1.) Komposition von Abbildungen

Aufgaben: 2

```
> restart;
```

Definition der Komposition

Von fundamentaler Bedeutung ist die Tatsache, dass man zwei Abbildungen zu einer dritten komponieren kann, wenn der Wertebereich der einen gleich dem Definitionsbereich der anderen ist. Es gibt kaum ein Teilgebiet der Mathematik, in welchem hiervon nicht permanent Gebrauch gemacht wird.

```
> f:=x->x^2+1;
```

$$f := x \to x^2 + 1 \qquad \textbf{(1.1.1)}$$

```
> g:=y->y^3-y^2;
```

$$g := y \to y^3 - y^2 \qquad \textbf{(1.1.2)}$$

```
> (g@f)(x);
```

$$(x^2+1)^3 - (x^2+1)^2 \qquad \textbf{(1.1.3)}$$

```
> g(f(x));
```

$$(x^2+1)^3 - (x^2+1)^2 \qquad \textbf{(1.1.4)}$$

MATH: Sind $f: X \to Y$ und $g: Y \to Z$ Abbildungen, so ist die **Komposition** oder Hintereinanderausführung von f mit g definiert als

$g \circ f: X \to Z: x \mapsto g(f(x))$.

Mit anderen Worten (in der Paarmengensprechweise):

$g \circ f := \{(x, z) \in X \times Z \mid \exists\, y \in Y: (x, y) \in f \text{ und } (y, z) \in g\}$

oder mittels Funktionswerten formuliert:

$g \circ f := \{(x, z) \in X \times Z \mid \exists\, y \in Y \text{ mit } y = f(x) \text{ und } z = g(y)\}$

DENKANSTOSS: Beweise, dass $g \circ f$ wirklich eine Abbildung von X nach Z ist.

Wichtiger Spezialfall: Selbstkomposition einer Abbildung einer Menge in sich (Dynamisches System)

MATH: Die Komposition einer Abbildung mit sich selbst ist immer möglich, wenn Definitions- und Wertebereich übereinstimmen. Dies ergibt eine besonders anschauliche, fast könnte man sagen, dynamische Vorstellung von der Funktion:

```
> f:=x->x^2+1 mod 12;
```

$$f := x \to (x^2+1) \bmod 12 \qquad \textbf{(1.2.1)}$$

Wir wollen diese Abbildung als Abbildung von $X := \{0, \ldots, 11\}$ in sich unter dem Gesichtspunkt der Komposition untersuchen. Da die Menge endlich ist, muss für jedes $x \in X$ die Folge $i \mapsto f(f(\ldots f(x)\ldots))$ (f wird i-mal angewandt)

irgendwann (eventuell nach einer Vorperiode) periodisch werden.
Man sollte sich diese Aussage klar machen bzw. beweisen, da sie in Zukunft noch häufig gebraucht werden wird.

MAPLE fasst die Abbildung f als Prozedur auf:

```
> whattype(eval(f));
```

procedure **(1.2.2)**

```
> Folg:=proc(f::procedure, m::integer)
    local F,s;
    F:=m;
    s:=m;
    while not (f(s) in {F}) do
      s:=f(s);
      F:=F,s;
    end do;
    return F,f(s);
  end proc:
> for i from 0 to 11 do Folg(f,i) end do;
```

$$0, 1, 2, 5, 2$$
$$1, 2, 5, 2$$
$$2, 5, 2$$
$$3, 10, 5, 2, 5$$
$$4, 5, 2, 5$$
$$5, 2, 5$$
$$6, 1, 2, 5, 2$$
$$7, 2, 5, 2$$
$$8, 5, 2, 5$$
$$9, 10, 5, 2, 5$$
$$10, 5, 2, 5$$
$$11, 2, 5, 2$$ **(1.2.3)**

