Programming by Example

Überlegungen zu Grundlagen einer Didaktik der Tabellenkalkulation

von

Michael Gieding, Heidelberg

Zusammenfassung: Hinsichtlich der Nutzung von Tabellenkalkulationssystemen im Mathematikunterricht beschränkte sich die mathematikdidaktische Forschung bisher vor allem auf die Untersuchung von Fallbeispielen. Wissenschaftliche Forschung sollte jedoch über eine Beispielsammlung hinausgehen, von speziellen Beispielen abstrahieren, grundlegende Prinzipien aufzeigen und allgemeine Fragen beantworten können.

Ein Ansatz für eine allgemeinere Sicht auf die Nutzung von Tabellenkalkulationssystemen ergibt sich daraus, dass es sich bei dieser Nutzung um eine spezifische Art des Programmierens handelt. Der vorliegende Artikel untersucht diese spezielle Art des Programmierens. Insbesondere werden dabei die Beziehungen zum funktionalen und zum imperativen Programmierparadigma aufgezeigt.

Summary: Regarding the use of spreadsheet systems in mathematics instruction the mathematics-didactical research was limited so far particularly to the investigation of case examples. Scientific research should go out however over an example collection, should abstract of special examples, show basic principles and answer general questions.

An extension for a more general visibility on the topic arises from the fact that using spreadsheet systems is a special kind of programming. The article investigates this special kind of programming. Especially the relations are shown to the functional and to the imperative programming paradigm.

1 Tabellenkalkulationssysteme und mathematikdidaktische Forschung

1.1 25 Jahre Tabellenkalkulation

Die Idee ist bereits ein viertel Jahrhundert alt. Sie stammt aus dem Jahr 1978 von Daniel Bricklin. Zusammen mit Robert Frankston setzte er seine Idee 1979 in die Tat um. Heraus kam die Mutter aller Tabellenkalkulationssysteme: VisiCalc.
Heute, im Jahre 2003, ist Mutter VisiCalc in Rente. Die in ihr verwirklichten genialen Bricklin'schen Ideen hat sie an ihre Kinder und

Kindeskinder weitergegeben. Stellvertretend wird in den vorliegenden Ausführungen Excel verwendet. Die veranschaulichten grundlegenden Ideen gelten für alle modernen Tabellenkalkulationssysteme (TKS).

1.2 Tabellenkalkulation in der allgemeinbildenden Schule

Der Entwicklung der Tabellenkalkulationssysteme zu einer bedeutenden Standardsoftware musste zwangsläufig bei den Überlegungen zur Einbeziehung von Elementen der Informatik und Computernutzung in die schulische Allgemeinbildung berücksichtigt werden.
Wie bei anderen Computeranwendungen auch, lassen sich die folgenden Aspekte der Einbeziehung von TKS in den Unterricht allgemeinbildender Schulen kennzeichnen:

- TKS als Werkzeug

Wie Taschenrechner, Formelsammlungen, etc. können TKS als „Werkzeug“ bei der Lösung mathematisch, naturwissenschaftlicher Probleme einsetzt werden.
Auf etwa Mitte der 80-ger Jahre sind diesbezügliche erste Überlegungen der Nutzung von TKS anzusetzen. Stellvertretend sei der Einsatz von TKS bei Zinseszinsproblemen genannt.

- TKS als Gegenstand

Wegen ihrer Bedeutung als Standardsoftware sind TKS integraler Bestandteil einer informationstechnischen Grundbildung. Spätestens seit Mitte der 90-ger Jahre ist in allen Ländern der Bundesrepublik Deutschland eine solche Bildung zu vermitteln.

- TKS als Medium

Wie Videofilme und Folienfolgen können bereits erstellte Tabellenkalkulationsdateien genutzt werden, um die Vermittlung bestimmter Lehrstoffe zu unterstützen. Dieser Medienaspekt wurde bisher weniger beachtet, könnte jedoch dadurch an Bedeutung gewinnen, dass neuere Versionen etwa von Excel die Generierung von im Internet lauffähigen TKS-Applikationen erlauben.
Zusammenfassend lässt sich konstatieren, dass TKS seit einigen Jahren im Rahmen der Vermittlung von schulischer Allgemeinbildung berücksichtigt werden. Wohl an jeder allgemeinbildenden Schule der

Bundesrepublik sind Tabellenkalkulationssysteme für den Unterricht verfügbar.

1.3 Tabellenkalkulation in der mathematikdidaktischen Forschung

"TKP[1] sind die - gerade im deutschsprachigen Raum - noch am meisten unterschätzten Programme für den Mathematikunterricht."

/Weigand 2001/

Die Anerkennung der Relevanz und die Verfügbarkeit von TKS für den Mathematikunterricht lässt eine didaktische Auseinandersetzung mit Tabellenkalkulationssystemen im Kontext der Vermittlung von mathematischer Allgemeinbildung vermuten. Auf der Suche nach entsprechender Literatur wird man jedoch enttäuscht.

Ausführungen über die Nutzung von Tabellenkalkulationssystemen als Werkzeug im mathematischen und naturwissenschaftlichen Unterricht kommen über die Schilderung exemplarischer Fallstudien nicht hinaus.

Die derzeit vorhandenen didaktischen Überlegungen zum Einsatz von Tabellenkalkulationssystemen im Mathematikunterricht gehen im wesentlichen von speziellen mathematischen Problemen aus. Man zeigt, dass diese Probleme durchaus mit einem TKS lösbar sind und dass dieses gewisse Vorteile für deren Behandlung im Unterricht bietet. Ergebnis der didaktische Forschung zur Nutzung von Tabellenkalkulationssystemen im Mathematikunterricht ist eine gewisse Sammlung exemplarischer Beispiele, wie etwa Möglichkeiten des Einsatzes von TKS zur Zinseszinsrechnung, zur Berechnung der Kreiszahl π … . Die angesprochenen Beispiele sind dem suchenden Lehrer sicherlich eine große Hilfe. Insofern ist eine Erweiterung der „Beispielsammlung“ wünschenswert und notwendig.

Wissenschaftliche Forschung sollte jedoch über eine Beispielsammlung hinausgehen, von den speziellen Beispielen abstrahieren, grundlegende Prinzipien aufzeigen und allgemeine Fragen beantworten können.

---

[1] Tabellenkalkulationsprogramme

1.4 Tabellenkalkulation, Problemlösen, Programmieren

"Visicalc is an intellectual modeling clay. It lets you program without knowing it."

/Gassee 1987, S. 29/

Um TKS erfolgreich einsetzen zu können, bedarf es auf Seiten des Users grundlegender Kenntnisse zum Gebrauch derartiger Systeme. Exemplarisch seien diesbezüglich die Begriffe absoluter und relativer Zellbezug genannt.

Hat man sich die entsprechenden Kenntnisse angeeignet und weiß z.B. wie man Formeln im allgemeinen und Formeln mit Zellbezügen im speziellen generiert, befindet man sich zunächst in einer ähnlichen Lage wie der Einsteiger in die Nutzung eines Bildbearbeitungssystems. Diesem ist zwar prinzipiell klar, wie man Montagen aus mehreren Bilddateien erstellt, ästhetisch befriedigend werden sie in der Regel bei den ersten Versuchen nicht ausfallen.

Analog verhält es sich mit der Nutzung von TKS: Der rein „technische" Umgang mit dem System ist das eine, ein Problem mit ihm zu lösen das andere. Letzteres bedarf eines gewissen Know-how, das über die Erläuterungen eines Excel-Handbuchs hinausgeht.

Die derzeitigen Bildungspläne gehen in bezug auf die Nutzung von TKS über den mehr oder weniger technischen Aspekt der Bedienfähigkeit dieser Systeme nicht hinaus.

Strategien des Problemlösens mittels TKS lassen sich unter einem allgemeineren Blick auf die TKS-Nutzung finden. Unter einem solchen Blick ist die Verwendung von TKS nichts anderes als eine spezielle Art des Programmierens.

Die vorliegenden Ausführungen stellen sich zur Aufgabe, diese spezifische Art des Programmierens zu untersuchen.

2 TKS und Programmieren am Beispiel des Euklidischen Algorithmus

2.1 Programmieren

Programmieren ist ein Prozess, der häufig kurz und prägnant mit den Worten „Vom Problem zum Programm" beschrieben wird. Am An-

fang des Prozesses steht ein Problem bzw. eine ganze Klasse von Problemen. Er endet mit einem lauffähigen Computerprogramm zur Lösung des Problems. Ein Computerprogramm ist nichts anderes als ein Algorithmus, der in einer Programmiersprache formuliert ist.
Programmieren beinhaltet damit im wesentlichen das Finden bzw. den Entwurf eines Algorithmus zur Lösung eines Problems. In diesem Sinne ist die Verwendung einer speziellen Programmiersprache zur Formulierung des Lösungsalgorithmus zweitrangig und Programmieren vor allem algorithmisches Arbeiten im Sinne des Findens von Lösungsalgorithmen.
Aus prinzipieller Sicht sind alle Arten der Formulierung von Algorithmen zueinander gleichwertig („Churchsche These“). Unter Berücksichtigung praktischer Randbedingungen haben sich unterschiedliche Konzepte und Formen der Programmierung herausgebildet.
Ziegenbalg benennt als mögliche und gängige Formen des Programmierens neben der Verwendung „klassischer“ Programmiersprachen (z.B. BASIC, Pascal, C, …) und der Verwendung von Sprachen mit Spracherweiterungskonzept (z.B. Prolog, Lisp, Logo, …) u.a.
„die Erstellung („Programmierung“) von elektronischen Kalkulationsblättern (engl. „spreadsheets“)“ /Ziegenbalg 2000, S. 11/.
Zur Untersuchung dieser speziellen Art des Programmierens sei zunächst die Umsetzung eines klassischen Algorithmus mittels eines TKS dargestellt.

