Prof. Dr. Stephan Kleuker
Hochschule Osnabrück
Fakultät Ing.-Wissenschaften und Informatik
- Software-Entwicklung -

**Logik**

Sommersemester 2023
**8. Lernnotiz**

Hinweis: Diese Lernnotiz enthält einen sehr sinnvollen Vorschlag um den Lehrstoff der 8. Woche der Veranstaltung zu erlernen. Er ist gegliedert in die generellen Ziele und die Arbeitsschritte. Es ist notwendig, dass Sie die in dieser Lernnotiz genannten Videos bis zum Ende der offiziellen Vorlesungszeit (Do 16:30) durchgearbeitet haben. Während der Vorlesungszeit besteht die Möglichkeit in Zoom Fragen zu stellen und weitergehende Themen zu diskutieren.

https://hs-osnabrueck.zoom.us/my/kleuker

Einzelne Termine können kurzfristig per E-Mail vereinbart werden.

**Ziele**

- Verständnis und Fähigkeit zur Erstellung von rekursiven Prädikaten in Prolog mit verschiedenen Ansätzen.
- Verständnis und Fähigkeit zur Nutzung der Negation in Prolog mit ihren besonderen Randbedingungen.
- Verständnis von und Fähigkeit zur Nutzung in Prolog vordefinierter Prädikate zu Typ-Prüfungen sowie Ein- und Ausgaben.

**Arbeitsschritte**

- *Laden Sie sich das folgende Video zuerst herunter, wenn Sie die HS-Plattform nutzen* und schauen Sie sich diese an. Es ist sinnvoll die Folien danach nochmals durchzugehen.

  Folien 169 – 189:
  http://kleuker.iui.hs-osnabrueck.de/Videos/Logik/Logik07Prolog2.mp4 (83:22), auch https://youtu.be/4U9i_fxgqYE

- Lesen Sie das zur Vorlesung gehörende Fragen-Und-Antworten-Dokument, das meist kurz nach der Vorlesung auf der Veranstaltungsseite in der Nähe dieser Lernnotiz steht.

- Bearbeiten Sie Aufgabenblatt 8. Denken Sie daran, dass ich für Fragen meist kurzfristig erreichbar bin.

- Prüfen Sie, ob Sie die angegebenen Lernziele erreicht haben.

Falls es zu den Ansätzen in Prolog noch Fragen gibt, kann die folgende Aufbereitung der Fibonacchi-Umsetzung hilfreich sein.
Im ersten Schritt muss geklärt werden, welche Daten in die Funktionalität hineinfließen und wieviele Ergebnisse es geben soll. Hieraus ergibt sich die Stelligkeit des Prädikats. Bei der Fibonacchi-Entwicklung gibt es eine Zahl zum Aufruf x, der gesuchten x-ten Fibonacchi-Zahl und es wird ein Ergebnis erwartet, daraus folgt die Stelligkeit 2, es soll fib(X, Y) aufrufbar sein. Danach muss ein Algorithmus gefunden werden, der das Problem löst. Diese Ansätze für Algorithmen können sich in Prolog von den klassischen imperativen Ansätzen unterscheiden, das muss aber nicht immer der Fall sein. Hier kann die imperative Idee gut umgesetzt werden. Es werden zwei Hilfsvariablen benötigt, die letzten beiden Fibonacchi-Zahlen enthalten. Weiterhin wird ein Schleifenzähler benötigt, der von 1 bis X zählt. Daraus lässt sich ableiten, was für ein Prädikat zur Berechnung benötigt wird. Dieses Prädikat hat die beiden am Anfang genannten Variablen zur Berechnung und die drei gefundenen Hilfsvariablen. Da Name des Prädikats und Reihenfolge der Variablen ist frei wählbar, es kann auch der gleiche Name für das Prädikat genutzt werden. Die Entscheidung ist hier:

```
fib(Nummer der gesuchten Fibonacchi-Zahl
```

```
        , Schleifenzähler
        , aktuelle Fibonacchi-Zahl
        , vorherige Fibonacchi-Zahl
        , Ergebnis).
```

Daraus kann der eigentliche Aufruf schon spezifiziert werden:

```
fib(X, Y) :- fib(X, 1, 1, 0, Y).
```

