Mixed Pickles

— Sammlung verschiedener bewiesener und noch nicht bewiesener Ausssagen —

Stefan Pudritzki
Göttingen

24. August 2014

Inhaltsverzeichnis

Kapitel 1

Unbewiesene Aussagen

1.1 Tangensprodukte

Sei $n \in \mathbb{N}$ und sei

$$m := \begin{cases} \frac{n}{2} - 1 & : \quad n \text{ gerade} \\ \frac{n}{2} - \frac{1}{2} & : \quad n \text{ ungerade} \end{cases}$$

Behauptung:

$$\left\{ \prod_{j=1}^{m} \tan^2\left(\frac{j}{n}\pi\right) \right\} = \begin{cases} 1 & : \quad n \text{ gerade} \\ n & : \quad n \text{ ungerade} \end{cases}$$

Status: noch nicht selbst bewiesen, entdeckt im April 1998

Es liegt nahe, einen Beweis durch vollständige Induktion zu versuchen, indem die Behauptung in zwei getrennte Behauptungen $A_g(n_g)$ für gerade natürliche Zahlen n_g und $A_u(n_u)$ für ungerade natürliche Zahlen n_u aufgeteilt wird. Der Induktionsschluss müsste dann zeigen, dass jeweils die folgenden Aussagen gelten:

$$A_g(n_g) \implies A_g(n_g + 2)$$
$$A_u(n_u) \implies A_u(n_u + 2)$$

1.2 Differenzenquotienten höherer Grade

Implizite und induktive Definition des Differenzenquotienten n-ten Grades mit konstantem $\varepsilon > 0$:

$$f^{[0]}(x_0) := f(x_0)$$

$$\varepsilon f^{[n+1]}(x_0) := f^{[n]}\left(x_0 + \frac{\varepsilon}{2}\right) - f^{[n]}\left(x_0 - \frac{\varepsilon}{2}\right)$$

Im hochgestellten Index werden die eckigen Klammern [und] benutzt, um sich von der Bedeutung der runden Klammern (und) zu unterscheiden.

Die Idee ist, zur Berechnung höherer Grade immer dasselbe ε zu benutzen, ohne den Grenzübergang $\varepsilon \to 0$ auszuführen. Die Fragestellung lautet: Welcher Ausdruck beschreibt den Differenzenquotienten $(n+1)$-ten Grades unter ausschließlicher Verwendung der Ursprungsfunktion $f(x)$?

Für den ersten Grad erhalten wir mit $n = 0$:

$$\varepsilon f^{[1]}(x_0) = f\left(x_0 + \frac{\varepsilon}{2}\right) - f\left(x_0 - \frac{\varepsilon}{2}\right)$$

Für den zweiten Grad muss $n = 1$ gesetzt werden:

$$\varepsilon f^{[2]}(x_0) = f^{[1]}\left(x_0 + \frac{\varepsilon}{2}\right) - f^{[1]}\left(x_0 - \frac{\varepsilon}{2}\right)$$

Durch Multiplikation dieser Gleichung mit ε erhalten wir zunächst:

$$\varepsilon^2 f^{[2]}(x_0) = \varepsilon f^{[1]}\left(x_0 + \frac{\varepsilon}{2}\right) - \varepsilon f^{[1]}\left(x_0 - \frac{\varepsilon}{2}\right)$$

Nun wird erneut die Definiton verwendet, so dass nur noch Ausdrücke mit f vorkommen:

$$\varepsilon^2 f^{[2]}(x_0) \;=\; f(x_0+\varepsilon) - 2f(x_0) + f(x_0-\varepsilon)$$

Auf gleiche Art und Weise verfahren wir mit dem Differenzenquotienten dritten Grades und erhalten durch fortlaufendes Multiplizieren mit ε und Einsetzen der Definition:

$$\varepsilon^3 f^{[3]}(x_0) \;=\; f\left(x_0+\frac{3}{2}\varepsilon\right) - 3f\left(x_0+\frac{\varepsilon}{2}\right) + 3f\left(x_0-\frac{\varepsilon}{2}\right) - f\left(x_0-\frac{3}{2}\varepsilon\right)$$