Es ergibt sich also folgende Beschreibung der Abbildung f: Die Elemente 2 und 5 bilden eine Periode der Länge zwei, d. h. $f(2) = 5, f \circ f(2) = 2$. Die übrigen Elemente verteilen sich in Vorperioden: $6 \mapsto 1 \mapsto 2, 0 \mapsto 1 \mapsto 2, 11 \mapsto 2$ sowie $9 \mapsto 10 \mapsto 5, 3 \mapsto 10 \mapsto 5, 8 \mapsto 5, 4 \mapsto 5$. Übrigens kann man diese Beschreibung auch ziemlich gut aus der ursprünglichen Definition von f als Teilmenge von $\{0,\ldots, 11\} \times \{0,\ldots, 11\}$ ablesen:

```
> {seq([i,f(i)],i=0..11)};
```

$$\{[0, 1], [1, 2], [2, 5], [3, 10], [4, 5], [5, 2], [6, 1], [7, 2], [8, 5], [9, 10], [10, 5], [11, 2]\}$$ **(1.2.4)**

ÜBUNG [01]:

1) Man fertige eine Papierskizze, genauer ein Pfeildiagramm, von f an, indem man die Punkte 0 bis 11 jeweils einmal aufzeichnet und zwei Punkte i und j genau dann durch einen Pfeil von i nach j verbindet, wenn $f(i) = j$.
(Wer sich sich an einen gerichteten Graphen erinnert fühlt, hat völlig recht).
2) Wiederhole die obige Beschreibung der Iteration der Funktion f mit Hilfe dieser Skizze.

MAPLE: Für Abbildungen von $\{1,\ldots, n\}$, die wir durch Listen in Maple repräsentiert hatten, ist die Komposition wie folgt zu realisieren:

```
> f:=[2,1,3,4,2]; g:=[5,4,3,3,1];
```

$$f := [2, 1, 3, 4, 2]$$

$$g := [5, 4, 3, 3, 1] \quad \textbf{(1.2.5)}$$

```
> fg:=[seq(f[g[i]],i=1..5)];
```

$$fg := [2, 4, 3, 3, 2] \quad \textbf{(1.2.6)}$$

Das Assoziativgesetz gilt, das Kommutativgesetz nicht

MATH: Wir sprechen jetzt wieder von dem allgemeinen Fall, wo Definitions- und Wertebereich nicht übereinstimmen müssen. Hier gibt es nun einen ganz fundamentalen Satz für die Komposition von Abbildungen:

Sind $f: A \to B$, $g: B \to C$ und $h: C \to \mathrm{D}$ drei Abbildungen, so gilt das **Assoziativgesetz**:

$$h \circ (g \circ f) = (h \circ g) \circ f$$

als Abbildungen von A nach D.

Der Beweis ist eine sehr einfache Verifikation: Sei $x \in X$ beliebig. Dann gilt

$$(h \circ (g \circ f))(x) = h((g \circ f)(x)) = h(g(f(x))) = (h \circ g)(f(x)) = ((h \circ g) \circ f)(x),$$

was die Behauptung beweist. Wir wollen sehen, ob Maple dies auch weiß. Zunächst müssen wir die Wertzuweisung an die Variablen rückgängig machen.

```
> f:='f':g:='g':h:='h':
> evalb(h@(g@f)=(h@g)@f);
```

$$true \quad \textbf{(1.3.1)}$$

MATH: Damit zwei Abbildungen f und g kommutieren können, müssen beide Abbildungen Selbstabbildungen der selben Menge sein, also $f: A \to A$ und $g: A \to A$ für eine geeignete Menge A. Aber selbst unter diesen Voraussetzungen ist im Allgemeinen $f \circ g \neq g \circ f$, wie wir an der folgenden Übung sehen.

ÜBUNG [02]:

Gib Abbildungen f und g der Menge $\{1, 2, 3\}$ in sich an mit

1.) $f \circ g = g \circ f$
2.) $f \circ g \neq g \circ f$
Teil (2) zeigt, dass bei Abbildungen im Allgemeinen nicht das Kommutativgesetz gilt.
(Hinweis: Listenschreibweise statt geschlossene Form macht es leichter.)

MATH: Für eine Teilmenge T der Menge X heißt die Abbildung
$e: T \to X: t \mapsto t$
die **Einbettung** von T in X. Für jede Abbildung $f: X \to Y$ ist die Komposition $f \circ e$ gleich der **Einschränkung** von F auf T:
$f_{|T} = f \circ e: T \to Y: t \mapsto f(t)$

2.) Fasern

Aufgaben: 4

```
> restart;
```

Bijektive Abbildungen

Unter den Abbildungen nehmen die bijektiven Abbildungen eine besondere Rolle ein, weil sie die Grundlage für Vergleiche zwischen Mengen sind. Sie können als Umbenennung der Elemente einer Menge durch die Elemente einer anderen Menge aufgefasst werden. Sie verallgemeinern den Zählvorgang, den man schon als Kind kennenlernt.

