Beispielaufgabe: Graf Rüdeger

Der schrullige Graf Rüdeger ist bekannt für seine spleenigen Ideen, mit denen er die Besucher auf seiner Burg in Lüne immer wieder überrascht. Jetzt hat er sich für seine Eingangspforten verspielte, aufwändige Schließanlagen konstruieren lassen. Eine Schließanlage besteht aus N Riegeln, deren Stellungen (auf oder zu) von außen sichtbar sind. Sie können einzeln nur nach den folgenden Regeln bewegt werden:

1. Riegel 1 kann immer bewegt werden.
2. Riegel 2 kann nur bewegt werden, wenn Riegel 1 auf ist.
3. Jeder andere Riegel kann genau dann bewegt werden, wenn der Riegel mit der nächst kleineren Nummer auf ist und alle mit noch kleineren Nummern zu sind.
4. Der Riegel N kann außerdem noch bewegt werden, wenn alle Riegel 1 bis N-1 zu sind.

Eine Pforte lässt sich nur öffnen, wenn alle Riegel auf sind.
Beispiel für N = 4
(in der letzten Zeile der Tabelle bedeutet Ri, dass Riegel i bewegt wurde):

	Pforte (ordentlich) zu									Pforte auf
Riegel 1	z	a	a	z	z	a	a	z	z	a	a
Riegel 2	z	z	a	a	a	a	z	z	z	z	a
Riegel 3	z	z	z	z	a	a	a	a	a	a	a
Riegel 4	z	z	z	z	z	z	z	z	a	a	a
z = zu **a = auf**		R1	R2	R1	R3	R1	R2	R1	R4	R1	R2

Graf Rüdeger hat mehrere Schließanlagen mit unterschiedlich vielen Riegeln bauen lassen. Leider kommt es häufig vor, dass Besucher einige Riegel bewegen, die Pforte aber nicht öffnen können und die Schließanlage unordentlich hinterlassen. Für Graf Rüdeger ist eine Pforte nur dann ordentlich geschlossen, wenn alle Riegel zu sind. Jeden Abend muss deshalb der Diener von Graf Rüdeger durch die Burg gehen und an allen Pforten die Riegel schließen. Du sollst ihm helfen.

→ Aufgabe

Schreibe ein Programm, das für ein beliebiges N und einen beliebigen Zwischenzustand einer Schließanlage die Folge derjenigen Riegel ausgibt, die der Diener bewegen muss, um die Pforte ordentlich zu schließen. Erzeuge für drei verschiedene Schließanlagen mit N > 4 und je zwei Zwischenzustände solche Bewegungsfolgen. Eine Bewegungsfolge davon soll für N = 6 und den Zwischenzustand (zu,zu,zu,auf,auf,zu) erzeugt werden.

→ Lösungsidee

Beim Betrachten der vier Regeln zum Verschieben der Riegel fallen zwei Dinge auf:

1. Regel 4 ist überflüssig. Sie dient lediglich dem Bewegen von Riegel N, der aber bereits mit Regel 3 (für N>2) bzw. Regel 2 (N=2) oder Regel 1 (N=1) bewegt werden kann. Regel 4 ermöglicht unter bestimmten Voraussetzungen ein „Abkürzen" – da aber in der Aufgabenstellung nicht nach der kürzesten Schließfolge gefragt war, kann man auf das Anwenden dieser Regel verzichten. Dadurch vereinfacht sich das Programm, da es nun nur noch drei Regeln zu beachten gibt.

2. Um einen beliebigen Riegel x bewegen zu können, ist lediglich die Stellung der Riegel mit Nummern kleiner x von Bedeutung. Diese müssen in eine bestimmte Stellung entsprechend der Regeln 1 bis 3 gebracht werden, bevor Riegel x bewegt werden kann. Andererseits beeinflusst die Stellung eines Riegels die Beweglichkeit der Riegel mit niedrigerer Nummer nicht. Es müssen also zuerst die Riegel mit hohen Nummern in die Endstellung gebracht werden, bevor dann die Riegel mit kleineren Nummern in ihre Zielstellung gebracht werden können.

