In: Christian Kassung (Hrsg.): Die Unordnung der Dinge. Eine Wissens- und Mediengeschichte des Unfalls. Bielefeld: transcript, 2009, S. 325-353.

Unsichtbar wird der Fehler, wenn sich alle daran gewöhnt haben

Wolfgang Coy

»ACCIDENT, n. An inevitable occurrence due to the action of immutable natural laws.«
(Ambrose Bierce)

»GLEITKOMMAFEHLER, der [Subst]. Unvorhersehbares Ergebnis, das aus einer Reihe exakt definierter maschineller Rechenschritte folgt und im Widerspruch zu Gesetzen der Arithmetik steht.«
(Ambrose Bierce)

Alan M. Turings mathematisch-logische Maschine und Konrad Zuses Rechenmaschine des Ingenieurs

Rechnen gilt als recht mechanische Tätigkeit. Mit Hilfsmitteln wie Multiplikationstafeln, Logarithmentafeln, Schemata zur Matrizenmultiplikation oder doppelten Buchführung erfordert das Rechnen zwar Aufmerksamkeit, aber keine besondere Intuition. Auf der Suche nach einer allgemeingültigen Beschreibung des Begriffs der Berechenbarkeit dachte sich Alan M. Turing deshalb eine abstrakte mathematisch-logische Maschine aus, eine *paper machine*, mit der die Vorgänge des schriftlichen Rechnens präzise definierbar sind. Ein unbegrenzt langes, beschreibbares und lesbares Band, heute sagen wir Speicherband, sollte die Rechenkästchen eines Schulheftes darstellen, Zeile an Zeile hintereinander gereiht. Zahlen ließen sich auf dieses Band wie in ein Rechenheft schreiben, z. B. ziffernweise als ganze Zahlen beliebiger Länge, bei Bedarf auch mit Dezimalkomma, aber auch als rationale Zahlen in Form von Brüchen, deren Zähler durch Divisionszeichen vom Nenner getrennt wurden.

Auch das regelhafte Vorgehen zur Berechnung wurde als *Rechentafel* in solche Kästchen geschrieben. Hinzu kamen einige naheliegende Einschränkungen. Zwar war das Speicherband unbeschränkt, die Zahl der verwendeten Zeichen musste aber endlich sein, ebenso wie die Rechentafel, die die Vorschrift zum Rechnen enthielt.

Abbildung 1: Anordnung der ganzen Zahlen auf einem beidseitig unbegrenzten Zahlenstrahl.

-15 -14 -13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +10 +11 +12 +13 +14 +15

Turing gelang es damit, nicht nur die arithmetischen Rechenschritte der Addition, Subtraktion, Multiplikation und Division als exakte Vorschrift zu notieren, sondern auch viel kompliziertere Algorithmen darauf zurückzuführen. Als geschulter Logiker konnte er nachweisen, dass sogar Maschinen denkbar waren, die jede Rechenvorschrift, die neben den Eingabedaten auf dem Speicherband notiert wurde, automatisch zum Ergebnis führen konnte. Er nannte diese *paper machine* deshalb Universalmaschine.

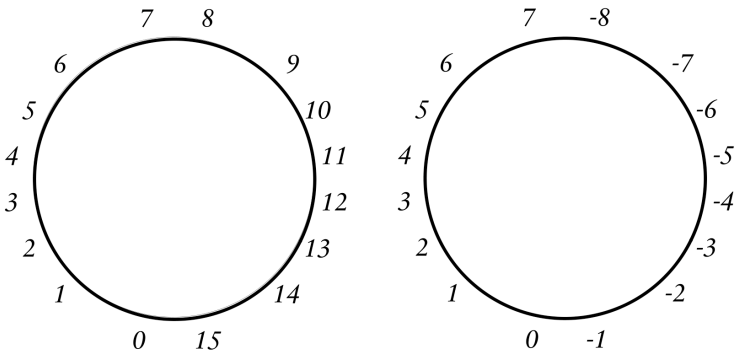
Aber das Ganze war nur eine Gedankenkonstruktion, bestenfalls eine Papiermaschine, und er hätte diese Überlegungen wohl nicht in den angesehenen »Proceedings« der Londoner Mathematical Society veröffentlichen dürfen, wenn er nicht quasi nebenbei ein von David Hilbert im Jahre 1900 aufgeworfenes Problem, das Entscheidungsproblem, unter den Annahmen seiner Maschine hätte lösen können.[1] Er zeigte nämlich, dass es entgegen Hilberts Erwartungen kein allgemeines Entscheidungsverfahren gibt, das in endlicher Zeit feststellt, ob eine bestimmte Größe x ein Element einer definierten Menge M ist. Turing untersuchte anstelle dieses umfassenden Problems die spezielle Frage, ob von einem gegebenen Rechenverfahren mit gegebenen Anfangsdaten vorhergesagt werden kann, ob die Rechnung überhaupt zu einem Ergebnis führen wird – oder eben nicht, weil sie niemals zu einem Ende kommt. Turings überraschende Antwort war: Das können wir im Allgemeinen nicht wissen. Das Halteproblem ist für beliebige Verfahren und Anfangsdaten unlösbar. Ist schon das Halteproblem unentscheidbar, so ist auch das umfassendere Entscheidungsproblem unlösbar. Dies gilt als mathematisch bemerkenswert. Dass Turing dazu einen umfassenden Begriff der Berechenbarkeit auf eine, freilich abstrakte Maschine zurückführte, war dagegen nachgeordnet. Mit dem Kriegsbeginn musste sich Turing mit konkreten Rechenmaschinen be-

1 | Vgl. Turing 1936/37a und 1936/37b.

fassen, und seine *paper machine* blieb erst einmal ein bloßes Gedankenexperiment.

Auch der Bauingenieur Konrad Zuse dachte an eine »universelle Rechenmaschine«, die unterschiedlichste »Rechenpläne« ausführen können sollte und vielleicht sogar zu einem »mechanischen Gehirn« würde, das Ketten von Rechenplänen aus Grundoperationen abarbeiten oder sogar selbst erstellen könnte.[2] Zuses Maschine war primär als Arbeitshilfe des rechnenden Ingenieurs gedacht. Er wollte Matrizen, Gleichungssysteme, Reihenrechnungen und Ähnliches mit Hilfe fester Rechenpläne berechnen. Im Fortgang der Arbeiten von 1935 bis 1945 wurde ihm jedoch immer klarer, wie nahe er Fragen der mathematischen Logik und von Anwendungen weit jenseits der Ingenieurmathematik kam. Dabei verlor er das Ziel freilich nicht aus den Augen, praktisch einsetzbare und bezahlbare Maschinen zu bauen. Dazu musste er technische Entscheidungen treffen und technische Kompromisse eingehen. Seine Maschinen basierten auf binären Zahlen, also Nullen und Einsen, und sie verarbeiteten Zahlen fester Länge. 22 Bit schien ihm ein brauchbarer Kompromiss für ein Speicherwort. Damit sind im Dezimalbereich bis zu 2^{21}, also rund 2 Millionen unterschiedliche Zahlen darstellbar, wobei ein Bit für das Vorzeichen reserviert bleibt. Die Darstellung entspricht einer modifizierten Modula-Arithmetik, deren Zahlen sich entlang eines Kreises darstellen lassen. Wird die maximal darstellbare Zahl Max einer Modula-Arithmetik überschritten, folgt darauf die minimal darstellbare Zahl Min.

Abbildung 2: Modula-16 Arithmetik und 3-bit-Zweierkomplement.



2 | Zuse 1984: 41.

In modernen Rechnern wird die Modula-Darstellung leicht verändert. Um die Unterscheidung von +0 und –0 zu vermeiden, die bei Abfragen nach der Gleichheit zweier Werte sonst eine Sonderbehandlung erfordert, wird meist +0 = –0 = 0 gesetzt, und der Überlauf, der selbstverständlich bei realen Rechnungen vermieden werden soll, geht von Max nach Min.

Heutige Maschinen haben typische Speicherwortlängen von 32 oder 64 Bit, aber es gibt auch Prozessoren für Steuerungen (*embedded systems*) mit 4, 8 oder 16 Bit. Auch die ersten PCs begannen mit 8 und 16 Bit. Werden dabei einige Nachkommastellen reserviert, z. B. zwei Stellen für einfache Finanzbuchhaltung, so schrumpft der darstellbare Bereich auf plus/minus Zwanzigtausend.

Tabelle 1: Zahlenräume unterschiedlicher Prozessoren.

Prozessor	Bit	Unterscheidbare Zahlen
Intel 8008	8	256
Intel 8086	16	65536
Zuse Z3	22	4194304
IBM S/360	32	4294967296
PowerPC	64	18446744073709551616
experimentell	128	340282366920938463463374607431768211456

Der sehr beschränkte Zahlenbereich bei kleinen Wortlängen war einer der Gründe, warum Ingenieure nicht gerne mit mechanischen Rechenmaschinen arbeiteten, sondern Rechenschieber oder Rechenstab bevorzugten – jedenfalls bis zur Erfindung des elektronischen Taschenrechners. Rechenschieber arbeiten halblogarithmisch: Multiplikationen lassen sich als Additionen darstellen. Wenn zwei gegeneinander verschiebbare Stäbe mit logarithmischer Teilung richtig eingestellt werden, lassen sich Multiplikationen auf eine Stelle genau ablesen. Professionelle Rechenschieber können noch mehr, und da, wo sie versagten, wurden umfangreiche gedruckte mathematische Tafelwerke eingesetzt.

