Institut für Programmstrukturen und Datenorganisation
Prof. Dr.-Ing. Gregor Snelting
gregor.snelting@kit.edu
Prof. Dr. Ralf Reussner
reussner@kit.edu

Programmierparadigmen – WS 2021/22

https://pp.ipd.kit.edu/lehre/WS202122/paradigmen/uebung

Blatt 7: Prolog

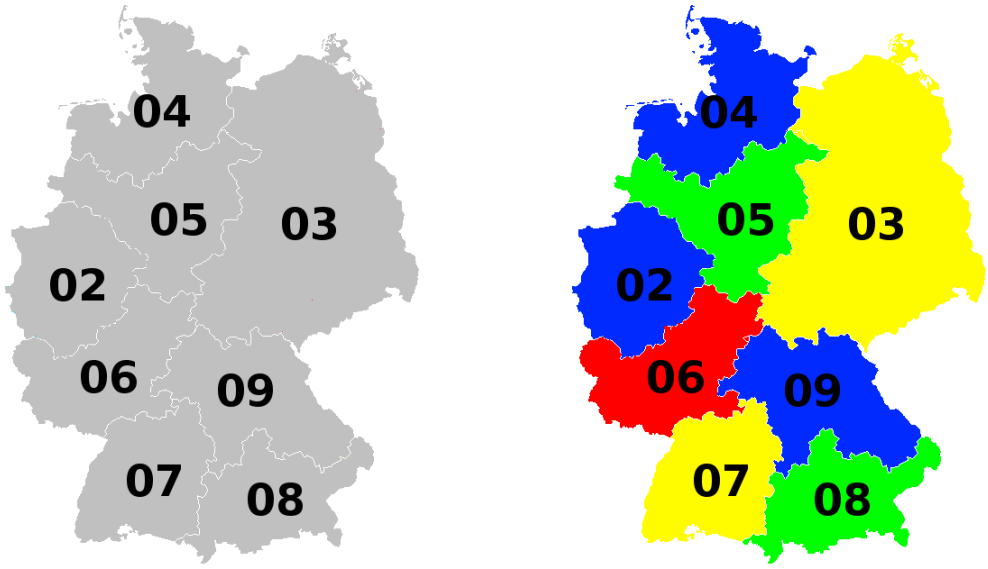
Abgabe: 10.12.2021, 14:00
Besprechung: 13.12. – 14.12.2021

Reichen Sie Ihre Abgabe bis zum 10.12.2021 um 14:00 in unserer Praktomat-Instanz unter https://praktomat.cs.kit.edu/pp_2021_WS ein.

1 Vier Farben

Der Vier-Farben-Satz besagt, dass für jede beliebige Landkarte vier Farben ausreichen, um diese so einzufärben, dass keine aneinandergrenzenden Gebiete die gleiche Farbe bekommen. In dieser Aufgabe sollen Sie mit Hilfe logischer Programmierung solch eine Einfärbung der Karte der Telefonvorwahlbereiche der BRD ausrechnen.

Abbildungen: nach https://de.wikipedia.org/wiki/Datei:Telefonvorwahlbereiche-Deutschland.png



1. Definieren Sie ein Prolog-Prädikat `farbe`, welches die vier zu verwendenden Farben festlegt.

   Beispiellösung:

   ```
   farbe(blau).
   farbe(gelb).
   farbe(gruen).
   farbe(rot).
   ```

2. Definieren Sie nun eine Regel für ein Prädikat `nachbar(X,Y)`, welches wahr ist, wenn X und Y unterschiedliche Farben sind.

   Beispiellösung:

```
nachbar(X, Y) :- farbe(X), farbe(Y), X \= Y.
```

3. Definieren Sie zuletzt ein 8-stelliges Prädikat `deutschland`, welches die Topologie der Karte der 8 Vorwahlbereiche beschreibt.

