Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Informatik

Programming Concepts and Paradigms (PCP)

PCP-Übung zu Prolog 3 + 4

Hauptthemen: Optimierungen (Endrekursion, Memoization), Listen, Cut-Operator
Zeitfenster: ca. 2-3 Lektionen

Ruedi Arnold

Diese Übung repetiert und vertieft den in Prolog 3 und 4 vermittelten Stoff. Gehen Sie zur Vorbereitung nochmals durch die Unterrichtsfolien durch und schauen Sie insbesondere, dass Sie alle auf den Folien angegebenen Code-Beispiele verstanden haben.

1. Endrekursive Fibonacci-Berechnung mit Ein- und Ausgabe

In der Vorlesung wurde eine endrekursive Prolog-Implementierung für Fibonacci-Zahlen besprochen. Implementieren Sie nun ein Prädikat io_fib/0 so, dass der Benutzer zuerst eine Zahl eingeben muss, und dann die dazugehörende Fibonacci-Zahl ausgegeben wird. Hinweis: Verwenden Sie dazu das im Unterricht gezeigte, endrekursive Prädikat fib_tr/2 und die beiden eingebauten Prolog-Prädikate read/1 und write/1.

Hier ein Beispiel-Aufruf von io_fib/0 in der Prolog-Konsole:

```
?- io_fib.
Gib eine Zahl ein: 7.
Die 7. Fibonacci-Zahl ist 13
```

2. Memoization: Optimierung der Fakultätsberechnung durch Assertions

Mit Hilfe von Assertions können neue Fakten in der Wissensdatenbank gespeichert werden. Dies kann verwendet werden, um Berechnungen zu optimieren, indem (Zwischen-)Resultate gespeichert werden.

a) Erweitern Sie das in Vorlesung besprochene Prädikat fak/2 zur Berechnung der Fakultät so, dass bereits berechnete Funktionswerte mit Hilfe vom eingebauten Prädikat asserta/1 gespeichert werden. Verwenden Sie zum speichern der berechneten Werte ein neues Prädikat fak_as/2 und testen Sie also beim einem Aufruf von fak/2 entsprechend zuerst, ob bereits ein passender Wert für fak_as/2 vorhanden ist. Falls ein gespeicherter Wert vorhanden ist, produzieren Sie mit Hilfe von write/1 eine entsprechende Konsolen-Ausgabe wie die folgende: `Hinweis: Fakultät von 7 war gespeichert`.

b) Ergänzen Sie ihr Fakultätsprogramm um ein Prädikat fak_clear/0, welches mit Hilfe vom eingebauten Prädikat retractall/1 alle gespeicherten Werte für fak_as/2 wieder löscht. Geben Sie dazu in die Konsole folgenden Text aus: `Hinweis: Alle gespeicherten Werte wurden gelöscht`.

Hier eine Aufruf-Sequenz aus der Prolog-Konsole, welche die Funktionsweise von den zu programmierenden Prädikaten fak/2 und von fak_clear/0 aufzeigt, spielen Sie diese Sequenz testweise mit ihrem Programm durch:

```
?- fak(7, N).
N = 5040 .
?- fak(7, N).
(Hinweis: Fakultät von 7 war gespeichert)
N = 5040 .
?- fak(8, N).
(Hinweis: Fakultät von 7 war gespeichert)
N = 40320 .
?- fak_clear.
(Hinweis: Alle gespeicherten Werte wurden gelöscht)
true.
?- fak(7, N).
N = 5040 .
```

3. Listen-Operationen: Hinzufügen, Löschen, Umkehren

In der Vorlesung wurden Listen eingeführt und die beiden Operationen "Test auf Mitgliedschaft" (mem/2) und "Listen zusammenfügen" (conc/3) besprochen. Bei dieser Aufgabe sollen Sie nun weitere Listen-Operationen selbst implementieren. Hinweis: Falls nicht explizit anders angegeben, sollen Sie die folgenden Listen-Operationen rein unter Verwendung der [|]-Notation implementieren, insbesondere OHNE Verwendung von eigenen oder vordefinierten Listen-Prädikaten wie conc/3, mem/2 oder member/2.