Zuse übersetzte nun diese Ingenieurtradition in einen binären Ziffernrechner:

> Hat man sich einmal von der Tradition gelöst, so liegt es nahe, diesen Weg konsequent bis zum Ziel zu verfolgen. Der Übergang vom dezimalen zum binären Zahlensystem bedeutet nur einen Schritt. Wissenschaftler und Ingenieure arbeiten viel mit Zahlen verschiedener Größenordnung, wobei jedoch die Genauigkeit auf verhältnismäßig wenige Stellen beschränkt sein kann. Diese Bedingung erfüllt der Rechenstab in idealer Weise. Auf digitale Rechengeräte übertragen, bedeutet das die Verwendung logarithmischer anstelle normaler Werte. Meine Versuche in diese Richtung scheiterten jedoch an der Addition zweier Zahlen, die gegeben sind. Der konstruktive Aufwand, um die an sich bekannten Additionsalgorithmen darzustellen, ist zu hoch. Als brauchbare Zwischenlösung fand ich die *halblogarithmische* Form, bei der die Zahl folgendermaßen dargestellt wird:
>
> $$Y = 2^a \times b$$
>
> Hierbei ist a als Exponent von 2 der ganzzahlige Teil des Logarithmus von Y in Bezug auf die Basis 2. b ist ein Faktor, der zwischen eins und zwei liegt.
>
> Den Mathematikern war diese Schreibweise längst bekannt. Aber in der Rechenmaschinenindustrie wurde sie noch nicht verwendet, weil man dort den technisch-wissenschaftlichen Rechnungen ohnehin keine Bedeutung beimaß. Heute findet man diese Darstellung der Zahlen unter dem Namen gleitendes Komma in fast allen Computern.
>
> Die theoretisch einfache Lösung erfordert aber komplizierte Rechengeräte beziehungsweise umständliche arithmetische Operationen. Aus diesem Grund plante Babbage sein Gerät noch mit festem Komma, jedoch für fünfzig Dezimalstellen. Demgegenüber baute ich das Gerät Z1 mit sieben Binärstellen für den Exponenten und sechzehn Binärstellen für die Mantisse (a = Wert und b = Wert in obiger Formel). Der verhältnismäßig geringe Aufwand erfordert leider wesentlich kompliziertere Steuerungsorgane.[3]

Soweit Konrad Zuse in seiner Autobiographie. Das gleitende Komma, die Gleitkommazahlen also, wurde auch Bestandteil seiner umfassenderen Patentanmeldung Z391 von 1941.[4] Gleitkommazahlen überdecken einen viel größeren Bereich als die ganzen Zahlen – mit dem

3 | Zuse 1984: 182f.

4 | Der Patentantrag Z391 wurde vom Reichspatentgericht zum Bundespatentgericht durchgereicht und dort letztinstanzlich Mitte 1967 wegen ›mangelnder Erfindungshöhe‹ abgelehnt, wenngleich ›Neuheit und Fortschrittlichkeit‹ bestätigt wurden.

Preis erheblicher Lücken zwischen benachbarten Gleitkommazahlen. Bei 22 Bit langen Speicherworten lassen sich mit Zuses Definition Zahlen von 10^{-19} bis 10^{+19} auf 4–5 Stellen genau darstellen.[5] Da sich aber bei aller Ingeniosität der Zuse'schen halblogarithmischen Darstellung gerade so viele ganze Zahlen wie Gleitkommazahlen darstellen lassen, geht diese enorme Vergrößerung des Darstellungsbereichs auf Kosten der unterscheidbaren Gleitkommazahlen. Es folgt, dass zwischen benachbarten Gleitkommazahlen mit wachsendem Wert immer größere Lücken auftreten. Bei unbeschränkter Wortlänge lässt sich zwischen zwei Brüchen immer noch eine weitere Zahl unterbringen. Zwischen 0,4 und 0,5 liegen 0,41 oder 0,45. Bei ganzen Zahlen gilt das nicht – zwischen 4 und 5 liegt keine weitere ganze Zahl. Bei Gleitkommazahlen kann das ebenfalls vorkommen. Zwischen 1,23456 und 1,23457 lässt Zuses 22-bit-Arithmetik keinen weiteren Wert zu, da nicht mehr als 5 Nachkommastellen präzise darstellbar sind.

Gleitkommazahlen sind also nicht beliebig genau darstellbar. Der einzugebende wird wie der berechnete Wert einer Gleitkommazahl zugeordnet, die ihm nahekommt. Das Problem taucht bereits bei den alltäglichen Dezimalbrüchen auf. Der Bruch 1/7 kann nur durch eine unendliche Folge als Dezimalzahl dargestellt werden. Um 1/7 als Dezimalbruch darzustellen, geben wir deshalb eine Folge von Ziffern nach dem Komma an, die wir irgendwann abbrechen und eventuell runden. $1/7 = 0{,}14285714285714285714\ldots$, also brechen wir mit $1/7 \approx 0{,}14285$ nach der fünften Nachkommastelle ab, runden zu $1/7 \approx 0{,}14286$ auf oder treffen eine andere geeignete Entscheidung. Bei periodischen Dezimalbrüchen wird manchmal der periodische Teil überstrichen, aber es gibt natürlich auch nicht periodische Dezimalbrüche wie die Kreiszahl $\pi = 3{,}14159265358979323846\ldots$, wo Abbrechen oder Runden zwingend notwendig wird.

Reale Computer sind keine arithmetischen Maschinen

Zwei Gründe bewegten die Computerkonstrukteure dazu, Gleitkommanotationen einzusetzen. Zum einen war es die Erweiterung des darstellbaren Zahlenbereichs und zum anderen die höhere Rechengeschwindigkeit, wenn Gleitkommazahlen mit geeigneter Hardware berechnet wurden – wie es schon in Zuses Z3 geschah. Nicht alle Konstrukteure erachteten diesen *trade-off* als sinnvoll. Die Mathematiker um John v. Neumann, Herman Goldstine und Arthur W. Burks sahen

5 | Gespeichert wurden nur 15 Binärstellen für die Mantisse, da die Zahl immer so *normalisiert* wird, dass die erste Binärziffer eine 1 ist, die nicht gespeichert werden muss. Eine Stelle der Mantisse wird als Vorzeichen interpretiert.

ebenso wie Alan Turing große Probleme in der schlechten mathematischen Fassbarkeit der Gleitkommaarchitekturen: Wie sollte man sicherstellen, dass numerische Algorithmen mit solchen Zahldarstellungen korrekt arbeiten?

Andere Informatiker wie Wilkinson und Backus waren optimistischer. John W. Backus führte 1953 mit dem Speedcoding-System die Gleitkommaarithmetik für die IBM 701, den Defense Calculator der IBM ein. 1954 wurde dies Teil der Programmiersprache FORTRAN, eine Entscheidung, die bis heute für fast alle wissenschaftlichen Programmiersprachen gilt. Die Gleitkommazahlen bezeichnete Backus mit dem etwas unpassenden Name *real numbers*; heute werden sie meist mit *float* oder *double* bezeichnet.

So sicher, wie $1+1=2$ ist, sind Rechnungen im Computer nicht. Es gilt zwar auch in binärer Zahldarstellung $1+1=10$; ein Ergebnis, das ins Dezimale übersetzt die Zwei darstellt. Aber 1/5 ist als Dezimalbruch geschrieben 0,2, als Binärbruch dagegen eine unendlich lange Folge: 0,0011001100… Brüche aus zwei ganzen Zahlen werden rationale Zahlen genannt. Doch auch wenn sie als Bruch eine präzise kompakte Darstellung besitzen wie etwa 1/5 oder 2/3, müssen sie nicht unbedingt als Dezimalbruch oder Binärbruch ausdrückbar sein; ihre Darstellung kann dann unpräzise, nämlich unendlich lang werden. So ist die rationale Zahl 2/3 sowohl als Dezimalbruch 0,666… wie als Binärbruch 0,101010… nicht mit endlich vielen Ziffern anschreibbar. Wer mit solchen Zahlen rechnet, muss sich Rundungsstrategien überlegen und mit Rundungsfehlern umgehen.

Mit moderner Computerhardware geht man Kompromisse ein. Auf der Prozessorebene sind Computer keine mathematischen Maschinen wie Turing sie sich vorstellte. Um hohe Rechengeschwindigkeit und gleichförmig adressierbare Speicher zu realisieren, fließen zwei Kompromisse in die Hardware ein:

- feste Wortlängen für Speicher, Bus und Prozessor (typischerweise 4, 8, 16, 32 oder 64 Bit),
- Gleitkommaarithmetik, um den darstellbaren Zahlenbereich zu erweitern (zumeist aus Geschwindigkeitsgründen in Hardware realisiert).

Beide Entscheidungen haben dazu geführt, dass Computer nicht immer das tun, was Nutzer aus alltäglicher Erfahrung mit Rechenoperationen erwarten. Folge dieser Architekturentscheidung sind zwei immer wieder auftretende Fehlerarten: Überlauf, wenn Zahlen größer als die in einem Bitformat darstellbare Maximalgröße werden, und

Rundungsfehler bei Gleitkommazahlen. Freilich lässt sich eine so frei programmierbare Maschine wie der Computer per Software auch in seinen Hardwareeigenschaften modifizieren, aber das kostet Rechenzeit und/oder Speicherplatz und geschieht deshalb nur selten.

Fehler verursachen Unfälle – manche werden erst dadurch entdeckt

Unfälle lassen sich häufig auf harmlos erscheinende Anfangsfehler zurückführen, die sich dann durch eine ›Kette unglücklicher Ereignisse‹ erheblich auswirken können.

> Numerical software is central to our computerized society. It is used, for example, to design airplanes and bridges, to operate manufacturing lines, to control power plants and refineries, to analyze financial derivatives, to determine genomes, and to provide the understanding necessary for the treatment for cancer. Because of the high stakes involved, it is essential that the software be accurate, reliable, and robust.[6]

Ziel verantwortungsbewusster Technik ist es, denkbare Anfangsfehler auszuschließen oder in ihren Wirkungen zu begrenzen. So rechnen moderne Programmiersprachen mit Ausnahmefehlern (*exceptions*), die etwa durch Hardwarefehler oder durch unbedachte Programmierung (z. B. Division durch Null, Größenüberlauf) entstehen können und lassen eine Fehlerbehandlung (*exception handling*) zu, bevor der Prozessor in einen unkontrollierten Zustand gerät. Nur: Die Ausnahme muss vorher gedacht werden.