Nun liegt die folgende Vermutung für beliebiges $n \in \mathbb{N}$ nahe:

$$\varepsilon^n f^{[n]}(x_0) \;=\; \left\{ \sum_{k=0}^{n} \left(\begin{array}{c} n \\ k \end{array} \right) (-1)^k f\left(x_0 + \left(\frac{n}{2} - k\right)\varepsilon\right) \right\}$$

Außerdem gilt offensichtlich beim Grenzübergang

$$\left\{ \lim_{\varepsilon \to 0} f^{[n]}(x_0) \right\} \;=\; f^{(n)}(x_0) \;=\; \{\, \partial_x^n f \,\}(x_0)$$

Status: noch nicht selbst bewiesen, 4. Dezember 2004

Auch hier liegt ein Induktionsbeweis nahe, da ja gerade die Definition des Differenzenquotienten $(n+1)$–ten Grades auf induktive Weise erfolgt ist.

Kapitel 2

Selbstbewiesene Aussagen

2.1 Schwingungen mit äquidistanten Frequenzen

Nach dem Buch *„Formeln + Hilfen zur Höheren Mathemtaik“*, Binomi Verlag, Springe, S. 41, lauten zwei Additionstheoreme für trigonometrische Funktionen:

$$\sin(\alpha+\beta) = \sin(\alpha)\cos(\beta)+\cos(\alpha)\sin(\beta) \tag{2.1}$$
$$\cos(\alpha+\beta) = \cos(\alpha)\cos(\beta)-\sin(\alpha)\sin(\beta) \tag{2.2}$$

Diese beiden Gleichungen lassen sich leicht durch die Beziehung der Exponentialfunktion zu den trigonometrischen Funktionen herleiten:

$$e^{i\phi} = \cos(\phi)+i\,\sin(\phi)$$

Aus (2.1) bzw. aus (2.2) folgen durch Substitution von β durch $+\beta$ in einem Fall und $-\beta$ im anderen Fall die vier Sätze

$$2\sin(\alpha)\cos(\beta) = \sin(\alpha+\beta)+\sin(\alpha-\beta) \tag{2.3}$$
$$2\cos(\alpha)\cos(\beta) = \cos(\alpha+\beta)+\cos(\alpha-\beta) \tag{2.4}$$
$$2\cos(\alpha)\sin(\beta) = \sin(\alpha+\beta)-\sin(\alpha-\beta) \tag{2.5}$$
$$2\sin(\alpha)\sin(\beta) = -\cos(\alpha+\beta)+\cos(\alpha-\beta) \tag{2.6}$$

Die Subtraktion der beiden Gleichungen (2.4) und (2.6) ergibt

$$\cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta) \;=\; \cos(\alpha+\beta) \tag{2.7}$$

Aus der Addition der beiden Gleichungen (2.3) und (2.5) erhalten wir

$$\sin(\alpha)\cos(\beta) + \cos(\alpha)\sin(\beta) \;=\; \sin(\alpha+\beta) \tag{2.8}$$

Zur einfachen und wenig rechenintensiven Erzeugung von Cosinuswerten und Sinuswerten äquidistanter Winkel kann ein induktives Verfahren angewendet werden. Wir wollen den Winkel β als Winkelabstand verwenden und definieren zunächst die beiden Konstanten

$$\begin{aligned} c_\beta &:= \cos(\beta) \\ s_\beta &:= \sin(\beta) \end{aligned}$$

Der Winkel α dient als Startwinkel. Wir definieren die beiden Anfangswerde

$$\begin{aligned} c_0 &:= \cos(\alpha) \\ s_0 &:= \sin(\alpha) \end{aligned}$$

Für alle natürlichen $n \geq 0$ können dann die Cosinuswerte und Sinuswerte aller äquidistanten Winkel auf induktive Weise erzeugt werden:

$$\begin{aligned} c_{n+1} &:= c_n c_\beta - s_n s_\beta \\ s_{n+1} &:= s_n c_\beta - c_n s_\beta \end{aligned}$$