2.2 Der Euklidische Algorithmus

Beispiel 1

Der Euklidische Algorithmus bestimmt den größten gemeinsamen Teiler zweier ganzer Zahlen a und b. Am Beispiel der Zahlen a = 221 und b = 130 sei die Arbeitsweise des Algorithmus illustriert:

Schritt 1	221	:	130	=	1	Rest 91
Schritt 2	130	:	91	=	1	Rest 39
Schritt 3	91	:	39	=	2	Rest 13
Schritt 4	39	:	13	=	3	Rest 0

An dieser Stelle bricht das Verfahren ab. Der ggT von 221 und 130

ist der letzte von 0 verschiedene Rest, also 13.
Allgemein bestimmt man r, den Rest der sich bei der Division a/b ergibt. Falls r = 0, ist b der ggT. Andernfalls wird a neu mit dem alten Wert von b belegt. Der neue Wert von b ist der alte Rest r. Es wird wiederum r, der Rest bei der Division a/b bestimmt. Das Verfahren endet, wenn r = 0 ist. Der letzte Wert von b ist dann der ggT der Zahlen a und b.
Exakter lässt sich der Algorithmus mittels einer Programmiersprache[2] formulieren:

```
1  function ggt(a, b){
2      rest = a%b; //a modulo b
3      while(rest != 0) { // solange Rest verschieden von 0
4          a = b; b = rest; rest = a%b;
5      }
6      return b;
7  }
```

Programm 1

2.3 Umsetzung des Euklidischen Algorithmus mittels eines TKS

Zur Realisierung des Algorithmus mit Hilfe eines TKS wird ein Excel-Blatt wie folgt initialisiert:

	A	B	C	D
1	Euklidischer Algorithmus			
2	a	b	Rest	ggT
3	221	130	=REST(A3/B3)	=WENN(C3=0;B3;" ")
4	=B3	=C3	=REST(A4/B4)	=WENN(C4=0;B4;" ")

Tabelle 1

Der Zeile 2 von Programm 1 entspricht die Formel in Zelle C3.

[2] Zur Formulierung von Algorithmen wird in den vorliegenden Ausführungen Actionscript verwendet. Actionscript ist eine Teilmenge des Multimediaautorensystems „Flash" der Firma Macromedia. Unter der Voraussetzung, dass auf seinem System der (kostenlose) Flash-Player 6 installiert ist, kann der Leser unter http://www.ph-heidelberg.de/wp/gieding/mathematicadidactica/ alle Programme testen.

In Spalte D wird vermerkt, ob der ggT bereits gefunden wurde. Dieses ist dann der Fall, wenn der in Spalte C bestimmte Rest gleich 0 ist. Falls nicht, ist das Verfahren weiterzuführen. Man erkennt leicht die Schleifenabbruchbedingung aus Zeile 3 von Programm 1.
Zeile 4 der Kalkulationstabelle nimmt die Wertzuweisungen des Schleifenkörpers (Zeile 4 des Programms) auf:

- Zelle A4: „Neues a wird altes b.“;
- Zelle B4: „Neues b wird alter Rest.“;
- Zelle C4: „Neuer Rest wird a modulo b.“ .

Auf der Werteebene[3] stellt sich Tabelle 1 wie folgt dar:

	A	B	C	D
1	Euklidischer Algorithmus			
2	a	b	Rest	ggT
3	221	130	91	
4	130	91	39	

Tabelle 2

Wenn das Verfahren wie im speziellen Fall von Tabelle 2 noch nicht abbricht, werden weitere Zellen zur Berechnung des ggT verwendet. Hierzu ist es nicht nötig, neue Formeln direkt über die Tastatur einzugeben. Da die zu weiteren Berechnungen benötigten Formeln dieselben relativen Zellbezüge[4] wie die Formeln der Zeile 4 verwenden, genügt es, die Inhalte der Zellen A4 bis D4 zu kopieren und in darunter liegende Zellen einzufügen. Zu derartigen Zwecken stellen TKS besondere, effiziente Formen des Kopierens und Einfügens zur Verfügung („Unten Ausfüllen“[5]).

3	221	130	91		
4	130	91	39		„Unten Ausfüllen“
5	91	39	13		↓
6	39	13	0	13	

Tabelle 3

[3] Die Begriffe Werte- bzw. Formelebene werden in 3.2 näher erläutert.
[4] Relative und absolute Zellbezüge werden in 3.3 erläutert.
[5] Die Technik „Unten Ausfüllen“ wird in 3.4 erläutert.

2.4 Enaktivität vs. Symbolik – ein erster Vergleich der beiden Algorithmusrealisierungen

"An algorithm must be seen to be believed, and the best way to learn what an algorithm is all about is to try it."

/Knuth ,1973, S. 4/

In einem ersten Vergleich der beiden Realisierungen des Euklidischen Algorithmus erkennt man zwei unterschiedliche Prozessoren zur Ausführung des Algorithmus. Während die Programmiersprachenvariante für eine Maschine bestimmt ist, greift der Mensch bei der TKS-Variante in den Ablauf des Algorithmus ein. Insbesondere realisiert der User die Schleife durch Kopieren und Einfügen („Unten ausfüllen"). Die TKS-Variante zielt damit auf eine Mensch-Maschine-Symbiose als Prozessor ab.

In gewisser Weise ähnelt die Arbeit mit einem TKS der Ausführung von Algorithmen mittels Papier und Taschenrechner. Das Rechnen selbst wird jedoch in stärkerem Maße und vor allem durch die Möglichkeit der Generierung von Zellbezügen wesentlich komfortabler durch das TKS durchgeführt.[6]

In diesem Sinn beinhaltet das Arbeiten mit einem TKS eine gewisse enaktive Komponente gegenüber der stärker abstrakt/symbolischen Formulierung eines Algorithmus in einer Programmiersprache.

2.5 „Programming by Example" – eine erste Klärung der Metapher

Die Spezifik algorithmischen Arbeitens mittels TKS lässt sich recht gut durch die Metapher „Programming by Example" beschreiben.[7]

Der User trägt spezielle Eingabewerte in bestimmte Zellen ein und wendet den Algorithmus auf diese Werte an. Für andere Eingabewerte sind ggf. weitere Zellen zu belegen oder auch bereits vorhandene Zellinhalte zu löschen. Im Beispiel des Euklidischen Algorithmus

---

[6] Wir beziehen uns dabei auf den allgemein in der Praxis üblichen Gebrauch der TKS. Folgende Ausführungen werden zeigen, dass die TKS-Nutzung auch anders gestaltet werden kann.

[7] David Reed gebraucht diese Beschreibung in einer E-Mail-Diskussion zur Frage, ob VisiCalc das erste TKS war. Zu finden unter: http://www.bricklin.com/firstspreadsheetquestion.htm

müsste eventuell für andere Zahlen a und b die Anzahl der Schleifendurchläufe erhöht werden, d.h. das „Unten Ausfüllen“ auf weitere Zellen angewendet werden.
Wie stark das Kalkulationsblatt auf ein ganz spezielles oder ein mehr allgemeines Problem ausgerichtet wird, hängt von den Intentionen des Users und von seinen Kenntnissen zur Tabellenkalkulation ab.
Für den Euklidischen Algorithmus ergibt sich beispielsweise die folgende Bandbreite der Algorithmusgestaltung von ganz speziell bis ganz allgemein:

- „Durchrechnen“ des Algorithmus für zwei spezielle Zahlen. Der User arbeitet wie mit Taschenrechner und Papiertabelle ohne Zellbezüge zu verwenden. Das Kalkulationsblatt ist nur auf den einen speziellen Fall ausgerichtet.[8]
- Verwendung von Zellbezügen wie im gezeigten Beispiel. Das Blatt funktioniert auch für andere Zahlen a und b. Die Schleife im Algorithmus ist in der Regel auf weitere Zellen auszudehnen oder auf weniger Zellen zu beschränken.
- Unter Nutzung rekursiver Zellbezüge wird das Kalkulationsblatt so gestaltet, dass es für alle zulässigen Eingabegrößen a und b funktioniert, ohne dass der User nach Eingabe dieser Werte noch in den Ablauf des Algorithmus eingreifen müsste.[9]

Aus der Sicht von Programmierparadigmen entspricht die angesprochene Bandbreite „von konkret bis abstrakt“ den Programmierstilen „imperativ bis funktional“.
Die folgende Ausführungen erläutern die imperativen und funktionalen Programmieraspekte der TKS-Nutzung genauer.
Aus Gründen der Verständlichkeit dieser programmiertechnischen Erläuterungen werden zunächst jedoch die Arbeit mit TKS noch einmal aus allgemeinerer Sicht dargestellt und gewisse grundlegende Begrifflichkeiten TKS-Nutzung erläutert.