MATH: Sei $f: X \to Y$ eine Abbildung (also insbesondere ist $f \subseteq X \times Y$). f heißt **bijektiv**, falls
$g := \{(y, x) \in Y \times X | (x, y) \in f\}$
eine Abbildung von Y nach X ist. In diesem Fall heißt g die **Umkehrabbildung** von f oder die zu f **inverse Abbildung** und wird mit f^{-1} bezeichnet.

DENKANSTOSS: $f: X \to Y$ ist genau dann bijektiv, wenn es eine Abbildung $g: Y \to X$ gibt mit
$f \circ g =$ Identität auf Y
und $g \circ f =$ Identität auf X
In diesem Fall ist g eindeutig und damit die oben definierte Umkehrabbildung.

MAPLE u. MATH: Eine Menge M heißt endlich, wenn es eine nicht-negative ganze Zahl n gibt mit einer bijektiven Abbildung
$M \to \{1, \ldots, n\}$.
In diesem Fall heißt $|M| := n$ die **Kardinalität** von M. (**DENKANSTOSS**: Warum ist

die Kardinalität eindeutig?) Die Herstellung einer solchen Abbildung heißt im Volksmund "zählen'', daher nennen wir eine solche Abbildung (manchmal auch ihre Inverse) eine **Abzählung**. Wie zählt man mit Maple?

```
> m:=seq(x^2+x-1,x=-10..10);
```

$$m := 89, 71, 55, 41, 29, 19, 11, 5, 1, -1, -1, 1, 5, 11, 19, 29, 41, 55, 71, 89, 109 \quad \textbf{(2.1.1)}$$

```
> M:={m};
```

$$M := \{-1, 1, 5, 11, 19, 29, 41, 55, 71, 89, 109\} \quad \textbf{(2.1.2)}$$

```
> nops(M);
```

$$11 \quad \textbf{(2.1.3)}$$

```
> {seq([i,M[i]],i=1..nops(M))};
```

$$\{[1, -1], [2, 1], [3, 5], [4, 11], [5, 19], [6, 29], [7, 41], [8, 55], [9, 71], [10, 89], [11, 109]\} \quad \textbf{(2.1.4)}$$

```
> {seq([M[i],i],i=1..nops(M))};
```

$$\{[-1, 1], [1, 2], [5, 3], [11, 4], [19, 5], [29, 6], [41, 7], [55, 8], [71, 9], [89, 10], [109, 11]\} \quad \textbf{(2.1.5)}$$

Hier haben wir also eine bijektive Abbildung und ihre Inverse angegeben. Indem wir $M[i]$ schreiben, haben wir bereits benutzt, dass Maple intern die Elemente der Menge sortiert und in einer bestimmten Reihenfolge abgespeichert hat, d. h. Maple kennt bereits eine Abzählung. Wir haben sie nur explizit hingeschrieben. Benutzt man die Listenschreibweise, so geht es noch einfacher:

```
> [op(M)];
```

