Die Computerprogramme von Charles Babbage

Raúl Rojas

Babbages Rechenmaschinen

Charles Babbage (1791–1871) ist allen Informatikern gut bekannt. Er hat vor fast 200 Jahren die erste vollautomatische Rechenmaschine ersonnen, die nah zu heutigen Computern einzuordnen wäre. Mit der Freigabe von Digitalisaten des Babbage-Archivs durch das Londoner Wissenschaftsmuseum können wir heute am Heimrechner die Blaupausen der „Difference Engine“ (1819 von Babbage erdacht) und der „Analytical Engine“ (mit Entwürfen ab 1835) bestaunen. Dazu gehören aber auch 26 Programme für die Analytical Engine, die Babbage zwischen 1836 und 1841 aufsetzte. Es sind die ersten und ältesten Computerprogramme der Welt. So, wie der gesamten Entwurf der *Analytical Engine* im Alleingang erfolgte, so stammen auch diese Programme ausschließlich aus der Feder von Babbage. In diesem Aufsatz wollen wir anhand von Babbages Code die Programmierarchitektur der *Analytical Engine* enträtseln, d. h. wir möchten erschließen, welche Struktur die Analytische Maschine aus der Sichtweise des Programmierers aufgewiesen hat.

Babbages ursprüngliche Idee von 1819–1822 war die Automatisierung der Erstellung von Tabellen von Funktionen [1]. Beispielsweise können Polynomtabellen durch die sogenannte Differenzmethode berechnet werden. Will man beispielsweise die quadratische Funktion $n^2 + n$ für n zwischen 1 und 100 berechnen, so fängt man manuell an und errechnet die ersten drei Werte der Funktion (für $n = 1, 2, 3$) in einer Spalte. In der zweiten Spalte rechnet man die Differenzen der Einträge in der ersten Spalte. In der dritten die Differenzen der Werte in der zweiten Spalte usw. Irgendwann ist die Differenz konstant (in unserem Beispiel eine 2) (Tab. 1).

Tabelle 1

Berechnung der Differenzen

n	$n^2 + n$	1. Diff	2. Diff
1	**2**		
2	6	**4**	
3	12	6	**2**
4	20	8	2

Tabelle 2

Berechnung der Funktion $n^2 + n$ mit der Differenzmethode

n	$n^2 + n$	1. Diff	2. Diff
1	**2**		
2	6	**4**	
3	12	6	**2**
4	20	8	2

Jetzt kann man rückwärts vorgehen und die Tabelle allein durch Additionen erzeugen, indem die sukzessiven Differenzen von rechts nach links aufaddiert werden. Die Berechnung kann mit den drei Zahlen, die fett gedruckt sind, gestartet werden (Tab. 2).

Babbages Idee war, eine Kette von mechanischen Addierern so zu verschalten, dass die Ausgabe eines jeden Addierers als Eingabe für den nächsten dienen könnte. Im Fall der Funktion $n^2 + n$ bräuchte man drei in einer Kette geschaltete Ad-

DOI 10.1007/s00287-017-1018-5

Raúl Rojas
Freie Universität Berlin,
Department of Mathematics and Computer Science,
Arnimallee 7, 022, 14195 Berlin
E-Mail: rojas@inf.fu-berlin.de

Zusammenfassung
Der britische Mathematiker und Erfinder Charles Babbage hat von 1836–1841 bis zu 26 Programme für die nie zu Ende gebaute „Analytische Maschine“ niedergeschrieben. In diesem Beitrag leiten wir die Programmierarchitektur des mechanischen Computers aus dem von Babbage hinterlassenen Code ab.

dierer, wobei im oberen Beispiel der letzte davon immer dieselbe Zahl, d. h. eine Zwei, enthält. Diese Maschine wurde von Babbage „Difference Engine“ genannt und für seine Realisierung erhielt der englische Mathematiker eine Finanzierung der britischen Regierung [2].

Obwohl von der „Difference Engine“ ein kleiner Prototyp erstellt wurde, hat Babbage das Projekt aufgegeben, als ihm klar wurde, dass er eine mächtigere Maschine bauen könnte, die „Analytical Engine“. Diese sollte praktisch alles enthalten, was einen Computer heute auszeichnet [3]. Auf die Idee ist Babbage gekommen, als ihm auffiel, dass, wenn die Kette der Addierer der Difference Engine im Kreis angelegt werden würde, die Funktion selbst als n-te Differenz in die Maschine zurückgeführt werden könnte. Man hätte dann viel komplexere rekursive Funktionen als solche mit konstanter n-ter Differenz rechnen können. So gelangte Babbage relativ schnell zu einem neuen Design für eine Rechenmaschine mit folgenden Merkmalen [4]:

- Die Analytical Engine sollte mit Dezimalzahlen arbeiten.
- Ein Speicher für bis zu 1000 Dezimalzahlen zu 40 Ziffern wäre vorhanden.[1]
- Der vom Speicher getrennte Prozessor würde die vier Grundrechenarten ausführen können.
- Befehle, Adressen und Konstanten sollten über Lochkarten in die Maschine einfließen.
- Der bedingte Sprung sowie Schleifen wären im Code möglich.

Babbage hat jedoch die Komplexität der konstruktiven Aufgabe völlig unterschätzt. Während die Grundarchitektur klar war, waren die dazugehörigen mechanischen Aufbauten sehr aufwendig. So blieb die Analytical Engine letztendlich unvollendet – es existieren jedoch nicht weniger als 28 Entwürfe der gesamten Maschine (Abb. 1 zeigt den Plan Nummer 25) und dazu die bereits oben erwähnten Programme, die Babbage „Berechnungsnotation“ nannte. Diese Programme wollen wir besprechen.