---

[8] Dieser Fall ist zwar prinzipiell möglich, jedoch theoretischer Natur. Er ist als didaktische Überhöhung anzusehen, welche die Metapher „Programming bei Example“ verdeutlichen soll.
[9] s. 5.10 bzw. 5.11

3 TKS-Grundlagen

3.1 Nutzung von TKS aus allgemeiner, abstrakter Sicht

Jedes Tabellenkalkulationssystem stellt dem User Kalkulationstabellen zu Verfügung, welche aus Zeilen und Spalten aufgebaut sind. Der Schnitt einer Zeile mit einer Spalte heißt Zelle. Jede Zelle kann gewisse Informationen (Zellinhalte) aufnehmen. Bei der Nutzung eines TKS ändert der User die Inhalte ausgewählter Zellen.
In diesem Sinne ist TKS-Nutzung eine gezielte Manipulation von Zellinhalten. Aus rein syntaktischer Sicht sind die Zellinhalte nichts anderes als eine Aneinanderreihung von numerischen und alphanumerischen Zeichen (Zeichenketten).
Die grundlegenden Datentypen, die als Zellinhalte verwendet werden dürfen, sind Text, Zahl, Wahrheitswert und Formel.[10]
Das TKS erkennt den jeweiligen Datentyp anhand syntaktischer Besonderheiten:

- Zeichenketten, die an irgendeiner Stelle ein alphanumerisches Zeichen enthalten, sind vom Typ Text.[11]
- Zeichenketten, die nur aus numerischen Zeichen bestehen, sind vom Typ Zahl.
- Die Zeichenketten „WAHR“ oder „FALSCH“ sind Wahrheitswerte.
- Zeichenketten, die mit einem wohldefinierten Zeichen beginnen (in der Regel das Gleichheitszeichen), sind vom Typ Formel.

Die Manipulation der Zellinhalte erfolgt über gewisse Methoden (Tätigkeiten):

- direkte Eingabe über die Tastatur,
- Kopieren und Einfügen bereits vorhandener Zellinhalte,
- Falls der Zellinhalt bereits als Formel gekennzeichnet ist: „Zeigen auf andere Zellen“.

---

[10] Mitunter werden auch Matrizen als spezielle Datentypen von TKS aufgefasst. Eine Matrix ist die Zusammenfassung eines rechteckigen Bereiches einer Kalkulationstabelle.
[11] Eine Ausnahme sind die Zeichenketten „WAHR“ bzw. „FALSCH“. Sie werden als Wahrheitswerte interpretiert.

Schematisch lässt sich die Arbeit mit einem TKS wie folgt darstellen:

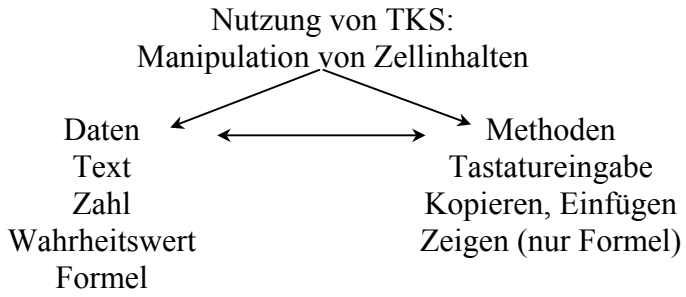

Abbildung 1

Der Doppelpfeil zwischen Daten und Methoden in Abbildung 1 ist insbesondere in der Richtung Daten → Methoden semantischer Natur: Welche Zellen man in welcher Art und Weise mit speziellen Daten der Typen Text, Zahl oder Formel belegt, wird durch das zu lösende Problem und eventuell bereits vorhandene Zellinhalte bestimmt. Entsprechend des zu lösenden Problems werden Beziehungen zwischen den Zellen eines Kalkulationsblattes hergestellt.
Grundlage für die Lösung eines Problems mittels eines TKS ist zunächst die Generierung eines Beziehungsgeflechts zwischen ausgewählten Zellen eine Kalkulationsblattes.
Hergestellt wird dieses Geflecht mittels Formeln, die auf die Werte anderer Zellen zugreifen. Die Zellen, auf die zurückgegriffen wird, können wiederum Formeln enthalten, die sich auf die Werte von Formeln beziehen, die von weiteren Zellen beinhaltet werden …. Zellinhalte und ihre Beziehungen untereinander sind damit auf verschiedenen Ebenen zu verstehen: Die Formelebene und die Werteebene.

3.2 Das Zwei- und das Dreiebenenmodell von Tabellenkalkulationsblättern[12]

Eine Zelle eines Kalkulationsblattes sei mit einem Wert vom Typ

[12] Die Ausführungen dieses Abschnitts gehen auf einen Artikel von E. Neuwirth (/Neuwirth 1995/)zurück.

Zahl belegt. Je nach speziellen Formatierungseinstellungen wird dieser Wert in bestimmter Art und Weise dem User durch das Blatt angezeigt. Drei Excel-Beispiele mögen diesen Sachverhalt illustrieren:

- Der User trägt in A5 die Zahl 1 ein. Das Format von A5 möge auf das spezielle Datumsformat „Tag.Monatsname.Jahr“ eingestellt sein. In A5 wird 01.Januar 1900 angezeigt.
- A7 sei mit 0,3 belegt. Das Zahlenformat von A7 sei auf „Bruch“ eingestellt. In A7 wird 3/10 angezeigt.
- B11 möge den Wert 3,141592654 beinhalten. Das Zahlenformat von B11 sei Zahl (zwei Nachkommastellen). Dem User präsentiert sich der Inhalt von B11 als 3,14.

Analoge Beispiele lassen sich für die Datentypen Wahrheitswert und Text finden, wobei sich die Formatierungseigenschaften im wesentlichen auf Schriftart, -stil etc. beschränken.

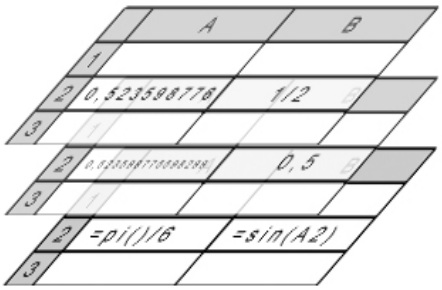

Abbildung 2

Entsprechend dieser Ausführungen unterscheidet man zwischen der Werte- und der Anzeigeebene eines Kalkulationsblattes.

Beinhaltet eine Zelle eine Formel, so kommt eine weitere unter diesen beiden Ebenen liegende Formelebene dazu. Abbildung 2 illustriert dieses „Dreiebenenmodell“.

Ein grundlegendes Verständnis der verschiedenen Zellinhalte ist für den sinnvollen Einsatz der Verfahren des Cut, Copy, Paste (CCP)[13] im Rahmen der TKS-Nutzung unerlässlich. Wie am Beispiel des Euklidischen Algorithmus bereits gezeigt, sind spezielle TKS-spezifische Formen des CCP („Unten ausfüllen“) geeignet, aus den klassischen imperativen Programmiersprachen bekannte Schleifen-

[13]Die Verfahren des Ausschneiden, Kopierens und Einfügens sind grundlegend für die Arbeit mit komplexerer Software. (s. /Gieding 2002/)

strukturen mittels TKS zu generieren: Insbesondere werden Formel aus einer Zelle kopiert und in andere Zellen eingefügt. Dort verhalten sich diese „analog“ zu der Formel der Ausgangszelle. Verständlich wird dieses durch eine genauere Untersuchung der Zellbezugsarten.

3.3 Relative und absolute Zellbezüge

Die verschiedenen Zellbezugsarten lassen sich am besten an einem Beispiel erläutern:

Beispiel 2:
Bauer Schulze will auf seiner Weide einen rechteckigen Bereich einzäunen. Ihm stehen hierfür 19m Maschendraht zur Verfügung. Was für ein Rechteck muss er wählen, um eine möglichst große Fläche einzuzäunen?
Die Aufgabe wäre in der SII eine typische Extremwertaufgabe. In der SI kann man die Lösung Quadrat zunächst durch systematisches Annähern plausibel machen. Die folgende Exceltabelle zeigt den ersten Iterationsschritt:

	A	B	C	D
1	u	19		
2	increment	1		
3		a	b	A
4	Start	1	8,5	8,5
5		2	7,5	15
6		3	6,5	19,5
7		4	5,5	22
8		5	4,5	22,5
9		6	3,5	21

Tabelle 4

In den Zellen C4 bis C10 wird aus der Seitenlänge a und dem Umfang des Rechtecks die Seitenlänge b berechnet. Die Seitenlänge a befindet sich dabei immer in der linken Nachbarzelle. Der Umfang u immer in der Zelle B1. Der Bezug auf die Seitenlänge a ist damit relativ, der Bezug auf den Umfang absolut. Tabelle 5 zeigt die Syntax der entsprechenden Zellbezüge.

	A	B	C	D
1	u	19		
2	increment	1		
3		a	b	A
4	Start	1	=$B$1/2-B4	=B4*C4
5		=B4+$B$2	=$B$1/2-B5	=B5*C5
			…	
13		=B12+$B$2	=$B$1/2-B13	=B13*C13

Tabelle 5

Beim Prozess des Kopierens und Einfügens einer bereits generierten Formel in weitere Zellen errechnet das TKS aus relativen Bezügen die entsprechenden konkreten Adressen und gibt diese in der Syntax relativer Bezüge in der Formel an.
Kopieren und Einfügen in TKS-spezifischer Form ist eine grundlegende Tätigkeit des Users bei der Arbeit mit TKS. Mit dem folgenden Abschnitt zu TKS-spezifische Formen des CCP schließt der Autor den Exkurs in die Grundlagen der TKS-Nutzung ab, um danach die Beziehungen zum imperativen Programmieren detaillierter aufzuzeigen.