$$[-1, 1, 5, 11, 19, 29, 41, 55, 71, 89, 109] \quad \textbf{(2.1.6)}$$

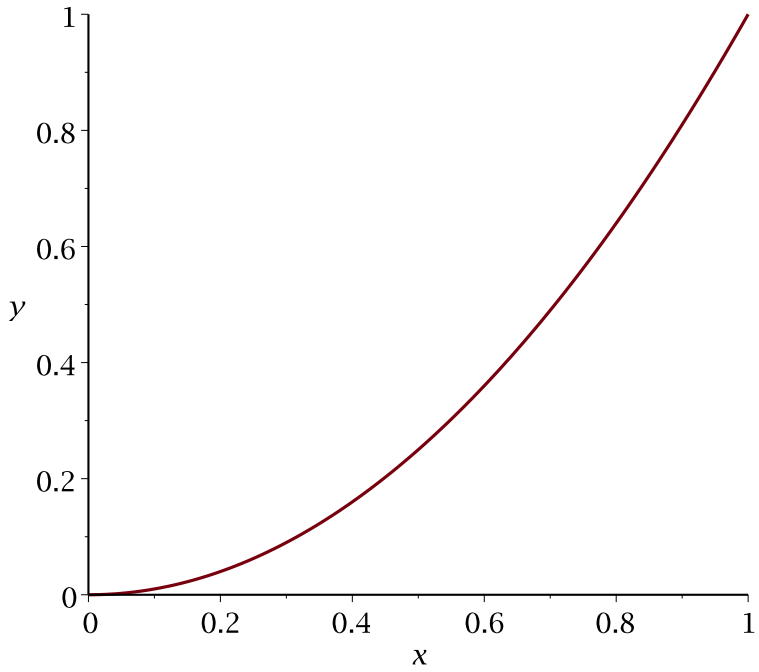
Leider brauchen wir für die Umkehrabbildung dann eine andere Datenstruktur. Man beachte, dass eine Bijektion deshalb vorliegt, weil alle Zahlen in der Liste verschieden (im Sinne von paarweise unterschiedlich, und nicht im Sinne von gestorben) sind.

MATH u. MAPLE: Wir wollen uns visuell davon überzeugen, dass $f_1 : [0, 1] \to [0, 1] : x \mapsto x^2$ bijektiv ist, jedoch $f_2 : [-1, 1] \to [0, 1] : x \mapsto x^2$ nicht.

```
> f:=x->x^2;
```

$$f := x \to x^2 \quad \textbf{(2.1.7)}$$

```
> plot(f(x),x=0..1,y=0..1);
```



Man "sieht", dass die Funktion jeden Wert in $[0, 1]$ genau einmal annimmt. (Dies werden wir viel später mit den Konzepten der strengen Monotonie und des Zwischenwertsatzes exakt beweisen.). Maple kennt auch die Umkehrfunktion:

```
> g:=x->sqrt(x);
```

$$g := x \to \sqrt{x} \qquad \textbf{(2.1.8)}$$

```
> (f@g)(x);
```

$$x \qquad \textbf{(2.1.9)}$$

```
> (g@f)(x);
```

$$\sqrt{x^2} \qquad \textbf{(2.1.10)}$$

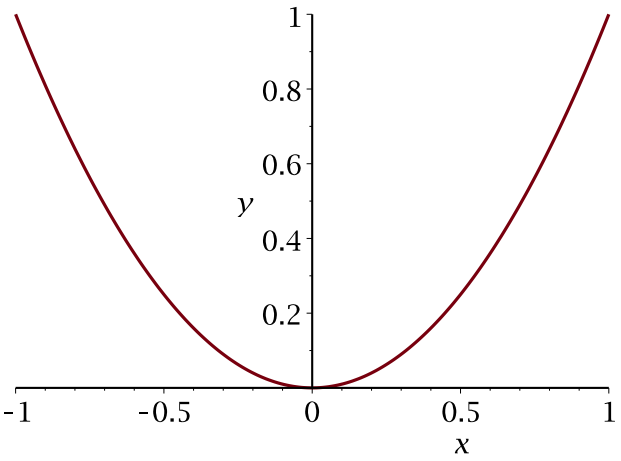
MAPLE weiß nicht, dass wir uns unter x eine reelle Zahl zwischen 0 und 1 vorstellen. Daher ist es vorsichtig. Mit etwas Hilfe geht es:

```
> (g@f)(x) assuming x>=0;
```

$$x \qquad \textbf{(2.1.11)}$$

Falls wir Maple glauben, haben wir jetzt gezeigt, dass f_1 in der Tat bijektiv ist. Wir wollen f_2 untersuchen:

```
> plot(f(x),x=-1..1,y=0..1);
```