An einem teuren Beispiel lässt sich das Problem des Speicherüberlaufs, aber auch mangelnder Vorhersicht illustrieren. Am 4. Juni 1996 fand der Jungfernflug der Ariane 5-Rakete statt. Sie war mit vier Satelliten im Wert von etwa 850 Millionen US-Dollar bestückt, der Start alleine kostete etwa 250 Millionen Dollar. Ariane 5 sollte ohne weitere Tests in Dienst genommen werden. Zahlreiche Bauteile waren vom erfolgreichen Vorgängermodell Ariane 4 übernommen worden, auch große Teile der eingesetzten, über viele Starts bewährten Software. 30 Sekunden nach dem Abheben, in einer Höhe von 3.700 Metern änderten die Feststoff-Booster abrupt ihre Bahn und brachen in einer Explosion von der Hauptrakete ab. Der auslösende Fehler wurde in der Software des *Inertial Reference Systems* gefunden, dessen Hardware zwar

6 | Einarsson 2005: 3.

redundant ausgelegt war, aber in beiden Systemen die gleiche Software verwendete. Erst schaltete die Backup-Einheit ab, dann die primäre Einheit. In beiden Fällen versuchte der Rechner, eine 64-bit-Gleitkommazahl in ein Register für 16-bit-Ganzzahlen zu schreiben. Da der zu konvertierende Wert die maximal zulässige Größe überschritt, stellte der Rechner wegen des ausgelösten Fehlersignals seine Arbeit ein und übergab von da an Testsignale statt realer Messwerte an den On-Board-Computer, der die Rakete steuerte. Diese für den Flug bedeutungslosen Testsignale führten zu einer abrupten Richtungsänderung der Booster und dann des Haupttriebwerks. Als letzte Maßnahme startete die Rakete erfolgreich eine Selbstvernichtungssequenz.

Zwei aufeinander folgende Fehler führten zu einer Katastrophe, die einschließlich der Nachbesserungskosten für die späteren Modelle fast zwei Milliarden Dollar kostete:

- Es war nicht daran gedacht worden, dass die viel größere und schnellere Ariane 5 größere Werte verarbeiten würde als die Ariane 4. Dies führte zu einem Speicherüberlauf.
- Der erkannte Überlauffehler wurde nicht durch ein Hilfsprogramm *abgefangen*, sondern führte zum Abbruch der Prozessortätigkeit, die wiederum das Fehlverhalten der Raketensteuerung verursachte.

Ein dritter Entwurfsfehler lag tiefer. Die Ergebnisse des fehlerhaft programmierten Prozessors waren nach dem Abheben der Rakete überhaupt nicht mehr relevant, wurden aber noch weiter abgefragt:

> The error occurred in a part of the software that only performs alignment of the strap-down inertial platform. This software module computes meaningful results only before lift-off. As soon as the launcher lifts off, this function serves no purpose.[7]

Überlauffehler

Computerarithmetik, die mit endlich langen Speicherworten arbeitet, wie es aus Effizienzgründen bei allen heutigen Rechnern üblich ist, gehorcht nicht den Gesetzen der Arithmetik, wie wir sie aus der Schule kennen. Bei den dominierenden C-artigen Programmiersprachen (C, C++, Java, C#) ist die Zahlenwelt wie eingangs erläutert zyklisch angeordnet, wobei die Null unabhängig vom Vorzeichen zumeist einen

7 | Lions 1996.

einzigen Wert belegt und der maximal darstellbare Wert direkt an den minimal darstellbaren Wert grenzt.[8] Mit der hardwaremäßigen Darstellung negativer Zahlen im heute üblichen Zweierkomplement, gilt unabhängig von der Wortlänge für die Grenzwerte Max = Min–1 bzw. Min = Max+1. Da praktisch alle Rechner keine Kontrolle über die absolute Größe eines ganzzahligen Wertes übernehmen, können Addition und Multiplikation die Maximal- und Minimalwerte ohne Fehlermeldung überschreiten. Im Beispiel werden die benachbarten darstellbaren ganzzahligen Werte für 8-, 16-, 32- und 64-bit-Speicherworte durch wiederholte Addition von 1 erzeugt – in der Programmiersprache Java entspricht das den Datentypen (signed) Byte, Short, Integer und Long. Max sei die jeweils größte darstellbare Zahl, Min die kleinste.

Tabelle 2: 8-bit-Ganzzahlen (maximaler Wert 2^7–1, minimaler Wert 2^{-7}).

…	Max–2	Max–1	**Max**	Max+1	Max+2	Max+3	…
…	+125	+126	+127	–128	–127	–126	…
…	Min–3	Min–2	Min–1	**Min**	Min+1	Min+2	…

(Falsches) Addieren über den zulässigen Maximalwert hinaus springt also zum minimal darstellbaren Wert, so wie umgekehrt Subtrahieren von minimalen Werten zu maximalen springt. Computerprogramme bemerken diesen Fehler normalerweise nicht; Menschen rechnen nicht mit einem solchen Verhalten. Das Gleiche gilt für 16-, 32- und 64-bit-Ganzzahlen:

Tabelle 3: 16-bit-Ganzzahlen (maximaler Wert 2^{15}–1, minimaler Wert 2^{-15}).

…	Max–2	Max–1	**Max**	Max+1	Max+2	Max+3	…
…	+32765	+32766	+32767	–32768	–32767	–32766	…
…	Min–3	Min–2	Min–1	**Min**	Min+1	Min+2	…

Tabelle 4: 32-bit-Ganzzahlen (maximaler Wert 2^{31}–1, minimaler Wert 2^{-31}).

…	Max–1	**Max**	Max+1	Max+2	…
…	+2147483646	+2147483647	–2147483648	–2147483647	…
…	Min–2	Min–1	**Min**	Min+1	…

In der internen Darstellung fällt Max mit dem nicht korrekt darstellbaren Min–1 und Min mit dem nicht korrekt darstellbaren Max+1 zusammen. Ein solcher Überlauf der Grenzwerte muss nicht bemerkt

8 | Einige wenige Programmiersprachen gehen anders mit dem Überlaufproblem um. Ein Überlauf wird bemerkt und führt zur Unterbrechung des Programms, zum Setzen eines Flags, oder es werden die überlaufenden Werte durch den maximal bzw. minimal darstellbaren Wert ersetzt.

Tabelle 5: 64-bit-Ganzzahlen (maximaler Wert 2^{63}–1, minimaler Wert 2^{-63}).

…	Max–1	**Max**	Max+1	…
…	+9223372036854775806	+9223372036854775807	–9223372036854775808	…
…	Min–2	Min–1	**Min**	…

werden, er kann auch im Rechenvorgang wieder verschwinden. So erwarten wir aus der Sicht der Arithmetik, dass stets a+b–(a+b–1) = 1 gilt. Für 32-bit-Ganzzahlen in Java gilt das sogar, wenn a = Max gesetzt wird, obwohl die Operation Max+b undefiniert ist. Sei b = 5, so rechnet die Java Virtual Machine Max+5–Max–4 = 1. Werden die Zwischenergebnisse Max+5 und Max+4 ausgedruckt, so gilt freilich Max+5 = –2147483644 und Max+4 = –2147483645, was nun doch stark von der Alltagserwartung abweicht.

Die zyklische Anordnung der begrenzten Ganzzahlen führt gelegentlich zu völlig unerwarteten Phänomenen. Arithmetisch ist a+b nur dann gleich a–b, wenn b = 0 ist, ansonsten gilt immer $a+b \neq a-b$. Im Rechner gibt es allerdings eine weitere Zahl b, so dass für beliebige a die Gleichung a+b = a–b gilt, nämlich $b = -2^{n-1}$, sofern der Prozessor intern mit dem gebräuchlichen Zweierkomplement arbeitet. Nicht alle Prozessoren machen das, und nicht alle Nutzer wissen, was ihr Computer so macht. Da ein Überlauf beim Berechnen der beiden Werte c = a+b und d = a–b oft nicht bemerkt wird, kann eine nachfolgende Abfrage »c = d?« katastrophale Folgen für die Ausführung des Programmes haben.

Aus der zyklischen Anordnung der positiven und negativen Zahlen lassen sich weitere arithmetische Absonderlichkeiten herleiten, die wie das vorherige Beispiel von der konkreten Prozessorarchitektur abhängen. Mit dem Zweierkomplement kann es bei bestimmten zulässigen Werten von b und für beliebige positive Eingabewerte von a vorkommen, dass die eigentümliche Beziehung

$$a+b+c+1 = a-1$$

gilt. Arithmetisch ist dies natürlich Unsinn, da die Addition positiver Zahlen keine Subtraktion bewirken kann. Sei a = +100, b = +127 und c = +127. Dann gilt im 8-bit-Format mit Vorzeichen a+b+c+1 = –a–1, also zum Beispiel +100+127+127+1 = +99. Ähnliches lässt sich für alle Wortlängen beobachten. Werden beispielsweise bei 64-bit-Wörtern die Variablen b = +9223372036854775807 und c = +9223372036854775807 gesetzt, so folgt etwa bei Intel-Prozessoren unter Java mit der Eingabe a = +6789012 als Ergebnis +6789011, also a–1.

Rundungsfehler

Bei Gleitkommazahlen und bei anderen Datentypen mit einer festen Anzahl von Stellen hinter dem Komma müssen Werte zur Darstellung gerundet werden. Offensichtlich ist dies bei den schon erwähnten Brüchen, die nicht in endlicher Form als Binärbruch darstellbar sind, also z. B. 1/3, 1/5, 1/6 oder bei jedem Bruch, dessen Nenner nach vollständiger Kürzung eine andere Zahl als eine 2er-Potenz ergibt.

