Klein, aber O

Henning Thielemann

25. Juni 2004

Drei Semester sind schon wieder ins Land gegangen seit meinem ersten Aufruf zu mehr Sorgfalt beim Verwenden mathematischer Symbole. Es war natürlich nur ein Tropfen auf den heißen Stein. Noch immer kann die Einstellung „Das wollen wir jetzt mal nicht so genau nehmen.“, zu deutsch: „Korrekte Schreibweisen würde doch sofort jeder verstehen!“, einen großen Fanclub um sich vereinen. Es ist also an der Zeit, neues Futter nachzuschieben. Wie wäre es zur Abwechslung mit der beliebten O-Notation?

Wir wollen die Laufzeiten verschiedener Algorithmen bei verschiedenen Eingaben untersuchen, und weil wir über diesen Zusammenhang jetzt öfter sprechen werden, geben wir ihm einen Namen:

1 Definition. Für einen bestimmten Algorithmus nenne die Funktion, welche zu einer gegebenen Problemgröße die benötigte Laufzeit angibt, die *Laufzeitfunktion*. Wir wollen davon ausgehen, dass die Problemgrößen immer natürliche Zahlen sind, und die Laufzeiten positive reelle Zahlen.

Um die Laufzeiten bestimmter Algorithmen zu vergleichen, hat man die Symbole O, Θ, Ω eingeführt. In der Mathematik verwendet man die ähnlichen Landau-Symbole O und o oder auch $\lesssim$, $\gtrsim$. Nehmen wir ein Beispiel:

- Ein Algorithmus A zur Lösung eines Problems der Größe $n \in \mathbb{N}$ benötige die Zeit $f(n) \in \mathbb{R}_+$.
- Ein Algorithmus B zur Lösung eines möglicherweise anderen Problems der Größe $n \in \mathbb{N}$ benötige die Zeit $g(n) \in \mathbb{R}_+$.

Dann soll die Beziehung

> „A braucht zur Lösung seines Problems im Großen und Ganzen weniger als oder genau so viel Zeit wie B“

durch

$$f \in O(g)$$

ausgedrückt werden. Das $O(g)$ bezeichnet also eine Menge von Funktionen und zwar die Menge aller Funktionen, die „im Wesentlichen kleiner als oder gleich“ g sind. Nun soll dieses „im Wesentlichen kleiner“ so beschaffen sein, dass man sich nicht bei Kleinigkeiten aufhalten muss, andererseits muss es schon exakt definiert sein, damit man später auch etwas beweisen kann.

Fangen wir von vorne an. Es ist zum Beispiel sinnvoll f als „kleiner als oder gleich“ g zu bezeichnen, wenn der Algorithmus A bei jeder Problemgröße mindestens genaus so schnell wie Algorithmus B ist, kurz:

$$\forall n \in \mathbb{N} : f(n) \leq g(n)$$

Eigentlich können wir aber ganz gut damit leben, wenn A erst bei großen Problemen schneller als B ist, denn erst bei großen Problemen und mithin langen Laufzeiten wird es kritisch. Begnügen wir uns damit, dass es ein N gibt, ab dem $f(n)$ immer kleiner als $g(n)$ ist:

$$\exists N \in \mathbb{N} : \forall n \in \mathbb{N} : n \geq N \Rightarrow f(n) \leq g(n)$$

Zum Schluss wollen wir noch ein weiteres Auge zudrücken (mehr als zwei haben die meisten von uns ohnehin nicht (Hühneraugen, Fettaugen, Würfelaugen zählen nicht mit!)) und es zudem tolerieren, wenn Algorithmus A um einen konstanten Faktor langsamer ist als B, weil wir uns zum Beispiel nicht darauf festlegen wollen, wie lange die Grundoperationen in den Algorithmen A und B dauern. A könnte zum Beispiel mit Multiplikationen zum Ziel kommen und B mit Hilfe von Vergleichsoperationen. Um wie viel schneller Vergleiche als Multiplikationen sind, wollen wir also offen lassen und in einer nebulösen Konstanten c zusammenfassen. Damit wären wir angelangt bei:

$$\exists c \in \mathbb{R}_+ : \exists N \in \mathbb{N} : \forall n \in \mathbb{N} :$$
$$n \geq N \Rightarrow f(n) \leq c \cdot g(n)$$

Das werden wir nun als Definition für unser O

verwenden:

$$O(g) = \{f : \exists c \in \mathbb{R}_+ \wedge N \in \mathbb{N} : \forall n \in \mathbb{N} : n \geq N \Rightarrow f(n) \leq c \cdot g(n)\}$$

Übung 2. Wenn man einmal den konstanten Faktor c in die Definition von O aufgenommen hat, braucht man dann überhaupt noch dieses N, also die Einschränkung auf hinreichend große Probleme?

