Werner Rockenbach
odswer@uni-koblenz.de
Herzog-Johann-Gymnasium
55469 Simmern

10. September 2000

Beispiele zur funktionalen Programmierung in CAML

Im Lehrplanentwurf "Informationstechnische Grundbildung (Gymnasium)" wird gefordert, dass im Sekundarbereich I eine informationstechnische Grundbildung vermittelt wird. Die Inhalte werden dabei in das Lernangebot der bestehenden Fächer eingebettet und integriert.
In der Mathematik werden u.a. einfache Algorithmen in lauffähige Programme umgesetzt. Die Entscheidung über die Programmiersprache liegt beim Fachlehrer. Mit Hilfe der funktionalen Programmiersprache CAML können die mathematischen Algorithmen umgesetzt werden. CAML (Categorical Abstract Machine Language) unterstützt dabei den mathematischen Funktionsbegriff.
Eine Behandlung der funktionalen Programmierung ist im regulären Informatikunterricht in mehreren Phasen möglich. In der Einstiegsphase (Klassenstufe 11) kann eine funktionale Programmiersprache als Modellierungswerkzeug benutzt werden. Für die Programmierphase (Klassenstufe 11) wird im Lehrplan eine imperative Programmiersprache vorgesehen, andere Zugänge werden aber zugelassen. Schließlich kann das Abschlussprojekt so gewählt werden, dass Aspekte der funktionalen Programmierung behandelt werden können. Im Bereich der theoretischen Informatik (Berechenbarkeit) kann eine funktionale Programmiersprache zur Programmierung der rekursiven Funktionen sinnvoll eingesetzt werden.
Mit Hilfe von Funktionen werden (komplexe) Berechnungsverfahren beschrieben. Bei der Konstruktion der Berechnungsvorschrift werden die Konstruktionsprinzipien Komposition, Fallunterscheidung und Rekursion benutzt. Die funktionale Programmierung eines Problems liegt näher an der logischen Sicht, während die Programmierung in einer imperativen Sprache näher an der Hardware erfolgt. Die funktionale Programmierung verzichtet auf die Wertzuweisung. Funktionen dienen dazu, Ein- und Ausgabesituationen zu spezifizieren. Mit Hilfe einer Zuordnung wird die Ein- und Ausgabesituation beschrieben. Im Vordergrund steht der Zuordnungsaspekt, wobei den Eingabedaten die Ausgabedaten zugeordnet werden. Mit Hilfe einer mathematischen Definition wird das Ein- oder Ausgabeverhalten präzise festgelegt. Die Festlegung muss ausführbar sein.
CAML-light ist eine leicht portierbare, typisierte funktionale Sprache. Das komplette CAML-light-System kann für den privaten Gebrauch kostenlos vom Server der INRIA (ftp://ftp.inria.fr/lang/caml-ligth/) geladen werden Es gibt Versionen für LINUX, DOS, MS-WINDOWS und Macintosh-Systeme. Zusatzinformationen findet man im WWW unter http://pauillac.inria.fr.

Der Algorithmenentwurf ist ein wesentlicher Teil des Lösungsvorgangs zu einem Problem. Den Problemlösungsprozess kann man in vier Schritte unterteilen:

(1) Analyse des Problems und genaue Darstellung (Spezifikation)
(2) Herausfinden eines Lösungsweges und Formulierung eines Algorithmus
(3) Übersetzung des Problems in eine computerverständliche Sprache
(4) Einsatz des Computers zur Erstellung der Lösung

1. Prozentrechnung

Bei der Prozentrechnung geht es stets um die Berechnung einer der drei Größen Prozentwert, Grundwert oder Prozentsatz. Dabei ergeben sich für die Berechnung der einzelnen Größen drei verschiedene Gesetzmäßigkeiten:

$$Prozentwert = Grundwert \bullet \frac{Prozentsatz}{100}$$

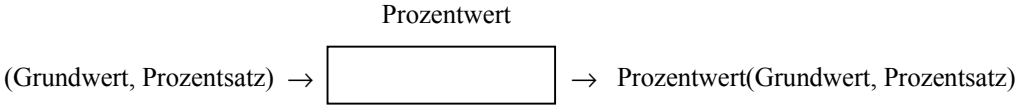
$$Grundwert = Prozentwert \bullet \frac{100}{Prozentsatz}$$

$$Prozentsatz = \frac{Prozentwert}{Grundwert} \bullet 100$$

Nach einer unterrichtlichen Behandlung (Klassenstufe 7) eignet sich die Prozentrechnung zur Erstellung linearer Programme. In der ITG (Informationstechnische Grundbildung) soll an einfachen Beispielen exemplarisch an die Erstellung von Algorithmen und deren Programmierung herangeführt werden.