Arithmetik mit Zahnrädern

Alle frühen mechanischen Rechenmaschinen haben mit einer internen Dezimaldarstellung gearbeitet. Es ist auch klar warum: Da ein dafür vorbereitetes Zahnrad zehn Positionen einnehmen kann, lässt sich damit eine Dezimalziffer durch die Drehung speichern. Eine Konversion (z. B. Dezimalbinär, wie heute) entfällt, und der Benutzer kann die Zwischenresultate direkt an den Zahnrädern der Maschine ablesen.

Die Addition mithilfe von Zahnrädern ist besonders einfach. Ist die Zahl 2 bereits im Zahnrad durch eine entsprechende Drehung gespeichert und will man eine 3 addieren, braucht man nur das Rad drei Positionen weiter zu drehen. Die Anzeige direkt über dem Zahnrad würde dann auf 5 weisen. Damit man mit 40 Dezimalstellen rechnen kann, muss ein Zahnrad für eine jede solche Dezimalstelle zur Verfügung gestellt werden und auch eine Vorrichtung für die Weitergabe des Übertrags, wenn ein Zahnrad eine komplette Drehung absolviert hat.

Babbage wollte in seiner Analytical Engine Dezimalzahlen in Stapeln von bis zu 40 Zahnrädern speichern. Es war keine Fließkommadarstellung vorgesehen – gerechnet werden sollte mit ganzen Zahlen. Die Maschine würde allerdings die Lage eines „virtuellen“ Kommas verwalten, sodass der Benutzer z. B. 20 Dezimalzahlen vor und 20 nach dem Komma verwenden könnte. Bei Additionen und Multiplikationen konnte durch das Resultat die notwendige Anzahl von Stellen automatisch verschoben werden („shifted“).

Allerdings ist ein Problem der Speicherung mit Zahnrädern, dass das Auslesen der Inhalte destruktiv erfolgt. Will man die in einem Zahnrad gespeicherte Zahl ablesen, wird diese zurückgedreht bis zur Anzeige Null, d. h. rückwärts in Bezug auf die ursprüngliche Speicherbewegung. Durch die Drehung können andere „am Bus“ angekoppelte Zahnräder in Bewegung versetzt werden. Zum Beispiel kann eine gespeicherte „5“ so fünf Bewegungspulse für weitere verschaltete Zahnräder

[1] Die Anzahl der Speicherworte und die Anzahl der Ziffern haben sich im Laufe der Jahre geändert.

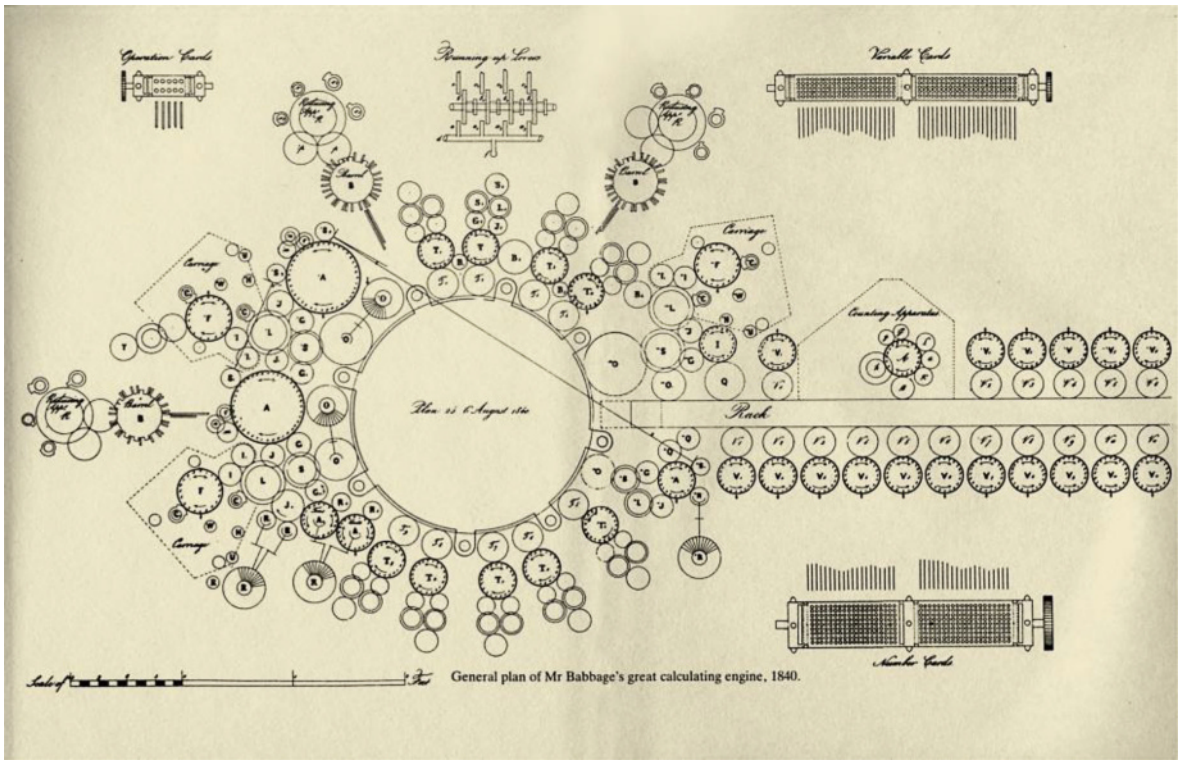

Abb. 1 Plan 25 der Analytischen Maschine. Aufsicht (1840, www.computerhistory.org)