   Beispiellösung:

```
deutschland(VW02,VW03,VW04,VW05,VW06,VW07,VW08,VW09) :-
        nachbar(VW04, VW05), nachbar(VW04, VW03),
        nachbar(VW05, VW02), nachbar(VW05, VW06), nachbar(VW05, VW03),
        nachbar(VW02, VW06),
        nachbar(VW03, VW06), nachbar(VW03, VW09),
        nachbar(VW06, VW09), nachbar(VW06, VW07),
        nachbar(VW09, VW07), nachbar(VW09, VW08),
        nachbar(VW07, VW08).
```

4. Wie bekommen Sie nun eine gültige Einfärbung der Karte?

   Beispiellösung: Durch Abfrage des `deutschland`-Prädikats:
   `?deutschland(VW02,VW03,VW04,VW05,VW06,VW07,VW08,VW09).`

5. Was müssen Sie tun, um zu testen, ob die Karte auch mit nur drei Farben färbbar ist?

   Beispiellösung: Man entferne eines der Fakten von `farbe` und frage das `deutschland`-Prädikat erneut ab.

2 Reguläre Ausdrücke [Klausuraufgabe vom WS 2013/14, 15 Punkte]

Reguläre Ausdrücke lassen sich als Prolog-Terme in Präfixnotation darstellen. Bei Prolog-Systemen, welche die Zeichen $\cdot$, $\star$ und $\cup$ als Funktor erlauben (wie beispielsweise SWI-Prolog), lassen sich z.B.

$a \cdot b \cdot c$	als	`·(a,·(b,c))`
a^*	als	`⋆(a)`
$\varepsilon \cup b$	als	`∪(ε,b)`
$a^* \cup (a \cdot b \cdot c) \cup (\varepsilon \cup b)^*$	als	`∪(⋆(a),∪(·(a,·(b,c)),⋆(∪(ε,b))))`

darstellen. Zeichen $a, b, c, \ldots \in \Sigma$ werden also als Prolog-*Atome* `a,b,c,...` dargestellt.

Ein regulärer Ausdruck α *akzeptiert* eine Folge von Zeichen, falls diese in der durch den Ausdruck definierten Sprache $L(\alpha)$ enthalten ist:

- Ausdruck ε akzeptiert die leere Zeichenfolge
- Ausdrücke a, b, c, ... akzeptieren jeweils die Zeichenfolge „a“, „b“, „c“, ...
- Ausdrücke $\alpha \cup \beta$ akzeptieren eine Zeichenfolge s, falls s durch α oder β akzeptiert wird
- Ausdrücke $\alpha \cdot \beta$ akzeptieren eine Zeichenfolge $s_1 \cdot s_2$, falls α die Folge s_1 und β die Folge s_2 akzeptiert
- Ausdrücke α^* akzeptieren
  - die leere Zeichenfolge, sowie
  - Zeichenfolgen $s_1 \cdot s_2$, falls s_1 nicht die leere Folge ist, α die Folge s_1 akzeptiert, und α^* die Folge s_2 akzeptiert

Aufgabe Implementieren Sie ein Prolog-Prädikat `matches(Regexp,S)` das für Ausdrücke `Regexp` und Zeichenfolgen `S` bestimmt, ob `S` durch `Regexp` akzeptiert wird. Beispiel:

```
?matches(⋆(·(a,·(b,c))),[a,b,c]).   ?matches(⋆(·(a,·(b,c))),[]).
true .                               true .

?matches(⋆(·(a,·(b,c))),[a,b,c,d]). ?matches(⋆(·(a,·(b,c))),[a,b,c,a,b,c]).
false.                               true .
```

Hinweis: Die Prädikate `append(S1,S2,S)`, **`not`** und **`atom`** könnten nützlich sein!