1.1. Berechnung Prozentwert

Modellierung (Berechnung Prozentwert)



Definition:

$$Prozentwert(Grundwert, Prozentsatz) = Grundwert \bullet \frac{Prozentsatz}{100}$$

CAML-Programm:

```
let prozentsatz = 5.00;;
let prozentwert = function
        grundwert  -> grundwert *. prozentsatz /. 100.0;;
```

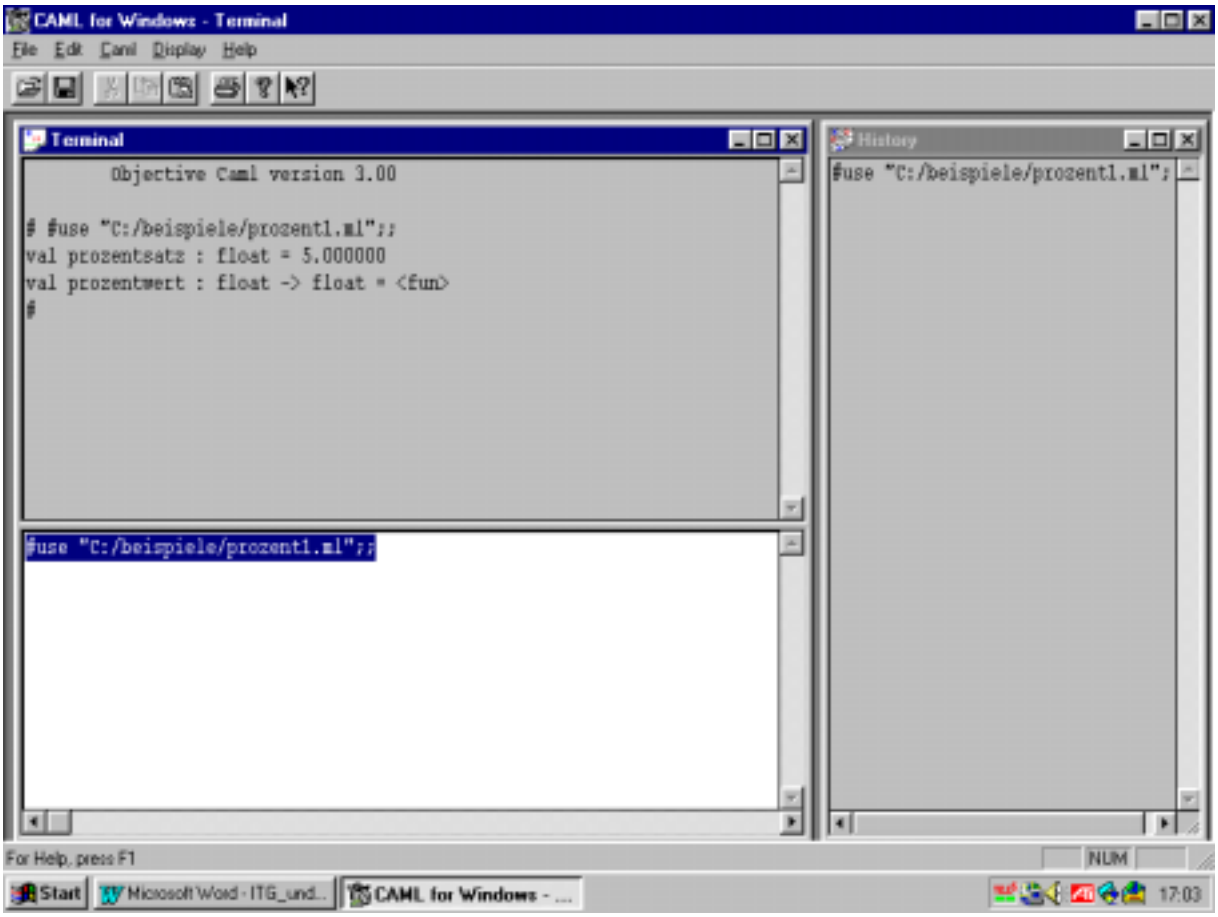
Hierbei wird der Prozentsatz als Konstante festgelegt. Für das Rechnen mit Gleitkommazahlen werden die Rechenzeichen mit Punkt geschrieben, beim Rechnen mit ganzen Zahlen werden die Rechenzeichen wie üblich geschrieben.

