Universidade do Minho
Escola de Engenharia
Departamento de Informática

Knowledge Representation

Prolog

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA
Sistemas de Representação de Conhecimento e Raciocínio

Fevereiro 2020

ISLab
Synthetic Intelligence Lab

Lists

- Introduce lists, an important recursive data structure often used in Prolog programming;
- **member/2** predicate, a fundamental Prolog tool for manipulating lists;
- Recursing lists.

ISLab
Synthetic Intelligence Lab

Lists

- A list is a finite sequence of elements;
- Elements are enclosed in square brackets;
- Number of elements → length;
- List can have all sort of prolog elements;
- Empty list: [ ] .

ISLab
Synthetic Intelligence Lab

Lists

- Example:
- [ana, paulo miguel, sara]
  - [ana, peluche(coelhinho), X, 2, ana, [ ]]
  - [ana, [miguel, juliana], [rosa, amigo(rosa)]]
  - [[ ], feliz(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

ISLab
Synthetic Intelligence Lab

Head and Tail

- A non-empty list consists of 2 parts:
  - The head;
  - The tail.
- Head → first item in the list;
- Tail → everything else.
  - tail is the list that remains when we remove the first element;
  - tail of a list is always a list!

ISLab
Synthetic Intelligence Lab

Head and Tail

- [ana, paulo, marco, miguel]

  - Head→ **ana**
  - Tail→ **[paulo, marco, miguel]**

ISLab
Synthetic Intelligence Lab

Head and Tail

- [[ ] , feliz(z), [2, [b,c]], [ ], Z, [b,c]]
  - Head→ [ ]
  - Tail→[feliz(z), [2, [b,c]], [ ], Z, [b,c]]

ISLab
Synthetic Intelligence Lab

Head and Tail

- [feliz(z)]

Head: **feliz(Z)**

Tail: **[ ]**

ISLab
Synthetic Intelligence Lab

Head and tail

- The empty list has neither a head nor a tail;
- For Prolog, [ ] is a special simple list without any internal structure;
- The empty list plays an important role in recursive predicates for list processing in Prolog.

ISLab
Synthetic Intelligence Lab

The built-in operator |

- Prolog has a special built-in operator | which can be used to decompose a list into its head and tail;
- The | operator is a key tool for writing Prolog list manipulation predicates.

ISLab
Synthetic Intelligence Lab

The built-in operator |

```
?- [Head|Tail] = [ana, julia, miguel, patricia].

Head = ana
Tail = [julia,miguel,patricia]
yes

?-
```

ISLab
Synthetic Intelligence Lab

The built-in operator |

```
?- [X|Y] = [ana, julia, miguel, patricia].

X = ana
Y = [julia,miguel,patricia]
yes

?-
```

ISLab
Synthetic Intelligence Lab

The built-in operator |

```
?- [X,Y|Tail] = [[ ], feliz(z), [2, [b,c]], [], Z, [2, [b,c]]] .

X = [ ]
Y = feliz(z)
Tail = [[2, [b,c]], [ ], Z, [2, [b,c]]]

?-
```

ISLab
Synthetic Intelligence Lab

Anonymous variable

```
?- [X1,X2,X3,X4|Tail] = [mara, ana, julia, joana, marco].
X1 = mara
X2 = ana
X3 = julia
X4 = joana
Tail = [marco]
yes
?-
```

ISLab
Synthetic Intelligence Lab

Anonymous variable

```
?- [_,X2,_,X4|_] = [mara, ana, julia, joana, marco].
X2 = ana
X4 = joana
yes
?-
```

- Only the 2nd and 4th element of the list;
- _ indicates anonymous variable.

ISLab
Synthetic Intelligence Lab

Anonymous variable

- When a variable is needed, but we are not interested in what Prolog instantiates it to;
- Each occurrence of the anonymous variable is independent, i.e. can be bound to something different.

ISLab
Synthetic Intelligence Lab

Member

- Something is an element of a list or not?
- Given a term X and a list L, tells us whether or not X belongs to L

- **member/2**

member/2

```
member(X,[X|T]).
member(X,[H|T]):-member(X,T).
```

```
?- member(ana,[joana,tania,ana,julia]).

 yes

?-
```

member/2

```
member(X,[X|T]).
member(X,[H|T]):-member(X,T).
```

```
?- member(marco,[joana,tania,ana,julia]).

 no

?-
```

member/2

```
member(X,[X|T]).
member(X,[H|T]):-member(X,T).

?- member(X,[ana,marco,paulo,julia]).
X = ana;
yes
```

ISLab
Synthetic Intelligence Lab

Rewriting member/2

```
member(X,[X|_]).
member(X,[_|T]):-member(X,T).
```

ISLab
Synthetic Intelligence Lab

Recursing down lists

- member/2 predicate works by recursively working its way down a list;
  - doing something to the head, and then
  - recursively doing the same thing to the tail.
- This technique is very common in Prolog.