Man "sieht", dass die Funktion jeden Wert in $[0, 1]$ (mit der Ausnahme von 0) genau zweimal annimmt. Also ist f_2 nicht bijektiv. Später werden wir sehen, dass die Tatsache, dass jeder Wert mindestens einmal angenommen wird, und die Existenz des Rechtsinversen (siehe g oben) äquivalent zueinander sind. Der gebräuchliche Begriff hierfür ist Surjektivität, den wir bald kennenlernen werden.
Hier nochmals ein verzweifelter Versuch, die Umkehrfunktion von f_2 mit MAPLE zu konstruieren. Das Scheitern, genauer die Art des Scheiterns, zeigt uns dann auch, dass f_2 nicht bijektiv ist.

```
> solve(y=f(x),x);
```

$$\sqrt{y}, -\sqrt{y} \qquad \textbf{(2.1.12)}$$

MATH: Wir wollen jetzt an zwei Beispielen die Philosophie der bijektiven Abbildungen demonstrieren. Wir hatten früher ein Programm geschrieben, welches die Potenzmenge von $\{1,\ldots, n\}$ erzeugen konnte. Jetzt wollen wir aber stattdessen die Potenzmenge der n-elementigen Menge $\{x_1,\ldots, x_n\}$ erzeugen, ohne die Arbeit nochmal zu machen.

```
> Pot := proc(n::nonnegint)
    local P, Q;
    if n = 0 then
      return {{}};
    end if;
    Q := Pot(n-1);
    P := map(a -> a union {n},Q);
    return Q union P;
  end proc:
> x:='x':
  S:=seq(i=x[i],i=1..4);
```

$$S := 1 = x_1, 2 = x_2, 3 = x_3, 4 = x_4 \qquad \textbf{(2.1.13)}$$

```
> subs([S],Pot(4));
```

$$\{\{\}, \{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}, \{x_1, x_2\}, \{x_1, x_3\}, \{x_1, x_4\}, \{x_2, x_3\}, \{x_2, x_4\}, \{x_3, x_4\}, \{x_1, x_2, x_3\}, \{x_1, x_2, x_4\}, \{x_1, x_3, x_4\}, \{x_2, x_3, x_4\}, \{x_1, x_2, x_3, x_4\}\} \quad \textbf{(2.1.14)}$$

ÜBUNG [03]:

1.) Verstehe das Programm Pot genau und kommentiere es gegebenenfalls.
2.) Schreibe ein neues Programm, welches das Programm **Pot** und das Konzept der bijektiven Abbildungen *benutzt*, um die Potenzmenge einer beliebigen endlichen Menge zu bestimmen.
Hinweis: Eine Bijektion zwischen Mengen induziert eine Bijektion zwischen den Potenzmengen. Nutze **subs**.
3.) Ändere das Programm **Pot** so ab, dass es direkt (mit obiger Idee des Programmes **Pot**) die Potenzmenge einer beliebigen endlichen Menge direkt bestimmt und ohne **subs** auskommt.

```
> PotM1 := proc(M::set) # Aufgabenteil 2)
    # ...
  end proc:
> PotM2 := proc(M::set) # Aufgabenteil 3)
    # ...
  end proc:
```

MATH: Wir wollen uns überlegen, dass eine n-elementige Menge genau 2^n Teilmengen hat. Was wir gerade demonstriert haben, zeigt, dass es genügt, $\{1,\ldots, n\}$ als n-elementige Menge zu nehmen. Wir erinnern uns an die charakteristische Funktion einer Teilmenge. Die Zuordnung Teilmenge $\rightarrow$ charakteristische Funktion ist eine Bijektion, wie wir sehen werden. Dies ist eine Abbildung der Potenzmenge von $\{1,\ldots, n\}$ in die Menge der 0-1-Folgen der Länge n, denn Abbildungen von $\{1,\ldots, n\}$ nach $\{0, 1\}$ hatten wir als Listen der Länge n mit Einträgen 0 oder 1 dargestellt. Wir wissen bereits, dass es 2^n solche Listen gibt (Darstellungen der Zahlen von 0 bis 2^n-1 im Dualsystem). Man kann sich jetzt überlegen, dass jede Teilmenge ihre charakteristische Funktion eindeutig festlegt und jede charakteristische Funktion umgekehrt die Teilmenge. In anderen Worten: Es liegt eine Bijektion vor.

