Contents

Lists

- A list is a finite sequence of elements
- Examples of lists in Prolog:

```
[mia, vincent, jules, yolanda]
[mia, robber(honeybunny), X, 2, mia]
[]
[mia, [vincent, jules], [butch, friend(butch)]]
```

Important things about lists

- List elements are enclosed in square brackets
- The length of a list is the number of elements it has
- All sorts of Prolog terms can be elements of a list
- There is a special list: the empty list `[]`

Head and Tail

- A non-empty list can be thought of as consisting of two parts
  - The head
  - The tail
- The head is the first item in the list
- The tail is everything else
  - The tail is the list that remains when we take the first element away

    – The tail of a list is always a list

```
[mia, vincent, jules, yolanda]
```

- Head: mia
- Tail: [vincent, jules, yolanda]

Empty List

- 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

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

```
?- [X|Y] = [mia, vincent, jules, yolanda].

X = mia
Y = [vincent, jules, yolanda]
yes

?- [X|Y] = [].
no

?-
```

Anonymous variable

- Suppose we are interested in the second and fourth element of a list

```
[I, J, K, L|X] = [mia, vincent, marsellus, jody, yolanda].
I=mia
J=vincent
K=marsellus
L=jody
X=[yolanda]
```

```
yes

?-
```

- There is a simpler way of obtaining only the information we want

```
[_, J, _, L|_] = [mia, vincent, marsellus, jody, yolanda].
J=vincent
L=jody
yes

?-
```

- The underscore is an anonymous variable
- Is used when you need to use a variable, but you are not inerested in what Prolog instantiates it to
- Each occurence of the anonymous variable is independent, i.e. can be bound to something different

Member

- One of the most basic things we would like to know is whether something is an element of a list of not
- So let's write a predicate that when given a terms `X` and a list `L`, tells us whether or not `X` belongs to `L`
- This predicate is usually called `member/2`

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

?- member(vincent, [yolanda, trudy, vincent, jules]).
yes
?- member(zed, [yolanda, trudy, vincent, jules]).
no
?-
```

- The `member/2` predicate works by recursively working its way down a list
  - Doing something to the head then tail