Rechen mit Gleitkommazahlen:	
Addition:	+.
Subtraktion:	-.
Multiplikation:	*.
Division:	/.

Rechnen mit ganzen Zahlen:	
Addition:	+
Subtraktion:	-
Multiplikation:	*
Division:	/
ganzzahliger Rest	mod

Die Definition einer Funktion wird mit ;; abgeschlossen.
Die Funktionsdefinitionen werden mit einem Texteditor erstellt und in einer eigenen Datei mit der Erweiterung .ml abgespeichert. Im CAML-System erfolgt der Aufruf mit dem CAML-Befehl "include" (CAML-Menü) wobei das System eine Rückmeldung liefert.

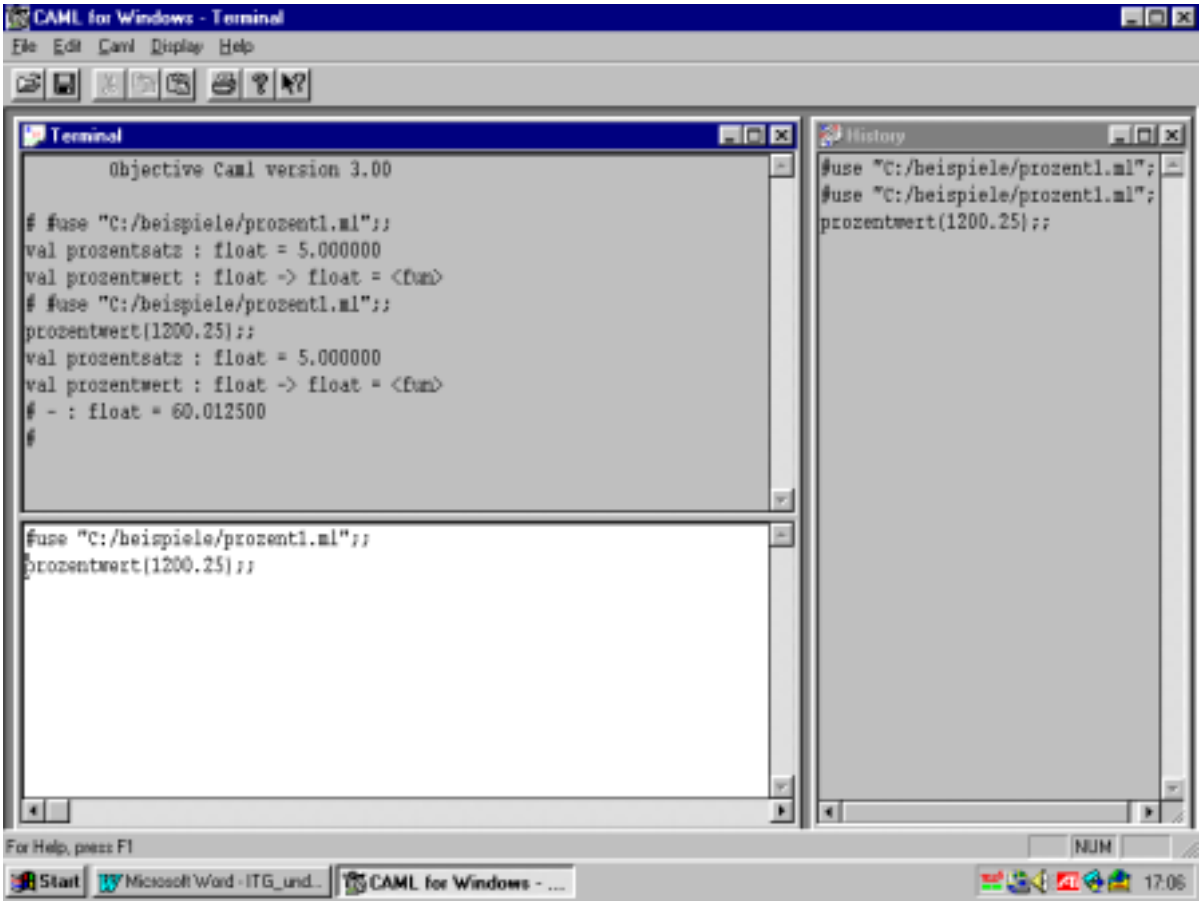
Meldung CAML-System:



Das System liefert eine Kurzbeschreibung der definierten Konstanten (erkennbar am Gleichheitszeichen) und Funktionen (erkennbar am Zusatz <fun>) mit der Definitions- und Wertemenge. Abschluss bildet das Promptzeichen (#).
Mit Hilfe der definierten Funktionen erfolgt nun die Berechnung. Im unteren Fenster im CAML-System gibt man den Funktionsaufruf ein und erhält im oberen Fenster die entsprechende Rückmeldung.

