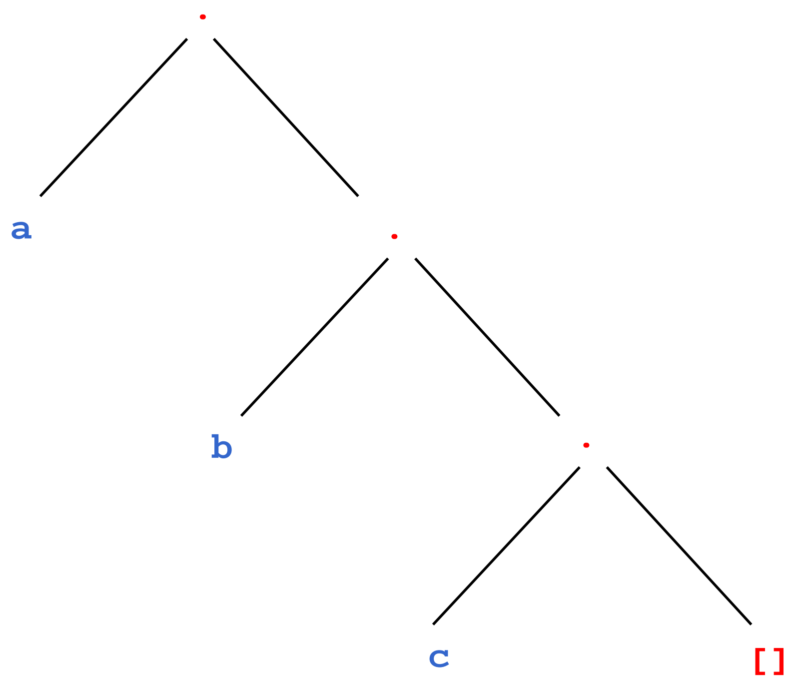
?-reachable(oxford_circus,charing_cross,R).
R = route(tottenham_court_road,route(leicester_square,noroute));
R = route(piccadilly_circus,noroute);
R = route(picadilly_circus,route(leicester_square,noroute))
```



Structured terms (2)

- Built-in data type in Prolog
- Functor: „.“
- Tree notation with functor as well as linear notation possible
- Empty list: `[]`
- `.(a, .(b, .(c, [])))`
- Linear: `[a, b, c]`
- `.(First, Rest)`
- `[First|Rest]`
- `[First,Second,Third|Rest]`

Lists (1)



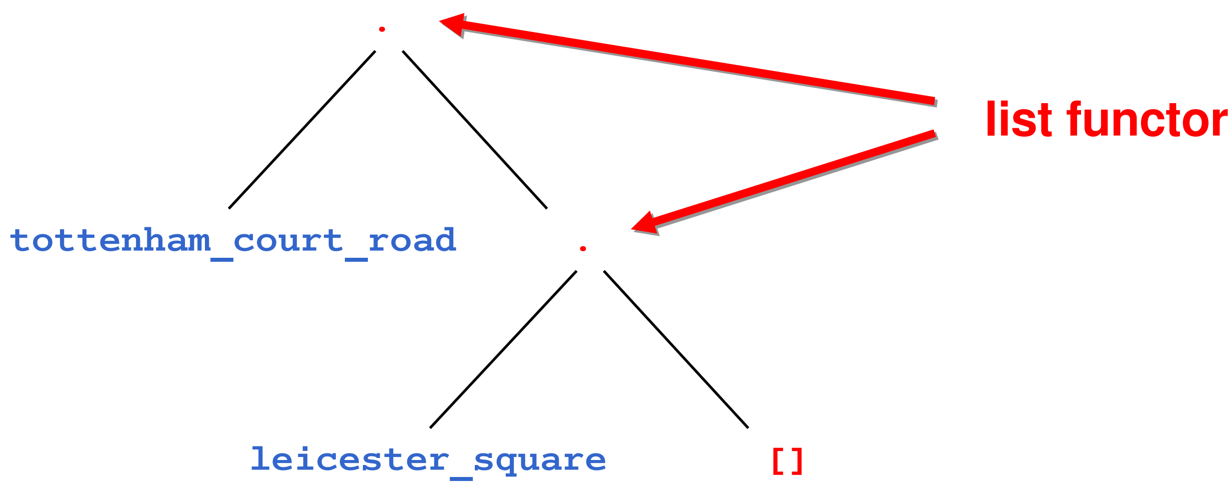
This list can be written in many ways:

- .(a,.(b,.(c,[])))
- [a|[b|[c|[]]]]
- [a|[b|[c]]]
- [a|[b,c]]
- [a,b,c]
- [a,b|[c]]
- …

Lists (2)

```
reachable(X,Y,[]):-connected(X,Y,L).
reachable(X,Y,[Z|R]):-connected(X,Z,L),
                      reachable(Z,Y,R).
```

```
?-reachable(oxford_circus,charing_cross,R).
R = [tottenham_court_road,leicester_square];
R = [piccadilly_circus];
R = [picadilly_circus,leicester_square]
```



Lists (3)

Definition of predicate `append`:

`append(X, Y, Z)` is true, if appending lists `X` and `Y` gives list `Z`:

```
append([], X, X).
append([X|R], Y, [X|Z]) :- append(R, Y, Z).
```

```
?- append([1,2], [3,4], [1,2,3,4]).
yes
```

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

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

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


- Prolog has very simple syntax
  - constants, variables, and structured terms refer to objects
    - variables start with uppercase character
    - functors are never evaluated, but are used for naming
  - predicates express relations between objects
  - clauses express true statements
    - each clause independent of other clauses
- Queries are answered by matching with head of clause
  - there may be more than one matching clause
    - query answering is search process
  - query may have 0, 1, or several answers
  - no pre-determined input/output pattern (usually)

Summary