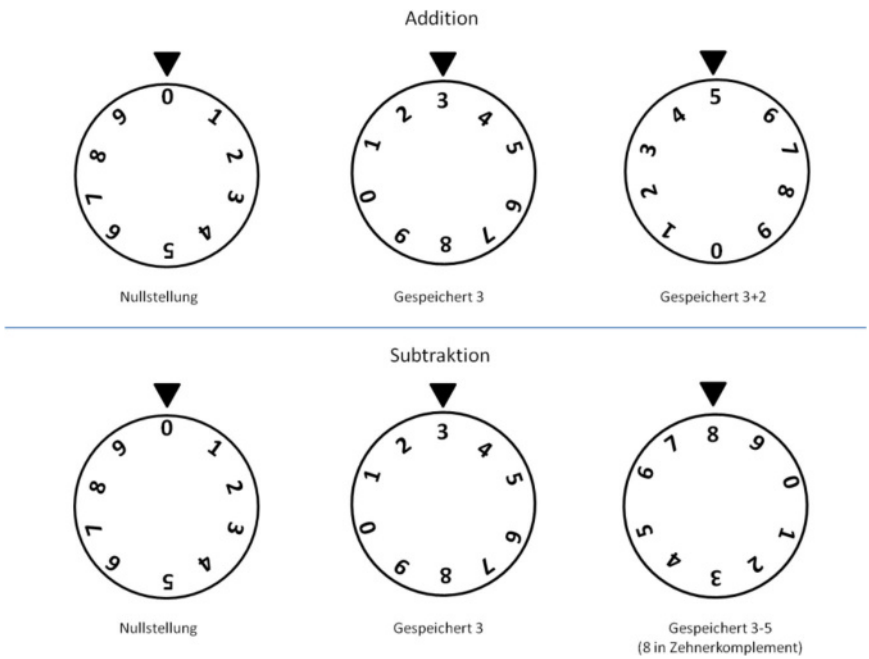

Abb. 2 Rechnen mit Zahnrädern

erzeugen. Der „destruktive" Zugriff auf den Speicher bedeutet, dass, falls die ursprüngliche Zahl nochmal verwendet werden soll, diese in einer Hilfsadresse zwischengespeichert werden muss.

Abbildung 2 zeigt Beispiele für die Addition bzw. Subtraktion mit Zahnrädern. So wie die Ziffern angeordnet sind, erfolgt eine Addition durch eine Rotation gegen den Uhrzeigersinn, eine Subtrak-

tion durch eine Rotation im Uhrzeigersinn. Bei dem Beispiel mit der Subtraktion 3 – 5 ist das Ergebnis negativ. Die Zehnerkomplementdarstellung (d. h. 8 statt – 2) ergibt sich direkt aus der Drehung der Speicherräder.

Beim Rechnen mit Zahnrädern muss man auch das Problem des Übertrags lösen. Babbage hat dafür den „carry look ahead" erfunden, eine Methode, um alle Übertragstellen parallel zu berechnen, sodass die Addition in konstanter Zeit ablaufen kann, unabhängig von der Anzahl der Dezimalstellen.

In der Analytical Engine gab es aber zwei Arten von Übertrag. Zahlen im Speicher wurden als ganze Zahlen gespeichert. Das Vorzeichen, negativ bzw. positiv, wurde in einem zusätzlichen Rad festgehalten. Bei Addition von zwei positiven Zahlen könnte dann der Additionsübertrag eintreten. Bei Subtraktion könnte dagegen der Subtraktionsübertrag notwendig sein. Beim „carry look ahead" werden die Additionen bzw. Subtraktionen in zwei Schritten berechnet: Zunächst wird die Addition/Subtraktion ziffernweise ausgeführt. Gleichzeitig werden die Überträge notiert (eine Spalte in der Dezimalnotation weiter nach links). Am Ende werden alle Additionsüberträge ziffernweise zur vorläufigen Summe addiert. Bei Subtraktion werden die Überträge ziffernweise vom vorläufigen Resultat abgezogen. Das Resultat wird in Zehnerkomplementdarstellung angegeben. Zwei Beispiele illustrieren die Technik.

Additionsübertrag	**Subtraktionsübertrag**
783 Argument 1	558 Argument 1
+ 558 Argument 2	– 783 Argument 2
231 Summe ohne Überträge	0875 Subtraktion ohne Überträge
+ 1110 positive Überträge	– 1100 negative Überträge
1341 Resultat	9775 Resultat (– 225 in Zehnerkomplement)

Überträge können sich über mehrere Spalten fortpflanzen (z. B. bei der Addition 9999 + 1). Eine „9" in der Summe ohne Überträge setzt den Übertrag weiter nach links fort. Die Technik des „carry look ahead" ist gut bekannt, deswegen gehe ich nicht ausführlich darauf ein.

Arithmetische Befehle

Die Programme im Babbage-Archiv beim Science Museum in London sind von L01 bis L027 durchnummeriert. Ein Programm für die Analytical Engine besteht, im Wesentlichen, aus einer Liste von Zuweisungen von Resultaten von arithmetischen Operationen mit zwei Argumenten. Speicherzellen werden mit „v" angegeben. Der Subindex benennt die Adresse im Speicher. Ein Befehl für die Addition der Zahlen in den Adressen 2 und 3 und Speicherung in Adresse 1 wäre:

$$v_1 = v_2 + v_3.$$

Die Maschine konnte die vier Grundrechenarten ausführen, d. h. die vier möglichen Operatoren sind + für Addition, – für Subtraktion, × für Multiplikation und ÷ für Division.

Allerdings verraten die Programme von Charles Babbage, dass es einige architektonische Einschränkungen gab: Es gab keine Register in der Maschine, in der Zwischenergebnisse für alle weiteren Operationen gespeichert werden konnten. Was im Prozessor berechnet wurde, musste in der Regel sofort als Resultat zurück zum Speicher gebracht werden (mit der Ausnahme von Additionen, die in „carry units" berechnet wurden).

