Grußwort des 1. Vorsitzenden

Heute komme ich gleich zum kritischen Punkt. Wenn Sie Physik studiert haben, erinnern Sie sich bestimmt an das Experiment, in dem in einem abgeschlossenen Glasgefäß z. B. ein Fluorchlorkohlenwasserstoff (FCKW) oder das PFAS aus einer Wärmepumpe sowohl flüssig als auch in einer darüberliegenden Dampfatmosphäre gasförmig vorliegt und erhitzt wird. Wenn der Druck im Gefäß richtig gewählt ist, kann man es einrichten, dass beim Erwärmen die Grenzlinie zwischen der Flüssigkeit und dem Gas auf der gleichen Höhe bleibt bis – beim kritischen Punkt – die Trennline plötzlich verschwindet. Davor befand sich unten Flüssigkeit und oben Gas, klar getrennt, und danach? Ist das Gas, oder Flüssigkeit, oder deren dialektische Aufhebung?

Bevor man den Phasenübergang am kritischen Punkt zu frei auf alle möglichen Unterschiede überträgt, sollte man sich klar machen, dass metaphorisches Denken fehleranfällig ist. So vorgewarnt stellt sich die Frage, ob es das Phänomen in der Mathematikdidaktik auch gibt. Ich selbst würde die Frage bejahen, sonst hätte ich sie nicht gestellt. Ein Beispiel, das mich in den letzten Monaten beschäftigt hat, ist die Beziehung von Beweisen und Programmen. Nun wird niemand erwarten, dass man in einem „Siehe-Beweis“ (einem „proof-without-words“) einen Algorithmus erkennt. Ebenso wenig dürfte man beim Anblick einer Programmzeile, in der eine Funktion auf ein Argument angewendet wird, ein Stück mittelalterlicher Logik sehen. Die Welten von Programmen und Beweisen scheinen ebenso grundverschieden wie die Aggregationsformen Flüssigkeit und Gas. Der Parameter, an dem man drehen muss, um zum kritischen Punkt zu kommen, ist der der Formalisierungsgrad. Wenn man Beweise formal korrekt aufschreibt, und wenn man Programme in einer hinreichend ausdrucksstarken Programmiersprache (an dieser Stelle sind C, Java und Python leider draußen, man denke an Racket, Haskell oder OCaml), dann verschwindet die Grenzlinie. Dies ist eine bemerkenswerte Erkenntnis, die eine lange Genese gebraucht hat. Vor 89 Jahren hat der Logiker Haskell Curry gesehen, dass die Regeln für Datentypen in einer abstrakten Programmiersprache (dem typisierten Lambda-Kalkül) und die Schlussregeln der intuitionistischen Aussagenlogik formal von gleicher Struktur sind. Später hat man gesehen, dass sich das präzise zu einem Isomorphismus von Beweisen und Programmen ausbauen lässt und sogar, dass sich auch die Prädikatenlogik erschließt, wenn das System der Datentypen hinreichend flexibel ist. Schon das ist eine beeindruckende Theorie und man mag sich fragen, was es bedeutet, dass Logiker und Informatiker unabhängig voneinander exakt die gleiche Theorie geschaffen haben. Lange gab es aber den Makel, dass das ganze Theoriegebäude nur für intuitionistische Varianten der Logik gilt, d. h. der Satz vom ausgeschlossenen Dritten, also auch Widerspruchsbeweise und damit erhebliche Teile der Beweise der klassischen Mathematik, waren jenseits der Reichweite der Theorie. Aber auch diese Grenze ist gefallen. Diesmal war es so, dass die Informatiker schon lange eine Kontrollstruktur erfunden hatten (für Kenner: Continuations), von der sich zeigte, dass sie genau das leistet, was man auf der Seite der Logik braucht, um die volle klassische Prädikatenlogik zu erschließen. Seitdem kann man also mit Fug und Recht festhalten, dass Beweise und Programme genau das gleiche sind, nur in unterschiedlicher Syntax hingeschrieben. Was für eine kopernikanische Kränkung! Beweise, die Träger des konzeptuellen Wissens der Mathematik, sind nichts anderes als schnöde Produkte von Hackern.