ISLab
Synthetic Intelligence Lab

Example: a2b/2

- The predicate a2b/2 takes two lists as arguments and succeeds:
  - if the first argument is a list of a's, and
  - the second argument is a list of b's of exactly the same length.

ISLab
Synthetic Intelligence Lab

Example: a2b/2

```
?- a2b([a,a,a,a],[b,b,b,b]).
 yes
?- a2b([a,a,a,a],[b,b,b]).
no
?- a2b([a,c,a,a],[b,b,b,t]).
no
```

ISLab
Synthetic Intelligence Lab

Defining a2b/2

```
a2b([],[]).
a2b([a|L1],[b,L2]):-a2b(L1,L2).
```

a2b/2

```
a2b([],[]).

a2b([a|L1],[b,L2]):-a2b(L1,L2).

?- a2b([a,a,a],[b,b,b]).

yes

?-
```

a2b/2

```
a2b([],[]).

a2b([a|L1],[b,L2]):-a2b(L1,L2).

?- a2b([a,a,a],[b,c,b]).

no

?-
```

ISLab
Synthetic Intelligence Lab

a2b/2

```
a2b([],[]).
a2b([a|L1],[b,L2]):-a2b(L1,L2).
```

```
?- a2b([a,a,a,a,a], X).
X = [b,b,b,b,b]
yes
?-
```

append

- **append/3 (**whose arguments are all lists)
- Declaratively:
  - append(L1,L2,L3) is true if list L3 is the result of concatenating the lists L1 and L2 together

ISLab
Synthetic Intelligence Lab

Append

- **append/3 (**whose arguments are all lists)
- Declaratively:
  - append(L1,L2,L3) is true if list L3 is the result of concatenating the lists L1 and L2 together

ISLab
Synthetic Intelligence Lab

append/3

append([ ], L, L).

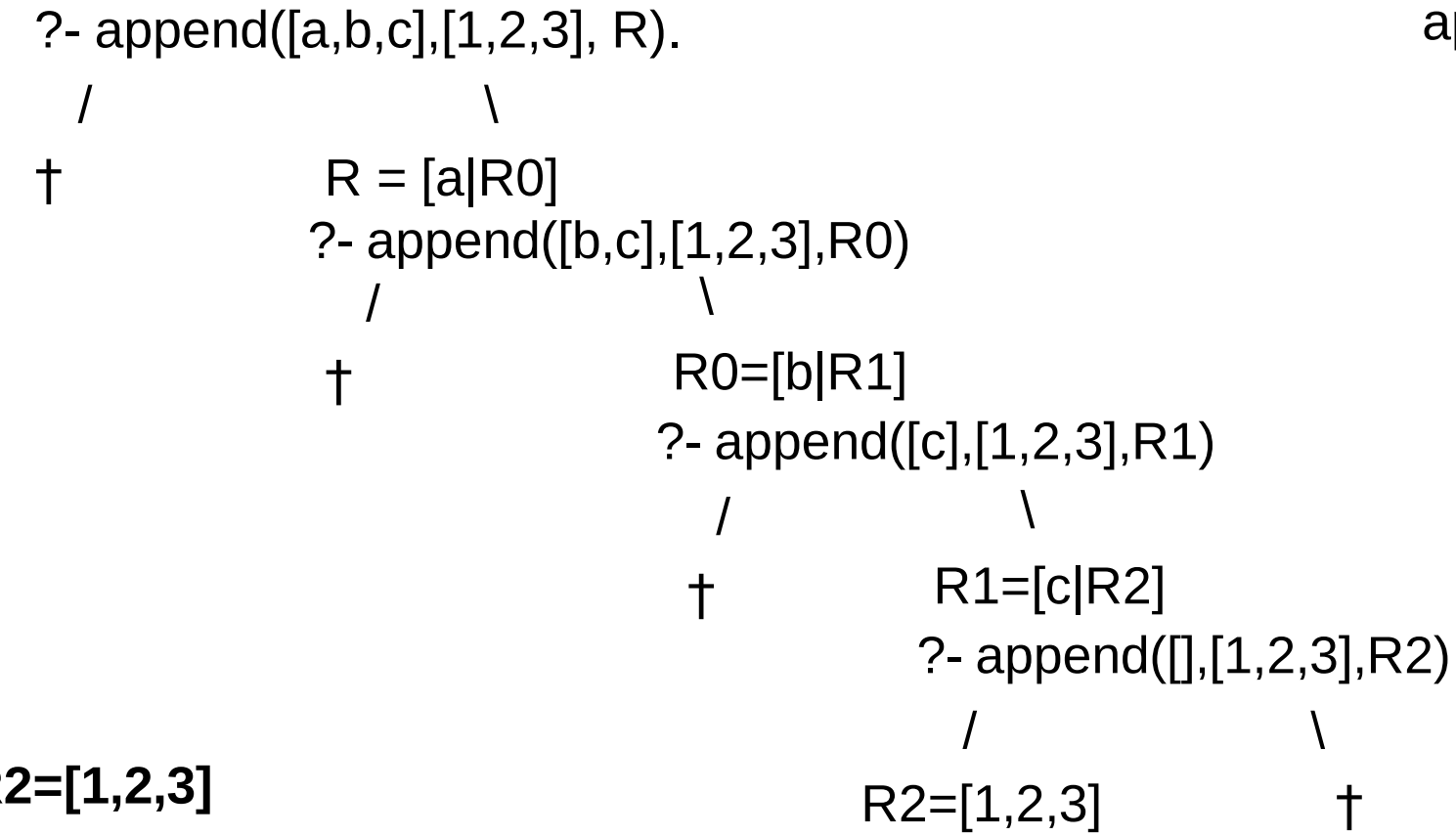
append([H|L1], L2, [H|L3]):- append(L1, L2, L3).