MATH u. MAPLE: Hier ist ein Programm, welches alle bijektiven Abbildungen von $\{1, 2,\ldots, n\}$ nach $\{1, 2,\ldots, n\}$ in Listenschreibweise konstruiert.

```
> Sym:=proc(n::posint)
    if n=1 then
      return {[1]};
    end if;
    map(X->seq([op(X[1..j-1]), n, op(X[j..n-1])],j=1..n), Sym
  (n-1));
  end proc:
```

```
> Sym(3);
```

$$\{[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]\} \quad (2.1.15)$$

```
> nops(Sym(4));
```

$$24 \quad (2.1.16)$$

DENKANSTOSS: Benutze die Idee des Programmes, um eine Formel für die Anzahl der bijektiven Abbildungen von $\{1,\ldots, n\}$ in sich zu bestimmen.

Wir wollen jetzt zwei wichtige Begriffsbildungen kennenlernen, die die Abweichung einer Abbildung davon messen, bijektiv zu sein. Später werden wir sehen, dass eine Abbildung eine Aufteilung (genauer: Partition) ihres Definitionsbereich definiert.

Fasern

MATH: Ist $f: X \to Y$ eine Abbildung, so nennt man für jedes $y \in Y$

$$f^{-1}(\{y\}) := \{x \in X \mid f(x) = y\}$$

die **Faser** über y unter der Abbildung f oder das **volle Urbild** von y unter f. Allgemeiner definiert man für eine Teilmange $T \subseteq Y$ das **Urbild** von T unter der Abbildung f als

$$f^{-1}(T) := \{x \in X \mid f(x) \in T\}.$$

KOMMENTAR: f^{-1} ist in diesem Fall nur eine Notation und bezeichnet **nicht** die Umkehrabbildung, da diese im Allgemeinen gar nicht existiert.

DENKANSTOSS: Benutze die charakteristische Funktion einer Teilmenge von X, um zu zeigen, dass jede Teilmenge von X als Faser einer geeigneten Funktion vorkommen kann. Genauer: Hat Y mehr als ein Element, so existiert zu jeder Teilmenge T von X eine Abbildung $f: X \to Y$ und ein $y \in Y$ mit $f^{-1}(\{y\}) = T$.

MATH u. MAPLE: Die Bestimmung einer Faser kann man als Lösen eines Gleichungssystems auffassen:

```
> {solve(x^2=3,x)};
```

$$\{\sqrt{3}, -\sqrt{3}\} \quad (2.2.1)$$

Wir sehen eine 2-elementige Faser für die Abbildung $x \to x^2$ der reellen Zahlen in sich. Verkleinert man den Definitionsbereich, so kann die Faser kleiner werden. (**DENKANSTOSS**: Warum?)

MAPLE: Hier ist ein Programm zur Bestimmung der Faser einer Abbildung von $\{1, 2,\ldots, n\}$ in eine Menge ganzer Zahlen, wobei die Abbildung f als Liste $[f(1), \ldots, f(n)]$ gegeben sein soll:

```
> Faser:=proc(f::list(integer),y::integer)
```

```
    local F,i;
    F:=NULL;
    for i from 1 to nops(f) do
      if f[i]=y then
        F:=F,i;
      end if;
    end do;
    return {F};
  end proc:
```

```
> f:=[2,1,3,1,4,1,6,1];
```

$$f := [2, 1, 3, 1, 4, 1, 6, 1] \quad \textbf{(2.2.2)}$$

```
> Faser(f,1);
```

$$\{2, 4, 6, 8\} \quad \textbf{(2.2.3)}$$

```
> Faser(f,2);
```

$$\{1\} \quad \textbf{(2.2.4)}$$

```
> Faser(f,9);
```

$$\{\,\} \quad \textbf{(2.2.5)}$$

Eine etwas elegantere Form ist das folgende Programm.

```
> Faser:=proc(f::list(integer),y::integer)
    return map(i->if f[i]=y then i fi,{$1..nops(f)});
  end proc:
```

Oder noch kürzer:

```
> Faser:=proc(f::list(integer),y::integer)
    return select(i->f[i]=y,{$1..nops(f)});
  end proc:
```

ÜBUNG [04]:

Seien M und N endliche Mengen der jeweiligen Kardinalität $m := |M|$ und $n := |N|$. Gib jeweils eine Abbildung $f: M \to N$ an oder begründe, warum keine solche Abbildung existiert.

