Functional and
Logic programming

Tutorial 2: Unification, Lists, Proof trees for recursive predicates.

Unification algorithm

Where do we meet the algorithm?

1) When matching a query against a rule:

```
    connected(picadilly_circus, bank_street).
?- connected(X,Y).
```

2) Using the *equals* predicate.

```
    picadilly_circus = X.
```

Input: Two terms *A* and *B*.

Output: A minimal set of substitution to make the terms *A* and *B* equal.

Unification algorithm: Examples 1/2

Input: *X* = ***picadilly***. Output: {X = picadilly}.

Input: ***plus(X,Y)*** = ***plus(Z,4)***
Output: {X \ 3, Y \ Z}.

Input: ***cons(first,cons(second,nil))*** = ***cons(A,cons(B,nil))***
Output: {A \ first, B \ second}

Unification algorithm: Examples 2/2

Input: *cons(first,cons(second,nil)) = cons(A,B)*

Output: {A \ first, B \ cons(second,nil)}

Input: *cons(first,nil) = cons(A,cons(B,nil))*
Output: false.

Input: *cons(X,cons(Y,nil)) = cons(Y,cons(element,nil)*
Output: {X \ element, Y \ element}

Unification algorithm

1) A variable and a constant do unify.

```
?- X = piccadily_circus.
true.
```

2) A variable and a variable do unify.

```
?- X = Y.
true.
```

3) Two functions unify if

a) predicate symbols and arities (="signature") are equal

b) their arguments unify

Unification for lists

- Prolog vs. LISP:
  - `'.'` instead of `cons()`
  - `[]` instead of `nil`.
- Syntactic sugar:
  - Unification `[X|Y] = [a,b,c]` makes `X` correspond to `car` and `Y` correspond to `cdr`.
  - Without shorening, we can write `'.'(X,Y) = '.'(a,'.'(b,'.'(c,[])))`
  - When hesitating, just recall: `[X|Y] == '.'(X,Y)` and the unification algorithm.

Underground journey 1/2

- Let's use the code from the last tutorial to declare predicate **`journey/2`** which finds a route between two stations and, if successful, writes them down.
- Example:
- **`?- journey(bond_street, leicester_square).`**
  `leicester_square tottenham_court_road oxford_circus`

Underground journey 2/2

- Now define predicate **`journey/3`** which returns a list of visited stations on a journey
- **`?- journey(bond_street, leicester_square, LS).`**
  `LS = [oxford_circus, tottenham_court_road, leicester_square]`

Is 'a' member of a list?

- Define your own implementation of the **`member/2`** predicate.

- Example:
- **`?- my_member(a, [b, a, c]). yes.`**
- **`?- my_member(a, []). no.`**
- **`?- my_member([1,a], [b, a, c, [1,a]]). yes.`**

Homework #1

Implement **`select`** predicate. Its functionality overlaps with **`member`**, but additionally, it returns the “rest of the list”.

```
?- my_select(Element, [a,b,c], Remaining).
Element = a,
Remaining = [b, c] ;
Element = b,
Remaining = [a, c] ;
Element = c,
Remaining = [a, b] ;
false.
```

Reverse a list

- Define predicate **`addtoend/2`** which adds element to the end of given list.
- Example:
  `?- addtoend(0, [1,2,3], L). L = [1,2,3,0].`
- Define predicate **`my_reverse/2`** that reverse elements of given list (you can use the addtoend/2 predicate)
- Example:
  `?- my_reverse([1,2,3], L). L = [3,2,1].`

Append two lists

- Define your own implementation of the **`append/3`** predicate
- Example:
```
?- my_append([a,b,c], [d,e], L).
L = [a,b,c,d,e].
```
- What would **`my_append(L1, L2, [a, b, c])`** do?

Solution of `append/3`

```
append(A,[],A).

append(A,[H|T],[H|L]):-
append(A,     T,      L).
```

Homework #2

1. Study the `=..` metapredicate:
```
?- connected(bond_street,
        oxford_circus, central) =.. X
X = [connected, bond_street,
        oxford_circus, central].
```
2. What does `var(·)` do: `?-var(a).` `?-var(X).`
3. Easy: extract variables from a flat term:
```
?- varsflat(term(a,b,X,c,Y,z),V). … V=[X,Y].
```
4. Hard: extract variables from a structured term:
```
?- allvars(f(a,X,g(c,h(Y)),z), V). … V=[X,Y].
```

Food for thought

- Study the astonishing beauty of **`sublist/3`** predicate:
  http://www.cs.bris.ac.uk/~flach/SL/labs/labs3.pl
- Convince yourself about the syntactic sugar
  `[X|Y] = '.'(X,Y)` using the `=..` metapredicate.
- What are the differences between
  a list **`[a,b,c]`** and
  a tuple **`(a,b,c)`** ?
  Again, using `=..` is highly recommended!