3.4 CCP in TKS

Jede mehr oder weniger komplexe Software bietet die Möglichkeit, Teilmengen von Dateien, ganze Dateien oder Zusammenfassungen von Dateien auszuschneiden (Cut), zu kopieren (Copy) und die damit im Zwischenspeicher abgelegten Informationen an anderen Stellen einzufügen (Paste).
CCP ist eine wesentliche Grundlage für kreatives und effizientes Arbeiten mit Computern im allgemeinen und mit TKS im speziellen.
Zunächst bieten TKS die üblichen, aus anderen Programmen bekannten, Möglichkeiten des CCP. Grundsätzlich werden beim Kopieren die Informationen aus allen drei Ebenen eines TKS gespeichert. Beim Einfügen kann der User entscheiden, ob er z.B. nur die Informationen der Werteebene übertragen möchte.[14]

[14] Beispiel: Jede weitere Iteration in Beispiel 2 beginnt mit einem neuen Startwert.

Eine große Stärke von Tabellenkalkulationssystemen besteht darin, dass eine einmal generierte Formel sehr schnell in weitere Zellen eingefügt werden kann. Hierzu stehen Befehle wie „Unten -“ bzw. „Rechts Ausfüllen“ zur Verfügung.
„Unten-„ bzw. „Rechts-Ausfüllen“ sind tabellenkalkulationsspezifische Varianten des Kopierens und Einfügens. „Unten ausfüllen“ bewirkt ein Kopieren der Inhalte der in einem markierten Tabellenbereich am weitesten oben liegenden Zellen und ein abschließendes Einfügen dieser Inhalte in alle weiteren darunter liegenden Zellen des markierten Bereiches. „Rechts ausfüllen“ funktioniert analog.

4 TKS im Kontext imperativen Programmierens

4.1 Der imperative Programmierstil

Kurz und knapp ist ein imperativer Programmierstil dadurch gekennzeichnet, dass eine wohldefinierte Abfolge von Anweisungen angegeben wird:

1. Anweisung 1
2. Anweisung 2

   …

n. Anweisung n

Die Reihenfolge der Anweisungen ist strikt einzuhalten.
Rein imperativ ausgerichtet waren die Klassiker der Programmiersprachen Basic, Fortran, Cobol[15] etc.. Aber auch die heute praktisch bedeutsamen objektorientierten Sprachen wie etwa C++ oder Java beinhalten in starkem Maße Elemente eines imperativen Programmierstils.
Die vorn angegebene Programmiersprachenversion des Euklidischen Algorithmus ist ein Beispiel für imperatives Programmieren:

In der zweiten Iteration wäre das etwa der Wert aus Zelle B7. Normales Einfügen der Informationen in die Startwertzelle würde die Formel aus B7 in diese Zelle übertragen, welche in der Startwertzelle keinen Sinn machen würde. Das Einfügen nur der Informationen aus der Werteebene löst das Problem.

[15] in ihrer ursprünglichen Form

...; Belege a mit b!; Belege b mit r!;
Belege r mit dem Rest der Division a/b!; …
Ein Vertauschen der Reihenfolge dieser Befehle, hätte ein Nichtfunktionieren des Algorithmus zur Folge.

4.2 Variablen in imperativen Programmen

Der Variablen a möge der Wert 5 zugewiesen sein (a=5). Diese Wertzuweisung zeigt zwei Aspekte des Variablenbegriffs in einem Programm. Es muss unterschieden werden zwischen der Bezeichnung „a“ und dem sich hinter dieser Bezeichnung befindlichen Wert 5. Physikalisch sind Variablen letztlich nichts weiter Speicherplätze im Rechner und ihre Bezeichnung die codierte Adresse des Speicherplatzes.

4.3 Algorithmengrundstrukturen

Jeder Algorithmus lässt sich aus den folgenden drei Algorithmengrundstrukturen zusammensetzten:

- Sequenz
  Unter Sequenzen versteht man die Aufeinanderfolge von Anweisungen:
  Beispiel: a=b; b=Rest; Rest=a%b.
- Wiederholung
  Häufig ist ein und dieselbe Folge von Anweisungen mehrfach hintereinander abzuarbeiten. Man fasst diese mehrfache Abarbeitung in Wiederholungen zusammen. Hierfür stellen Programmiersprachen gewisse Schleifenkontrollstrukturen zur Verfügung.
  Beispiel:

```
while(Rest != 0) { // solange Rest verschieden von 0
    a=b; b=Rest; Rest= a%b;
}
```

- Verzweigung
  Falls eine gewisse Bedingung erfüllt ist, wird eine Anweisungsfolge I abgearbeitet, im Falle der Nichterfüllung der Bedingung eine Anweisungsfolge II:

```
    if(Bedingung erfüllt){Anweisungsfolge I!}
    else{Anweisungsfolge II!}
Beispiel:
if (x<0) {
       betrag = -x
       }
    else {
       betrag = x
       }
```

4.4 Die Kalkulationstabelle zum Euklidischen Algorithmus in imperativer Interpretation

Die folgende Übersicht zeigt Zusammenhänge zwischen der Programm- und der TKS-Version des Euklidischen Algorithmus.

<table>
<tr><td>Struktur</td><td>Programm</td><td>TKS</td></tr>
<tr><td>Sequenz</td><td>a = 2345;</td><td>A2 = 2345</td></tr>
<tr><td></td><td>b = 28;</td><td>B2 = 28</td></tr>
<tr><td></td><td>Rest = a%b</td><td>C2 = rest(A2;B2)</td></tr>
<tr><td>Schleife</td><td>while Rest<>0{
a =b;
b = Rest;
Rest = a%b;
}</td><td>
<table>
<tr><td></td><td>A</td><td>B</td><td>C</td></tr>
<tr><td></td><td colspan="3">…</td></tr>
<tr><td>3</td><td>=B2</td><td>=C2</td><td>=rest(A3;B3)</td></tr>
<tr><td>4</td><td>=B3</td><td>=C3</td><td>=rest(A4;B4)</td></tr>
<tr><td>5</td><td>=B4</td><td>=C4</td><td>=rest(A5;B5)</td></tr>
<tr><td>6</td><td colspan="3">…</td></tr>
</table>
</td></tr>
</table>

4.5 Interpretation von Sequenzen in TKS

Eine Interpretation von Zelladressen als Variablennamen ergibt sich kanonisch. Die Reihenfolge in der die Zellen aufeinander Bezug nehmen, kann als Widerspiegelung der Anweisungsreihenfolge eines imperativen Programms gesehen werden.
Es sei an dieser Stelle betont, dass es sich dabei um gewisse Modellvorstellungen handelt, die von den realen Verhältnissen im Kalkulationsblatt abweichen können.
Im Abschnitt 4.14 „Kritik an der imperativen Sicht auf Kalkulationsblätter“ wird diese Problematik noch einmal aufgegriffen.

4.6 Interpretation von Schleifen in TKS

Zunächst werden Zellinhalte generiert, welche der Anweisungsfolge entsprechen, die wiederholt auszuführen ist. Anders ausgedrückt: Es wird einmal der sogenannte Schleifenkörper in Form von Zellbelegungen erstellt. Danach wendet man gewisse Formen des CCP an, um die Schleife auszuführen. Dem User obliegt es dabei zu entscheiden, wann die Schleife zu beenden ist.

4.7 Zellbezüge und Schleifen

Ein Verständnis für absolute und relative Zellbezüge ist grundlegend für die Arbeit mit TKS. Diesbezüglich sind zwei Aspekte der Zellbezüge zu sehen. Zum einen muss der User den Unterschied zwischen relativen und absoluten Zellbezügen verstanden haben. Zum anderen ist, im Kontext des zu lösenden Problems, zu entscheiden, welcher Zellbezug relativ und welcher absolut sein muss. Letzteres ist der schwierigere Aspekt von Zellbezügen. Die folgenden Ausführungen werden zeigen, dass eine Betrachtung von Kalkulationsblättern im Sinne der Ausführung von Schleifen hilfreiche Anhaltspunkte liefert, zu entscheiden, welcher Art konkrete Zellbezüge sein müssen.

4.8 Einfache, nicht verschachtelte Schleifenstrukturen

Ausgangspunkt sei eine einzige Schleife, die keine weitere Schleife enthält.
Die Variablen die im Schleifenkörper verwendet werden, lassen sich in zwei Klassen einteilen:

/1/ Variablen, deren Werte durch die Abarbeitung der Anweisungen des Schleifenkörpers verändert werden.
Beispiel: Die Schleife in Beispiel 1 (Euklidischen Algorithmus) ändert die Werte aller im Algorithmus verwendeten Variablen.
(s. Programm 1)

/2/ Variablen, deren Werte durch die Abarbeitung der Anweisungen des Schleifenkörpers nicht verändert werden.
Beispiel: Der Wert des Umfangs u in Beispiel 2 wird durch die Iterationsschleife nicht geändert. Innerhalb der Schleife wird jedoch zur Berechnung der Länge b auf u zurückgegriffen.

```
1  n=10; a=1; increment=1; u=19; ausgabe="";
2  for(i=1; i<n; i++){
3      b=u/2-a; F=a*b;
4      ausgabe=ausgabe+"a= "+a+" b="+b+" A= "+F+newline;
5      a=a+increment;
6  }
```

Programm 2

Eine Schleife wird im TKS durch „Unten-„ bzw. „Rechts ausfüllen“ realisiert. Sich verändernde Variablenwerte findet man in verschiedenen Zellen wieder. Eine Variable, deren Wert sich nicht ändert, benötigt demgegenüber nur eine einzige Zelle. Wird ihr Wert in einer mehrere Zeilen bzw. Spalten beanspruchenden Schleife immer wieder verwendet, so realisiert man dieses am besten über einen absoluten Zellbezug.

