Programming in Prolog
List Aggregation

Claudia Schulz

Logic and AI Programming
(Course 518)

What you will learn in this lecture

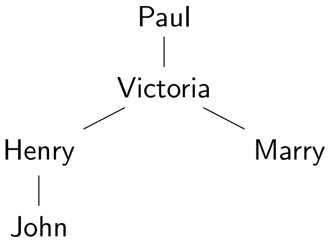
How to collect solutions in a list.

The Ancestor Example

Ancestor Example

```
is_ancestor_of(Parent, Person) :-
    is_parent_of(Parent, Person).

is_ancestor_of(Ancestor, Person) :-
    is_parent_of(Parent, Person),
    is_ancestor_of(Ancestor, Parent).
```



Paul

Victoria

Henry

Marry

John

- find all ancestors of Henry
  ⇒ all Ancestor s.t.
  is_ancestor_of(Ancestor,henry)
- find all people of which Victoria is an ancestor
  ⇒ all Person s.t.
  is_ancestor_of(victoria,Person)

Aggregate 1 – `findall/3`

`findall(+Object,+Goal,-List)`

find all instances of `Object` which satisfy `Goal` and store them in `List`
⇒ as if you query `Goal` and collect `Object` from all possible solutions using `;`

- `Object` and `Goal` usually **share at least one variable** – but it's not necessary
  1. `findall(Z, is_ancestor_of(A,henry), List)`
  2. `findall(hello, is_ancestor_of(A,henry), List)`

Aggregate 1 – findall/3

findall(+Object,+Goal,-List)

find all instances of Object which satisfy Goal and store them in List

- Object **does not have to be a plain variable**
  ⇒ to get the solutions in a certain format:
  1. `findall(is_ancestor(P), is_ancestor_of(A,henry),`
     `          List)`
  2. `findall(P-A, is_ancestor_of(A,P), List)`

Aggregate 1 – findall/3

findall(+Object,+Goal,-List)

find all instances of Object which satisfy Goal and store them in List

- all free variables in Goal (which are not in Object) are "**existentially quantified**"
  1. findall(P, is_ancestor_of(A,P), List)

  - find all instances of P and all existing A s.t. is_ancestor_of(A,P) and put P in List
  - find all ways of proving is_ancestor_of(A,P) and for every solution store P in List
  - if P appears various times with different (or even the same A), it is stored various times

Aggregate 1 – findall/3

findall(+Object,+Goal,-List)

find all instances of Object which satisfy Goal and store them in List

- all free variables in Goal (which are not in Object) are “**existentially quantified**”

  2. findall(A, is_ancestor_of(A,P), List)

  - find all instances of A and all existing P s.t. is_ancestor_of(A,P) and put A in List
  - find all ways of proving is_ancestor_of(A,P) and for every solution store A in List
  - if A appears various times with different (or even the same P), it is stored various times

Aggregate 1 – `findall/3`

`findall(+Object,+Goal,-List)`

find all instances of `Object` which satisfy `Goal` and store them in `List`

- if `Goal` cannot be proven, then `List = ∅`
  1. `findall(A, is_ancestor_of(A,paul), List)`

Aggregate 2 – bagof/3

`bagof(+Object,+Goal,-List)`

find all instances of `Object` which satisfy `Goal` (for some particular free variable) and store them in `List`
⇒ as if you query `Goal` with one particular (succeeding) instantiation of free variables and collect `Object` from all possible solutions using ;

- Same behaviour as `findall/3` except free variables in `Goal`:
  1. `bagof(hello, is_ancestor_of(A,henry), List)`
  2. `bagof(is_ancestor(A), is_ancestor_of(A,henry), List)`
  3. `bagof(P-A, is_ancestor_of(A,P), List)`
  4. `bagof(P, is_ancestor_of(A,P), List)`
  5. `bagof(A, is_ancestor_of(A,P), List)`

Aggregate 2 – bagof/3

`bagof(+Object,+Goal,-List)`

find all instances of `Object` which satisfy `Goal` (for some particular free variable) and store them in `List`

- to get exactly the same behaviour as `findall/3`, free variables need to be “existentially quantified”
  ⇒ using ^ before the `Goal`
  ⇒ `X^Goal` means “there exists some `X` such that `Goal` holds”
  1. `bagof(P, A^is_ancestor_of(A,P), List)`
  2. `bagof(A, P^is_ancestor_of(A,P), List)`
- BUT: `bagof/3` fails if `Goal` cannot be proven:
  1. `bagof(A, is_ancestor_of(A,paul), List)`

Aggregate 3 – setof/3

setof(+Object,+Goal,-List)

find all Objects which satisfy Goal (for some particular free variable) and store them ordered in List

- same as bagof/3 but List is ordered and contains no duplicates
  1. setof(is_ancestor(P), is_ancestor_of(A,henry), List)
  2. setof(P-A, is_ancestor_of(A,P), List)
  3. setof(P, is_ancestor_of(A,P), List)
  4. setof(P, A^is_ancestor_of(A,P), List)
  5. setof(A, is_ancestor_of(A,paul), List)

more on aggregates

- list of all people and for each of them all their ancestors:
  - `findall(P-AList, bagof(A, is_ancestor_of(A,P), AList), List)`
- `setof/3` more powerful than `bagof/3` more powerful than `findall/3`
- `setof/3` less efficient than `bagof/3` less efficient than `findall/3`

A word of caution

Don't use aggregates if not necessary!

No-Go:

```
findall(X, member(X, L), List)
```

What you should know now

How to collect solutions in a list

- findall/3
- bagof/3
- setof/3
- the difference between these three aggregates