Beispiel:

```
prozentwert(1200.25);;

prozentwert(1200.25);;
- : float = 60.012500
#
```



Eine Konstante wird nach folgender Syntax formuliert:

let <Konstantenname> = Konstante

Die Funktionen werden nach folgender Syntax formuliert:

let <Funktionsname> = function
<Argumentname> -> <Berechnung des Resultats>

Treten mehrere Argumente auf, so sind diese Argumente in Klammern zu setzen, sie entsprechen so einem Argument aus dem als kartesisches Produkt zusammengefassten Definitionsbereich.

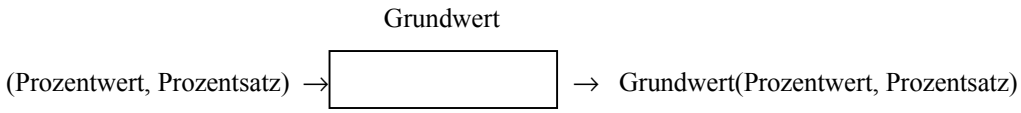
Die Konstante Prozentsatz und die Funktion Prozentwert können auch zu einer Funktion zusammengefasst werden.

```
let Prozentwert = function
  (Grundwert, Prozentsatz) -> Grundwert *. Prozentsatz /. 100.0;;
```

Aufruf: prozentwert(100.25, 5.5);;

1.2. Berechnung Grundwert

Modellierung (Berechnung Grundwert)



Definition:

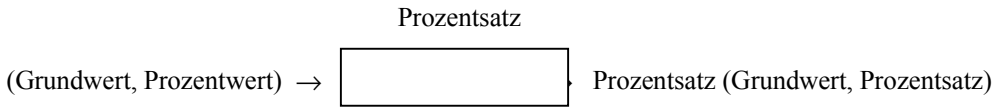
$$Grundwert(Prozentwert, Prozentsatz) = Prozentwert \bullet \frac{100}{Prozentsatz}$$

CAML-Programm

```
let Grundwert = function
  (Prozentwert, Prozentsatz) -> Prozentwert *. 100.0 /. Prozentsatz;;
```

1.3. Berechnung Prozentsatz

Modellierung (Berechnung Grundwert)



Definition

$$Prozentsatz(Grundwert, Prozentwert) = \frac{Prozentwert}{Grundwert} \bullet 100$$

CAML-Programm

```
let Prozentsatz = function
(Grundwert, Prozentwert) -> Prozentwert /. Grundwert *.100;;
```

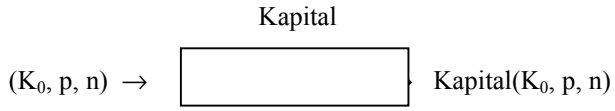
1.4. Zinseszinsrechnung

Ein Kapital von 3000,00 DM wird mit einem Zinssatz von 7,5 % verzinst. Auf welchen Betrag wächst es in 6 Jahren mit Zinsen und Zinseszins an?

	Guthaben [DM]	Zinssatz	Zinsen [DM]	neues Guthaben [DM]
1. Jahr	3000,00	7,5 %	225,00	3225,00
2. Jahr	3225,00	7,5 %	241,88	3466,88
3. Jahr	3466,88	7,5 %	260,02	3726,90
4. Jahr	3726,90	7,5 %	279,52	4006,42
5. Jahr	4006,42	7,5 %	300,48	4306,90
6. Jahr	4306,90	7,5 %	323,02	4629,92

Modellierung (Zinseszins)

K_0: Anfangskapital
p: Zinssatz
n: Laufzeit in Jahren
K_n: Endkapital



Definition (Rekursion)
Die Zinseszinsrechnung kann mit folgender Rekursion beschrieben werden:

$$K_n = \begin{cases} K_0 & \text{für } n = 0 \\ K_{n-1} * (1 + p / 100) & \text{für } n > 0 \end{cases}$$

allgemeine Beschreibung:

$$\text{Kapital} = \begin{cases} K_0 & \text{für } n = 0 \\ (1+p/100)*\text{Kapital}\ (K_0, p, n-1) & \text{sonst} \end{cases}$$