Geeignetes Runden kann eine hohe Kunst sein. Buchhaltern und Ingenieuren ist die Praxis des Rundens vertraut. Preise werden auf feste Einheiten zurückgeführt, typischerweise auf zwei Stellen hinter dem Komma gekürzt, eventuell ergänzt um eine dritte, vierte oder gar fünfte Stelle, um die Rundung zu präzisieren. Selbst wenn Bankkunden den Eindruck haben mögen, ihre Konten würden stets abgerundet, gibt es doch unterschiedliche Verfahren. Runden kann erhebliche Auswirkungen haben. Noch heute wird den Grünen bei der Schleswig-Holstein-Wahl im April 1992 ein Stimmenanteil von 5 % bescheinigt, wie es auch als vorläufiges Wahlergebnis im ARD-Wahlstudio mitgeteilt wurde.[9] Im amtlichen Endergebnis verloren die Grünen ihre vorläufigen Sitze, da sie die 5 %-Hürde mit 4,97 % verfehlten, einer Differenz von 398 Stimmen. Der SPD verschaffte dies eine absolute Mehrheit – und 4,97 % der Stimmen blieben ohne Einfluss. Die Hamburger FDP erzielte bei der Bürgerschaftswahl 2001 mit 43.214 Stimmen 5,08 % aller abgegebenen gültigen und nur 5,04 % der abgegeben Stimmen. Sie lag damit ganze 303 Stimmen über der gesetzlichen Sperrklausel.

Korrektes Runden ist sogar Teil der DIN-Norm 1333, die allerdings nur die Ausgabe von Zahlen festlegt, nicht aber den rechnerischen Umgang innerhalb von Computerprogrammen. Dabei soll die neu entstehende Zahl vom Ausgangswert nicht stärker als einen halben Stellenwert der Abbrechstelle abweichen. Eine weit verbreitete Rechenpraxis rundet Ziffern von 6 bis 9 um eine Einheit der nächst größeren Ziffer auf, also etwa 7,49 zu 7,5. Die Ziffern 0 bis 4 führen zum Abrunden. Da 0,5 aber genau in der Mitte zwischen 0 und 1 liegt, wird bei Gleichverteilung statistisch eher auf- als abgerundet. Dies kann zu erheblichen Fehlern führen, wenn viele gerundete Zahlen summiert werden. Eine ältere Regel verlangte deshalb, dass zur nächsten geraden Zahl hin auf- bzw. abgerundet wird. 1,5 würde also aufgerundet, 2,5 dagegen abgerundet. Doch was geschieht mit negativen Zahlen? Soll –2,45 nun zu –2,5 werden oder zu –2,4? Entweder wird nur in eine Richtung aufgerundet oder zur Null hin. Beides ist üblich, muss

9 | Vgl. Schröder 2008 sowie http://www.spiegel.de/politik/deutschland/0,1518,63421,00.html (5.9.2008).

aber vereinbart werden. Eine brutale Art der Rundung, die freilich besonders leicht zu programmieren ist, besteht im Abschneiden oder Abbrechen. Hier wird 2,19 eben zu 2,1.

Ein Beispiel für einen eklatanten Fehler durch nachlässiges Runden lieferten die Programmierer der Börse im kanadischen Vancouver. Die Börse gehört nicht zu den ganz großen, aber sie hat dennoch im Januar 1982 einen eigenen Börsenindex eingeführt, der sich von klassischen Indizes dadurch unterscheidet, dass er die Bewegungen aller rund 1.400 Aktienwerte erfasst. Eine derart präzise Berechnung war nur möglich, weil alle Einzelkäufe mit einem Computersystem erfasst wurden. Der Index startete mit einem Punktestand von 1.000.

Ein solcher, bislang unüblicher Index musste jedoch täglich mehrere tausend Buchungen berücksichtigen. Statt den Index in einem Minutenraster als Summe aller aktuellen Aktienwerte auszurechnen, beschlossen die Programmierer präziser vorzugehen und jede einzelne Transaktion zur Neuberechnung zu verwenden. Sie verwendeten die Eingabedaten in kanadischen Dollar mit bei Finanzdaten üblichen fünf Nachkommastellen, bei denen jedoch die letzten zwei Stellen für die Indexberechnung abgeschnitten wurden, also auf ein Zehntel Cent genau. Trotz der positiven Entwicklung vieler gehandelter Aktien verlor der Index sukzessive an Wert. Am 25. November 1983 betrug er nur noch 524 Punkte, woraufhin der Effekt des vieltausendfachen Abschneidens der täglichen Buchungswerte überdacht wurde. Jede Transaktion verursachte durch das Kürzen im Zehntel-Cent-Bereich einen Rundungsfehler zwischen 0,00000 und 0,00099 Dollar. Bei knapp 2.800 Buchungen am Tag akkumulierte sich dies zu einem Fehler von rund 1,25 Punkten täglich, die sich nach rund 480 Handelstagen zu einem rechnerischen Fehler von etwa 600 Punkten aufaddierten.

Der reale Fehler betrug 575 Punkte. Nach einer Neuberechnung eröffnete der Vancouver-Index am 28. November 1983 mit 1098,892 Punkten, einem zahlenmäßigen Gewinn von über 100 %, dem freilich keinerlei wirtschaftliche Tätigkeit zu Grunde lag. Hätte es 1983 bereits Exchange Traded Funds auf Börsenindizes gegeben, hätte dies in Kanada eine schwere Finanzkrise zur Folge haben können. Welche Fehlentscheidungen auf die zwei Jahre falscher Indexzahlen zurückzuführen sind, lässt sich nicht klären. So beschränkten sich die direkten Folgen wohl auf den Personalbestand der Börse Vancouver. Diese ging 1999 durch eine Fusion mit der Alberta Stock Exchange in der Canadian Venture Exchange auf.

Noch weit gravierendere Folgen hatte eine fehlerhafte Akkumulation von Gleitkommarundungen beim Einsatz von MIM-104 PATRI-

OT-Abwehrraketen im ersten Irakkrieg, der Operation Desert Storm.[10] Am 25. Februar 1991 wurde ein amerikanisches Feldlager in Dharan in Saudi-Arabien von einer irakischen SCUD-Rakete getroffen. 28 Soldaten starben. Die zum Schutz des Lagers installierte PATRIOT-Batterie suchte die Rakete an einer falsch berechneten Position und schoss die SCUD nicht ab. Der Abwehrcomputer schätzte an Hand der vom Radar gemessenen Geschwindigkeit der Angriffsrakete die erwartete Flugbahn, um den feindlichen Sprengkopf mit wenigstens einer von jeweils zwei abgefeuerten PATRIOTs zu treffen. Dieser Anfang der 1970er Jahre entworfene Computer verwendet 24-bit-Gleitkommazahlen, mit denen die Fluggeschwindigkeit in Meilen pro Sekunde auf vier Nachkommastellen genau und die Zeit der Alarmbereitschaft in Zehntelsekunden verarbeitet werden.

Die Messung in Zehntelsekunden führt zu massiven Rundungsfehlern, da 1/10 im Binärsystem ein unendlich langer Bruch ist, der wegen der 24-bit-Register zu 0,00011001100011001100011001 gerundet wird. Die fortwährende Addition von Zehntelsekunden akkumulierte die Rundungsfehler mit jeder Addition um etwa 0,0000001 Sekunden mit entsprechenden Folgen für die vorausberechnete Position der anfliegenden Rakete. Nach 8 Stunden sind dies 0,0025 Sekunden, wodurch aufgrund der hohen Geschwindigkeit die Position der SCUD bereits um 55 Meter verfehlt wird. Nach hundert Stunden verhindern 0,3433 Sekunden Verzögerung oder 687 Meter Fehlpositionierung die Ortung und jede weitere Verfolgung der SCUD.

Diese Problematik war bekannt – die PATRIOTs waren ursprünglich zur Ortung und zum Abschuss von viel langsameren Flugzeugen und Flugdrohnen entworfen worden. Die SCUD flog mit Mach 5, also etwa dreimal so schnell wie die Flugzeuge der ursprünglichen Einsatzplanung. Um dem entgegenzuwirken, wurde die Software streckenweise so umgestellt, dass die Zeitmessung in zwei verketteten 24-bit-Registern stattfand. Zudem wurde der Fehler zum Teil durch ein Unterprogramm in den umgerüsteten PATRIOTs korrigiert. Zur Verfolgung der angreifenden Rakete genügt eine relative Messung ab Entdeckung, so dass sich akkumulierte Fehler während der unmittelbaren

10 | Obwohl es einen umfangreichen Fehlerreport des U. S. General Accounting Office gibt, der die Softwareproblematik der PATRIOT beschreibt, bleiben die Erfolgsquoten des Raketenabwehrsystems ungewiss. Nach einer von der Wikipedia kolportierten Äußerung des U. S.-Präsidenten bei einem Besuch der Herstellerfirma Raytheon wurden 40 von 41 abgefeuerten SCUDs abgefangen. Nach Aussagen vor einem Untersuchungsausschuss des U. S.-Senats lag die Erfolgsrate vermutlich unter 10 %, möglicherweise sogar bei Null, d. h. kein einziger Einsatz einer PATRIOT führte zu einer vorbehaltlos bestätigten Vernichtung einer feindlichen Rakete. Die PATRIOTs der Operation Desert Storm waren möglicherweise eher eine Waffe der psychologischen Kriegsführung als eine reale Option.

Operation eigentlich ausgleichen sollten. Weil die Software aber nicht vollständig angepasst worden war, verarbeitete sie korrigierte Werte und nicht korrigierte Werte parallel, was die tödlichen Fehlberechnungen zur Folge hatte.[11]

Der vordergründig entscheidende Rundungsfehler, der eigentlich ein Umwandlungsfehler ist, weil Zehntelsekunden nicht exakt binär darstellbar sind, hängt mit schlechten frühen Entwurfsentscheidungen zusammen, fehlerhaften Hardware- und Software-Updates, fehlerhaftem Einsatz der Software unter erweiterten Bedingungen und wohl auch mit der verspäteten Fertigstellung einer neuen Programmversion, die diesen Fehler vermieden oder zumindest in seiner Wirkung reduziert hätte. Dieses Update, eine von sechs Auslieferungen während des Krieges, wurde erst am 26. Februar, einen Tag nach dem Unfall, bereitgestellt.