- Recursive definition:
  - Base clause: appending the empty list to any list produces that same list;
  - When concatenating a non-empty list [H|T] with a list L, the result is a list with head H and the result of concatenating T and L

ISLab
Synthetic Intelligence Lab

Search tree example

```
append([], L, L).
append([H|L1], L2, [H|L3]):-
        append(L1, L2, L3).
```



R2=[1,2,3]

R1=[c|R2]=[c,1,2,3]

R0=[b|R1]=[b,c,1,2,3]

R=[a|R0]=[a,b,c,1,2,3]

ISLab
Synthetic Intelligence Lab

append/3

- Splitting up a list:

```
?- append(X,Y, [a,b,c,d]).

X=[ ]      Y=[a,b,c,d];
X=[a]      Y=[b,c,d];
X=[a,b]    Y=[c,d];
X=[a,b,c]  Y=[d];
X=[a,b,c,d]   Y=[ ];
no
```

prefix/2

```
prefix(P,L):- append(P,_,L).
```

- A list P is a prefix of some list L:
  - there is some list such that L is the result of concatenating P with that list.
- Note the use of the anonymous variable.

ISLab
Synthetic Intelligence Lab

prefix/2

```
prefix(P,L):- append(P,_,L).
```

```
?- prefix(X, [a,b,c,d]).
X=[ ];
X=[a];
X=[a,b];
X=[a,b,c];
X=[a,b,c,d];
no
```

suffix/2

```
suffix(S,L):-append(_,S,L).
```

- A list S is a suffix of some list L:
  - there is some list such that L is the result of concatenating that list with S.
- Again, the anonymous variable.

suffix/2

```
suffix(S,L):-append(_,S,L).

?- suffix(X, [a,b,c,d]).
X=[a,b,c,d];
X=[b,c,d];
X=[c,d];
X=[d];
X=[];
no
```

ISLab
Synthetic Intelligence Lab

sublist/2

```
sublist(Sub,List):-
    suffix(Suffix,List),
    prefix(Sub,Suffix).
```

- The sub-lists of a list L are simply the prefixes of suffixes of L

ISLab
Synthetic Intelligence Lab

append/3

- **append/3** can be source of inefficiency:
  - Concatenating a list is not done in one simple action;
  - But by traversing down one of the lists.

ISLab
Synthetic Intelligence Lab

reverse

```
reverse([],[]).
reverse([H|T],R):- reverse(T,RT),  append(RT,[H],R).
```

- This definition is correct, but it does an awful lot of work
- It spends a lot of time carrying out appends
- But there is a better way…

ISLab
Synthetic Intelligence Lab

Reverse using an accumulator

- The better way is using an accumulator;
- The accumulator will be a list, and when start reversing it will be empty;
- Take the head of the list to reverse and add it to the head of the accumulator list;
- Continue this until reaching the empty list;
- At this point the accumulator will contain the reversed list!

ISLab
Synthetic Intelligence Lab

Reverse using an accumulator

```
accReverse([ ],L,L).
accReverse([H|T],Acc,Rev):-
    accReverse(T,[H|Acc],Rev).
```

ISLab
Synthetic Intelligence Lab

reverse

```
accReverse([ ],L,L).
accReverse([H|T],Acc,Rev):-
    accReverse(T,[H|Acc],Rev).

reverse(L1,L2):- accReverse(L1,[ ],L2).
```

ISLab
Synthetic Intelligence Lab

reverse

- List: [a,b,c,d] Accumulator: []
- List: [b,c,d] Accumulator: [a]
- List: [c,d] Accumulator: [b,a]
- List: [d] Accumulator: [c,b,a]
- List: [] Accumulator: [d,c,b,a]

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Knowledge Representation

Prolog

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA
Sistemas de Representação de Conhecimento e Raciocínio

Fevereiro 2020