So in etwa läuft meistens die Einführung dieses Symbols in der Informatik-Vorlesung ab, dann aber passiert nicht selten etwas ganz scheußliches: Der Dozent meint, dass er ab jetzt der korrekten Schreibweise

$$f \in O(g)$$

die „einfachere" Schreibweise

$$f = O(g)$$

vorziehen wird. Die Vereinfachung besteht sicher darin, dass man den gebogenen Strich vom Enthaltensein-Zeichen nun gerade zeichnen und im Weiteren auch die schwierigsten, wenn nicht sogar alle denkbaren Behauptungen „beweisen" kann. Denn nach wie vor steht das Gleichheitszeichen für Identität oder aber zumindest für eine Äquivalenzrelation. Es widerstrebt mir aber schon ein bisschen, eine Funktion und eine Menge von Funktionen als äquivalent zu bezeichnen.

Beispiel 3. Es seien

$$\begin{aligned} f(n) &= n \\ g(n) &= n^2 \\ h(n) &= n^3 \end{aligned}$$

Es gilt offensichtlich (mit der „vereinfachten" Notation)

$$\begin{aligned} f &= O(h) \\ g &= O(h) \end{aligned}$$

Wenn „=" eine Äquivalenzrelation bezeichnet, gilt die Symmetrie, konkret

$$O(h) = g$$

und die Transitivität

$$f = O(h) \wedge O(h) = g \Rightarrow f = g$$

Also gilt

$$\forall n \in \mathbb{N} : n = n^2$$

w.z.b.w. (was zu befürchten war)

In diesem Beispiel war der Fehler offensichtlich, denn wenn man korrekt

$$\begin{aligned} f &\in O(h) \\ g &\in O(h) \end{aligned}$$

geschrieben hätte, wäre keiner auf die Idee gekommen, hieraus $f = g$ zu folgern. Aber man kann die falsche Schreibweise sicher auch geschickter unterbringen, zum Beispiel in einem falschen Beweis eines richtigen Sachverhaltes.

Nehmen wir der Vollständigkeit halber noch die Definitionen der anderen beiden Funktionsmengen hinzu:

- $\Omega(g)$ ist die Menge aller Funktionen f, die im Wesentlichen größer als oder gleich g sind:

$$\Omega(g) = \{f : \exists c \in \mathbb{R}_+ : \exists N \in \mathbb{N} : \forall n \in \mathbb{N} : n \geq N \Rightarrow f(n) \geq c \cdot g(n)\}$$

- $\Theta(g)$ ist die Menge aller Funktionen f, die im Wesentlichen genau so groß sind wie g: [1]

$$\Theta(g) = \{f : \exists \{c_0, c_1\} \subset \mathbb{R}_+ : \exists N \in \mathbb{N} : \forall n \in \mathbb{N} : n \geq N \Rightarrow c_0 \cdot g(n) \leq f(n) \leq c_1 \cdot g(n)\}$$

Eine Äquivalenzrelation zwischen Funktionen und Funktionenmengen zu basteln ist sicher nicht so glorreich, aber vielleicht bekommt man eine Äquivalenzrelation zwischen zwei Funktionen zu Stande. Die Relation

$$f \sim_h g :\iff (f \in O(h) \iff g \in O(h))$$

täte es zum Beispiel, aber dafür bräuchte ich auch gar keine O-Notation und könnte statt $O(h)$ irgendeine andere Funktionenmenge nehmen. Deshalb folgende kleine

[1] Solche aneinandergehängten Relationen wie in der Mengendefinition sparen zwar schön Schreibarbeit und sind übersichtlich, einem strengen Typtest halten sie aber nicht stand. Der Term bedeutet strenggenommen, dass der Wahrheitswert des ersten Vergleiches mit der dritten Zahl verglichen werden soll. – C/C++ erlaubt solchen Unsinn.

Übung 4. Wie kann man mit den Mengen O, Θ, Ω eine Äquivalenzrelation der Art „zwei Funktionen haben die gleiche Größenordnung“ formulieren?