Additives Rauschen

Ein Grundproblem der Gleitkommaarithmetik sind die riesigen Unterschiede zwischen den darstellbaren Zahlen. Bei normalisierter 64-bit-Arithmetik reicht das von winzigen 10^{-323} bis zu gigantischen 10^{+308}, also über rund 631 Zehnerpotenzen. Diese sind freilich nur auf etwa 16 Stellen genau unterscheidbar.

Addiert man nun zu großen Zahlen kleine Zahlen, so wird diese Operation unsichtbar, weil die kleinen Zahlen erst in der siebzehnten Stelle der großen Zahlen wirksam werden. Notgedrungen gilt bei 32-bit-Gleitkommaarithmetik $12345678{,}0+x = 12345678{,}0$ für alle $x \leq 0{,}1$, und bei 64 Bit ist $1234567890{,}123456+x = 1234567890{,}123456$ für alle $x \leq 0{,}0000001$. Die gängige Informatikererklärung ist, dass der Bodensee nicht messbar steigt, wenn ein Glas Wein hineingeschüttet wird. Das gilt jedoch auch schon für eine Badewanne, wenn der Wasserstand mit einem Filzstift markiert wird. Die Badewanne mag bei einem Quadratmeter Oberfläche 100 l Wasser enthalten; werden 0,2 Liter hinzugefügt, so steigt der Wasserpegel um den Faktor 100,2/100, also um 0,2 %. Das sind weniger als ein Millimeter. Bei einer Flasche Wein wäre der Unterschied bereits sichtbar (3–4 mm), bei drei Flaschen dann ganz deutlich. Die wiederholte Addition sehr kleiner Größen kann sich also sehr wohl bemerkbar machen. Wird sie jedes Mal abgerundet, bleibt sie freilich unsichtbar. Nun sind tausendfache Additionen, ja sogar millionenfache Additionen bei heutigen Programmen nicht auszuschließen, so dass ein deutliches Fehlerpotenzial in der Addition kleiner Werte zu großen Werten lauert.

11 | Vgl. Skeel 1992.

Auslöschungsfehler

Eine verwandte Problematik entsteht bei der Subtraktion großer Zahlen. Da Gleitkommazahlen x mit wachsendem Absolutwert immer größere Lücken zu ihren Nachbarn aufweisen, kann eine solche Lücke größer als der Diminuend y der Subtraktion x–y werden. Damit bleibt die Subtraktion ergebnisneutral, und es gilt x = x–y. Die folgenden fünf 32-bit-Gleitkommazahlen sind benachbart, d. h. es liegen keine Werte dazwischen. Die Hexadezimaldarstellung auf der rechten Seite belegt diese Nachbarschaft.

Tabelle 6: 32-bit-Gleitkommazahlen.

Dezimal	Hexadezimal
1,999999761581421	0X1.FFFFFCP0
1,999999880790710	0X1.FFFFFEP0
2,000000000000000	0X1.000000P1
2,000000238418579	0X1.000002P1
2,000000476837158	0X1.000004P1

Für Subtraktionen, die gerundet zwischen diese darstellbaren Werte fallen, gilt dann beispielsweise 2,0–1,99999995 = 0, was natürlich arithmetisch fehlerhaft ist.

Auch die Mittelwertbildung verläuft atypisch, wenn die Körnigkeit der Gleitkommazahlen keinen dazwischenliegenden Wert mehr zulässt. So sind die mit 32 Bit codierten Gleitkommazahlen 17825810, 17825812, 17825814, 17825816, 17825818, 17825820 benachbart. Es gibt keine exakt darstellbaren Zahlen dazwischen und auch keine Zahlen mit signifikanten Nachkommastellen in diesem Bereich. Also gilt für den Mittelwert (17825812+17825814)/2 = 17825812, weil die arithmetisch korrekte Lösung 17825813 als 32-bit-Gleitkommazahl schlichtweg nicht existiert.

Die Gesetze der Arithmetik gelten ewig – aber nicht für die Gleitkommaarithmetik

Zwei Baugruppen spielen bei Computerprogrammen eine herausragende Rolle: die Rechenoperationen und die logische Steuerung des Ablaufs der einzelnen Rechenschritte. Der Standardprozess besteht in der stufenweisen Abarbeitung von Programmschritten als Rechensequenz. Gelegentlich wird eine solche Sequenz durch Fallunterscheidungen oder durch mehrfache Wiederholung einer Sequenz – beides abhängig von den berechneten Zwischenergebnissen – ergänzt. Die Bedingungen, die zu diesen Abweichungen führen, werden als logische Ausdrücke entschieden. Die einfachste Abfrage ist die Gleichheit zweier Werte, etwa k < 7 oder i = j. Für ganze Zahlen ist dies normalerweise unproblematisch – einzig die Antwort auf die Frage +0 = –0 muss wie eingangs erwähnt gesondert behandelt werden. Problematisch dagegen wird die Abfrage bei Gleitkommazahlen, da die gewohnten arithmetischen Gleichheiten nicht immer gelten.

Gängige Rechenregeln wie die Assoziativität oder Distributivität gelten nur eingeschränkt, wegen unentdeckter Überläufe können diese verletzt werden. Reihenfolge und Klammerung spielen auch bei arithmetisch gleichwertigen Rechnungen eine Rolle, als Beispiel sei die Gleichung (maxint–maxint)+1 ≠ (maxint+1)–maxint genannt. Ebenso ist a·(b+c) unter Umständen etwas anderes als a·b+a·c. Zum Beispiel rechnet Java in 32-bit-Darstellung:

$$9000\cdot(20{,}00005-5{,}1) = 134100{,}437500, \text{ aber}$$

$$9000\cdot 20{,}000050+9000\cdot(-5{,}1) = 134100{,}453125$$

Wir halten fest: Die Gesetze der Arithmetik gelten ewig. Sie gelten auch im Computer, aber nicht immer. Solche Abweichungen sind zumeist kaum zu entdecken. Sie sind das Fundament schweren Fehlverhaltens bis hin zu Unfällen.

Weitere Gleitkommafehler

Als Beschleuniger bei der Akkumulation von Rundungsfehlern wirken Schleifen. Das folgende Beispiel zeigt, dass schon nach wenigen Iterationen Gleitkommarechnungen *aus dem Ruder laufen* können. Mit h = 1/n wird nach n-maligem Durchlaufen der Schleife

```
for (int i=1; i<=n; i++){
  h=(float)(n+1)*h-1;
}
```

das Ergebnis x = n·h berechnet, also x = n·1/n (= 1). Dieses korrekte Ergebnis Eins taucht immer dann auf, wenn die Anzahl der Iterationen eine Zweierpotenz ist. Für alle anderen n wachsen die Abweichungen vom erwarteten korrekten Wert Eins, bis schließlich scheinbar wahllos Ergebnisse im ganzen Wertebereich von minus unendlich bis plus unendlich erzeugt werden.

Das Beispiel ist in dieser reduzierten Form leicht durchschaubar, aber es ist klar, dass deutlich unauffälligere Fehlerquellen ähnlicher Komplexität auch von geübten Programmierern nicht entdeckt werden. Dabei scheinen 34 Schleifendurchläufe äußerst harmlos. In der Praxis sind sehr viel umfangreichere Iterationen üblich.

Tabelle 7: Iteration einer Gleitkommarechnung.

1	+1,0000000000000000	18	+2613251644850176,0
2	+1,0000000000000000	19	+1,8749997558267904e+17
3	+1,0000019073486328	20	+1,8952130195370280e+19
4	+1,0000000000000000	21	+8,4093745517052300e+20
5	+1,0003089904785156	22	+6,5875304671052850e+22
6	+1,0080142021179200	23	–5,5206143847263180e+23
7	+1,0937500000000000	24	+2,7105053601424050e+26
8	+1,0000000000000000	25	–8,6848121124991490e+27
9	+48,683715820312500	26	+4,3506622167952400e+29
10	+563,17852783203120	27	–2,5199321265116980e+31
11	+22144,375000000000	28	+8,7350875133534380e+33
12	+854569,75000000000	29	–2,7271266449023390e+35
13	+40550648,000000000	30	+4,6582402242008350e+37
14	+2320045824,0000000	31	–Infinity
15	+60129542144,000000	32	+1,0000000000000000
16	+1,0000000000000000	33	–Infinity
17	+115813767970816,00	34	+Infinity

Das zweite Beispiel kommt ohne sichtbare rundungsfehlerträchtige Additionen aus. Verwendet werden allein Multiplikationen und die im Prozessor als Basisoperation angebotene Quadratwurzel, bei der wir eine optimale Rundungsstrategie erwarten können. Die zugebenermaßen etwas artifizielle Aufgabe besteht darin, beginnend mit einer Zahl x wiederholt die Quadratwurzel zu ziehen und anschließend aus dem Ergebnis durch wiederholte Quadrierung wieder den Originalwert herzustellen. Arithmetisch korrekt gilt $(\sqrt[k]{x})^k = x$ für positive x und beliebige k.

Das Programm ist hier, wie im vorherigen Beispiel in Java auf einem Intelprozessor programmiert, dessen Gleitkommaeinheit weitgehend dem IEEE-754-Standard folgt. Benutzt wurden 32-bit-Gleitkommazahlen *float*; bei *double* treten die gleichen Effekte bei etwa doppelter Iterationshäufigkeit auf.