Danach kann entweder erst über die Beendung der Berechnung oder über den Berechnungsweg nachgedacht werden. Da es sicher sein muss, dass es ein Ende gibt, wird hier mit dieser Idee angefangen. Das Ende wird erreicht, wenn der Schleifenzähler die Nummer des gesuchten Werts erreicht hat. Das Ergebnis steht dann im aktuellen Wert. Der erste funktionierende, etwas naive Weg sieht wie folgt aus.

```
fib(X, Zaehler, Aktuell, Vorher, Ergebnis):-
  X = Zaehler,
  Ergebnis = Aktuell.
```

Wichtig ist, dass das „=“ keine Zuweisung ist, es wird hier überprüft, ob die beiden Terme unifizierbar sind, was bei einer Konstanten und einem Wert immer gegeben ist. Deshalb ist genau nur diesem Kontext der Ansatz dies als Zuweisung zu sehen, in Ordnung.
In einer kürzeren Darstellung, kann die Forderung, dass an verschiedenen Stellen der gleiche Wert stehen soll, verkürzt dargestellt werden. Weiterhin sind in diesem Kontext uninteressante Variablen durch einen _ zu ersetzen. Das Ergebnis ist dann:

```
fib(X, X, Aktuell, _, Aktuell).
```

Für die Berechnung kann durchaus ein imperativer Ansatz genutzt werden, der dann zu einer Rekursion führt. Der Aufruf der Rekursion muss dabei näher am Ergebnis sein. Im konkreten Fall kommt man der Lösung einen Schritt näher, indem der Schleifenzähler erhöht wird.

```
ZaehlerNeu is Zaehler + 1
```

Um Endlosschleifen zu vermeiden, muss geprüft werden, ob der gesuchte Wert schon erreicht wurde.

```
not(X=Zaehler)
```

Weiterhin muss die neue Fibonacchi-Zahl berechnet werden. Für alle Berechnungen werden immer neue Variablen genutzt.

```
FibNeu is Aktuell + Vorher
```

Damit ist die Berechnung als Regel darstellbar.

```
fib(X, Zaehler, Aktuell, Vorher, Y) :-
  not(X = Zaehler),
  FibNeu is Aktuell + Vorher,
  ZaehlerNeu is Zaehler + 1,
  fib(X, ZaehlerNeu, FinNeu, Aktuell, Y).
```

Bei den Testprädikaten ist zu beachten, dass diese oft nicht für den vollständigen Test der Funktionalität geeignet sind.

```
testFib:-
    fib(1,1),
    fib(2,1),
    fib(3,2).
```

Mit obigem Test wird geprüft, ob das Prädikat fib zur Überprüfung von fib geeignet ist. Dazu müsste fib die Fibonacchi-Zahlen nicht berechnen. In einfachen Test kann die oft zusätzlich gewünsche Berechnungsfähigkeit nicht geprüft werden, die z. B. bei der Eingabe `fib(4, X)` benötigt wird. Deshalb muss zusätzlich zu den Tests eine interaktive Prüfung stattfinden. Weiterhin ist bei Tests zu beachten, dass immer auch Negativfälle mit not geprüft werden. Ist

das nicht der Fall, muss das zu testende Prädikat nur „aus irgendeinem Grund“ true zurückliefern. Im Beispiel reicht die folgende Realisierung aus, um den Test zu erfüllen.

```
fib(_, _).
```

Ergänzender Hinweis zur Negation: Es ist fast immer nur sinnvoll eine Negation zu nutzen, wenn es zum Zeitpunkt des Aufrufs keine freien Variablen mehr gibt. Dies soll folgendes Beispiel verdeutlichen.

```
c(42,42).
b(42).
b(43).
a(X) :- b(X), not(c(X, X)).
```

Die folgende Anfrage gibt das nachstehende Ergebnis.

```
?- a(Y).
Y = 43.
```

Bei der Abarbeitung von not hat X immer einen konkreten Wert zur Überprüfung.

Wird das Programm nur leicht modifiziert, genauer wird in einer Konjunktion die Reihenfolge vertauscht, passiert etwas anderes.

```
c(42,42).
b(42).
b(43).
a(X) :- not(c(X, X)), b(X).
```

Die folgende Anfrage gibt das nachstehende Ergebnis.

```
?- a(Y).
false.
```

Bei der Abarbeitung von not ist der Wert von X noch ungebunden, da es ein Paar für das Prädikat c gibt, ist das Ergebnis von not false und die zu erwartende Lösung wird nicht gefunden.