Bleiben wir im Folgenden besser bei der korrekten Schreibweise mit dem Enthaltensein-Zeichen. Sie liefert im übrigen auch den Vorteil, viele sonst geläufige Notationen im Zusammenhang mit Mengen direkt weiterverwenden zu können.

$$O(f) \subset O(g)$$

bedeutet zum Beispiel, dass alle Funktionen, die im Wesentlichen kleiner sind als f, auch im Wesentlichen kleiner sind als g und dass die Funktionenklasse $O(g)$ noch mehr Funktionen enthält (wegen der echten Teilmengenbeziehung).

$$O(f) + O(g)$$

entspricht der Summe zweier Mengen. Die Summenmenge enthält alle Elemente, die sich als Summe aus einem Element der einen Menge und einem Element der anderen Menge darstellen lassen:

$$O(f) + O(g) = \{f' + g' : f' \in O(f) \wedge g' \in O(g)\}$$

Damit lassen sich dann solche Zusammenhänge wie $O(f) + O(g) = O(f + g)$ formulieren. Ja ja, das ist durchaus nicht trivial. Ihr könnt Euch ja mal daran versuchen.

Übung 5. Beweise, dass für Laufzeitfunktionen f und g stets

$$O(f) + O(g) = O(f + g)$$

gilt.

Lasst uns doch mal schauen, was man noch für Unfug mit dieser O-Notation anstellen kann. Mit $O(1)$ kann man alle Laufzeitfunktionen bezeichnen, welche zu Algorithmen gehören, deren Bearbeitungszeit überhaupt nicht von der Größe des Problems abhängt. Diese kann man guten Gewissens als Grundoperationen bezeichnen. Beispielsweise benötigt ein Rechner für die Addition zweier Zahlen immer die gleiche Zeit (sofern die Addition überhaupt korrekt, also ohne Überlauf ausführbar ist). Benötigt die Addition in der Wirklichkeit t Sekunden, dann setzt man in der Definition von O eben $c = t$, und es ist sofort klar, dass die zur Addition gehörige Laufzeitfunktion f in $O(1)$ enthalten ist.

Auf die gleiche Weise überprüft man leicht, dass folgende Beziehungen gelten:

$$\begin{aligned} 1 &\in O(1) \\ 4 &\in O(2) \\ 9 &\in O(3) \\ &\vdots \\ n^2 &\in O(n) \end{aligned}$$

Das ist zwar eine etwas schwächere Aussage als die von oben ($n = n^2$), klingt aber immer noch ziemlich gewagt. Aber ich denke, dass wir uns langsam damit abfinden sollten, man kann es eben einfach beweisen. :-) Das wollt Ihr gerne sehen, gell? Ich werde es Euch zeigen!

Um nicht in Konflikt mit den Variablennamen in der Definition von O zu kommen, benennen wir n in m um und skizzieren nun den Beweis für $m^2 \in O(m)$. Die Behauptung ist also:

$$m^2 \in \{f : \exists c \in \mathbb{R}_+ : \exists N \in \mathbb{N} : \forall n \in \mathbb{N} : n \geq N \Rightarrow f(n) \leq c \cdot m\}$$

oder kurz

$$\exists c \in \mathbb{R}_+ : \exists N \in \mathbb{N} : \forall n \in \mathbb{N} : n \geq N \Rightarrow m^2 \leq c \cdot m$$

Damit ist sofort klar, dass man für die geforderten Konstanten am einfachsten $c := m$ und $N := 1$ einsetzt, und damit ist die Behauptung auch schon bewiesen.

Manche Dozenten mögen diese simple Tatsache (in Worten: $n^2 \in O(n)$) nicht einsehen, aber sie stimmt einfach, wir haben doch den Beweis! Ja aber …das **kann** doch nicht stimmen! Das widerspricht doch dem gesunden Menschenverstand!

Ich räume zumindest ein, dass es unserer Intuition widerspricht. Aber an welcher Stelle hat uns die Intuition auf das Glatteis geführt? Kommt Ihr selber darauf? Argh, wie ich Euch kenne, habt Ihr *heute* sicher keine Lust zum Nachdenken und lest einfach weiter.