Es war auch nicht möglich, die Adresse von Argumenten der Operationen für die Speicherung des Resultats zu verwenden. Zum Beispiel war $v_1 = v_1 + v_3$ aus unbekannten Gründen mechanisch nicht möglich. Stattdessen konnte man das Ergebnis nur in einer anderen Variablen unterschiedlich von v_1 und v_3 speichern, z. B. so:

$$v_7 = v_1 + v_3.$$

Durch das destruktive Lesen musste man im gleichen Zyklus manchmal mehrere Sachen tun, und zwar die Berechnung und die Zwischenspeicherung eines Argumentes:

$$v_7 = v_1 + v_3; \quad v_1 \to v_8; \quad v_8 \to v_1.$$

Oben ist angedeutet, dass die Addition stattfindet. Im gleichen Arbeitsgang werden die vom Zahnrad erzeugten Pulse beim Auslesen von v_1 in Adresse 8 zwischengespeichert. Nach der Addition können die in Adresse 8 gespeicherten Pulse zurück zu v_1 übertragen werden, womit v_1 am Ende nicht destruktiv gelesen würde. Man muss allerdings anmerken, dass, wenn der Inhalt einer Speicherzelle über den

Bus zum Prozessor und gleichzeitig an eine andere Speicherzelle übertragen wird (wie v_1 oben), letztere als ziffernweises Zehnerkomplement ankommt. In $v_1 \to v_8; v_8 \to v_1$ kommt der Inhalt von v_1 ziffernweise komplementiert in v_8 an. Bei der Rückübertragung $v_8 \to v_1$ hebt das neue Komplement die vorherige Komplementbildung auf und v_1 bleibt erhalten. Das war eine mechanische Einschränkung in der Analytical Engine, die allerdings keine negative Auswirkung hat.

Datenfluss-Architektur

Die größte Überraschung für den Programmierer ist die Trennung von Operatoren und Adressen. Nehmen wir die Befehlsfolge:

$$v_7 = v_1 + v_3; \quad v_1 \to v_8; \quad v_8 \to v_1$$
$$v_3 = v_7 \times v_1.$$

Das heißt, addiere Adresse 1 und 3 und speichere in Adresse 7, wobei der Inhalt von Adresse 1 erhalten bleibt, während Adresse 3 auf Null reduziert wird. Multipliziere danach Adresse 7 mit Adresse 1, speichere in Adresse 3 und lass Adresse 1 und 7 zu Null übergehen. Für die Analytical Engine bestünde dann das Programm einerseits aus der Folge der arithmetischen Operatoren, andererseits den Adressenangaben. In Babbages Notation für die Variablenadressen wurden diese Operationen wie folgt beschrieben:

Operationen	Adressen
+	$(v_1, v_8)(v_3, 0)(0, v_7)(v_8, v_1)$
×	$(v_7, 0)(v_1, 0)(0, v_3)$

Das Paar $(v_3, 0)$ bedeutet, dass Adresse 3 zum Prozessor weitergereicht wird und das Zahnrad am Ende bei Null bleibt. Das Paar (v_1, v_8) gibt an, dass Adresse 1 zum Prozessor weitergereicht wird, aber deren Inhalt in Adresse 8 zwischengespeichert wird. Das Paar $(0, v_7)$ bedeutet, dass ein Ergebnis aus dem Prozessor in Adresse 7 gespeichert wird. Das Paar (v_8, v_1) ist das Zurückschreiben vom alten Inhalt von Adresse 1 in die selbe Adresse, wobei Adresse 8 dann auf Null reduziert wird.

Babbage hat auch Lochkarten für die Übertragung von Zahlen in den Speicher vorgesehen, so wie „combinatorial cards“ für die bedingten Sprünge, aber mehr darüber später.

Die erste Spalte in der Tabelle oben würde die Kette der Operationen darstellen, die in einem Strang von Lochkarten vom Prozessor gelesen werden sollte. Die zweite Spalte benennt die Adressen in Paaren, die vom Speicher über einen getrennten Strang von Lochkarten gelesen wird. Es ist sehr wichtig zu verstehen, dass Prozessor und Speicher autonome Einheiten sind. Der Prozessor wird vom ersten Plusoperator in den Additionszustand versetzt und wartet auf zwei Argumente. Sobald diese eintreffen, wird die Addition ausgeführt und das Ergebnis zurück zum Speicher geschickt. Der Prozessor weiß eigentlich nicht, welche Adressen verarbeitet werden, er sieht nur die über den Bus ankommenden Zahlen. Der Speicher weiß nicht, welche Operation ausgeführt wurde, er sieht nur das Ergebnis. Eine solche Verschaltung ähnelt einer Datenflussarchitektur, wo die Operanden die Operation in jedem Knoten einer Berechnung auslösen. Das Abb. 3 zeigt deswegen die abstrakte Programmierarchitektur der Analytischen Maschine. Prozessor („Mill“) und Speicher sind getrennte Einheiten, jeder mit seinem Strang von Lochkarten. Dazu gibt es auch Lochkarten, um Zahlen in den Speicher einzulesen und spezielle

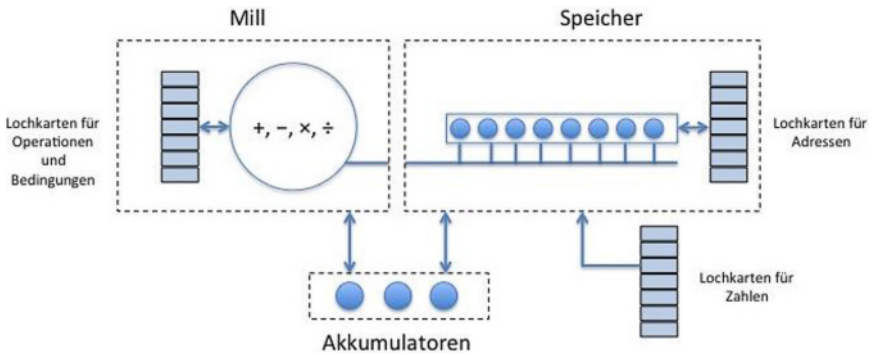

Abb. 3 Programmierarchitektur der Analytischen Maschine

Karten für Bedingungen und Sprünge (gemischt mit den Operationskarten). Drei Akkumulatoren (d. h. Zahnräder, die mit Übertrag rechnen können und Babbage „carry units“ nannte) gehören zum Prozessor, können aber auch unabhängig von ihm in Kombination mit dem Speicher arbeiten. Damit konnte man direkt im Speicher Zahlen addieren, während der Prozessor etwas anderes tat. Man konnte z. B. den Speicher als Difference Engine betreiben und Tabellen von Funktionen erzeugen, als ob diese Tabellen im Speicher wären und so direkt vom Prozessor für andere Aufgaben benutzt werden könnten.