Um die Pforte ordentlich zu schließen, beginnt man also mit Riegel N. Wenn man Glück hat, so ist Riegel N bereits geschlossen. Hat man Pech und Riegel N ist geöffnet, so müssen zuerst die Riegel N-1 bis 1 in die benötigte Stellung entsprechend Regel 3 bzw. 2 gebracht werden, bevor Riegel N endgültig geschlossen werden kann. Auf die gleiche Weise schließt man danach der Reihe nach Riegel N-1 bis Riegel 1.

Es bleibt nun noch zu klären, wie man die Riegel mit den kleineren Nummern in die benötigen „Zwischenpositionen" bekommt. Dazu kann man auch das soeben beschriebene (rekursive) Verfahren verwenden, das von der zu erzielenden Stellung und der Menge der zu bearbeitenden Riegel unabhängig ist. Das folgende Pseudoprogramm formalisiert diese Verallgemeinerung:

```
Bringe m Riegel in Stellung s_m, …, s_1:
  Wenn Riegel m nicht in Stellung s_m dann
      t_(m-1), …, t_1 sei Stellung um Riegel m
      bewegen zu können
      Bringe m-1 Riegel in Stellung t_(m-1), …, t_1
      Bewege Riegel m
  end
  Bringe m-1 Riegel in Stellung s_(m-1), …, s_1
end
```

Um Graf Rüdegers Diener mit dieser Prozedur zu helfen, muss dann nur noch

```
Bringe N Riegel in Stellung zu, …, zu
```

augerufen werden.

→ Programm-Dokumentation

Das Lösungsprogramm ist in Prolog verfasst, da sich das Problem sehr gut logisch formulieren lässt.

Der Zustand des Schlosses wird als Liste von Werten aus der Menge {auf, zu} dargestellt. Dabei steht der Zustand des Riegels N an erster Stelle in der Liste, der Zustand des Riegels N-1 an zweiter Stelle und so weiter. Der Zustand des Schlosses aus dem Pflichtbeispiel der Aufgabenstellung sieht zum Beispiel so aus:

```
[zu, auf, auf, zu, zu, zu]
```

Der Zustand „ordentlich geschlossen" sieht entsprechend so aus:

```
[zu, zu, zu, zu, zu, zu]
```

Die Folge der bewegten Riegel wird ebenfalls als Liste repräsentiert. Dabei steht an erster Stelle der Liste die Nummer des Riegels, der als erster bewegt wurde, an zweiter Stelle der Riegel, der als zweiter bewegt wurde, und so weiter. Die im Aufgabenblatt durchgeführten Bewegungen sehen zum Beispiel so aus:

```
[1, 2, 1, 3, 1, 2, 1, 4, 1, 2]
```

Zur Lösung des Problems wird ein Prädikat graf(Ist, Bewegungen, Soll) definiert. Dieses Prädikat ist erfüllt, wenn man durch Bewegen der in Bewegungen festgelegten Riegel entsprechend obiger Strategie vom Ist- in den Soll-Zustand gelangt. Lässt man bei einer Anfrage mit diesem Prädikat den zweiten Parameter variabel, wird das Prolog-Programm diese Variable mit einer Bewegungsfolge belegen.

Um die Beispielaufgabe zu lösen, muss also folgende Anfrage an das Prolog-System gerichtet werden:

```
graf([zu, auf, auf, zu, zu, zu], Bewegungen,
[zu, zu, zu, zu, zu, zu]).
```

→ Programm-Ablaufprotokolle

Nach Starten des Interpreters wird zuerst der Quellcode eingeladen.

```
?- consult('bwinf21.1').
% bwinf21.1 compiled 0.01 sec, 2,324 bytes
```

Nun kann das in der Aufgabenstellung geforderte Beispiel gelöst werden.

```
?- graf([zu, auf, auf, zu, zu, zu], Bewegungen,
[zu, zu, zu, zu, zu, zu]).
Bewegungen = [5, 1, 2, 1, 3, 1, 2, 1, 4, 1, 2,
1, 3, 1, 2, 1]
Yes
```

Belegt man alle Parameter, kann man zum Beispiel herausfinden, ob das in der Aufgabenstellung genannte Beispiel tatsächlich korrekt ist.

```
?- graf([zu, zu, zu, zu], [1, 2, 1, 3, 1, 2, 1,
4, 1, 2], [auf, auf, auf, auf]).
Yes
```