Aus dieser Definition folgt mit den beiden Gleichungen (2.7) und (2.8) dann

$$
\begin{aligned}
c_{n+1} &= \cos(\alpha + n\beta) \\
s_{n+1} &= \sin(\alpha + n\beta)
\end{aligned}
$$

Wenn ω_0 die Startkreisfrequenz und ω_d der Kreisfrequenzabstand sind, dann folgt also für zeitabhängige Schwingungen:

$$
\begin{aligned}
c_{n+1} &= \cos(\omega_0 t + n\,\omega_d t) \\
s_{n+1} &= \sin(\omega_0 t + n\,\omega_d t)
\end{aligned}
$$

Kapitel 3

Diskussion um bestehende Definitionen

3.1 Nullte Potenz von Null

Der Umgang mit der Operation Null hoch Null, also der nullten Potenz der Zahl Null, ist in der mathematischen Literatur und auf numerischen Maschinen unterschiedlich gehandhabt, wie die folgenden Beispiele zeigen:

$a^0 := 1\ (a \neq 0)$	Schülkes Tafeln, 57. Auflage, B.G. Teubner Verlag, Stuttgart, S. 46
$a^0 := 1$ für alle $a \in \mathbb{R}$ (auch für $0^0 := 1$)	Repetitorium der Analysis (Timmann), Teil 1, Feldmann Verlag, Springe, S. 34
$\mathrm{pow}(0.0, 0.0) == 1.0$	Implementation der Standardfunktion pow des C–Compilers des Xcode–Entwicklungssystems unter Mac OS X auf einem Apple Mac mini
Tastatureingabe: $0, x^y, 0, =$ Anzeige: $"-\mathrm{E}-"$	Taschenrechner CASIO fx-992s

3.1.1 Eindeutigkeit durch Interpretation als unären Operator

Ich möchte hier eine Definition angeben, die ein eindeutiges Ergebnis liefert. Der Gedankenansatz besteht darin, die Multiplikation einer Zahl y mit einer Zahl x als unären Operator in der Form $(x\cdot)$ aufzufassen. Wir benötigen dazu ein System von Grundvoraussetzungen, die wir als Axiome in Quantorenlogik formulieren.

Axiome

Als erstes werden die Wertebereiche aller verwendeten Variablen festgelegt:

$$\begin{array}{ll} (1) & \forall\, x: \quad x \in \mathbb{R} \\ (2) & \forall\, y: \quad y \in \mathbb{R} \\ (3) & \forall\, n: \quad n \in \mathbb{N}_0 \end{array} \tag{3.1}$$

Wir unterscheiden zwischen einem Multiplikationsoperator $(x\cdot)$ und einer Zahl x. Im folgenden Axiom legen wir fest, wie ein Produkt mit diesem Operator aufgeschrieben wird:

$$\forall\, x: \ \forall\, y: \quad \{(x\cdot)y\} \ \text{ ist das Produkt aus } x \text{ und } y \tag{3.2}$$

Im nächsten Axiom wird die Wirkung des neutralen Operators $(1\cdot)$ auf eine beliebige Zahl x festgelegt:

$$\forall\, x: \quad \{(1\cdot)x\} \ = \ x \tag{3.3}$$

Die Potenz des Operators $(x\cdot)$ wird induktiv über alle n definiert. Dazu benötigen wir zunächst als Induktionsanfang die Festlegung, dass dieser Operator zur nullten Potenz dieselbe Bedeutung wie die des neutralen Operators $(1\cdot)$ und damit per Definition keine Wirkung haben soll:

$$\forall\, x: \quad (x\cdot)^0 \ = \ (1\cdot) \tag{3.4}$$

Das folgende Axiom stellt den definierenden Induktionsschritt über alle n dar:

$$\forall\, n: \ \forall\, x: \quad (x\cdot)^{n+1} \ = \ (x\cdot)(x\cdot)^n \tag{3.5}$$