Arten von Programmen

Die 26 Programme von Babbage im Londoner Archiv lassen sich in folgende Gruppen einteilen. Es gibt:

- zwölf Programme für Linearalgebra (vor allem Simultangleichungen),
- vier für die Auswertung von Polynommultiplikation und Division,
- zwei für die Auswertung von Rekursionen (durch „loop unrolling“),
- drei für die Simulation einer Differenzmaschine im Speicher,
- drei für die Auswertung von rekursiven astronomischen Formeln,
- eines zeigt die Benutzung von „combinatorial cards“.

Die erste Gruppe von Programmen ist die einfachste: Gegeben sind mehrere lineare Gleichungen (mit zwei, drei oder mehr Variablen) – die Gleichungen werden für eine Variable gelöst (mithilfe der Cramer’schen Regel) oder eine Variable wird durch Gauß-Elimination entfernt, womit jedes Mal eine lineare Gleichung entfällt. Alle Berechnungen können mit arithmetischen Operationen durchgeführt werden.

Dasselbe gilt für die Polynomdivision und Multiplikation. Die Polynomkoeffizienten werden in sukzessiven Adressen gespeichert und die Produkte bzw. Quotienten werden direkt, eins nach dem anderen, explizit berechnet (ohne indexierte Adressierung).

Tabelle 3 fasst die verschiedenen Programme für die Analytical Engine, das Datum ihrer Erstellung und den Anwendungsbereich zusammen. Die erste Spalte gibt die Nummerierung der Programme im Babbage-Archiv an. Die Reihenfolge ist chronologisch.

Wie man aus Tab. 3 erkennt, ist der am intensivsten studierte Anwendungsbereich die Lösung von Simultangleichungen. Auch heute spielen solche Lösungsverfahren im Scientific Computing eine große Rolle. Die Programme für die Polynommultiplikation und -division sind auch bedeutend, da Babbage damit zeigen wollte, dass die Analytische Maschine algebraische Berechnungen durchführen konnte. Heute würden wir unter algebraischer Berechnung eher die Manipulation von abstrakten Symbolen verstehen. Da aber die numerischen Werte der benutzten Polynomkonstanten (die Koeffizienten für jede Potenz) bis zur Ausführung unbekannt sind, kann man solche Programme auch als eine Art algebraische Manipulation verstehen. Im Babbage-Archiv ist das Programm 14 als Multiplikation von zwei Kosinusreihen annotiert. Das Programm im Archiv ist aber eine Polynommultiplikation. Vielleicht ist das Original verwechselt worden.

Die astronomischen Programme sind interessant. Es geht bei allen um die Berechnung des „elliptischen Radius“ für astronomische Beobachtungen mithilfe einer rekursiven Funktion. Babbage arbeitet hier mit keiner Schleife, sondern mit „loop unrolling“. Das bedeutet, dass die sukzessiven Werte der rekursiven Funktion einfach eine nach der anderen, ohne Schleife, berechnet werden.

In den Programmen 8 bis 10 arbeitet der Speicher mit drei Addiereinheiten zusammen, sodass der Speicher als eine Art Differenzmaschine parallel mit dem Prozessor betrieben werden kann. Die Idee von Babbage war, dass, während der Prozessor komplizierte Operationen wie Multiplikation und Division durchführt, der Speicher in Zusammenarbeit mit den „carry units“ in derselben Zeit Polynome berechnen kann. Im Programm 12 wird diese Möglichkeit illustriert. Die Operationen im Prozessor sind nur Multiplikationen und Divisionen (bei der Berechnung $(P/Q)F$). Im Speicher werden zwei Polynome P und Q direkt ausgewertet und die Ergebnisse an den Prozessor für die Division P/Q und anschließende Multiplikation des Resultats mit F geliefert. Die Befehle dazu kommen aus dem Speicher, über die Variablenkarten, die leider in dem Beispiel nicht angegeben sind.

Die Programme 19 bis 22, die letzten in der Liste, versuchen zu zeigen, wie die Variablenkar-

Tabelle 3

Die Programme im Babbage-Archiv. Anwendungsgebiete sind Linearalgebra (LA), Polynommultiplikation und -division, Berechnung von rekursiven Funktionen durch „loop unrolling" (REC), Erstellung von Tabellen von Polynomen wie mit einer Differenzmaschine (DIFF), astronomische Berechnungen. Programm 27 enthält keinen Code, nur die Speicherbelegung für mehrere Beispiele. Programm 3 befindet sich nicht im Archiv.