CAML-Programm

```
let rec kapital = function
   (k,p,0) -> k
   | (k,p,n) -> kapital(k,p,n-1) *.(1.0 +. p /. 100.0);;
```

Der Aufruf kapital(3000.00, 7.5, 6);; liefert das Ergebnis.
Das Problem wird in einer imperativen Programmiersprache mit Hilfe einer Schleife gelöst. Eine funktionale Programmiersprache kommt grundsätzlich ohne Schleifen aus. Wiederholungen werden mittels der Rekursion formuliert.
Aus Effizienzgründen sind in CAML-LIGHT die gezählte und die abweisende Wiederholung vorhanden.
Nach Möglichkeit sollte bei der funktionalen Programmierung auf Schleifen als Strukturierungsmittel verzichtet werden.

Die Definition einer rekursiven, sich selbst aufrufenden Funktion wird wie folgt definiert:

```
let   rec  <Funktionsname> = function
        <Argumentliste mit Rekursionsausgang> ->  <Berechnung Rekursionsausgang>
        |  <Argumentliste >  ->  <Berechnung Rekursion>
```

2. Rekursionen

Im Folgenden sind **x** und **y** (positive) ganze Zahlen.

2.1. Summe (x, y)

$$\text{summe}(x, y) = \begin{cases} x & \text{für } y = 0 \\ \text{summe}(x, y-1) + 1 & \text{für } y > 0 \end{cases}$$

CAML-Programm:
```
let rec summe = function
        (x, 0) ->  x
        | (x, y) -> summe(x, y-1) + 1;;
```

Das Programm kann auch mit if-then-else formuliert werden.

CAML-Programm:
```
let rec summe = function
        (x, y) -> if y = 0 then x else summe(x, y-1) + 1;;
```

Aufruf: `summe(2, 10);;`

2.2. Produkt(x, y)

$$\text{produkt}(x, y) = \begin{cases} x & \text{für } y = 1 \\ \text{produkt}(x, y-1) + x & \text{für } y > 1 \end{cases}$$

CAML-Programm:
```
let rec produkt = function
        (x, 1) ->  x
        | (x, y) -> produkt(x, y-1) + x;;
```

Aufruf: `produkt(12,16);;`

Das Programm kann auch mit indirekter Rekursion formuliert werden.

CAML-Programm:
```
let rec summe = function
        (x, 0) -> x
        | (x, y) -> summe(x, y-1) + 1;;

let rec produkt = function
        (x, 1) -> x
        | (x, y) -> summe(produkt(x, y-1), x);;
```

Aufruf: `produkt(12, 18);;`

2.3. Potenz(x, y)

$$\text{potenz}(x, y) = \begin{cases} 1 & \text{für } y = 0 \\ \text{potenz}(x, y-1) * x & \text{für } y > 0 \end{cases}$$

CAML-Programm:
```
let rec potenz = function
        (x, 0) -> 1
        | (x, y) -> potenz(x, y-1) *x;;
```

Aufruf: `potenz(2, 5);;`

Die Potenz kann auf die Multiplikation zurückgeführt werden.

CAML-Programm:
```
let rec produkt = function
        (x, 1) ->  x
        | (x, y) -> produkt(x, y-1) + x;;

let rec potenz = function
        (x, 0) -> 1
        | (x, y) -> produkt(potenz(x, y-1), x);;
```

Aufruf: `potenz(2, 8);;`

Die Potenz kann auf die Multiplikation, die Multiplikation auf die Addition zurückgeführt werden.

CAML-Programm:
```
let rec summe = function
        (x, 0) -> x
        | (x, y) -> summe(x, y-1) +1;;

let rec produkt = function
        (x, 1) -> x
        | (x, y) -> summe(produkt(x, y-1), x);;

let rec potenz = function
        (x, 0) -> 1
        | (x, y) -> produkt(potenz(x, y-1), x);;
```

Aufruf: `potenz(2, 10);;`

2.4. Fakultät(x)

$$\text{fakultaet}(x) = \begin{cases} 1 & \text{für } x = 0 \\ \text{fakultaet}(x-1) * x & \text{für } x > 0 \end{cases}$$

CAML-Programm:
```
let rec fakultaet = function
        0 -> 1
        | x -> fakultaet(x-1)*x;;
```

Aufruf: fakultaet(6);;

Die Multiplikation in der Fakultaet kann auf die Addition zurückgeführt werden.