4.9 Zwei ineinander geschachtelte Schleifen

Eine Schleife möge genau eine weitere Schleife enthalten. Mit einem TKS lässt sich ein solches Konstrukt durch „zweidimensionales“ Kopieren und Einfügen realisieren: Eine der beiden Schleifen durch „Rechts ausfüllen“ die andere durch „Unten ausfüllen“.
Aus Gründen der Übersicht sei die Problematik anhand eines sehr einfachen Beispiels erläutert:
Beispiel 3:

Die Gruppentafel der Gruppe [$\boldsymbol{Z}/_4$;+] ist zu generieren.

	A	B	C	D	E
1	Modul	4			
2	+	0	1	2	3
3	0	=Rest($A3+B$2;$B$1)	1	2	3
4	1	1	2	3	0
5	2	2	3	0	1
6	3	3	0	1	2

Tabelle 6

Ein Algorithmus zur Generierung aller möglichen Verknüpfungen a+b (a,b œ $\boldsymbol{Z}/_4$) lässt sich mittels Actionscript wie folgt formulieren:

```
1  Modul = 4; new Array;
2  Tafel = [[0, 0, 0, 0],[0, 0, 0, 0],[0, 0, 0, 0],[0, 0, 0, 0]];
3  for(i=0;i<Modul;i++){
4      for(j=0;j<Modul;j++){
5          Tafel[i][j]=(i+j)%Modul;
6      }
7  }
```

Programm 3

Zur Erstellung der Gruppentafel mittels eines TKS muss lediglich eine einzige Formel eingegeben werden.[16] Das Einfügen der Formel in zwei Richtungen, nach unten und nach rechts, impliziert die Verwendung gemischter Zellbezüge. Hierunter versteht man einen absoluten Spaltenbezug zusammen mit einem relativen Zellbezug bzw. umgekehrt.
Die Formel, die in B3 einzugeben wäre, lautet:

=Rest($A3+B$2;$B$1).

Der erste Summand ($A3) entspricht der Zählvariablen der äußeren Schleife. Er steht immer in Spalte A ($A, absoluter Bezug auf Spalte A)[17] und relativ in derselben Zeile wie die Summe (3, relativer Bezug auf die Zeile, in der auch die Summe steht). Der zweite Summand (B$2) entspricht der Zählvariablen der inneren Schleife. Er steht immer in Zeile 2 ($2, absoluter Bezug auf die Zeile 2) und in derselben Spalte wie die Summe (B, relativer Spaltenbezug)[18].
Der Modul wird vor den beiden Schleifendurchläufen festgelegt und durch die Schleifen nicht geändert. Der Bezug auf den Modul ($B$1) ist demzufolge rein absolut.

4.10 Mehrfach geschachtelte und komplexere Schleifenkonstrukte

Das Beispiel des vorangegangenen Abschnitts war bewusst einfach gewählt, wobei das Adjektiv einfach unter zwei Aspekten zu sehen ist:

---

[16] Der nicht grau unterlegte Teil von Tabelle 6 sei bereits initialisiert.
[17] „Unten ausfüllen“ generiert die äußere Schleife.
[18] „Rechts ausfüllen“ generiert die innere Schleife.

- Die obere Schleife enthielt nur eine einzige Schleife und daneben keine weiteren Anweisungen.
- Die innere Schleife enthielt genau eine Anweisung.

Falls wenigstens eine der beiden Schleifen mehrere Anweisungen enthält, die nicht in einer einzigen Formel zusammengefasst werden können, benötigt man mehr als eine Zelle zur Abbildung der entsprechenden Anweisungsfolge im TKS. Die Realisierung der ineinander geschachtelten Schleifen mittels „zweidimensionalen" Kopierens und Einfügens gestaltet sich jetzt etwas komplizierter.[19] Diesbezügliche Lösungsmöglichkeiten offerieren TKS u.a. in Form von

- If-Then-Else-Funktion,
- Möglichkeiten der gleichzeitigen Verwendung mehrerer Kalkulationsblätter,
- Matrixfunktionen.

Durch die Verwendung von mehreren Kalkulationsblättern und/oder Matrixfunktionen lassen sich auch mehrfach verschachtelte Schleifen mittels TKS realisieren.

Im Rahmen des vorliegenden Artikels können und sollen entsprechende Lösungsmöglichkeiten nicht näher untersucht werden.

4.11 Imperative Schleifenstrukturen in TKS - eine Zusammenfassung

Zusammenfassend lässt sich konstatieren:

- Schleifenstrukturen imperativer Programmiersprachen lassen sich durch die Optionen „Unten/Rechts ausfüllen" mittels TKS realisieren. Dem User obliegt die Überprüfung der Schleifenbedingung.
- Verschachtelte Schleifenstrukturen können durch „mehrdimensionales" „Unten/Rechts ausfüllen" in TKS generiert werden.
- Es besteht ein Zusammenhang zwischen der Art des Zugriffs einer Schleife auf bestimmte Variablen und der Zellbezugsart in einem entsprechenden Kalkulationsblatt:

[19] „Unten ausfüllen" kann innerhalb einer Spalte den Inhalt von nur einer einzigen Zelle in weitere Zellen übertragen. Analoges gilt für „Rechts ausfüllen".

Programmiersprache	TKS
• Variable v wird außerhalb der Schleifen initialisiert, Anweisungen eines Schleifenkörpers greifen auf v zu, ändern den Wert von v jedoch nicht.	• Verwendung eines absoluten Zellbezuges
• Variablenwert wird innerhalb der Schleife geändert, Schleife ist nicht Teil einer Schleifenverschachtelung	• Verwendung eines relativen Zellbezuges
• Variablenwert wird innerhalb zweier geschachtelter Schleifen verändert.	• Verwendung eines gemischten Zellbezuges
• Mehr als zwei Schleifen sind verschachtelt.	• Verwendung „höher dimensionaler“ Bezüge, d.h. Zugriff auf die Zellen weiterer Kalkulationstabellen

4.12 Schleifen als didaktische Hilfe für das Erkennen von Zellbezügen?

Zellbezüge haben hinsichtlich der Vermittlung eines grundlegenden Verständnisses für die TKS-Nutzung eine zentrale Bedeutung.
Die Schwierigkeit der Begriffe absoluter und relativer Zellbezug besteht weniger in einem Verständnis für die Begriffe an sich als vielmehr in der Entscheidung ob ein bestimmter Bezug in einem Kalkulationsblatt relativ oder absolut sein sollte.
Die Betrachtung von auftretenden Schleifen in Problemlösungen könnte entsprechend der vorangegangenen Ausführungen hilfreich hinsichtlich der Auswahl von Zellbezugsarten sein.
Diesbezügliche schulpraktische Untersuchungen seitens der informatik- und mathematikdidaktischen Forschung erscheinen lohnenswert.

4.13 Imperative Schleifenstrukturen, TKS, Programming by Example

Die Verwendung einer einzigen Formel ist zur Lösung von Bei-

spiel 3 (Gruppentafelerstellung [$\mathbf{Z}/_4$;+]) nicht zwingend nötig. Der User kann genauso gut, ohne dann gemischte Zellbezüge verwenden zu müssen, viermal separat die innere Schleife generieren.
Die Lösung des Umzäunungsproblems enthält von der Struktur her zwei ineinandergeschachtelte Schleifen. Diese Struktur kann in TKS durch „zweidimensionales" Ausfüllen realisiert werden.[20] Es obliegt dem User, ob er eine derartige Lösungsvariante anwenden möchte. Er kann natürlich auch den Part der übergeordneten Schleife selbst übernehmen. Hierzu wählt er nach jedem Durchlauf der Iteration, den neuen Startwert und das neue Inkrement selbst aus und belegt die entsprechenden Zellen „händisch" mit den ausgewählten Werten.
In Abschnitt 2.5 wurde die Art und Weise der TKS-Nutzung mit der Bandbreite „von konkret bis abstrakt" beschrieben. Aus der Sicht der Generierung von Schleifenstrukturen lässt sich diese Bandbreite genauer spezifizieren:

konkreter	abstrakter
• viele einfache Formeln	• wenige komplexe Formeln
• hoher Useranteil beim Algorithmusablauf, dabei insbesondere inhaltlich geprägte Entscheidungen des Users	• geringerer Useranteil beim Algorithmusablauf, Useranteil eher formal: „Unten/Rechts ausfüllen"
• relative Zellbezüge reichen aus	• relative, absolute und gemischte Zellbezügen

4.14 Kritik an der imperativen Sicht auf Kalkulationsblätter

In 4.5 wurden Zellbezügen als spezielle Realisierungen imperativer Folgen von Arbeitsanweisungen (Sequenzen) gesehen.
Eine solche Sichtweise ist möglich und beim Entwurf von Kalkulationsblättern auch durchaus hilfreich, bleibt aber letztlich reine Interpretation, die dem Wissen um eine imperative Lösung des Problems entspringt.
Allein der Fakt, dass es nicht zwingend nötig ist, bei der Belegung