Programm Nummer	Datum	Gebiet	Kommentar
27	1836	Speicher	Speicherbelegung, ohne Operationen
26	8.1837	LA	Zwei Methoden für die Lösung von zwei Simultangleichungen
1	4.8.1837	LA	Zwei Simultangleichungen, Lösung für x und y. Cramersche Regel
2	14.8.1837	Polynome	Division eines Polynoms (Grad 6) durch ein lineares Polynom
4	16.8.1837	LA	Gaußreduktion von zwei Gleichung auf eine Gleichung in x bzw. y
5	16.8.1837	LA	Gaußreduktion von drei auf zwei Gleichungen
6	8.1837	LA	Wie Programm 5, kompakter
7	22.8.1837	LA	Gaußreduktion von vier auf drei Gleichungen
8	22.8.1837	DIFF	Tabelle einer quadratischen Funktion (Differenzmethode)
9	22.8.1837	DIFF	Tabelle einer kubischen Funktion (Differenzmethode)
10	23.8.1837	DIFF	Tabelle eines Polynoms vom Grad vier (Differenzmethode)
11	28.8.1837	REC	Berechnung einer einfachen Rekursion (abgerollt für $n = 0, 1, 2, \ldots$)
12	29.8.1837	REC-DIFF	Rekursion kombiniert mit Polynomberechnung durch Differenzmethode im Speicher
13	11.12.1837	Polynome	Produkt von zwei Polynomen (explizite Adressierung), partielles Resultat
14	7.9.1837	Polynome	Wie Prog. 13, aber vollständig für Grad 6
15	23.12.1837	LA	Kompakte Gaußreduktion von drei auf zwei Gleichungen, Lösung für x
16	6.1838	LA	Wie Prog. 15, mit „give off and retain" Notation
23	7.6.1838	Polynome	Polynom-Multiplikation mit „combinatorial cards" und impliziter Adressierung
17	1.10.1838–1841?	Astro, REC	Rekursion, elliptischer Radius (Notation mit geschweiften Klammern)
18	1.10.1838	Astro, REC	Wie Prog. 17, sauber
24	10.1838	REC	Rekursion
25	10.1838	Astro, REC	Rekursion, elliptischer Radius
19	31.7.1840	LA	Gaußreduktion von vier auf drei Gleichungen, mit Karten für Variablen
20	31.7.1840	LA	Fortsetzung von Prog. 19, mit Karten für Variablen
21	15.8.1840	LA	Wie Prog. 19, alternative Adressierung
22	15.8.1840	LA	Fortsetzung von Prog. 21, mit Karten für Variablen

ten zu codieren sind und in welcher Reihenfolge sie vorkommen. Die Notation ist nicht völlig konsistent, da z. B. das Endresultat manchmal als letztes Variablenpaar vorkommt, manchmal nicht. Es gibt auch Karten, die Informationen im Speicher von einer Adresse zu einer anderen bewegen, ohne Bezug zur aktuellen Operation. Wie viele solcher Speicheradressierungen in einem Arbeitsgang möglich waren, bleibt ungeklärt (bei heutigen Computern würden wir uns fragen, wie viel Information durch den Bus im selben Zyklus fließen kann).

Interessant in den Programmen 19 bis 22 ist, wie Babbage versucht, die Operationsschleifen sehr sau-

Tabelle 4

Protokoll der Ausführung der Berechnung von $(ab + cd)/(a + c)$. Die ersten vier Adressen enthalten die Konstanten a, b, c und d. Nach der Operation können die Variablen ihren Wert erhalten oder zu Null reduziert werden. Im ersten Fall wird eine Hilfsadresse verwendet (v_9 im Programm), wo das Komplement der Variable zwischengespeichert wird. Die Notation $0/a$ bedeutet, dass der ursprüngliche Wert wiederhergestellt wird. Das Programm führt fünf arithmetische Operationen aus.

Nummer	Op	Prozessor	v_1 a	v_2 b	v_3 c	v_4 d	v_5	v_6	v_7	v_8	v_9	v_{10}
1	×	ab	$0/a$	**0**			$\{ab\}$				com.a	
2	×	cd			$0/c$	**0**		$\{cd\}$			com.c	
3	+	$ab + cd$					**0**	**0**	$\{ab + cd\}$			
4	+	$a + c$	**0**		**0**					$\{a + c\}$		
5	÷	$\frac{ab+cd}{a+c} = x$							**0**	**0**	$\left\{\frac{ab+cd}{a+c}\right\}$	

ber zu definieren und die Variablenkarten für jede Schleife akribisch zu notieren. Letztendlich findet er, dass die wichtigste Schleife eine Pivot-Operation im Gauß-Verfahren ist. Gegeben die Gleichungen

$$aw + bx + cy + dz + e = 0$$
$$a'w + b'x + c'y + d'z + e' = 0$$

Es wird die erste Gleichung mit a' multipliziert, die zweite mit a und anschließend eine neue Gleichung durch Subtraktion der beiden skalierten Gleichungen ermittelt (es wird implizit angenommen, dass weder a noch a' Null sind). Die nicht-Null-Koeffizienten der neuen Gleichung sind $a'b - ab'$, $a'c - ac'$, $a'd - ad$, und $a'e - ae$. Für jeden Koeffizienten sind zwei Multiplikationen und eine Subtraktion notwendig. Die drei Operatoren $(\times, \times, -)$ definieren die notwendige Schleife für je zwei Gleichungen. Man erkennt hier deutlich, wie Babbage versucht, nicht mehr einfach die Pivotierung im Gauß-Verfahren ad-hoc zu programmieren, sondern bereits an den optimalen Code denkt, mit Schleifen und Oberschleifen. Da aber die Adressierung absolut und nicht indiziert ist, bleibt offen, wie man solche Operationsschleifen mit den Adressen in den Variablenkarten koordinieren könnte. Ich denke, Babbage hatte bis zuletzt keine konstruktive Lösung für das Problem.