Tabelle 8: Iteration einer Wurzelrechnung.

x	16 Iterationen	24 Iterationen	32 Iterationen
0,0	0,000000000000000	0,000000000000000	0,000000000000000
0,2	0,200000658631325	0,135302618145943	0,000000000000000
0,4	0,399281740188599	0,367835044860840	0,000000000000000
0,6	0,599422752857208	0,367835044860840	0,000000000000000
0,8	0,797248780727387	0,367835044860840	0,000000000000000
1,0	1,000000000000000	1,000000000000000	1,000000000000000
1,2	1,196805477142330	1,000000000000000	1,000000000000000
1,4	1,399189114570620	1,000000000000000	1,000000000000000
1,6	1,597844123840330	1,000000000000000	1,000000000000000
1,8	1,796627640724180	1,000000000000000	1,000000000000000
2,0	1,988544702529910	1,000000000000000	1,000000000000000

Während bei 16 Iterationen noch eine Annäherung auf ein oder zwei Nachkommastellen an das korrekte Ergebnis x sichtbar ist, sind bei 24 Iterationen die Ergebnisse ab x größer 1 konstant gleich eins, und bei 32 Iterationen werden nur noch die Ergebnisse $x = 0$ für $x < 1$ und $x = 1$ für alle größeren Werte erzeugt. Diese Fehler lassen sich mit anderen Rechnern, z. B. Taschenrechnern, reproduzieren. Die Ergebnisse sind oft ähnlich, wenngleich nicht identisch. Einige Prozessoren erzeugen auch ERROR-Werte. Kein Gleitkommaprozessor arbeitet bei größerer Iterationshäufigkeit auch nur annähernd korrekt.

Hardwarefehler

Geräte sind technische Artefakte und technische Artefakte können kaputt gehen. In der Frühzeit der Computer wurden bis in die 1950er Jahre viele Tests in die Programme eingebaut, um festzustellen, ob eine Rechenfolge *noch richtig* verlief. Eine schönes Beispiel ingenieursmäßiger Kontrolle des Rechenprozesses bestand darin, einen Lautsprecher an den Computer anzuschließen, so dass die Prozessoraktivitäten als akustisches Rauschen wahrnehmbar waren. Stille zeigte, dass der Rechner stand, periodische Töne wiesen auf Schleifen hin – die gefälligst nach einiger Zeit aufhören mussten. Die Phasen fehlerfreien Arbeitens hielten selten länger als eine Stunde an. Die aus diesen Laborrechnern abgeleiteten kommerziellen Großrechner besaßen Bänke kleiner Leuchten, die den Zustand der Bits in den Registerspeichern des Prozessors anzeigten. Blieb ein Birnchen über längere Zeit an, war dies ein Hinweis auf einen möglichen Hardwarefehler. Heute sind wir daran gewöhnt, dass die Prozessor- und Speicherhardware fehlerfrei

funktioniert. Einzig beim Einschalten werden einige Hardwaretests durchgeführt – oder wenn misstrauische Nutzer ein explizites Testprogramm starten. Der normale Nutzer geht von einer einwandfrei arbeitenden Hardware aus und schiebt ein erkennbares Fehlverhalten der Software zu – oder, wohlerzogen durch Gewohnheit, seinem eigenen Umgang mit dem Gerät.

Katastrophal können sich freilich Entwurfsfehler auswirken, die bereits in der Rechnerhardware fest verdrahtet sind. Heutige Prozessoren enthalten hunderte Millionen Schaltfunktionen. Derart komplexe Entwürfe können gar nicht fehlerfrei sein – auch wenn sich die Hersteller große Mühe geben, Fehler zu vermeiden oder zu entdecken und zu korrigieren. Als Intel 1993 den Pentium-Prozessor vorstellte, war dies ein neuer Meilenstein hochkomplexer Schaltungen. Er markierte einen Bruch mit der bisherigen 86er-Prozessorbaureihe und war doch rückwärtskompatibel, um ältere Software ausführen zu können. Unter anderem konnte der Pentium in einem Rechenschritt zwei Bits einer Division berechnen und nicht nur einen, wie bei den früheren 86er-Prozessoren.

Knapp anderthalb Jahre nach der Einführung wurde ein Schaltungsfehler in der Divisionshardware der Gleitkommaeinheit entdeckt, der eine Reihe von Maschinenbefehlen betraf, vor allem den Maschinenbefehl FDIV, der dem Fehler auch seinen Namen gab. Dieser »subtle flaw in the hardware divide unit« produzierte bei bestimmten Eingabedaten einen Fehler in der vierten Nachkommastelle – ein Problem, das sich zudem akkumulieren konnte und somit keineswegs auf die vierte Nachkommastelle beschränkt bleiben musste.[12] Die Rechengenauigkeit war in diesen Fällen mit 64-bit- oder 80-bit-Daten schlechter als mit 32 Bit. Nachdem der Fehler im Oktober 1994 in Compuserve- und anderen Internet-Diskussionsgruppen bekannt wurde, griffen ihn die Medien als Pentium-FDIV-Fehler auf. Intel wusste wohl schon länger davon und lieferte noch im selben Monat erste korrigierte Chip-Versionen aus.

Dies war kein einzelner Fehler, es wurden weitaus schlimmere Beispiele gefunden. So berechnete der Prozessor mit $x = 5505001{,}0$ und $y = 294911{,}0$ für den Ausdruck $z = (x/y)\cdot y - x$ den Wert $z = -256$ statt $z = 0$, wie man durch Kürzen sofort sieht. Als Fehlerursache wurden fünf falsche Hilfswerte in einer internen Hardwaretabelle identifiziert, also ein in Hardware gegossener Softwarefehler.

Die Relevanz der fehlerhaften Tabelle blieb umstritten. Während Intel davon sprach, dass der Fehler statistisch nur alle 27.000 Jahre aufträte, rechneten IBM-Forscher vor, dass er alle 24 Tage auftreten könne.

12 | Intel White Paper 1994: Section 1.

Nutzer, die den nachprüfbar fehlerhaften Prozessor umtauschen wollten, wurden zunächst von Intel brüsk abgewiesen.

> Elaborate surveys, analyses and empirical observation lead us to the overall conclusion that the flaw is of no concern to the vast majority of users of Pentium processor based systems.[13]

Zudem teilte Intel mit, die fehlerhaften Chips vorerst weiter zu verkaufen, bis die Lager geräumt seien.[14] Nach einem Entrüstungssturm in den Medien und ersten Androhungen einer *class action lawsuit* startete die Firma kurz vor Weihnachten dann doch eine Umtauschaktion, für die rund eine halbe Milliarde Dollar eingeplant wurde.

Der FDIV-Bug war nicht der erste Fehler in Intel-Prozessoren. Es gab dutzende, die allein die 386- oder 486-Prozessoren betrafen. Es war auch nicht der letzte. So gibt es einen Pentium II-Gleitkomma-Bug und viele weitere. Anderen Prozessorherstellern unterlaufen ähnliche Fehler. Der Pentium-FDIV-Bug war aber wohl der bislang teuerste, nicht zuletzt, weil der Prozessor mit der großen Werbekampagne Intel Inside verbunden war und weil Intel mehr als drei Viertel des Marktes für PC-Prozessoren beherrscht. Sichtbar wurden darüber hinaus einige allgemeinere Probleme des Computereinsatzes.

- Reale Fehler sind nicht nur Programmierfehler. Neben der Software kann auch die Hardware *falsch programmiert* sein. Intel wies mit einigem Recht darauf hin, dass andere Hardwarefehler gravierender seien, als dieser interne Prozessorfehler. Genannt werden Installationsfehler, Fehler in der Stromversorgung, falsche Rechnerkonfiguration, Hauptspeicher-, Platten- und Tastaturfehler, falsch angeschlossene Geräte und – besonderes pikant – weitere Prozessorfehler.[15] Doch gerade solche Probleme waren bis dahin gegenüber den Nutzern verharmlost worden, so dass der FDIV-Bug als besonders schwerwiegend erscheinen musste.

- Nutzer haben häufig keine Vorstellung davon, dass solche Fehler vorkommen und wie sie damit umgehen sollen – und die Industrie bekräftigt gern die Illusion, Computerfehler seien nahezu ausgeschlossen.

- Intel unterschätzte die Wirkung des 1994 gerade aufblühenden Internets, das schneller als die bisherigen Medien die kritischen

13 | Ebd.
14 | Vgl. Corcoran 1994.
15 | Vgl. Intel White Paper 1994: Section 5.

> Informationen genau an die Kunden verbreitete, die sich betroffen sahen. Da dies auch die Journalisten betraf, war das Medienecho enorm. In der Folge waren viele andere Nutzer verunsichert und haben Intel zu einer sehr teuren Umtauschaktion gezwungen.[16] Gleichzeitig haben viele Nutzer das Umtauschangebot von Intel nicht angenommen – warum auch immer.

»We take calculation in Excel very seriously«

Es gibt zwei Anwendungsbereiche, wo Computernutzer der Gleitkommaarithmetik ständig begegnen, ohne sich dessen wirklich bewusst zu sein. Alle Grafikprozessoren arbeiten mit Gleitkommazahlen, um Bildschirminhalte mit farbigen und schattierten Bildern und Animationen zu versehen. Diese Berechnungen unterliegen den Besonderheiten der Gleitkommaarithmetik, aber am Bildschirm fallen Fehler nur selten auf. Sie führen typischerweise zu kaum wahrnehmbaren visuellen Artefakten, die ansonsten keine weitere Bedeutung haben.

Tabellenkalkulationsprogramme, allen voran Microsoft Excel, sind das andere große Anwendungsfeld, wo einfache und höchst komplizierte Berechnungen mit Gleitkommaarithmetik ablaufen. Bei ganz einfachen Rechnungen fallen Fehler gelegentlich sogar unbedarften Nutzern auf. Im Internetforum wer-weiss-was.de schreibt Sabine:

> Irgendwie scheint mein Excel einen Rechenfehler zu haben. Wenn ich Felder mit den Werten 67,28+2,19+13,85 addiere, dann schreibt Excel ins Ergebnisfeld 83,31. Lt. Taschenrechner müsste es aber 83,32 sein. Gibt es auch hier eine Lösung des Problems?[17]

Das Ergebnis entsteht, wie sich schnell herausstellt, durch Rundung von Summanden, die mehr als zwei Nachkommastellen besitzen. Es ist offensichtlich: Die Nutzerin hat keine Ahnung, wie Excel rechnet, und sie verwendet zur Addition dreier kleiner Zahlen einen Taschenrechner. Schlaglichtartig wird das Problem der Programmiersprache Excel deutlich: Ungeschulte Nutzer mit geringen Vorkenntnissen müssen dem Programm trauen. Allerdings bemerken die Wenigsten überhaupt, ob ein Fehler auftritt. Und bei größeren Rechenaufgaben hilft dann auch der Taschenrechner nicht mehr beim Nachrechnen.

