Kontrollfragen

Warum kann eine endrekursive Prozedur von Prolog effizienter ausgeführt werden, als eine nicht-endrekursive Prozedur?

- Man bewältigt alle Berechnungen in einer Iteration (Schleife) und hat nur einmal am Ende der Prozedur einen rekursiven Aufruf
- Man muss gewisse Berechnungen nicht mehrmals machen

Beschreibe in eigenen Worten, auf was für einer Beobachtung "Optimierung mit Assertions" beruht und wie diese konkret in Prolog umgesetzt wird.

- Bereits berechnete Berechnungen werden mit Rekursion mehrmals berechnet
- Deswegen sollte man die Resultate solcher Berechnungen in der Wissensdatenbank ablegen für das nächste Mal, wenn dieselbe Berechnung gemacht werden muss.
- Optimierung mit Assertions heisst somit:
    - Bereits berechnete Berechnungen werden gespeichert und können wieder aufgeruft werden. Mit Anlegen von neuen Fakten (Prädikat muss dynamic sein)

Was antwortet Prolog auf die Anfrage `X = [a | [b]]` ?

- `X = [a, b].`
- Wir sagen Prolog, dass `a` der Kopf und `[b]` der Schwanz (Restliste) der Liste ist

Und auf die Anfrage `[1, 2, 3] = [_ | X]` ?

- `X = [2, 3].`
- Der Schwanz wird `X` zugewiesen

Und auf die Anfrage `[a, b, c] = [_, X | Y]` ?

- `X = b, Y = [c].`
- Für `X` wird nur nach dem Element gefragt
- Für `Y` wird nach der Rest der Liste gefragt

Warum ist es in Prolog i.A. effizienter auf das erste, als auf das letzte Element einer Liste zuzugreifen?

- Listen sind rekursiv aufgebaut
- Aus diesem Grund wird rekursiv nach dem ersten und letzten Listenelement gesucht
- Da das erste Element gleich zu Beginn der Rekursion gefunden wird, kann dort abgebrochen werden, beim letzten Element erst wenn es alle Elemente durchsucht hat

Was antwortet Prolog auf die Anfrage `conc(L,[c],[a,b,c])` ?

- `L = [a, b].`

Und auf die Anfrage `conc(Before, [d | After], [a, b, c, d, e, f, g, h])` ?

- `Before = [a, b, c]`
- `After = [e, f, g, h]`

Und was auf `conc([a], L, [b, c])` ?

- `false.`
- Es gibt keine Kombination wo eine Liste mit dem Element zusammen mit einer anderen Liste die Liste `[b, c]` ergeben kann

Bonusaufgabe (schwierig): Wie lässt sich `mem/2` unter Verwendung von `conc/3` ausdrücken? (D.h. als neues Prädikat der Form `mem_c(X, L) :- ...conc...`)

- TODO