Tabellen 4 und 5 geben eine Idee der Art von Code, der von Babbage geschrieben wurde. Die Programme an sich sind eher Protokolle der Ausführung der Befehle. Links werden die Einzelschritte gezählt und die jeweilige arithmetische Operation angezeigt. Rechts haben wir die Variablenadressen und ihre Belegung während des Programms. Arithmetische Operationen haben zwei Argumente. Das Resultat wird in geschweifter Klammer angezeigt. Wenn der Wert einer Variable erhalten werden soll, wird z. B. die Notation $0/a$ verwendet, was „give-off and retain“ bedeutet. Das Erhalten wird über den Umweg der Speicherung in einer Hilfsvariablen implementiert. Dorthin gelangt die Variable als Komplement und beim Zurückschreiben wird noch einmal das Komplement erzeugt, womit die Variable ihren ursprünglichen Wert behält. Wenn eine Variable nicht mehr notwendig ist, kann sie zu Null reduziert werden. Tabelle 5 gibt an, wie die Variablenkarten für das Programm in Tab. 4 vorbereitet werden könnten. Vor der Ausführung des Programms hätte man die Werte der Variablen aus den Lochkarten gelesen und diese im Speicher abgelegt.

Tabelle 5

Die Variablenkarten für die zugehörigen Operationen des Programms in Tab. 4

Operationen	Adressen
×	$(v_1, v_9)(v_2, 0)(0, v_5)(v_9, v_1)$
×	$(v_3, v_9)(v_4, 0)(0, v_6)(v_9, v_1)$
+	$(v_5, 0)(v_6, 0)(0, v_7)$
+	$(v_1, 0)(v_3, 0)(0, v_8)$
÷	$(v_7, 0)(v_8, 0)(0, v_9)$

Schleifen und bedingte Sprünge

Babbage hat für bedingte Sprünge und Schleifen die Möglichkeit vorgesehen, dass sowohl bei den Operationskarten als auch bei den Variablenkarten die normale Reihenfolge geändert werden könnte. Die Maschine könnte dann zu einer beliebigen Stelle in dem Strang der Karten „springen“. Diese Stellen könnten mit „index cards“ markiert werden. Der Sprung könnte durch eine Variable ausgelöst werden, wenn sie z. B. den Wert Null erreicht (durch einen Zähler, der nach unten zählt). Das wäre so etwas wie ein Befehl „IF $x = 0$ THEN GOTO index card“. Die Sprünge könnten sowohl vorwärts als auch rückwärts im Code erfolgen. Mit den Sprüngen rückwärts wäre es dann möglich, Schleifen zu implementieren.

Und tatsächlich, Programm 23 vom 7. Juni 1838 zeigt, wie die Polynommultiplikation mithilfe von „combinatorial cards“ hätte realisiert werden können. Babbage benutzt dafür zwei verzahnte Schleifen und, überraschenderweise, auch eine Art von indirekter Adressierung. Gegeben sind zwei Polynome

$$A + Bx + Cx^2 + \cdots$$
$$a + bx + cx^2 + \cdots$$

mit Koeffizienten $A, B, C, \ldots, Z$ und $a, b, c, \ldots, z$ (von der Nullten zur n-ten Potenz) muss man die korrespondierenden Koeffizienten miteinander multiplizieren. Nehmen wir an, dass wir die Koeffizienten in zwei Zeilen anordnen und dass wir den Koeffizienten von der Potenz x berechnen möchten. Der Koeffizient von x wäre $(Ab + Ba)$. Das kann man rechnen, wenn die Variablenkarten so verschoben werden, dass sowohl A und b als auch B und a sich gegenüberstehen, wie unten angedeutet. Danach muss man beide Stränge nach und nach durch den Lesekopf laufen lassen („advance“) (Abb. 4).

Das bedeutet, dass obwohl die Adressen der Variablen fest in den Lochkarten angegeben sind, durch Verschiebung derselben unterschiedliche Adressen paarweise angesprochen werden können. Das Paar (v_1, v_2) z. B. sollte man sich dann vorstellen als zwei getrennte Stränge von Adresskarten, die mitlaufen, aber auch relativ zueinander gegenlaufen können! Solche Bewegungen sollten durch die „combinatorial cards“ angestoßen werden. Eine mögliche Anwendung für solche Adressierung wäre z. B. ein ganzes Array von Zahlen zu addieren, indem immer die nächste Zahl im Array durch Verschiebung eines Strangs von Lochkarten als Argument gelesen werden könnte.

Nach Babbage sollten die „combinatorial cards“ auch das „Wiederholungsgerät“ steuern. In der Analytischen Maschine konnten Ketten von Operationen wiederholt werden (man konnte z. B. die Multiplikation siebenmal ausführen). Dafür wurde nur eine Operationskarte für Multiplikation und im Wiederholungsgerät eine Sieben eingestellt (wie, das wird in den vorliegenden Programmen nicht verraten).

Das Programm in Tab. 6 zeigt, wie die „combinatorial cards“ und Schleifen programmiert werden konnten. Es gibt eine interne Schleife und eine äußere Schleife. Die äußere Schleife berechnet die ersten drei Koeffizienten des Produkts. Die innere Schleife multipliziert die einzelnen Koeffizienten der Argumente und bewegt die Lochkarten mit den Adressen der Variablen in Bezug auf den Lesekopf. Die Adressen der Variablen im Speicher werden in dieser Tabelle nicht angezeigt.