[20] Entsprechende Lösungen unter:
http://www.ph-heidelberg.de/wp/gieding/mathematicadidactica/

der Zellen eine gewisse Reihenfolge einzuhalten[21] zeigt den interpretativen Charakter einer derartigen Sicht auf die Kalkulationsblätter.
Die ebenfalls in 4.5 angesprochene „kanonische“ Interpretation der Zelladressen als Variablennamen ist in gleichem Sinn als Möglichkeit einer imperativen Interpretation von Kalkulationsblättern zu sehen. Auch diese Interpretation kann für den Entwurf von Kalkulationsblättern hilfreich sein, ist zugleich aber nicht problemlos. Zum einen berücksichtigt die Vorstellung von einer Zelladresse als Platzhalter für Werte[22] nur die Werteebene von Kalkulationsblättern und abstrahiert von deren Formelebene. Zum anderen kollidieren die Möglichkeiten der Verwendung von Zelladressen mit dem Umfang des Variablenbegriffs imperativer Programmiersprachen. Die folgende imperative Kontrollstruktur (Verzweigung) ist beispielsweise nicht „1 zu 1“ mittels eines TKS umsetzbar:

```
if(Bedingung){a = 500;}else{b = 700;}
```

In TKS ist es schlichtweg nicht möglich, mittels einer Formel etwa in A1den Wert einer anderen Zelle zu manipulieren, ohne dass in dieser Zelle selbst eine Formel existiert, die sich auf A1 bezieht.
Eine entsprechende Möglichkeit käme der Manipulation der Formelebene einer Zelle durch die Formel einer anderen Zelle gleich. Würde man diese Idee konsequent zu Ende denken, hätte diese zur Konsequenz, dass sich eine Formel selbst (hinsichtlich ihres Quelltextes) manipuliert.
TKS wurden eher für Buchhalter als für Forscher auf dem Gebiet der Künstlichen Intelligenz konzipiert.
Trotzdem hat die Nutzung von TKS sehr viele Gemeinsamkeiten mit dem Gebrauch von funktionalen Programmiersprachen, die vornehmlich auch in der KI-Forschung verwendet werden.
Ein erstes Indiz hierfür ist das If-Then-Else Konstrukt. In TKS tritt es

---

[21] Im Gegensatz hierzu zwingt z.B. das mathematische Ingenieursystem MathCAD dem User eine gewisse imperative Arbeitsweise auf: MathCad-Dokumente werden von links nach rechts und von oben nach unten ausgewertet.
[22] Eine andere Bedeutung haben Variablen in imperativen Programmen nicht. Letztlich sind die Variablennamen nicht anderes als codierte Speicherplatzadressen.

als spezielle Funktion auf. Eine entsprechende Untersuchung der Realisierung von Verzweigungen in TKS soll aus diesem Grund dem nächsten Abschnitt zugeordnet werden. Dieser wird zeigen, dass TKS-Nutzung vor allem auch funktionales Programmieren ist.

5 TKS im Kontext funktionalen Programmierens

5.1 If-Then-Else und mehr

Wie bereits angedeutet tritt „If-then-else“ in TKS als spezielle Funktion auf. In Excelsyntax lautet diese:

=Wenn(Prüfung; Dann-Wert; Sonst-Wert).

Die folgende konkrete Formulierung einer „If-then-else“-Funktion berechnet z.B. den absoluten Betrag eines in B1 generierten Wertes:

(*) =WENN(B1<0;-B1;B1).

Der Wert in B1 möge durch die Funktion =sin(A1) berechnet sein. A1 sei mit x bezeichnet, vom speziellen Wert von x sei abstrahiert.[23] Aus dieser Sicht wurde durch * eine neue Funktion definiert:

$$f(x) := \begin{cases} \sin(x) & falls\ \sin(x) > 0 \\ -\sin(x) & sonst \end{cases}$$

f ist nichts anderes als die Nacheinanderausführung $abs \circ \sin$.

Die *if-then-else*-Funktion eines TKS kann als eine Vorschrift verstanden werden, mittels derer man zwei Funktionen f_1 und f_2 eine neue Funktion zuordnet:

$$(f_1, f_2) \xrightarrow{Wenn(w;f_1,f_2)} \begin{cases} f_1 \text{ falls } w \text{ den Wert wahr annimmt} \\ f_2 \text{ sonst} \end{cases}$$

Das Einsetzen der Funktionen w, f_1 und f_2 in die Funktion *if-then-else* realisiert man aus Übersichtsgründen am besten durch Bezüge auf Zellen, in denen w, f_1 bzw. f_2 zuvor definiert wurden.

5.2 Alles ist Funktion

Abstrahiert man davon, dass eine neu zu definierende Funktion na-

[23] Bei x könnte es sich letztlich um jeden Wert vom Typ Zahl handeln.

türlich einen gewissen Sinn haben soll, ist bei einer Funktionsdefinition mittels Zellbezügen keine Zelle vor einer anderen auszuzeichnen.
Letztlich steht jede beliebige Zelle für eine Funktion, die zur Definition weiterer Funktionen verwendet werden kann.[24]
In 3.1 wurde der Entwurf eines Kalkulationsblattes als die Generierung eines Beziehungsgeflechts zwischen den Zellen des Blattes charakterisiert.
Jetzt ist klar, was mit diesem Beziehungsgeflecht genau gemeint ist: Definition von Funktionen.
Die Zellbezüge dienen zur Einsetzung der Definition einer Funktion in die Definition einer weiteren Funktion.
Durch die Aufteilung der Definition einer komplexen Funktion auf mehrere Zellen wird eine gewisse Übersichtlichkeit und Handhabbarkeit der komplexen Funktionen gewährleistet.
Aus theoretischer, prinzipieller Sicht könnte man auch mit einer einzigen Zelle zur Funktionsdefinition auskommen.

5.3 Und was ist mit Variablen?

Kurz und knapp:
Variablen gibt es in der funktionalen Sicht auf Kalkulationsblätter nicht! Ebenso wenig gibt es Variablen in rein funktionalen Programmiersprachen.
Es wird Zeit, den Zusammenhang zu den funktionalen Programmiersprachen herzustellen. Letztere beruhen alle auf dem sogenannten λ–Kalkül.

---

[24] Im Falle, dass in einer Zelle eine Formel eingetragen wurde, ist diese Aussage einsichtig. Formeln sind letztlich Funktionen. Ist eine Zelle leer, ergibt die Auswertung des Bezuges auf eine solche Zelle den numerischen Wert 0 (bei den meisten TKS). Leere Zellen enthalten damit die konstante Funktion $f(x)= 0$. Ebenfalls konstante Funktionen enthalten die Zellen, in die ohne Formel ein Wert eines bestimmten Datentyps eingetragen wurde. Derartige „direkte“ Eintragungen können als benutzerfreundliche Vereinfachung gesehen werden. Soll in eine Zelle der Wert 7 bzw. die Zeichenkette „Hallo“ eingetragen werden, so kann das sowohl über die einfache Eingabe als auch über die Generierung der Formeln = 7 bzw. = “Hallo“ passieren.

5.4 Der λ-Kalkül

Üblicherweise definiert man in der Mathematik eine Funktion als eine Menge geordneter Paare:

> Es sei M eine nichtleere Menge. Eine Teilmenge aus dem Kreuzprodukt $M \times M$ heißt Relation auf M. Eine Relation R ist eine Funktion wenn sie die Bedingung der Eindeutigkeit erfüllt:
>
> $$\forall (x, y) \in R : (x, y_1) \in R \wedge (x, y_2) \in R \rightarrow y_1 = y_2 .$$

Die Definition ist eine Abstraktion über alle Spezialfälle, die man zunächst intuitiv als Funktion gesehen hat. Die Auswahl der geordneten Paare aus $M \times M$, die zu einer speziellen Funktion f gehören, erscheint dabei in gewisser Weise „Gott gegeben“. Wie man alle Paare $(x, y) \in f$ bestimmt, spielt in der Abstraktheit des Funktionsbegriffs keine Rolle.

Im praktischen Umgang mit Funktionen ist eine solch abstrakte Fassung des Funktionsbegriffs nicht sonderlich hilfreich.

Hier interessiert, wie man zu einem zulässigen x das y bestimmt, so dass $(x, y) \in f$ gilt.

Anders ausgedrückt: Gib mir ein *Verfahren*, das zu einem Argument x den zugehörigen Funktionswert y berechnet!

1930 verwirklicht Alonzo Church im sogenannten λ-Kalkül die Idee der Definition von Funktionen über elementare Rechenvorschriften. Der Kalkül erlangte in der Informatik grundlegende Bedeutung zur Formulierung von Algorithmen.

Im vorliegenden Rahmen kann und soll der λ-Kalkül nicht vollständig und exakt erläutert werden.

Es genügt, die Ideen des Kalküls kurz anzudeuten.

Church entwickelte den Kalkül im Kontext von Berechenbarkeitsuntersuchungen. Insbesondere ist mit der Nacheinanderausführung $g \circ f$ zweier berechenbarer Funktionen g und f eine Vorschrift gegeben, wie man aus zwei berechenbaren Funktionen eine weitere berechenbare Funktion erzeugen kann. Auf abstrakter Ebene lässt sich diese Vorschrift als Funktion auffassen, die zwei Funktionen auf eine neue Funktion abbildet. Funktionen sind in diesem Zusammenhang selbst Werte.

Derartige abstrakte Vorschriften, die selbst Funktionen sind und sich nur noch mit Funktionen und Funktionsauswertungen beschäftigen, heißen Kombinatoren.[25]
Ein Vergleich mit den vorausgegangenen Ausführungen zeigt, dass Ideen des λ-Kalküls in den Kalkulationsblättern der TKS aufgehoben sind.

