Lists and recursion

Lists

- Many algorithms (like sorting) need some memory space and data structure to work on
- Conventional *iterative* programming languages use *arrays*
- Functional/logic programming languages use *linked lists*

A list can store any number of data objects (if there is enough memory space of course !)

Examples of prolog lists: $[1, 2, 3, [-1, \mathsf{a}, [\,]\,], \text{'movie\_bd'}]$ $[\,]$ $[\_, X, Y, 1]$

- A list is enclosed in squared brackets
- The elements are separated by commas "**,**"
- Elements of all types can be put in a list
- A list can contain other lists
- Character strings are lists: Prolog interprets "abcd" as the list of ASCII codes $[97, 98, 99, 100]$.

Data abstraction

An example Consider a database for the timetable of a faculty:

- it could containt facts of the form $\mathsf{slot}(\mathsf{Start}, \mathsf{End}, \mathsf{Course}, \mathsf{Group}, \mathsf{Teacher})$
  where Start and End give the start and end times of the slot
- most predicats and queries do not need to know how time points are represented

$\Rightarrow$ these could be represented by $[H, M]$, or $[H, M, S]$ or the number N of sec. elapsed since Jan. 1st, 1970.

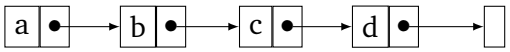
Remark [and] are special constructors, reserved for lists in Prolog
It is possible to use other constructors to use other structures, but, in principle, lists are sufficient.

Pattern Matching and Filtering

An example Consider a program that deals with colors, and that is able to use to systems: RGB and CMYK:

- RGB: colors can be represented by lists with three numbers
- CMYK: colors can be represented by lists with four numbers
- clauses for a predicate darken, that computes a color darker than a given one could have clause like:
  $\mathsf{darken}([R, G, B], \mathsf{New}) \leftarrow \ldots$ calculation for a color defined byR, G, B
  $\mathsf{darken}([C, M, Y, K], \mathsf{New}) \leftarrow \ldots$ calculation for a color defined by C, M, Y, K
- for the goal $\mathsf{darken}([123, 255, 27], N)$ prolog will not try the second rule, which expects a list with 4 elements as first parameter

Recursive structure

A list: 

Prolog representation: [a | [b | [c | d | [ ]]]]

- [ ] represents the empty list;
- $[X|L]$ represents a list, the first element of which is X, whereas the *tail* is a list L;
- the "," is a convenient shortcut to enumerate elements at the beginning of a list.

For instance, $[1, 2, 3, 4] = [1, 2, 3|\,[4]\,] = [1|\,[2|\,[3|\,[4|\,[\,]\,]\,]\,]\,]$. (But $[1, 2, 3, 4] \neq [\,[1, 2]\,|\,[3, 4]\,]$. Why ?)

A list is a *compound term*, we could use an atom "list" to build lists:
• $[X \mid L]$ would be written $\mathsf{list}(X, L)$ • $[a, b, c, d]$ would be written $\mathsf{list}(a, \mathsf{list}(b, \mathsf{list}(c, \mathsf{list}(d, \mathsf{empty}))))$
(where "empty" would be another atom to denote the empty list).

Examples of filtering with the recursive form of lists:

- $\mathsf{isFirstElmtOf}(X, L)$: true if X is the first element of the list $L \Rightarrow$
  $\mathsf{isFirstElmtOf}(X, L) \leftarrow L = [X|R]$.or simply $\mathsf{isFirstElmtOf}(X, [X|R])$.
- $\mathsf{isSecondElmtOf}(X, L)$ ...

Recursive programming

The classical example

We wish to retrieve the last element of a list: $\mathsf{isLastElmtOf}(X, L)$ must be true if X is the last element of L

- the linked list must be scanned until its last element is reached
- we do not know in advance how many steps will be needed $\Rightarrow$ *recursive programming:*

X is last element of [Y | R] if and only if X is last element of R
Termination: X is the last element of $[X]$

In prolog: $\mathsf{isLastElmtOf}(X, L) \leftarrow L = [X] \vee (L = [Y|R] \wedge \mathsf{isLastElmtOf}(X, R))$.

Example Write a definition for a predicate member/2, such that $\mathsf{member}(X, L)$ is true if X is element of list L.