Sei's drum. Das Problem ist bereits die Uneindeutigkeit der Notation $n^2 \in O(n)$. In durchdachten Programmiersprachen wie Modula ist man gewohnt, den Typ einer Variablen genau festzulegen und sich nachher auch daran halten zu müssen, bei C/C++ muss man ihn nur noch festlegen, sich aber nicht daran halten (*characters*, *booleans*, *integers*, *sets* (welche als *integers* implementiert werden müssen) dürfen alle bunt durcheinandergewürfelt werden), na und bei Skriptsprachen gibt es gleich gar keine Deklarationen mehr. Die Strafe folgt auf dem Fuße:

Satz 6. (Goldene Regel der Informatik) Was man bei der Vorüberlegung (etwa bei der Deklaration) einspart, muss man der Fehlersuche zulegen.

Ihr habt in n wahrscheinlich einen Ausdruck gesehen, der von n abhängt, eine lineare Funktion also. Warum aber? n hängt doch auch von jeder anderen denkbaren Variable ab, wie zum Beispiel x. n ist bezüglich x konstant, also ist n eine konstante „Funktion", und deswegen hat auch mein Beweis geklappt. Wenn wir genau hinschauen, stellen wir aber fest, dass n überhaupt **keine Funktion** ist! Ursprünglich hatte ich n zur natürlichen Zahl erklärt, und natürliche Zahlen sind ganz gewiss keine Funktionen. Zwar steht es einer Funktion f frei, immer dieselbe natürliche Zahl n zurückzuliefern (also $\forall t : f(t) = n$), aber die Funktion f an sich ist eben nicht identisch mit der Zahl n. Die Funktion f beschreibt nur die Tatsache, dass ein paar bestimmte Argumente in ein paar bestimmte Ergebnisse überführt werden.

Beispielsweise kann eine Funktion so beschaffen sein, dass sie das übergebene Argument quadriert. Die Funktion, die dazu das übergebene Argument mit sich selbst multipliziert, könnte man f nennen. Eine andere Funktion g, die für alle Argumente n die ungeraden Zahlen von 1 bis $2n-1$ addiert, erledigt das Gleiche. Es gilt also

$$\forall t \in \mathbb{N} : f(t) = g(t) = t^2$$

und damit ist $f = g$. Wie wir sehen, ist die Implementation der Funktionen egal, wichtig sind die Ergebnisse. Die Funktionen sind in gewisser Weise Black-Boxes, also Maschinen, in die man nicht hineinschauen kann, deren Funktionsweise man nur durch Hineinstecken von Eingaben und Beobachtung der Ausgaben erforschen kann. Und wir sehen auch, dass die Namen der Argumente irrelevant sind, ob das übergebene Argument und mithin die interne Variablenbezeichnung in der Funktionsimplementation t, x oder n heißt, spielt keine Rolle. Man kann also nicht sagen, dass f von t abhängt, sicher ist lediglich, dass die *Funktion* f von einer Variablen abhängt und dass die *Terme* $f(t)$ von t und $f(x)$ von x „abhängen".

In der Analysis gibt es das gleiche Theater, weil zum Beispiel die Bezeichnung f_x für die partielle Ableitung einer Funktion voraussetzt, dass eine bestimmte „Dimension" mit x bezeichnet ist. Dummerweise verwendet man dann x aber auch gleich weiter als Variable, die die Stelle bezeichnet, an der man die Ableitung wissen möchte, etwa $f_x(x)$. Das führt spätestens dann zum Chaos, wenn man noch mit einem weiteren Wert auf der x-Achse, vielleicht x_0 oder x' arbeiten möchte. Ebenso ist bei Funktionen in mehreren Veränderlichen oft nicht genau zu erkennen, über welches Funktionsargument sich eigentlich ein Skalarprodukt oder eine Norm erstreckt. Oder es gibt Schwierigkeiten, weil bei der unbestimmten Integration der gleiche Bezeichner als Integrationsvariable und als Argument der entstehenden Stammfunktion eingesetzt wird:

$$F(x) = \int f(x)\,\mathrm{d}\,x$$

Besser wäre wohl

$$F = \int f(x)\,\mathrm{d}\,x$$

Wieder zurück zum O-Symbol. Wie können wir denn nun das Beispiel mit der quadratischen Funktion retten? Man könnte vielleicht den Kunstgriff ansetzen, und n als Identitätsfunktion deklarieren, also

$$\forall t \in \mathbb{N} : n(t) = t$$

Dann müsste man das Quadrieren der Funktion n so definieren

$$\forall t \in \mathbb{N} : n^2(t) = (n(t))^2$$

– was durchaus üblich ist. Damit könnte man arbeiten und würde auch nachweisen können, dass $n^2 \notin O(n)$, allerdings darf man dann n nicht die Problemgröße nennen, stattdessen wäre t die Problemgröße, zumindest in den beiden vorangegangenen Ausdrücken.