Folgerung

Zunächst können wir das Axiom für den neutralen Operator (3.3) auch für die freie Variable y aufschreiben. Dies ist deshalb möglich, da durch die Festlegungen der Wertebereiche in (3.1) die beiden Variablen x und y Elemente derselben Menge sind:

$$\forall y: \quad \{(1\cdot)y\} = y \tag{3.6}$$

Wenn wir nun das Axiom über die nullte Potenz des Multiplikationsoperators (3.4) anwenden, dann erhalten wir zusätzlich die freie Variable x:

$$\forall x: \forall y: \quad \left\{(x\cdot)^0 y\right\} = y \tag{3.7}$$

Spezialisierung: Sei $x = 0$ gewählt. Dann erhalten wir die Schlussfolgerung:

$$\forall y: \quad \left\{(0\cdot)^0 y\right\} = y \tag{3.8}$$

Die n–te Potenz des Operators $(x\cdot)$ auf eine Zahl y bedeutet die n–fache Anwendung desselben Operators auf diese Zahl. Die nullte Potenz dieses Operators bedeutet die nullfache Anwendung des Operators auf eine Zahl y. Das Gleiche gilt auch für den speziellen Operator $(0\cdot)$. Die nullte Potenz bedeutet, dass dieser Operator gar nicht auf die Zahl y angewendet wird. (07.12.2011)

3.2 Aussagenlogische Variablen in Programmiersprachen

In manchen Programmiersprachen ist eine Implementation eines aussagenlogischen Datentyps *„bool“* üblich, bei dem die Wahrheitswerte *„false“* und *„true“* die Werte 0 bzw. 1 haben, z.B. nach Definition in der C++–Datei stdbool.h:

```
#define true 1
#define false 0
```

Die Wahl des Wertes *„true“* mit 1 halte ich für ungeschickt, da bei einer CPU mit fester Registerstruktur nur das niederwertigste Bit auf 1 und alle anderen Bits auf 0 gesetzt würden. Daher kann eine solche aussagenlogische Variable nicht unter Verwendung der häufig vorhandenen Bitverknüpfungen einer CPU in einer komplizierteren aussagenlogischen Funktion weiterverwendet werden.

Mein Vorschlag lautet, einen neuen Wahrheitswert mit geändertem Namen *„True“* mit -1 gleichzusetzen:

```
typedef int Bool;

#define True ((Bool)(-1))
#define False ((Bool)(0))
```

Da nun beim Wahrheitswert *„True“* alle Bits auf 1 und beim Wahrheitswert *„False“* alle Bits auf 0 gesetzt sind, können bequem kompliziertere aussagenlogische Funktionen mit mehreren aussagenlogischen Variablen unter Benutzung der Bitverknüpfen im Befehlsvorrat einer CPU konstruiert werden.

Außerdem können solche echten aussagenlogischen Variablen nun auch dazu benutzt werden, um zwischen verschiedenen Operanden mit derselben Wortbreite wie die der aussagenlogischen Variablen über logische Bitoperationen auszuwählen. Das folgende C–Quelltextbeispiel zeigt die praktische Anwendung:

```
/* aussagenlogische Funktion */

Bool    Logikfunktion(const Bool a, const Bool b)
{
        static Bool y;

        y = a&b;        /* Beispiel */
        return(y);
}


int main(int argc, char* argv[])
{
/* Voraussetzung: sizeof(Bool)==sizeof(float) */
        Bool    a,b,c;
        Bool    xb,yb,zb;
        float   x,y,z;

/*******************************************************/
/* Beispiel eines Programmteil zur Festlegung der
Operanden x,y und der Logikvariablen a,b: */

        a = False;
        b = True;

        x = 33.333f;
        y = 42.125f;
/*******************************************************/

        memcpy(&xb,&x,sizeof(Bool));
        memcpy(&yb,&y,sizeof(Bool));

        c  = Logikfunktion(a,b);

/* Auswahl eines Operanden: */
        zb = ((c&xb)|((~c)&yb));

        memcpy(&z,&zb,sizeof(Bool));

        printf("%f",z);
}
```

Wenn die Funktion *„Logikfunktion“* als Makro definiert würde, könnte man damit ganz ohne Verwendung von Sprungbefehlen eine if–else–Kontruktion durch reine Logik ersetzen. (24.08.2014)