a) Ein Element X am Anfang einer Liste L hinzuzufügen ist in Prolog trivial: [X | L]. Oder falls dafür explizit ein Prädikat verwendet werden soll: add(X, L, [X | L]). Implementieren Sie nun ein eigenes Prädikat add_tail(X, L, L1), welches ein Element X am Ende einer Liste L einfügt, so dass daraus Liste L1 entsteht. Hier ein Beispiel-Aufruf zur Illustration:

```
?- add_tail(x, [a, b, c], L).
L = [a, b, c, x].
```

b) Implementieren Sie ein Prädikat del(L, X, L1), welches ein Element X aus einer Liste L löscht, so dass daraus die Liste L1 entsteht. Hier ein Anwendungsbeispiel:

```
?- del([a, b, c], c, L).
L = [a, b]
```

Hinweis: Das hier zu implementierende Prädikat del/3 ist in SWI-Prolog eingebaut als delete/3. – Hier sollen Sie del/3 natürlich selber programmieren ohne Verwendung von delete/3.

c) Wie kann mit Hilfe von dem soeben programmierten del/3 auf Listen-Mitgliedschaft getestet werden? Implementieren Sie dazu ein Prädikat mem_d/2, welches ausschliesslich del/3 verwendet. Hinweis: Das geht in einer Zeile.
Zwei Anwendungsbeispiele dazu:

```
?- mem_d(a, [a, b, c]).
true .
```

```
?- mem_d(x, [a, b, c]).
false.
```

d) Implementieren Sie ein Prädikat rev_acc(L, A, R) zum Umdrehen von Listen mit Hilfe von einem Akkumulator: Die Liste R besteht also aus allen Elementen der Liste L in umgekehrter Reihenfolge, und im Akkumulator A bauen wir uns die umgekehrte Liste schrittweise auf. Initial wird rev_acc/3 also mit einer leeren Liste als Akkumulator aufgerufen, Anwendungsbeispiel:

```
?- rev_acc([a, b, c, d], [], L).
L = [d, c, b, a].
```

e) „Verpacken“ Sie das oben programmierte Prädikat rev_acc/3 in ein Prädikat rev(L, R), welches für eine Liste L die umgekehrte Liste R erstellt. Anwendungsbeispiel:

```
?- rev([a, b, c, d], L).
L = [d, c, b, a].
```

Hinweis: Das hier zu implementierende Prädikat rev/2 ist in SWI-Prolog eingebaut als reverse/2. – Hier sollen Sie rev/2 natürlich wie oben ausgeführt selbst programmieren ohne Verwendung von reverse/2.

4. Code mit und ohne Cut-Operator

In der Vorlesung wurde ein Beispiel gezeigt für ein Programm mit und ohne Cut-Operator bei einem Prädikat p/1. Führen Sie den Code aus und vollziehen Sie die beiden Suchbäume mit und ohne Cut vom Foliensatz „Prolog 4“ nach. Verwenden Sie dazu falls gewünscht ebenfalls das Debug-Hilfsprädikat trace/0 (oder guitracer/0) von SWI-Prolog.

5. Umschreiben ohne Cut (Green Cut)

Das folgende Prädikat warn/1 gibt in Abhängigkeit vom Argument einen Hinweis aus.

```
warn(T) :- T < 80, write('Temperatur ok'), !.
warn(T) :- T < 100, write('Temperatur sehr warm'), !.
warn(_) :- write('Temperatur zu heiss').
```

Schreiben Sie dieses Prädikat um, so dass es ohne Cut-Operator auskommt, aber für alle Argumente dieselbe Ausgabe wie das Originalprädikat produziert. Der Code oben ist ein Beispiel für einen Green Cut. Inwiefern ist der Code oben eine Optimierung vom umgeschriebenen Prädikat?

6. Red Cut: Anzahl Lösungen einschränken

In der Vorlesung haben Sie gesehen, wie mit Hilfe des Listen-Zugehörigkeitsprädikats mem/2 Permutationen erstellt werden können (Prolog 3, Folie 37). Wie dort gezeigt, geht das beispielsweise wie folgt für eine Liste der Länge 3 mit den drei Elementen a, b, und c:

```
L = [_, _, _], mem(a, L), mem(b, L), mem(c, L).
```

Ergänzen Sie obiges Code-Beispiel geeignet mit einem Cut-Operator, so dass nur eine Permutation erzeugt wird. Dieser Einsatz von einem Cut-Operator ist ein Beispiel für einen Red Cut. Wieso ist das so?