1.) $m = 42$, $n = 2$ und beide Fasern von f haben Kardinalität 21.
2.) $m = 3$, $n = 2$ und beide Fasern von f haben Kardinalität 2.
3.) $m = 3$, $n = 42$ und es gibt genau eine Faser von f, welche nicht leer ist.
4.) $m = 3$, $n = 4$ und jede Faser von f hat Kardinalität 1.

Bild und Surjektivität

MATH: Sei $f: X \to Y$ eine Abbildung. Man nennt

$$\text{Bild}(f) := f(X) := \{f(x) \mid x \in X\}$$

genauer

$$\{y \in Y \mid \exists\, x \in X \text{ mit } y = f(x)\}$$

das **Bild** von f. Im Falle $f(X) = Y$ heißt f **surjektiv** oder eine Abbildung auf Y. Letzteres ist offenbar äquivalent zu der Forderung, dass keine der Fasern von f leer ist.

MAPLE: Im Falle $X = \{1,\ldots, n\}$, wo wir f durch die Listenschreibweise gegeben haben, ist die Bestimmung von $\mathrm{Bild}(f)$ sehr einfach:

```
> f:=[seq((i-4)^2,i=1..12)];
```

$$f := [9, 4, 1, 0, 1, 4, 9, 16, 25, 36, 49, 64] \quad \textbf{(2.3.1)}$$

```
> B:={op(f)};
```

$$B := \{0, 1, 4, 9, 16, 25, 36, 49, 64\} \quad \textbf{(2.3.2)}$$

Der Definitionsbereich von f war $\{1,\ldots,$ **nops**$(f)\}$. Wir hatten den Wertebereich nicht spezifiziert. Aber jede Obermenge des gerade bestimmten Bildes B kann gewählt werden. Surjektiv ist die Abbildung nur, wenn man als Wertebereich B selbst wählt. Wir wollen letzteres tun und eine interessante Abbildung von B in $\{1,\ldots,$ **nops**$(f)\}$ bestimmen:

```
> FindErst:=proc(L::list(integer),a::integer)
    local i;
    if not(a in {op(L)}) then
      return NULL;
    end if;
    for i from 1 to nops(L) do
      if L[i]=a then
        return i;
      end if;
    end do;
  end proc:
> f ;
```

$$[9, 4, 1, 0, 1, 4, 9, 16, 25, 36, 49, 64] \quad \textbf{(2.3.3)}$$

```
> FindErst(f,1);
```

$$3 \quad \textbf{(2.3.4)}$$

```
> g:=map(a->[a,FindErst(f,a)], B);
```

$$g := \{[0, 4], [1, 3], [4, 2], [9, 1], [16, 8], [25, 9], [36, 10], [49, 11], [64, 12]\} \quad \textbf{(2.3.5)}$$

Wir stellen per Inspektion fest, dass g eine Abbildung von B nach $\{1,\ldots,$ **nops**$(f)\}$ ist. Wir bilden die Kompositition $f \circ g$. Zuvor aktivieren wir unser altes Programm, welches in der Paardatenstruktur den Funktionswert bestimmt:

```
> Wert := proc(F::set, x)
    local y, z;
    for y in F do
      if y[1] = x then
        return y[2];
      end if;
    end do;
  end proc:
```

```
> map(a->[a,f[Wert(g,a)]], B);
```
$$\{[0, 0], [1, 1], [4, 4], [9, 9], [16, 16], [25, 25], [36, 36], [49, 49], [64, 64]\} \quad \textbf{(2.3.6)}$$

So geht es auch:

```
> {seq([a,f[Wert(g,a)]],a=B)};
```
$$\{[0, 0], [1, 1], [4, 4], [9, 9], [16, 16], [25, 25], [36, 36], [49, 49], [64, 64]\} \quad \textbf{(2.3.7)}$$

Die Komposition $f \circ g$ ist also gleich der Identität auf B. **Dringende Aufforderung**: Versuche zu ergründen, warum das so ist. Dann hast du nämlich einen guten Teil des folgenden Satzes verstanden:

MATH: Sei $f: X \to Y$ eine Abbildung. Genau dann ist f surjektiv, wenn eine Abbildung $g: Y \to X$ existiert mit

$f \circ g = Id_Y :=$ Identitätsabbildung auf Y.