So weit taugt das Ganze um intellektuelle Langeweile zu vertreiben. Hat es weitergehende Bedeutung für die Didaktik? Man könnte geneigt sein, das zu verneinen: Die Temperatur in der Didaktik ist so niedrig, dass immer klar ist, ob man mit Flüssigkeit oder Gas kocht, ob man beweist oder programmiert. Trotzdem seien ein paar Argumente genannt, für die ich mich erwärmen kann: Da ist zunächst das Verständnis unseres eigenen Faches: Mathematik wird oft charakterisiert als „deduktiv-beweisende“ Wissenschaft. Warum nicht als „konstruktiv-algorithmische“ Wissenschaft, wie man mit gleichem Recht sagen könnte? Und: Logisches Denken und algorithmisches Denken sind nicht das Gleiche. Wo zwischen der Theorie, in der Identität herrscht, und der Praxis des Denkens und Argumentierens liegen die entscheidenden Unterschiede? Sollte es nicht eine didaktische Antwort darauf geben, einfach um mathematisches Denken besser zu verstehen?

Aus historisch-genetischer Perspektive ist es interessant, dass erst relativ spät (vor gut 20 Jahren) erkannt wurde, dass nicht nur die intuitionistische Logik, sondern auch die klassische Logik in das Bild „Program=Proof“ (wie eines der Bücher zum Thema heißt) passt. Die Erweiterung hat einen Preis: Was auf der Seite der Logik die fast selbstverständlich scheinende Aussage, dass $A \vee \neg A$ wahr ist, erfordert auf der Seite der Programme nicht-lokalen Kontrollfluss – eine Programmtechnik, die jeden Menschen mit begrenztem Arbeitsgedächtnis an Grenzen bringen kann. Liegt hier der tiefe Grund, warum Widerspruchsbeweise kognitiv anspruchsvoll sind? Auf Kalkülebene bedeutet der nicht-lokale Kontrollfluss, dass es nicht-lokale Beziehung gibt, die berücksichtigt werden müssen, wenn man die Programme ausführt. Man kann spekulieren, dass es deswegen besonders schwer ist, logische Argumentationsmethoden allein durch lokale Beobachtung des Sprachspiels (wie Sprachmodelle das tun) zu erlernen. Man kann aber auch in die andere Richtung denken: Small ist beautiful. Die Erweiterung auf die volle klassische Logik mag die Theorie zu einem beeindruckenden Gedankengebäude abrunden, aber besonders schön und stringent ist die Beziehung in der Tat, wenn man sich auf die intuitionistische Logik beschränkt, also konstruktive (nicht konstruktivistische!) Mathematik betreibt. Sollte man konstruktive Elementarmathematik genauer inspizieren? Brouwer hatte seinerzeit gegen die Autorität Hilberts keine Chance, aber vielleicht lohnt es sich, die unterschiedlichen Bildungsgehalte klassischer und konstruktiver Mathematik herauszuarbeiten und so vielleicht auch zu einem besseren Verständnis der Genese der Mathematik aus operationaler Sicht zu kommen.

Vielleicht kann die Erkenntnis, dass Programme und Beweise das gleiche sind, auch die Rivalität von schulischer Informatik und Mathematik überwinden helfen. Wer wie ich schon etwas länger in der Didaktik aktiv ist, weiß, dass früher die Beschäftigung mit Algorithmen einen ausgesprochen schlechten Ruf hatte. Wer dafür geworben hat, konnte sich schon mal den Vorwurf einhandeln, seine Ziele seien „verrückt“. In der Informatikdidaktik gab es lange Diskussionen um die Gewichtung von informationstechnischer Grundbildung, in der Programme verwendet werden, und den eigentlichen Informatikunterricht, in dem auch programmiert wird. Welche Einschätzung des Mathematikunterrichts die Anwendung des Isomorphismus zwischen Programmen und Beweisen auf diese Unterscheidung nahelegt, überlasse ich den Lesenden als Übungsaufgabe.

Nach diesem Ausflug in möglicherweise wenig bekanntes Terrain wünsche ich Ihnen nun eine gewinnbringende Lektüre des vorliegenden Heftes. Vielleicht ist auch der ein oder andere Artikel so interessant, dass Sie ihn zweimal lesen: Dann haben Sie mit Ihrem Lesealgorithmus bewiesen, dass $A \wedge A \equiv A$ – aber ich bin sicher, die inhaltliche Erkenntnis des Artikel wird höher sein als diese formal-logische. In jedem Fall aber beweist das Heft, dass Mathematikdidaktik eine vielfältige und interessante Wissenschaft ist.

Reinhard Oldenburg
(1. Vorsitzender der GDM)