Das konkret aufgeworfene Problem erschließt sich durchaus mit der Hilfe-Funktion von Excel. Dort steht:

16 | Intel hatte für dieses Pentium-Modell eine »Lifetime Replacement Guarantee« ausgesprochen. Mit dem Umtausch erlosch diese.

17 | http://www.wer-weiss-was.de/theme156/article2790517.html, 9.9.2008.

> Unabhängig von der Anzahl der angezeigten Ziffern speichert Excel Zahlen mit einer Genauigkeit von bis zu 15 Stellen hinter dem Komma. Besteht eine Zahl aus mehr als 15 signifikanten Ziffern, wandelt Excel die übrigen Stellen in Null (0) um.[18]

Ob die Nutzerin Sabine freilich diese Hilfe gelesen und ob sie ihre Bedeutung verstanden hat, bleibt offen. Microsoft sieht die Perspektiven des Excel-Einsatzes bei ahnungslosen Nutzern:

> Die Anwender vertrauen seit je her auf Excel sobald es darum geht, auf Informationen zuzugreifen und Geschäftsdaten zu verarbeiten, zu analysieren und anderen zugänglich zu machen. Allerdings beschränkt sich der Nutzwert von Excel nicht nur auf Bereiche wie Buchführung und Finanzen. Denn auch Privatanwender hantieren täglich mit Zahlen, die analysiert, geordnet und anschaulich dargestellt werden müssen. Auf diesem Hintergedanken aufbauend wurde Excel 2002 mit dem Ziel entwickelt, den Zugriff auf Informationen um ein Vielfaches zu vereinfachen – unabhängig davon, ob es sich dabei um eine einfache Berechnung oder eine umfassende Sammlung von aufeinander aufbauenden Arbeitsblättern handelt.[19]

Excel wurde 1985 von Microsoft für den Macintosh als Konkurrenzprodukt für Visicalc und Lotus entwickelt, die damals beide nur für PC erhältlich waren. Es löste Microsofts Tabellenkalkulation Multiplan ab, die auf 8-bit-Betriebssystemen erfolgreich war, aber unter DOS gegenüber Lotus 1-2-3 im Verkauf zurückfiel. 1987 war Excel 2.05 das erste Spreadsheet-Programm für Windows. Während der Mac 32-bit-Prozessoren der Baureihe Motorola 68000 verwendete, lief die Windows-Version auf Intel-Prozessoren vom i80286 bis zu den heutigen 64-bit-Maschinen. Excel wurde für 64-bit-PowerPC-Prozessoren implementiert. Seit 1993 ist es Bestandteil des Office-Pakets.

Bei der Entwicklung von Excel werden offensichtlich mehrere Richtlinien verfolgt, die nicht immer konfliktfrei umgesetzt werden können.[20]

- Excel rechnet intern mit Gleitkommaarithmetik der höchsten im Prozessor vorgegebenen Genauigkeit.

18 | Microsoft Excel 2000 Hilfe.

19 | Microsoft Excel 2002 Update: Produktbeschreibung des Herstellers.

20 | In den *knowledge bases* für Excel lassen sich diese Entwicklerkonflikte gelegentlich nachvollziehen.

- Nutzer sehen das Ergebnis dieser *Rechenlogik*. Excel visualisiert die Rechenergebnisse mit Programmteilen seiner *Darstellungslogik* auf Bildschirm und Drucker und wandelt die Berechnungen bei Bedarf um.

- Tabellenkalkulationsprogramme vermitteln die Illusion, alle Zellen würden *gleichzeitig* berechnet. Tatsächlich gibt es eine Auswertungsreihenfolge, typischerweise diagonal von Zelle (0,0) ab verlaufend, die den Nutzern freilich nicht bewusst ist. Des Weiteren gibt es, wie bei allen Programmiersprachen, eine Auswertungsreihenfolge für funktionale Ausdrücke in Zellen. Auch diese bleibt den Nutzern verborgen.

- Normale Nutzer sollen nicht das Gefühl haben, *programmieren* zu müssen. Der Verbund der Zellformeln ist aber eine Programmiersprache, wenngleich eine sehr eigentümliche. Für fortgeschrittene Nutzer gibt es seit Version 4.0 (1991) zusätzlich eine Makrosprache Visual Basic for Applications, die auch für andere Programme des Office-Pakets genutzt werden kann.

- Excel-Programme sollen die Nutzer sozusagen bei der Stange halten, d. h. bei der Office-Suite. Deshalb wird Rückwärtskompatibilität als wichtiges Element der Entwicklung gesehen. Alte Programme sollen unter neuen Versionen laufen können. Dies stößt nach etwa zwanzig Entwicklungsjahren und zehn Versionssprüngen an z. T. fehlerträchtige Grenzen.[21]

Die Probleme aufgrund der Gleitkommaarithmetik werden zusätzlich durch kaum durchschaubare Optimierungen zu berechnender Ausdrücke und Auswertungsreihenfolgen um weitere Fehler ergänzt. Beispielsweise kann die Klammerung von Ausdrücken in einzelnen Zellen zu schwerwiegenden Fehlern führen. So gilt in Excel 2008 korrekterweise $a = (11/10-1)\cdot 10-1 = 0{,}00000000000000000000$, aber es bleibt unklar, warum für $(a) = ((11/10-1)\cdot 10-1) = 0{,}00000000000000008882$ berechnet wird. Bei $a = (12/11-1)\cdot 11-1$ gilt dagegen, ob mit oder ohne Klammer, das gleiche falsche Ergebnis

21 | Zum Schutz des prorietären Programms gehörte es, dass das Speicherformat .xls nicht offengelegt wurde. Dies ist seit 2008 geändert. Freilich wurden mit Excel 2007 sowieso neue Formate eingeführt. Auch VBA dient als *strategisches Element*, um Nutzer vom Wechsel zu anderen Tabellenkalkulationsprogrammen abzuhalten, aber die Rückwärtskompatibilität bereitet große Schwierigkeiten. Interessant ist, dass seit Excel 2008 for Mac VBA nicht mehr unter Mac OS X läuft und *heavy users* damit bei Excel 2004 for Mac stehen bleiben müssen oder alle VBA-Makros in AppleScript umschreiben müssen. Auch der Austausch mit den Windows-Versionen von Excel ist mit Excel 2008 nicht mehr möglich, sofern VBA verwendet wird.

$$a = (a) = -0{,}00000000000000008882.$$

Da $a = x^{2n}$ den gleichen Wert wie $b = (x^n)^2$ erzeugt, gilt $a = b$, aber seltsamerweise nicht $(a = b)$. Microsoft könnte dieses fehlerhafte Verhalten sicher erklären; da die Auswertungsstrategie den Nutzern jedoch nicht erläutert wird, legt sich ein Schleier des Geheimnisvollen über diese Rechnung. Wir zeigen in Tabelle 9 diese Fehlberechungen anhand des Exponenten $n = 7$ und einigen ausgewählten Werten von x.

Tabelle 9: Fehlrechnungen mit Excel.

x	$a = x^{2n}$	$b = (x^n)^2$	a–b	(a–b)
95	48767497911553000000000000	48767497911553000000000000	0	0
96	56467331235511400000000000	56467331235511400000000000	0	0
97	65283627746060800000000000	65283627746060800000000000	0	0
98	75364194147490200000000000	75364194147490200000000000	0	0
99	86874581276897800000000000	86874581276897800000000000	0	–1099511627776
100	100000000000000000000000000	100000000000000000000000000	0	0
101	114947421323762000000000000	114947421323762000000000000	0	0
102	131947876306287000000000000	131947876306287000000000000	0	0
103	151258972485511000000000000	151258972485511000000000000	0	–2199023255552
104	173167644760280000000000000	173167644760280000000000000	0	0
105	197993159943940000000000000	197993159943940000000000000	0	–2199023255552
106	226090395575442000000000000	226090395575442000000000000	0	0

Die Trennung von Rechenlogik und Darstellungslogik bleibt dem Nutzer verborgen. Dies führt zwangsläufig zu nicht interpretierbaren Fehlern. Wenn diese sichtbar auftreten, lassen die Microsoft-Entwickler die besorgten Nutzer im Dunkeln und flüchten in allgemeine Kompetenzbezeugungen: »We take calculation in Excel very seriously«, war die Antwort des Microsoft-Mitarbeiters David Gainer auf die für viele Nutzer überraschende Entdeckung, dass das gerade vorgestellte Excel 2007 als Ergebnis der Multiplikation 850·77,1 den Wert 1000000 vorschlug. Kurze Zeit später wurde klar, dass dies nicht der einzige Rechenfehler war, der durch die Darstellungslogik von Excel erzeugt wurde.

Excel ist in gewisser Weise mit Office zum weltweiten *de facto*-Standard in den Büros geworden. Im Laufe der Jahrzehnte ist es ein barockes Programm geworden, das für die Entwickler kaum noch überschaubar ist. Kein Konkurrenzprodukt kann auf Import und Export von .xls-Dateien verzichten. Leider werden auch viele Absonderlichkeiten von Excel nachgeahmt – wozu eben auch typische Fehler gehören.