Man nennt g auch ein **Rechtsinverses** von f.

DENKANSTOSS: Die Rechtsinversen kommen durch Auswahl eines Elementes in jeder Faser zustande. Beachte: Bei einer surjektiven Abbildung ist keine Faser leer.

ÜBUNG [05]:

1.) Gib ein Rechtsinverses von $f: [-1, 1] \to [0, 1] : x \mapsto x^2$ an, wobei $[a, b] := \{x \in \mathbb{R} \mid a \leq x \leq b\}$.
2.) Gibt es eine inverse Abbildung zu f?

Fasern und Injektivität

MATH: Ist $f: X \to Y$ eine Abbildung, so heißt f **injektiv**, wenn gilt:

$(a, b \in X \wedge f(a) = f(b)) \Rightarrow a = b$.

In anderen Worten: Jede Faser von f besteht aus höchstens einem Element.

MATH u. MAPLE: Bei endlichem Definitionsbereich X ist f genau dann injektiv, wenn das Bild und der Definitionsbereich gleich viele Elemente enthalten. (**DENKANSTOSS**: Wie sieht es bei Surjektivität aus?) Deshalb haben wir bei $X = \{1, 2, \ldots, n\}$ und Listendatenstruktur einen bequemen Test für Injektivität:

```
> f:=[$2..10];
  g:=[seq(i^2,i=-4..5)];
```
$$f := [2, 3, 4, 5, 6, 7, 8, 9, 10]$$
$$g := [16, 9, 4, 1, 0, 1, 4, 9, 16, 25] \quad \textbf{(2.4.1)}$$

```
> evalb(nops(f)=nops({op(f)}));
```
$$true \quad \textbf{(2.4.2)}$$

```
> evalb(nops(g)=nops({op(g)}));
```
$$false \quad \textbf{(2.4.3)}$$

Kontrolle:

```
> map(i->Faser(f,i),{op(f)});
```

$$\{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}\} \quad \textbf{(2.4.4)}$$

```
> map(i->Faser(g,i),{op(g)});
```

$$\{\{5\}, \{10\}, \{1, 9\}, \{2, 8\}, \{3, 7\}, \{4, 6\}\} \quad \textbf{(2.4.5)}$$

MATH: Für eine nichtleere Menge X ist eine Abbildung $f: X \to Y$ ist genau dann injektiv, wenn eine Abbildung $h: Y \to X$ existiert mit

$$h \circ f = \mathrm{Id}_X := \text{Identität auf } X.$$

Jede derartige Abbildung h heißt **Linksinverse** von f.

Ebenso wie bei den Rechtsinversen kann man die Linksinversen aus den Fasern ablesen. Man kann es aber auch so sehen: f ist genau dann injektiv, wenn

$$ff: X \to f(X) : x \mapsto f(x)$$

bijektiv ist. Auf $f(X)$ definiert man h als ff^{-1} (keine andere Wahl möglich). Auf $Y - f(X)$ kann man h beliebige Werte aus X zuweisen.

ÜBUNG [06]:

1.) Warum zeigt der Befehl "**evalb(nops(f)=nops({op(f)}));**", dass f injektiv ist, und "**evalb(nops(g)=nops({op(g)}));**", dass g nicht injektiv ist?
2.) Fasse f von oben als Abbildung nach $\{1,\ldots, 11\}$ auf und konstruiere alle Linksinversen l mit Hilfe der Listenschreibweise.
Hinweis: Die $l(a)$ mit $a \in \mathrm{Bild}(f)$ liegen eindeutig fest. Wo kann man noch variieren?
3.) Allgemeiner: Sei $g: M \to N$ eine injektive Abbildung und M und N endliche Mengen $m := |M| \leq n := |N|$. Wie viele Linksinverse hat g?

MATH: Für eine Abbildung f sind folgende Aussagen äquivalent:
1) f ist bijektiv.
2) f ist injektiv und surjektiv.
3) Jede Faser von f besteht aus genau einem Element.