Babbage benutzt im Programm zwei „Backup“-Mechanismen, bezeichnet mit „z“ und „y“. Backup z bringt die Operationskarten zurück auf den Start der Operatorenschleife $(\times, +, -)$. Backup y bringt die Operationskarten zurück zu der Stelle, wo der Index der kombinatorischen Karten („comb“) um

Anfangsposition

			a	b	c	d	...
...	C	B	**A**				

Start der Berechnung für *Ab*

←back		a	**b**	c	d	e	...
...	C	B	**A**				

Berechnung von Ba nach Verschiebung

→adv			**a**	b	c	d	...
→adv	D	C	**B**	A			

Abb. 4

Tabelle 6

Programm für die Berechnung der Polynommultiplikation mithilfe von „combinatorial cards“

								...D,C,B,A		a,b,c,d			
Coeff	No	OP	Resultat	Tab	Comb		Speicher	adv	back	adv	back	z	y
1	1	+	0+1	1	1								
	2	x	*Aa*										
	3	+	0+*Aa*										
	4	–	1-1		0							back z	back y
2	5	+	1+1	2	2								
	6	x	*Ab*						0		1		
	7	+	0+*Ab*										
	8	–	2-1		1								back y
	9	x	*Ba*					1		1			
	10	+	(0+*Ab*)+*Ba*										
	11	–	1-1		0							back z	back y
3	12	+	2+1	3	3								
	13	x	*Ac*						0		2		
	14	+	0+*Ac*										
	15	–	3-1		2								back y
	16	x	*Bb*					1		1			
	17	+	(0+*Ac*)+*Bb*										
	18	–	2-1		1								back y
	19	x	*Ca*					1		1			
	20	+	(0+*Ac*+*Bb*)+*Ca*										
	21	–	1-1		0							back z	back y

Eins erhöht wird, sodass die innere Schleife einmal, zweimal und dreimal widerholt wird. Babbage gibt an, dass der kombinatorische Index durch spezielle Karten jedes Mal erzeugt wird, es bleibt aber etwas unklar, wie das geschieht.

Für die Variablenkarten mit den Adressen der Koeffizienten … *D*, *C*, *B*, *A*, gibt es einen Vorwärts- und einen Rückwärtsmechanismus. Auch für die Variablenkarten mit den Adressen der Koeffizienten *a*, *b*, *c*, *d* gibt es beide Mechanismen; „advance“ heißt, dass die Variablenkarten sich eine Stelle nach rechts bewegen, „back“ dass die Variablenkarten sich eine Stelle nach links bewegen.

Fazit

Die Analyse der Programme von Charles Babbage macht deutlich, dass er der erste Programmierer der Welt war. Wie könnte es auch anders sein? Er hat die Analytische Maschine entworfen, die mit Recht als erster Computer der Welt hätte gelten können, wenn sie zu Ende gebaut worden wäre. Auf dem Papier ist Babbages Design universell. Die Überprüfung der Funktionsweise könnte man durch Programmierung der Maschine testen. Dafür wurden die oben besprochenen Programme geschrieben.

Die Analytische Maschine wies eine Art Datenflussarchitektur auf, bei der Prozessor und Speicher getrennte Lochkartenstränge verarbeiten. Die Operationskarten haben den Prozessor in Additions-, Subtraktions-, Multiplikations- bzw. Divisionsmodus versetzt. Die ankommenden Argumente haben dann die jeweilige Operation gestartet. Nach erfolgter Operation wurde der Prozessor intern auf Null zurückgeführt. Der Speicher musste das Resultat aufnehmen.

Drei „carry units“, die man sich als Akkumulatoren für wiederholte Additionen vorstellen kann, konnte man mit dem Prozessor oder mit dem Speicher einbinden. Im Prozessor haben sie geholfen, Addition und Subtraktion durchzuführen. Mit dem Speicher verbunden, könnten sie diesen in eine Differenzmaschine für die Berechnung von Funktionstabellen verwandeln. Die so geschaffene Differenzmaschine konnte mit dem Prozessor als Funktionaleinheit zusammenarbeiten.

Babbages Programme sind als Protokoll der Ausführung geschrieben, da es durch die Trennung der Operatoren von den Argumenten zu schwer gewesen wäre, die Reihenfolge der Operationen gedanklich zu verfolgen.

Sehr wichtig ist, dass Babbage die bedingten Sprünge im Programmcode und auch in den Lochkarten für die Operatoren vorgesehen hat. So ließen sich Sprünge und Schleifen realisieren. Mit den Sprüngen bei den Variablenlochkarten lässt sich eine gewisse Art von indirekter Adressierung implementieren (z. B. für indizierte Schleifen).

Und selbstverständlich: Ada Lovelace hat mit all dem hier besprochenen nichts zu tun gehabt [7]. Die Analytische Maschine war bereits entworfen worden, bevor Ada ihre Übersetzung von Luigi Menabreas Beschreibung [5] der Analytischen Maschine mit einigen Anmerkungen veröffentlichte (im Jahr 1843). Adas Übersetzung [6] erschien also volle sechs Jahre nach den ersten Programmen von Babbage und ganze fünf Jahre nach Programm 23, das eingebettete Schleifen und indirekte Adressierung mithilfe von „combinatorial cards“ verdeutlicht. Für die Analytische Maschine wurden komplexere Programme als die 26 hier referierten nie geschrieben.

Die Analytische Maschine sowie ihre Programmierung waren ausschließlich geistige Kinder von Charles Babbage. Die Maschine konnte nie zu Ende gebaut werden, sie wäre aber großartig gewesen, nach alldem, was wir in den Programmen von Babbage zwischen den Zeilen lesen können.

Literatur

1. Babbage C (1864) Passages from the Life of a Philosopher. Longman, London
2. Babbage C (1822) Observations on the Application of Machinery to the Computation of Mathematical Tables. In: Memoirs of the Astronomical Society, London
3. Babbage C (1982) On the Mathematical Powers of the Calculating Engine, unveröffentlichtes Manuskript 1837. In: Randell B (ed) The Origins of Digital Computers. Springer, Berlin
4. Bromley A (1982) Charles Babbage's Analytical Engine 1838. Ann Hist Comp 4(3): 196–217
5. Menabrea LF (1843) Notions sur la machine analytique de M. Charles Babbage, Bibliothèque Universelle de Genève, Oktober, N. 82, 1842. Sci Mem 3:666–731
6. Menabrea LF (1843) Sketch of The Analytical Engine Invented by Charles Babbage
7. Stein D (1987) Ada – A Life and a Legacy. MIT Press