Beispiellösung:

```
matches(ε,[]).

matches(C,[C]) :- atom(C), C \= ε.

matches(∪(A,_),S) :- matches(A,S).
matches(∪(_,B),S) :- matches(B,S).

matches(·(A,B),S) :- append(S1,S2,S), matches(A,S1), matches(B,S2).

matches(⋆(_),[]).
matches(⋆(A),S) :- append(S1,S2,S), not(S1=[]), matches(A,S1), matches(⋆(A),S2).
```

Bemerkung: Diese Implementierung ist nicht besonders effizient. Eine bessere Implementierung ergibt sich, wenn man zunächst ein Prädikat `matchesPrefix(Regexp,S,S2)` definiert, das für Ausdrücke `Regexp` und Zeichenfolen `S` erfüllt ist, falls ein Anfangsstück s_1 von $\mathtt{S} = s_1 \cdot s_2$ durch `Regexp` gematcht wird. In diesem Fall enthält S2 dann die Restzeichenfolge s_2.

3 Wolf, Ziege, Kohl [Klausuraufgabe vom SS 2012] [25 Punkte]

Ein Mann mit einem Wolf, einer Ziege und einem Kohlkopf will mit seinem Boot einen Fluss überqueren. Neben dem Mann hat maximal eines der drei Dinge im Boot Platz. Weiterhin gilt:

- Befinden sich Wolf und Ziege unbeaufsichtigt am gleichen Ufer, so frisst der Wolf die Ziege.
- Befinden sich Ziege und Kohl unbeaufsichtigt am gleichen Ufer, so frisst die Ziege den Kohl.

(Wie) kann der Mann alle drei Dinge heil vom Ufer `links` an das Ufer `rechts` transportieren?

In Prolog lässt sich die Situation vor und nach jeder Flussüberfahrt als Tupel `(Mann,Ziege,Wolf,Kohl)` von Ufern darstellen. Die unerwünschte Situation, dass sich der Mann und der Kohl am rechten, die Ziege und der Wolf jedoch am linken Ufer befinden, z.B. als: `(rechts,links,links,rechts)`.

1. Definieren Sie einen einstelligen *Tester* [7 Punkte]

   `erlaubt(Situation)`

   welcher für genau die Situationen erfüllt ist, in denen keines der Dinge gefressen wird.

2. Wir benötigen einen dreistelligen *Generator* `fahrt(S1,F,S2)` [8 Punkte]
   der zu Situation `S1` bei Reerfüllung alle Situationen `S2` generiert, die durch einfache Flussüberfahrt entstehen können. Die Zeichenkette `F` beschreibt die Fahrt, besteht also aus dem mitgenommen Ding (oder „leer“, falls der Mann nichts mit nimmt).

   Definieren Sie

   `fahrt(S1,F,S2)`

   (In der Klausur mussten lediglich die Regeln für Überfahrten mit Ziege sowie die für leere Fahrten angegeben werden)

   Beispiel:

   ```
   ? fahrt((links,links,links,links),F,S2).
   F = 'Ziege',
   S2 = (rechts, rechts, links, links) ;

   F = 'leer',
   S2 = (rechts, links, links, links);

   ...
   ```

3. Das Prädikat `lösung(Fahrten)` generiert alle Lösungen des Problems: [10 Punkte]

   ```
   lösung(Fahrten) :- start(S), ziel(Z), erreichbar(S,[],Fahrten,Z).
   start((links,links,links,links)).
   ziel((rechts,rechts,rechts,rechts)).
   ```

   Definieren Sie das hierzu benötigte vierstellige Prädikat

   `erreichbar(S,Besucht,Fahrten,Z)`

welches für Start-Situation `S` und Ziel-Situation `Z` erfüllt ist, falls `Z` von `S` durch eine Folge von Flussüber**`fahrt`**en erreichbar ist. Dabei dürfen nur **`erlaubt`**e Zwischensituationen entstehen. Um Endlosschleifen zu vermeiden darf dabei weiterhin keine der in der Liste `Besucht` enhaltenen Situationen nochmals vorkommen. Die Liste `Fahrten` soll bei Erfüllung die Beschreibungen der einzelnen **`fahrt`**en in richtiger Reihenfolge enthalten.

