Übungsblatt 6

Abgabe der Lösungen: Tutorium in der Woche 16.12-20.12

Aufgabe 1 Prolog: Listenarithmetik (Präsenzaufgabe)

1. Programmieren sie ein Prädikat `laenge(List, Len)`, das die Länge `Len` einer Liste `List` bestimmt.
   ```
   ?- laenge([b, a, 1, 3, c, d], Len).
   Len = 6.
   ```

2. Programmieren sie ein Prädikat `summe(List, Sum)` , das die Summe `Sum` einer Liste aus Zahlen `List` bestimmt.
   ```
   ?- sum([42, 1337, 815], Sum).
   Sum = 2194.
   ```

3. Programmieren sie ein Prädikat `ElemAt(Pos, X, List)`, das das Element `X` an der Position `Pos` einer Liste `List` ausgibt. Die Indizierung soll bei 1 beginnen.
   ```
   ?- elemAt(3, X, [a, b, c, d, e]).
   X = c.
   ```

Aufgabe 2 Prolog: Tiefensuche (Präsenzaufgabe)

Folgende Fakten definieren einen gerichteten azyklischen Graphen:

```
kante(a,b,2).
kante(a,c,5).
kante(b,f,9).
kante(f,g,7).
kante(h,j,6).
kante(j,l,2).

start(a).
```

```
kante(a,d,7).
kante(e,f,2).
kante(g,h,18).
kante(i,j,1).
kante(j,m,5).

ziel(m).
```

```
kante(c,e,1).
kante(d,f,5).
kante(h,i,7).
kante(j,k,1).

ziel(k).
```

Ein Faktum `kante(X,Y,K)` definiert eine Kante von X nach Y mit den Kosten K, und `start(X)` markiert die Wurzel des Graphen. `ziel(X)` markiert die gewünschten Zielknoten.

Die Menge der Lösungen im Graphen ist durch folgende Relation charakterisiert:

$$\forall X \forall Y (\mathsf{loesung}(X,Y) \Leftrightarrow \mathsf{start}(X) \wedge \mathsf{erreichbar}(X,Y) \wedge \mathtt{ziel}(Y));$$

`erreichbar`(X,Y) ist wahr genau dann, wenn ein Pfad von X nach Y existiert; `loesung`(X,Y) ist also dann wahr, wenn X ein Startknoten und Y ein Zielknoten ist, so dass ein Pfad von X nach Y existiert.

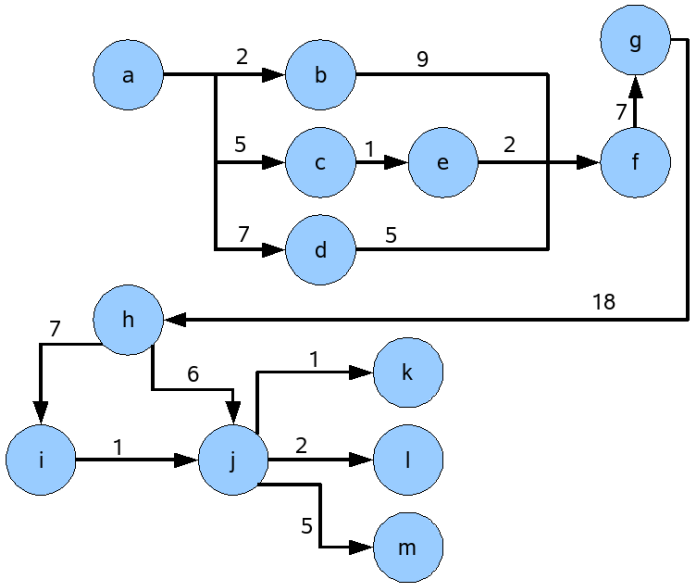

Abbildung 1: Ein gerichteter Graph mit Kosten.

1. Implementieren Sie `erreichbar`(X, Y) in Prolog.
2. Implementieren Sie `loesung`(X, Y) in Prolog.
3. Welches Ergebnis erhalten Sie für die Anfrage `loesung`(a, S)?
4. Erklären Sie das Ergebnis aus der vorherigen Teilaufgabe 3) mit Hilfe der Definition von `erreichbar`(X, Y), dem gegebenen Graphen und der Suchstrategie von Prolog.

Aufgabe 3 Substitution vertieft III (5 Punkte)

Zwei Substitutionen σ, θ *kommutieren*, wenn $\sigma\theta = \theta\sigma$. Geben Sie ein Beispiel von zwei nicht kommutierenden Substitutionen an (mit Beweis!).

Aufgabe 4 Listenunifikation (8 Punkte)

Verwenden Sie die in der Vorlesung eingeführte Kodierung von Listen durch die Operationen $[]/0$ (leere Liste) und $./2$ (Element vorn anhängen), um mittels des Algorithmus aus der Vorlesung für die folgenden Paare von Ausdrücken jeweils einen allgemeinsten Unifikator zu berechnen bzw. zu zeigen, dass sie nicht unifizierbar sind. Der Operator $./2$ ist die Präfixversion von $[-|-]/2$, d.h.

$$.(X, Y) = [X \mid Y]$$

(beide Variante sind vollkommen gültige Prolog-Ausdrucke.)

1. $[X, b \mid Z]$ und $[a, Y]$,
2. $[Y, [a, b] \mid Z]$ und $[X \mid X]$,

3. $[X \mid Y]$ und $[a, Y \mid X]$,
4. $[a, X, Y \mid []]$ und $[X, Y \mid Z]$,
5. $[X, [Y \mid Z] \mid Z]$ und $[[a, b \mid Z] \mid X]$.

Dabei ist es notwendig die gegebene Ausdrucke zunächst wie vorgeschrieben umformen und danach das Unifikationsverfahren anzuwenden.

Vergleichen Sie das Ergebnis mit der in SWI-Prolog eingebauten Unifikation. Welche Schlüsse kann man ziehen hinsichtlich des Occurs Check in SWI-Prolog?

Aufgabe 5 Lauflängencodierung (7 Punkte)

Bei der Lauflängencodierung werden mehrmals hintereinander auftretende Zeichen durch das Zeichen selbst und die Anzahl aufeinanderfolgender Vorkommen des Zeichens ersetzt. So wird beispielsweise die Folge *aaabbccccabbbb* als 3*a*2*b*3*c*1*a*4*b* codiert. Eine Gruppe von aufeinanderfolgenden gleichen Zeichen ist durch einen Term $[N, S]$ darzustellen, wobei N die Anzahl des Zeichens S ist. Programmieren Sie ein Prädikat, das die Lauflängencodierung implementiert.

Beispiel eines beabsichtigten Programmablaufs:

```
?- encode([a,a,a,b,b,c,c,c,a,b,b,b,b],X).
X = [[3,a],[2,b],[3,c],[1,a],[4,b]]
```