Kehrt man probehalber die Reihenfolge der Schließvorgänge um, so ergibt sich keine gültige Lösung.

```
?- graf([zu, zu, zu, zu], [2, 1, 4, 1, 2, 1,
3, 1, 2, 1], [auf, auf, auf, auf]).
No
```

Belegt man nun alle Parameter bis auf den letzten, kann man sich zu einem gegebenen Anfangszustand und einer gegebenen Bewegungsfolge den erreichten Endzustand der Schließanlage anzeigen lassen.

```
?- graf([zu, zu, zu, zu], [1, 2, 1, 3, 1, 2, 1, 4],
Schloss).
Schloss = [auf, auf, zu, zu]
Yes
```

Ist statt dessen nur der Endzustand und die Schließfolge bekannt, kann man den Anfangszustand der Schließanlage bestimmen.

```
?- graf(Schloss, [1, 2, 1, 3, 1, 2, 1, 4],
[auf, auf, zu, zu]).
Schloss = [zu, zu, zu, zu]
Yes
```

→ Programm-Text

Zunächst einige Worte zu Prolog-Programmen: In diesen beschreibt man Prädikate (quasi Funktionen, die nur die Wahrheitswerte wahr und falsch liefern können) und deren Erfüllbarkeit (wann der Wert wahr geliefert wird). Eine solche Beschreibung kann aus mehreren Fakten (wie im folgenden Programm bei ok und falsch, deren Gültigkeit sich allein aus festen Parameterwerten ergibt) oder Regeln (wie bei graf, wo hinter dem :- Vorbedingungen für die Erfüllbarkeit angegeben sind) bestehen. Die Notation [First|Rest] beschreibt eine Liste mit erstem Element First und der Liste der weiteren Elemente Rest. Bezeichner, die mit einem Großbuchstaben beginnen, sind Variablen, deren Belegung interessiert, während der Unterstrich _ eine beliebige Variable ist. Hat eine Prädikatbeschreibung mehrere Elemente, werden diese bei der Beantwortung einer Anfrage (die Überprüfung einer Regelbedingung ist wieder eine Anfrage) von oben abgearbeitet. Enthält eine Anfrage Variablen, werden automatisch deren mögliche Belegungen durchprobiert – mit dem Ziel, eine Belegung zu finden, mit der die Anfrage erfüllt wird. Nun aber das Programm:

```
% Welche Kombination aus Ist- und Soll-Zustand
% ist ok...
ok(auf, auf).
ok(zu, zu).

% ...und welche Kombination ist falsch.
falsch(auf, zu).
falsch(zu, auf).

% Wenn es keine Riegel gibt, braucht man auch keine
% zu bewegen.
graf([], [], []).

% Ist der oberste Riegel ok, so brauchen wir uns nur
% noch um die restlichen Riegel zu kümmern.
graf([X|Xs], S, [Y|Ys]) :-
  ok(X, Y),
  graf(Xs, S, Ys).

% Ist der oberste Riegel in der falschen Stellung,
% so müssen wir zuerst die kleineren Riegel ent-
% sprechend der Vorbedingungen der Regeln 1-3 ein-
% stellen, dann den obersten Riegel bewegen (also
% seine Nummer in die Liste mit den Bewegungen
% aufnehmen) und dann die restlichen Riegel in die
% endgültige Position bewegen.
graf([X|Xs], S, [Y|Ys]) :-
  falsch(X, Y),
  vorbedingung(Xs, Zs),
  graf(Xs, L1, Zs),
  length([X|Xs], N),
  append(L1, [N|L2], S),
  graf(Zs, L2, Ys).

% Für den ersten Riegel gibt es keine Vorbedingung.
vorbedingung([], []).

% Ab dem zweiten Riegel muß der nächstkleinere Riegel
% offen sein, und der Rest (wenn es noch kleinere
% Riegel gibt) geschlossen.
vorbedingung([_|Ls], [auf|Xs]) :-
  restzu(Ls, Xs).

% Prüfe, ob in einer Liste von Riegeln alle zu sind.
restzu([], []).

restzu([_|Ls], [zu|Xs]) :-
  restzu(Ls, Xs).
```

Lösung von Michael Schneider