Standards

Standardisierung war ein zentraler Schritt zur Vereinfachung der Softwareentwicklung. Die erste weit verbreitete standardisierte Rechnerarchitektur jenseits militärischer Anwendungen war das IBM S/360. Aus diesem Konzept entstand in der zweiten Hälfte des zwanzigsten Jahrhunderts praktisch die gesamte Mainframe-Architektur. Mikroprozessoren, die ab 1972 produziert wurden, waren von ihrer Herstellungsweise her standardisierte Produkte.

Die Gleitkommaarithmetik war und ist eine Spielwiese für Rechnerarchitekten. Durch die Konvergenz zu den offenen Standards des Institute for Electrical and Electronics Engineers besserte sich diese Situation langsam, speziell durch den IEEE-754-Standard von 1985 zur Beschreibung binärer Gleitkommaarithmetik, dessen revidierte Version IEEE 754r im Juni 2008 verabschiedet wurde. Die neueste Version lässt neben den 32-bit-, 64-bit- und 80-bit-Versionen binärer Gleitkommaarithmetik auch 16-bit- und 128-bit-Typen zu. Aus dem ergänzenden IEEE-854-Standard wurden Regeln für Dezimalgleitkommaarithmetik übernommen. Diese Dokumente beschreiben das Verhalten von Gleitkommaprozessoren mit nicht unumstrittenen Besonderheiten von Zahlwerten, die *infinity*, *negative zero* oder *NotANumber* heißen.

Am anderen Ende der Standardisierung stehen Regeln, die sich aufgrund von Marktpositionen bilden. Excel ist ein solcher proprietärer *Industriestandard*, der alle Konkurrenten zum Nachahmen ohne eigene Einflussnahme zwingen soll. Es fällt schwer, in solchen Standards Vorteile zu erkennen, die jenseits von kommerziellen Gewinnen der herstellenden Firma liegen.

Auch Java ist aus der Idee der Standardisierung heraus entstanden. Die Firma Sun, bei der die Programmiersprache Java 1995 entworfen wurde, hat sich mit einer offenen Standardisierung im strengen Sinne lange schwer getan, wenngleich sie ihre Schritte immer dokumentiert und freien Zugang zu ihren Programmen gewährt hat. Seit 2007 ist Java in einen offenen Standardisierungsprozess überführt worden.

Zur Ausführung von Java-Programmen wird zwischen der ausführenden Hardware und dem Java-Programm eine weitere Programmschicht, die Java Virtual Machine definiert. Die JVM ist ein Interpreterprogramm, das eine einheitliche virtuelle Hardwareschicht vorspiegelt, so dass Java-Programme keine (oder weniger) Rücksicht auf die reale Hardware nehmen müssen. »Compile once, run everywhere« ist der Werbespruch von Sun für dieses standardisierte Verhalten. Der Vorteil der Vereinheitlichung liegt auf der Hand. Fehler, die in der Definition von Java oder der JVM liegen, werden damit allerdings nicht beseitigt, sondern bloß verallgemeinert.

Standards haben den enormen Vorteil, dass sich die Dinge gleich verhalten – korrekt müssen die Rechenergebnisse damit noch lange nicht sein. Es bleibt die freilich schwache Hoffnung, dass Fehler, die häufiger, eben standardisiert, auftreten, eher erkannt und verhindert werden.

Ausblick: Andere Computer sind möglich

Strikte Hardware-Standards wie IEEE 794 für binäre Exponenten und IEEE 854 für dezimale Exponenten erlauben den erfolgreichen Einsatz von Gleitkommaarithmetik auf unterschiedlichen Rechnern, sofern Betriebssystem und Programmiersprachen diese standardisierte Hardware unterstützen. Zwei zentrale Fragen, die in allen Bereichen der Programmierung gelten, aber bei Anwendung der Gleitkommaarithmetik eine besondere Rolle spielen, werden häufig ignoriert:

- Wie können fehlerhafte Berechnungen erkannt und erklärt werden?
- Wer trägt die Verantwortung, um solche Fehler zu finden und zu korrigieren?

Es gibt Anwendungsbereiche wie etwa die Computergrafik, die gut mit der Gleitkommaarithmetik leben können. Moderne Grafikprozessoren sind ein sichtbares Ergebnis dieser Entwicklungen. Es gibt aber auch viele Anwendungsbereiche, in denen falsche, unsichtbare Zwischenergebnisse fatale Auswirkungen haben können. Tabellenkalkulationsprogramme sind ein solcher Anwendungsfall, bei dem Gleitkommazahlen selten nützen, aber große Fehlerpotenziale in sich tragen. Weitaus die meisten Nutzer derartiger Programme sind beim Umgang mit solchen Fehlerquellen überfordert. Im Rechenalltag sind ganzzahlige Darstellungen (auch mit einer festen Stellenzahl nach dem Komma) insbesondere bei modernen 64-bit-Rechnern vorzuziehen, sofern die Größenordungen der Rechnung abschätzbar sind und die Grenzwerte der Ganzzahldarstellung nicht überschreiten. 128-bit-Architekturen wären weitaus geeigneter – und sie sind produzierbar.

Es gibt Softwarelösungen, die mit gewissen Abstrichen bei der Rechengeschwindigkeit und mit etwas zusätzlichem Speicheraufwand präzise(re) Rechnungen erlauben. Soweit sie überhaupt vorhanden ist, wird eine Hardware-Unterstützung der für das kaufmännische Rechnen besser geeigneten BCD-Arithmetik wenig genutzt. Die einst von

IBM angebotene Hardware-Ergänzung für Intervallarithmetik ist mangels Nachfrage nicht mehr im Einsatz.[22]

Angesichts der immer weiter steigenden Rechnerleistung und Speicherkapazität stehen Software-Alternativen zur Verfügung, die eine klare Unterscheidung zwischen den Speicher- und den Rechenformaten vornehmen. Dazu gehören Zahldarstellungen beliebiger Genauigkeit (Arbitrary Precision) und Computeralgebrasysteme, die neben der numerischen Berechnung auch algebraische Umformungen beherrschen.

Die hier dargestellten Zustände sind eine Momentaufnahme. So wie es beschrieben wird, ist es. Andere Technik ist möglich, aber um sie einzuführen, muss den Nutzern erst einmal das vorhandene Risiko bewusst werden. Erwartungen an den sogenannten Fortschritt weisen noch immer in eine andere Richtung: schneller rechnen, mehr speichern. Um die latente und schwer zu kontrollierende Fehlerursache Gleitkommarechnung auszuschalten oder wenigstens zu mildern, muss der weitgehend irrationale Wunsch nach immer schnelleren Computern durch die Forderung nach zuverlässigeren Systemen abgelöst werden. Zur Zeit dominiert die blinde Hoffnung, Computer rechneten von Haus aus richtig und unfallträchtige Fehler seien *nur* Geräte- oder Programmierfehler. Die Problematik der standardisierten, aber fehlerbehafteten Rechnerarithmetik bleibt den meisten Nutzern verborgen – sie wird unsichtbar durch Gewöhnung.

Literatur

ANSI/IEEE Std 754-1985: Standard for Binary Floating-Point Arithmetic. The Institute of Electrical and Electronics Engineers, 1985.

ANSI/IEEE Std 854-1987: Standard for Radix-Independent Floating-Point Arithmetic. The Institute of Electrical and Electronics Engineers, 1987.

Corcoran, Elizabeth 1994: A Flaw Chips Away at Intel's Shiny Image. In: Washington Post, 2. Dez. 1994. S. A1.

DIN 1333: Abbrechen und Runden von Dezimalzahlen, Mai 1958, Neufassung 1970.

Einarsson, Bo (Hrsg.) 2005: Accuracy and Reliability in Scientific Computing. Philadelphia: SIAM Book.

Hammer, Rolf u. a. 1995: C++ Toolbox for Verified Computing I. Basic Numerical Problems: Theory, Algorithms, and Programs. Berlin u. a.: Springer-Verlag.

22 | Ich fürchte, dass die schwache Nachfrage vor allem auf ein mangelndes Problembewusstsein zurückzuführen ist.

Intel White Paper 1994: Statistical Analysis of Floating Point Flaw, 30. Nov. 1994.

Kahan, William 1983: Mathematics Written in Sand. In: Proc. Joint Statistical Meeting of the American Statistical Association, 1983. S. 12–26.

Lions, Jacques-Louis u. a.: 1996: Ariane 5 Flight 501 Failure. Report by the Inquiry Board, Paris, 19. Juli 1996. In: http://www.esrin.esa.it/htdocs/tidc/Press/Press96/ariane5rep.html, nicht mehr erreichbar.

Quinn, Kevin 1983: Ever Had Problems Rounding Off Figures? This Stock Exchange Has. In: The Wall Street Journal, 8. Nov. 1983. S. 37.

Pratt, Vaughan 2005: Anatomy of the Pentium Bug. Proc. TAPSOFT'95 (Aarhus), LNCS 915. Berlin u. a.: Springer.

Schröder, Valentin 2008: Politische Wahlen in Deutschland. In: http://www.Wahlen-in-Deutschland.de/, 3.9.2008.

Skeel, Robert 1992: Roundoff Error and the Patriot Missile. In: SIAM News, 25/4 (Juli 1992). S. 11.

Turing, Alan M. 1936/37a: On Computable Numbers, with an Application to the Entscheidungsproblem. In: Proceedings of the London Mathematical Society, Ser. 2, 42 (1936–37). S. 230–265.

Turing, Alan M. 1936/37b: On Computable Numbers, with an Application to the Entscheidungsproblem: A correction. In: Proceedings of the London Mathematical Society, Ser. 2, 42 (1936–37). S. 544–546.

Williams, Martyn 2000: Computer problems hit three nuclear plants in Japan. In: CNN.com News, 1. Jan. 2000, http://archives.cnn.com/2000/TECH/computing/01/03/japan.nukes.y2k.idg/, 1.9.2008.

Zuse, Konrad 1984: Der Computer – Mein Lebenswerk, Tagebuchnotiz vom 20.6.1937. Heidelberg u. a.: Springer-Verlag.

Abbildungen