Man kann aber auf eine der vielfältigen Schreibweisen zurückgreifen, welche Terme in Funktionen konvertieren. Die Operation, mehrere Werte zu einer Funktion zusammenzufügen, ist gewissermaßen die Umkehrung zum Einsetzen eines Funktionsargumentes und Ablesen des Funktionswertes.

1. Die erste Variante erreicht uns aus der Welt der Programmiersprachen, Unterwelt `LISP`, in Form des berüchtigten λ-Operators. [2] Zum Beispiel bezeichnet

   $$f = \lambda_{x,y}(x^2 + y^2)$$

   eine Funktion, für die

   $$f(\xi, \eta) = \xi^2 + \eta^2$$

   gilt. Damit könnte man die falsche Notation $O(n^2)$ elegant in ein korrektes

   $$O(\lambda_n(n^2))$$

   verwandeln.

[2] `http://www.csee.umbc.edu/471/lectures/7/sld016.htm`

2. In der Mathematik gibt es mehrere praktisch gleichbedeutende Notationen für diese Aufgabe. Leider konnte sich jede Notation immer nur für Spezialfälle durchsetzen. Ihr kennt sicher
$$\left(\frac{1}{n}\right)_{n\in\mathbb{N}}$$
für die harmonische *Folge*, aber habt vermutlich noch nie die stetige aber auch visuelle Fortsetzung auf die Hyperbelfunktion
$$\left(\frac{1}{x}\right)_{x\in\mathbb{R}_+}$$
zu Gesicht bekommen.

3. Bei geschweiften Klammern, traut man sich zur Definition von *Mengen*, wie der Menge der rationalen Zahlen, auch an eine solche Notation:
$$\left\{\frac{p}{q} : p \in \mathbb{Z} \wedge q \in \mathbb{N}\right\}$$
Wenig gebräuchlich, aber durchaus genutzt, ist die daran angelehnte Notation
$$\left(\frac{1}{n} : n \in \mathbb{N}\right)$$
für die harmonische Folge. Auch hier traut sich wohl keiner an die Verallgemeinerung auf stetige Funktionen.

4. Für stetige Funktionen durchaus üblich und für diskrete Folgen wiederum ausgegrenzt ist folgende Schreibweise:
$$x \mapsto 1/x$$
Damit sähe die O-Notation so aus:
$$O(n \mapsto n^2)$$

Jo, wer nicht mutig genug ist, eine der obigen Methoden zu verwenden, der muss eben zu Fuß zwei Funktionen einführen, um wieder zum Beispiel zurückzukommen f und g, die eine linear, die andere quadratisch, also
$$\forall n \in \mathbb{N} : f(n) = n$$
und
$$\forall n \in \mathbb{N} : g(n) = n^2$$
dann ist n wieder unsere vertraute Problemgröße und
$$g \notin O(f)$$
der uns geläufige Zusammenhang. Und nachdem das Weltbild wieder gerade gerückt war, lebten alle glücklich und zufrieden bis an ihr Lebensende . . .

…doch nein, da braut sich schon wieder eine neue Gewitterwolke zusammen! Der Kampf zwischen n^2 und $O(n)$ ist noch nicht entschieden! Gebt mir noch einen Versuch, und ich beweise Euch endgültig, dass $g \in O(f)$ nichts als die Wahrheit ist (f und g wie eben vorgeschlagen)!

Dazu brauchen wir die vollständige Intuition (na Ihr wisst schon, wie es richtig heißt) und folgendes

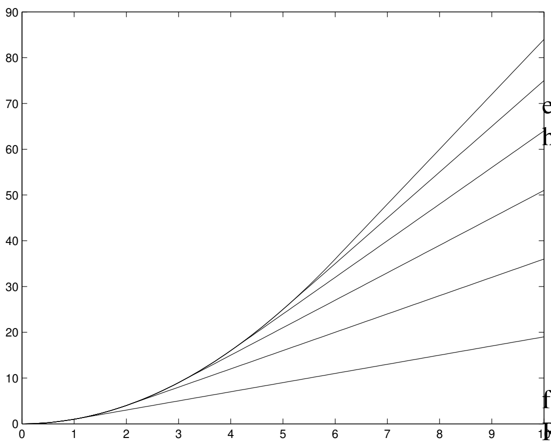
7 Lemma. Es seien f, g, h Funktionen, die von $\mathbb{N}$ nach $\mathbb{R}_+$ abbilden. Ferner gelte $f \in O(g)$ und $g \in O(h)$. Dann gilt auch $f \in O(h)$.