5.5 Variablen

Der Begriff der Variable ist hier im Sinne der Programmierung zu verstehen. Ein Variablenname ist nicht anderes als eine kodierte Speicheradresse. Der Wert der Variable d.h. der Speicherinhalt kennzeichnet einen gewissen Zustand der Maschine, die das Programm abarbeitet.
In einem System, in dem nur Funktionen und Funktionsauswertungen eine Rolle spielen, haben Zustände der Maschine und damit Variablen keine Bedeutung.

5.6 Funktionale Programmiersprachen

Die funktionalen Programmiersprachen von LISP (1957) über LOGO, Scheme, … bis Haskell (1991) basieren auf dem λ-Kalkül. Sie sind in mehr oder weniger reiner Form Umsetzungen des Kalküls.
Ein in einer funktionalen Sprache geschriebenes Programm besteht aus einer Menge von Funktionsdefinitionen, wobei diese Definitionen durchaus mathematisch zu verstehen sind. Für die Programmformulierung stehen einige „eingebaute“ Grundfunktionen sowie Möglichkeiten der Rekursion und Einsetzung zur Verfügung.
Ein Programm wird durch Eingabe eines Ausdrucks aktiviert, der ausgewertet wird:
Eingabe: ggT 45 50
Ausgabe: 5

[25] Alan Turing stellte seinerseits Berechenbarkeitsuntersuchungen mit Hilfe von Turingmaschinen an. Der λ-Kalkül ist so mächtig, dass man jede Turingmaschine *T* durch einen Kombinator des λ-Kalküls ersetzen kann, welcher das selbe wie die Turingmaschine leistet.

Zuvor muss die Funktion ggT natürlich definiert (programmiert) worden sein.

5.7 Rekursion

Mit der Möglichkeit der Anwendung von Rekursionen stellt der λ-Kalkül ein mächtiges Werkzeug zur Definition von Funktionen zur Verfügung.
Die folgende Definition des größten gemeinsamen Teilers zweier ganzer Zahlen *a* und *b* ist ein einfaches Beispiel zur Anwendung der Rekursion:

$$(\mathrm{I})\ \forall a,b \in \mathrm{Z} : ggT(a,b) := \begin{cases} a & falls\ b = 0 \\ ggT(b, a \bmod b) & sonst \end{cases}$$

Das „schwierige" Problem $ggT(a,b)$ wird auf seinen einfachsten Fall $ggT(a,0)$ zurückgeführt.[26]
Unschwer erkennt man in (I) eine sehr elegante Formulierung des Euklidischen Algorithmus.

5.8 Die Funktion ggT in Scheme und Haskell

Zur Implementierung des Euklidischen Algorithmus in einer funktionalen Programmiersprache kann Definition (I) quasi „1 zu 1" übernommen werden.

Scheme

```
(define (ggT a b)
  (if (= b 0)
    b
    (ggT b (remainder a b))))
```

Haskell

```
ggT :: Int ° Int ° Int
ggT a 0 = a
ggT a b = ggT b (mod a b)
```

Programm 4

5.9 Rekursionen mittels TKS?

Durch übermäßigen imperativen Programmiersprachengebrauch kanalisiert, ist man sofort geneigt, die Rekursion in Definition (I) bzw.

[26] Die Methode der Rekursion ist gleichzeitig eine Problemlösestrategie: Führe ein Problem auf seinen einfachsten Fall zurück! Man spricht in diesem Zusammenhang von „Teile-und-Herrsche-Problemen" bzw. –Algorithmen.

in den entsprechenden Programmiersprachenvarianten in imperative Schleifendurchläufe „aufzudröseln“.
In der bereits besprochenen Variante der Realisierung des Euklidischen Algorithmus mittels eines Kalkulationsblattes ist nichts anderes passiert.
Eine solch imperative Sicht auf Rekursionen ist im Rahmen des funktionalen Programmierparadigmas jedoch nicht nötig. Hat man (I) in der Syntax einer funktionalen Programmiersprache formuliert, muss man sich um die Auswertung der Funktion *ggT* keine weiteren Gedanken mehr machen. Letzteres erledigt die Maschine.[27]
Es stellt sich die Frage, ob man analog mit TKS arbeiten kann.

5.10 Eine rekursive Realisierung des Euklidischen Algorithmus mittel TKS

Rekursionen können in TKS durch sogenannte Zirkelbezüge realisiert werden. Bei einem Zirkelbezug bezieht sich eine Zelle letztlich auf sich selbst.[28]
Die folgende Excel-Tabelle berechnet den ggT zweier ganzer Zahlen a und b unter Nutzung rekursiver Zellbezüge:

	A	B	C	D
1	ggT	a	b	
2	Start	45	12	
3	0	=WENN(A3=0; B2; WENN(C3=0; B3; C3))	=WENN(A3=0; C2; D3)	=WENN(C3=0; 0; REST(B3;C3))

Tabelle 7

[27] Diese wiederum kann nicht wirklich rekursiv arbeiten. Sie löst das Problem, indem sie die rekursiven Funktionsaufrufe in einem Kellerspeicher (Stack) protokolliert und diesen dann in einer rein imperativen Schleife von oben nach unten abarbeitet. Komplizierte Rekursionen mit hoher Tiefe können dementsprechend eine Meldung „Stack-Overflow“ hervorrufen.
[28] Da unkontrollierte Rekursionen Rechnerabstürze verursachen können, müssen in manchen TKS (z.B. Excel) Zirkelbezüge zunächst durch den User erlaubt werden. Excel: Extras, Optionen, Berechnung, Iteration aktivieren.

Die Tabelle enthält eine Reihe von Konstrukten, die „lediglich“ einer korrekten „Arbeitsweise“ dieser Tabelle dienen. Mit dem Ziel, die eigentliche Rekursion

ggT a b = ggT b (mod a b) (Haskell)

herauszukristallisieren, werden derartige Konstrukte Stück für Stück eliminiert.

Hat man etwa in Haskell die Funktion *ggT* bereits definiert, kann man durch einen Aufruf von z.B. ggT 50 45 die Funktion auswerten lassen. Dieser Aufruf wird durch die Bezüge auf die Zelle A3 (Start) simuliert. B3 (a) und C3 (b) beziehen sich auf ihren eigenen Wert. Damit ein solcher sinnvoller Wert überhaupt vorhanden ist, übernehmen sie im Fall A3=0 die Anfangswerte von a und b. Ein vom User zu vollziehender Wechsel des Wertes in der Zelle A3 „ruft *ggT* auf“.

Im Fall A3 ∫ 0 sind nur die folgenden Formeln relevant:

	B	C	D
		…	
3	=wenn(C3=0; B3; C3)	=D3	=wenn(C3=0; 0; Rest(B3;C3)

Die Verwendung der *if-then-else*-Funktion in Zelle D3 ist nötig, um Fehlermeldungen (#DIV/0!) abzufangen. Die Notwendigkeit dieser Vorsichtsmaßnahme ist nicht unmittelbar einsichtig und erhellt sich nur vor dem Hintergrund einer tieferen Einsicht in die Handhabung von Rekursionen durch Excel.

Abstrahiert man von der Notwendigkeit des Abfangens der genannten Fehlermeldung und verwendet anstelle der Zellbezüge B3, C3, D3 die Bezeichnungen a, b, r[29], so stellen sich die wesentlichen Teile der Formeln wie folgt dar:

a	b	r
= Wenn(b = 0; a ; b)	= r	= Rest(a; b)

Die Rekursion wird nicht aufgerufen, wenn b=0 ist.

[29] Viele TKS erlauben die Definition von Namen für Zellen, die dann in Zellbezügen anstelle der Zelladressen verwendet werden können.

Für die eigentliche Rekursion bleiben letztlich die folgenden Formeln übrig:

a	b	r
=b	= r	= Rest(a; b)

Um in diesen Formeln die Haskell-Zeile

„ggT a b = ggT b (mod a b)“

wieder zu finden, kommt man nicht umhin, den Zirkelbezug zwischen a, b und r nachzuvollziehen.

Eine Stärke der funktionalen Programmierung besteht aber gerade darin, dass der „maschinelle“ Ablauf der Rekursion nicht nachvollzogen werden muss![30]

Um doch noch zu einer Formel zu kommen, die in stärkerem Maße der Haskell-Formulierung entspricht, ist man geneigt, nach einer TKS-Lösung zu suchen, die für die Definition der Rekursion nur eine einzige Zelle verwendet. Eine rein formale Reduktion der Formel aus a, b und r auf eine einzige Formel in Zelle a zeigt das Dilemma: Die Formel lautet =Rest(a;b). Eine Rekursion ist in dieser Formel nur bezüglich a zu erkennen. Da eine Zelle nur einen einzigen numerischen Wert enthalten kann, muss die Funktion b in einer weiteren Zelle rekursiv definiert werden.

Die Grenzen rekursiven Arbeitens mittels TKS sind erreicht.

5.11 Behind the Border: Definition der Funktion *ggT* in einer einzigen Zelle

Grenzen reizen dazu, durchbrochen zu werden. Eine Grenze bezüglich rekursiven Arbeitens mittels TKS wird dadurch gesetzt, dass in TKS ein komfortables Listenkonzept, wie es den funktionalen Programmiersprachen eigen ist, fehlt. [31]

Eine Möglichkeit, diese Schwierigkeit zu umgehen besteht darin, Listen durch Zeichenketten zu simulieren.