Hinweis: Verwenden Sie die Prädikate `member` und `not` aus der Vorlesung.

Beispiellösung:

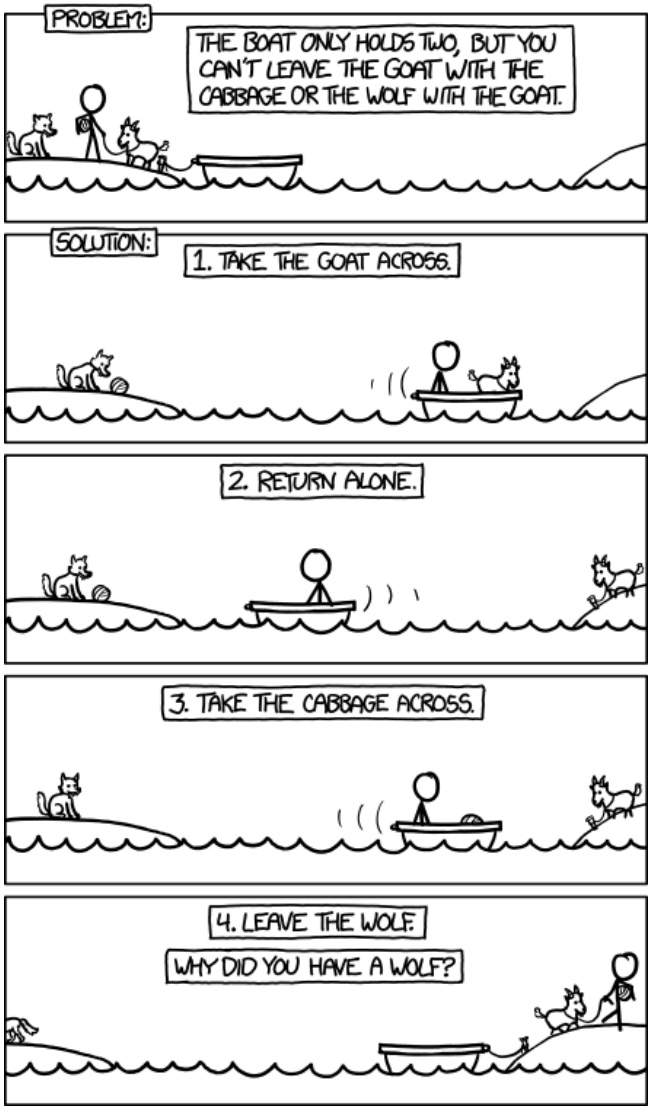
```
gegenueber(links,rechts).
gegenueber(rechts,links).

harmlos(_,X,Y) :- gegenueber(X,Y).
harmlos(Ufer,Ufer,Ufer).

erlaubt((Mann,Ziege,Wolf,Kohl)) :- harmlos(Mann,Ziege,Wolf), harmlos(Mann,Ziege,Kohl).

fahrt((U,U,Wolf,Kohl),  'Ziege', (UNeu,UNeu,Wolf,Kohl)) :- gegenueber(U,UNeu).
fahrt((U,Ziege,U,Kohl), 'Wolf',  (UNeu,Ziege,UNeu,Kohl)) :- gegenueber(U,UNeu).
fahrt((U,Ziege,Wolf,U), 'Kohl',  (UNeu,Ziege,Wolf,UNeu)) :- gegenueber(U,UNeu).
fahrt((U,Ziege,Wolf,Kohl), 'leer',  (UNeu,Ziege,Wolf,Kohl)) :- gegenueber(U,UNeu).

erreichbar(S,_,[],S).
erreichbar(S,Besucht,[Fahrt|Fahrten],Z) :-
        fahrt(S,Fahrt,ZwischenS), erlaubt(ZwischenS), not(member(ZwischenS,Besucht)),
        erreichbar(ZwischenS,[ZwischenS|Besucht],Fahrten,Z).
```



From http://xkcd.com/