Frage: Können Sie den Ansatz bei der Fibonacchi-Berechnung nochmal genauer darstellen?

Antwort: 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).
```

Frage: Berechnungen finden in Prolog ja nicht wirklich als Prädikate statt, geht das auch anders?

Antwort: Die Feststellung stimmt, es ist zwar ein Prädikat, aber hat die Randbedingung, dass die ersten beiden Terme mit Zahlen unifiziert sind. Mit den bekannten Sprachkonstrukten kann eine Lösung mit einer beliebigen freien Variable umgesetzt werden.

```
summe(X,Y,Z):- nonvar(X), nonvar(Y), Z is X + Y, !.
summe(X,Y,Z):- nonvar(X), nonvar(Z), Y is Z - X, !.
summe(X,Y,Z):- nonvar(Z), nonvar(Y), X is Z - Y, !.
```

```
?- summe(2,3,X).
X = 5.

?- summe(2,X,5).
X = 3.

?- summe(X,3,5).
X = 2.

?- summe(2,3,5).
true.

?- summe(X,Y,5).
false.
```

Sollte im letzten Fall eine Fehlermeldung ausgegeben werden, müssten dazu weitere Regeln ergänzt werden. Der Cut-Operator ! wird in der 10. Veranstaltung vorgestellt und kann hier ignoriert werden.

Generell kann der Ansatz um einen Generator, zumindest für positive ganze Zahlen erweitert werden.

```
summe(X,Y,Z):-var(X), var(Y), nonvar(Z), Z >=-1, variante(X,Y,Z,0).

variante(A,B,A,B).
variante(X,Y,A,B) :-
    A > 0,
    TmpA is A - 1,
    TmpB is B + 1,
    variante(X, Y, TmpA, TmpB).
```

Dann ist auch folgende Anfrage möglich.

```
?- summe(X,Y,4).
X = 4,
Y = 0 ;
X = 3,
Y = 1 ;
X = Y, Y = 2 ;
```

```
X = 1,
Y = 3 ;
X = 0,
Y = 4 ;
false.
```

Eine schwer zu beantwortende Frage ist, wie so eine Lösung gefunden wird. Generell gibt es relativ wenig Konzepte für die Entwicklung von Prädikaten. Die zentrale Idee ist meist, dass überlegt wird, welche Informationen für das Ergebnis benötigt werden, das sind die zuerst benötigten Variablen. Im zweiten Schritt wird ein Ansatz zur Berechnung der Ergebnisse überlegt, hieraus folgt eine sehr unterschiedliche Anzahl von Hilfsvariablen. Sollen dabei Informationen über mehrere Rechenschritte weitergegeben werden, wird dies Information typischerweise in Listen festgehalten.