

Structures:

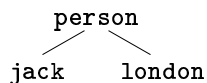
Atoms:

- is_town
- london
- tom

Variables:

- _, _123
- X, Y, Z, Variable, Solution

Records: person(jack, london)

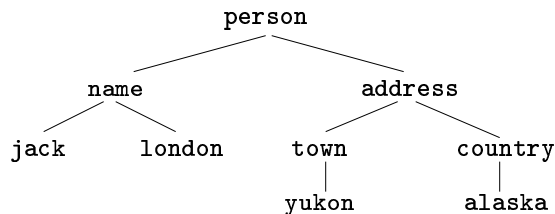


Note: formal resemblance to facts. In fact, facts are records in a certain context, namely in the readin (or consult) cycle of Prolog.

1

Record

```
X = person(name(jack, london),
            address(town(yukon),
                    country(alaska))).
```



Note: person is called the *principal functor* of the structure

2

Unification and Matching

Issue: match between structures

Example:

- $X=Y$ (X as in former slide) unify trivially
- $\text{person}(Y,Z)=X$ results in
 - $Y=\text{name}(\text{jack}, \text{london})$
 - $Z=\text{address}(\text{town}(\text{yukon}), \text{country}(\text{alaska}))$

Example 2: $\text{person}(\text{name}(\text{FN}, \text{SN}), _)=X$

- $\text{FN}=\text{jack}$
- $\text{SN}=\text{london}$

Applications:

- pattern matching
- rule selection

3

Unification and Matching II

Matching: is performed in Prolog according to the following algorithm. Two structures S and T *match* in one of the following three cases:

1. if S and T are constants, then exactly if they are the same object
2. if S is a variable and T is anything, they match, with S being instantiated to T . And vice versa.
3. if S and T are structures, they match if S and T have same principal functor and if all their subcomponents match.

4

Recursive List Analysis

Note: use of `|` allows utilization of the recursive structure of the list definitions by recursive application of prolog programs

Example: membership. Define `member(Elem,List)*`

```
member(Elem, [Elem|_]).    % (1.)
member(Elem, [_|Rest]) :-
    member(Elem, Rest).    % (2.)
```

1. in the recursive representation, we can only identify members beginning with the head (not the tail) of the list. Thus, we check first whether the first element of the list satisfies the membership and ignore the rest if yes.
2. if either the first element of the list is not matched or it is rejected on backtracking, ignore the first element and try membership in the rest.

Note: `member` is a `member(-E, -L)` predicate, i.e. it can be used to generate solutions.

*if you use that in SWI Prolog, you need to rename `member` as a standard predicate with that name already exists.

9

Question: what happens if the list is empty?

Analysis: remember, `[_|_]` never matches `[]`, i.e. it fails, as it should.

Task: how many members does a list have? Write `size(List, Length)`. Start with simple case, and then followup the recursion.

```
size([], 0).
size([H|T], Size) :-
    % reduce problem:
    % Size is 1 more than size of T
    % get size of T first
    size(T, Size1),
    Size = 1 + Size1.
```

Result: 1+1+1+0

Why: `+` is an uninterpreted predicate. Force evaluation of arithmetic expression using `is`: `Size is 1 + Size1` works as expected.

10

Generation of Solutions

Question: what happens with `size(List, 3)`?
And why?

```
size(List, 3)
=> size(List, 0) ?           Fail
=> size(T, Size1)?
=> size([], 0)?              Ok
T = [], Size1 = 0           Ok
=> 3 is 1 + 0                Fail
Redo => size(T, Size1)?
=> size(T', Size1')
=> size([], 0)               Ok
T1 = [], Size1' = 0
Size1 is 1 + Size1'
      \___/
      0
T = [_|[]] (= []), Size1 = 1
=> 3 is 1 + 1                Fail
      \___/
      Size1
```

etc.

11

Another Example: concat

```
concat([], [], []).          % minimal case
concat([], [H|T], [H|T]).    % have to treat this case in addition
                              % to case 3, because [] (in the first
                              % argument) cannot be
                              % matched with a [H|T] structure
concat([H1|T1], [H2|T2], L3) :-
    concat(T1, [H2|T2], L12),
    L3 = [H1|L12].
```

12