Beweis. Wenn die für $f \in O(g)$ benötigten Konstanten c_{fg} bzw. N_{fg} und die für $g \in O(h)$ benötigen Konstanten c_{gh} bzw. N_{gh} heißen, dann sind mögliche Konstanten für $f \in O(h)$ offensichtlich $c := c_{fg} \cdot c_{gh}$ und $N := \max\{N_{fg}, N_{gh}\}$. □

Nur als Hinweis: Mit Beendigung des Lemmas haben wir auch den Sichtbarkeitsbereich für die lokalen Bezeichner des Lemmas wie f, g und h verlassen und f und g sind wieder unsere beiden Funktionen, die lineare und die quadratische.

Ich werde mir jetzt eine Folge von Funktionen definieren. Die erste Funktion wird die Identitätsfunktion sein. Diese ist ihrerseits identisch zu f und damit auch in $O(f)$ enthalten. Alle weiteren Funktionen streben gegen die quadratische, bleiben aber trotzdem in $O(f)$! Jede Funktion g_j der Folge besteht bis zur Stelle j aus der quadratischen Funktion und geht danach mit dem gleichen Anstieg in eine lineare Funktion über:

$$g_j(t) := \begin{cases} t^2 & : t < j \\ (2t-j) \cdot j & : t \geq j \end{cases}$$



Nun zum

Beweis.

Induktionsanfang

$$g_1 = f \in O(f)$$

Induktionsschritt

Induktionsvoraussetzung

$$g_j \in O(f)$$

Induktionsbehauptung

$$g_{j+1} \in O(f)$$

Induktionsbeweis Da wir der Hilfe unseres Lemmas gewiss sind, reicht es nachzuweisen, dass sich g_{j+1} durch g_j beschränken lässt, sprich dass $g_{j+1} \in O(g_j)$ gilt.

Bis einschließlich zur Stelle j sind g_j und g_{j+1} identisch, bis dort kann man also den Faktor 1 ansetzen. Nach der Stelle j gehen die beiden Funktionen in $(2t-j) \cdot j$ bzw. $(2t-j-1) \cdot (j+1)$ über.

$$\begin{array}{rcl} 2t - j & \geq & 2t - j - 1 \\ (j+1)j \cdot (2t-j) & \geq & (j+1)j \cdot (2t-j-1) \\ \dfrac{j+1}{j} \cdot (2t-j) \cdot j & \geq & (2t-j-1) \cdot (j+1) \end{array}$$

Das heißt, dass wir ab der Stelle j den Faktor $\frac{j+1}{j}$ benötigen, welcher im Übrigen größer als 1 ist. Deswegen ist der Gesamtfaktor $\frac{j+1}{j}$. Damit ist gezeigt, dass

$$g_{j+1} \in O(g_j)$$

gilt, und mit der Induktionsvoraussetzung

$$g_j \in O(f)$$

erhalten wir über unser Lemma die Induktionsbehauptung

$$g_{j+1} \in O(f)$$

□

Da seid Ihr baff, was? Ich habe jetzt jedenfalls die Nase voll, und wenn jemand von Euch des Rätsels Lösung herausfindet, kann er diese an mich schicken!

Kleiner Tip: Versucht bei dem Beweis einmal ohne Induktion auszukommen.

PS: Wenn man es einmal selbst probiert, stellt man fest, dass es gar nicht so leicht ist, einen falschen Beweis überzeugend aufzubereiten. Ich beneide die Studenten im Grundstudium, deren Aufgaben ich manchmal kontrollierte, um diese Fähigkeit! :-)

PPS: „Falscher Beweis“ ist eigentlich ein Widerspruch in sich, besser sollte es wohl „gescheiterter Beweisversuch“ heißen.

See also: Simon Thompson: *Haskell: The craft of functional programming.* Chapter 19.1: Time and space behaviour / Complexity of functions