---

[30] Bei komplizierteren Rekursionen ist das ohnehin nur mit sehr großem Aufwand möglich.

[31] Als „Urvater“ der funktionalen Programmiersprachen kann LISP angesehen werden. LISP steht für LISt Prozessor.

Eine entsprechende Excel-Lösung ist mit der folgenden Formel gegeben:

```
=WENN(B8=0;
    VERKETTEN(C8;"|";D8);
       WENN(ISTFEHLER(FINDEN("|";C11));C11;
          WENN(b=0;VERKETTEN("Der ggT ist "; a);
             VERKETTEN(b;"|";REST(a;b)))
       )
)
```
[32]

Heureka! Im unterstrichenen Teil der Formel ist endlich die Rekursion „ggT a b = ggT b (mod a b)“ wieder zu erkennen.
Mit obiger Formel hat sich der Autor als „Formally-Basic-Programmer-Now-Spreadsheet-Junkie“ geoutet. Was bleibt ist Schadensbegrenzung: Die Aufnahme der Formel in die vorliegenden Ausführungen erfolgte aus Gründen der didaktischen Überhöhung und dient lediglich der besseren Verdeutlichung von Grenzen rekursiven Arbeitens mittels TKS.

5.12 Noch mehr Grenzen

In die vollständigen rekursiven TKS-Formeln wurden *if-then-else*-Funktionen eingebaut, die lediglich dazu dienen, Fehler abzufangen. Die Notwendigkeit derartiger Sicherheitsvorkehrungen ergibt sich aus dem Umgang von TKS mit Zirkelbezügen: Es ist völlig egal, ob die Rekursionstiefe erreicht wurde oder nicht, es wird in jedem Fall eine fest eingestellte Anzahl von Berechnungen durchgeführt. Diese kann der User bis zu einem gewissen Maximalwert einstellen. Das TKS stellt jedoch keinen Bezug zur Semantik der Rekursion her.

---

[32] Die Formel befindet sich in C11, die beiden Zahlen, von denen der ggT zu bestimmen ist, in C8 bzw. D8. Aus Gründen der Übersicht wurde die Formel vereinfacht widergegeben. Die Bezeichnungen a und b stehen für folgende Funktionen:
a = WERT(LINKS(C11;FINDEN("|";C11)-1));
b = WERT(RECHTS(C11;LÄNGE(C11)-FINDEN("|";C11)))
Ersetzt man im Quelltext a und b in entsprechender Weise erhält man die in Excel einzugebende Funktion. Eine Realisierung der Idee findet man unter:
http://www.ph-heidelberg.de/wp/gieding/mathematicadidaktica/

5.13 Rekursion und TKS, ein Fazit

TKS sind nur bedingt für komplexere rekursive Problemlösungen tauglich. Das Konzept rekursiven Arbeitens funktionaler Programmiersprachen ist in TKS nicht vollständig aufgehoben.
Hinsichtlich der Auswertung einer rekursiv definierten Funktion in einem Kalkulationsblatt hat der User zwei Möglichkeiten. Beide Möglichkeiten sind dadurch gekennzeichnet, dass der User die Rekursionen mehr oder weniger stark als imperative Schleifen verstehen muss. Auf dieser Grundlage kann er dann

- die Schleifendurchläufe durch spezielle Methoden des Kopierens und Einfügens selbst generieren,

oder

- die Zellen des Kalkulationsblattes so organisieren, dass das TKS die Schleifen ausführt.

Für letzteres stellen TKS die Methode der Zirkelbezüge zur Verfügung. Zirkelbezüge werden vom TKS stark formal ohne inhaltlichen Bezug zur angestrebten Rekursion mit einer festen Anzahl von Iterationen abgearbeitet.
Vor diesem Hintergrund erhellt sich, warum die imperative Sichtweise auf das Arbeiten mit TKS in bezug auf die Schleifen weniger problematisch als bei den Sequenzen und Verzweigungen war.

6 Programming by Example, eine Zusammenfassung

6.1 Aspekt 1 des Arbeitens mit Beispielen: Was wäre wenn

Entwurf von Kalkulationsblättern und Einsatz von Kalkulationsblättern zur Problemlösung ist Programmiertätigkeit. Zuvorderst ist diese Tätigkeit unter dem funktionalen Programmierparadigma zu verstehen: Jede Zelle beinhaltet eine Funktion, durch die Generierung eines Beziehungsgeflechts zwischen ausgewählten Zellen und die Verwendung von „eingebauten“ Funktionen werden neue Funktionen definiert. Die Auswertung dieser Funktionen liefert die angestrebte Problemlösung.
Die in 3.2 erläuterte Formelebene ließe sich aus dieser Sicht auch als Definitionsebene bezeichnen, während die Funktionsauswertung in

der Werte- bzw. Ausgabeebene stattfindet.
Die ständige Verfügbarkeit der Ausgabeebene ist ein Aspekt, der die Metapher „Programming by Example“ rechtfertigt. Im Gegensatz zur Verwendung einer Programmiersprache werden alle Funktionsauswertungen inklusive aller Zwischenschritte sofort und unmittelbar sichtbar gemacht. Dieses setzt natürlich voraus, dass die Eingabegrößen (konstante Funktionen) spezifiziert wurden[33]. Die Funktionsauswertung erfolgt dementsprechend für das Beispiel der Eingabegrößen.
Aus dieser unmittelbaren Rückkopplung zwischen Funktionsdefinition und Funktionsauswertung entsprechend der gewählten Beispieleingaben ergibt sich eine Möglichkeit experimentellen Arbeitens mit TKS: Was wäre wenn z.B. diese und jene Eingabebeispiele verwendet werden?

6.2 Aspekt 2 des Arbeitens mit Beispielen: Enaktivität

Das funktionale Programmierparadigma ist insofern nicht vollständig in TKS wieder zu finden, als die Möglichkeiten rekursiven Arbeitens gegenüber den Programmiersprachen stark eingeschränkt sind.
Rekursive Konstrukte werden in der Regel durch Schleifen des imperativen Programmierstils ersetzt. In die Abarbeitung der Schleifen ist der User integriert. Er organisiert die Schleifen durch tabellenkalkulationsspezifische Formen des Kopierens und Einfügens. Da andere Beispieleingaben in der Regel andere Anzahlen von Schleifendurchläufen erfordern, rechnet der User die Lösungen de facto mit Hilfe des TKS Beispiel für Beispiel durch.
Wie stark das Kalkulationsblatt auf das spezielle Beispiel ausgerichtet oder allgemeiner gestaltet wird, hängt von den Intentionen und TKS-Kenntnissen des Users ab.
Durch das mehr oder weniger starke beispielhafte Anwenden des Kalkulationsblattes lässt sich das Verständnis des Users für den Lösungsalgorithmus vertiefen.

---

[33] Ist dieses nicht der Fall, so werden die leeren Zellen als numerischer Wert 0 oder als leerer Text interpretiert (s. 5.2 „Alles ist Funktion“).

Insbesondere hinsichtlich der Vermittlung eines Verständnisses für spezielle Lösungsalgorithmen in der allgemeinbildenden Schule scheint das stärker konkret, beispielgebundene Arbeiten mittels TKS gegenüber dem Einsatz von eher abstrakteren Programmiersprachenversionen von didaktischer Bedeutung zu sein.

7 Literatur

/Brecht 1995/
Brecht, Werner: *Theoretische Informatik*, Vieweg, Braunschweig, Wiesbaden, 1995

/Gassee 1987/
Gassee Jean-Louis :*The Third Apple: Personal Computers & the Cultural Revolution*, Harcourt; 1987

/Gieding 2002/
Gieding, M.: *Bemerkungen zur informationstechnischen Grundbildung,* in: Abele, A., Selter, Ch. (Hrsg.): *Mathematikunterricht zwischen Tradition und Innovation,* Deutscher Studien Verlag, Weinheim, 2002

/Gieding 2003/
Gieding, M.: *Didaktik der Tabellenkalkulation!?*, in: *Beiträge zum Mathematikunterricht 2003*, Franzbecker, Hildesheim, Berlin, 2003.

/Knuth 1973/
Donald, E. Knuth: *Fundamental Algorithms*; Addison-Wesley, 1973

/Neuwirth 1995/
Neuwirth, Erich: *Tabellenkalkulation als alternative* Darstellungsform *für formale Strukturen*, in H.C. Reichel: *Computereinsatz im Mathematikunterricht*, Spektrum, Heidelberg, 1995

/Pepper 2000/
Pepper, Peter: *Funktionale Programmierung in OPAL, ML, HASKELL und GOFER*, Springer, Berlin, Heidelberg 2000

/Weigand 2001/
H.-G. Weigand: *Diskrete Mathematik und Tabellenkalkulation*, in: *Der Mathematikunterricht*, Jahrgang 47, Heft 3 /Juni 2001

/Ziegenbalg 1996/
Ziegengalg, J.: *Algorithmen – von Hammurapi bis Gödel*; Spektrum, Heidelberg, *Berlin*, Oxford 1996
/Ziegenbalg 2000/
Ziegenbalg J.: *Algorithmen - fundamental für Mathematik, Mathematikunterricht und mathematische Anwendungen*, Vortragsreihe Dresdner Kolloquium zur Mathematik und ihrer Didaktik, Heft 5, Nr. 20, 1 – 28

Anschrift des Verfassers
Michael Gieding
Schwetzinger Str. 136
69124 Heidelberg
gieding@ph-heidelberg.de
http://www.ph-heidelberg.de/wp/gieding/