CAML-Programm:
```
let rec produkt = function
        (x, 1) -> x
        | (x, y) -> produkt(x, y-1) + x;;

let rec fakultaet = function
        0 -> 1
        | x -> produkt(fakultaet(x-1), x);;
```

Aufruf: fakultaet(6);;

Die Multiplikation kann rekursiv mit der Summe ausgedrückt werden.

CAML-Programm:
```
let rec summe = function
        (x, 0) -> x
        | (x, y) -> summe(x, y-1) +1;;

let rec produkt = function
        (x, 1) -> x
        | (x, y) -> summe( produkt(x, y-1), x);;

let rec fakultaet = function
        0 -> 1
        | x -> produkt(fakultaet(x-1), x);;
```

Aufruf: fakultaet(5);;

2.5. Ackermann-Funktion

Die Ackermann-Funktion ist eine nicht primitiv-rekursive Funktion. Die Idee der Ackermann-Funktion besteht darin, eine berechenbare Funktion zu definieren, welche schneller wächst als alle primitiv-rekursiven Funktionn. Zu immer stärker wachsenden Funktionen führt die Folge: Summe, Produkt, Potenz. Da die Potenz aus dem Produkt auf ähnliche Weise entsteht, wie das Produkt aus der Summe, so lässt sich dieser Prozess weiterführen. Man kommt so zu einer Hyperpotenz.

Definition:
$$A(0, y) = y + 1$$
$$A(x, 0) = A(x - 1, 1)$$
$$A(x, y) = A(x - 1, A(x, y - 1))$$

CAML-Programm:
```
let rec acker = function
    (x,y) -> if x=0 then y+1
                    else if y=0 then acker(x-1,1)
                                else acker(x-1,acker(x,y-1));;
```

Aufruf: acker(3,2);;

Ergebnis: 29

3. Beispiele aus der Geometrie

3.1. Tetraeder

Ein Tetraeder ist eine Pyramide, deren Grund- und Seitenflächen gleichseitige Dreiecke sind. Die Oberfläche setzt sich aus der Summe der vier Dreiecksflächen zusammen.

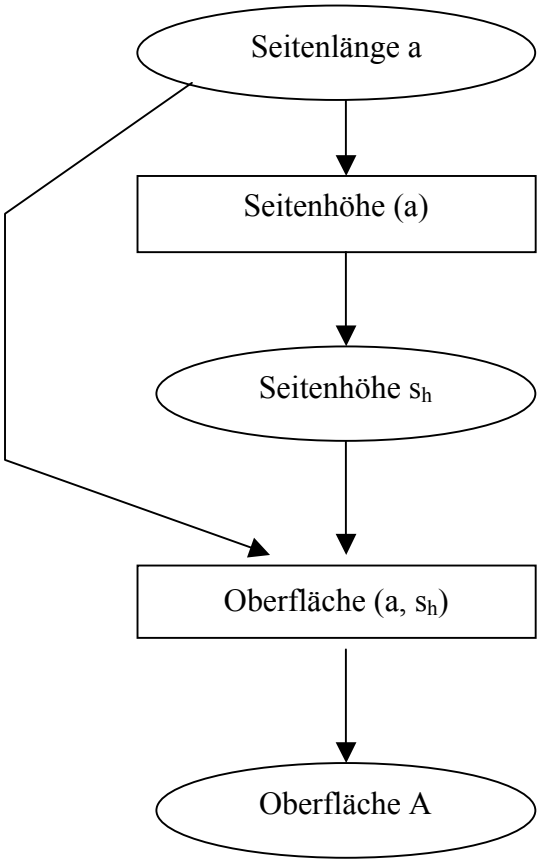
3.1.1. Oberfläche Tetraeder

Modellierung:

Seitenhöhe s_h: $s_h = \frac{a}{2}\sqrt{3}$

Seitenfläche A: $A = \frac{1}{2} a \bullet s_h = \frac{1}{4} a^2 \sqrt{3}$

Oberfläche A_O: $A_O = 4 \bullet A = a^2 \sqrt{3}$



CAML-Programm:
```
let  seitenhoehe = function
        a -> a /. 2.0 *. sqrt(3.0);;

let flaeche = function
        a -> 0.5 *. a *. seitenhoehe(a);;
```

Aufruf: flaeche(4.5);;

3.1.2. Volumen Tetraeder

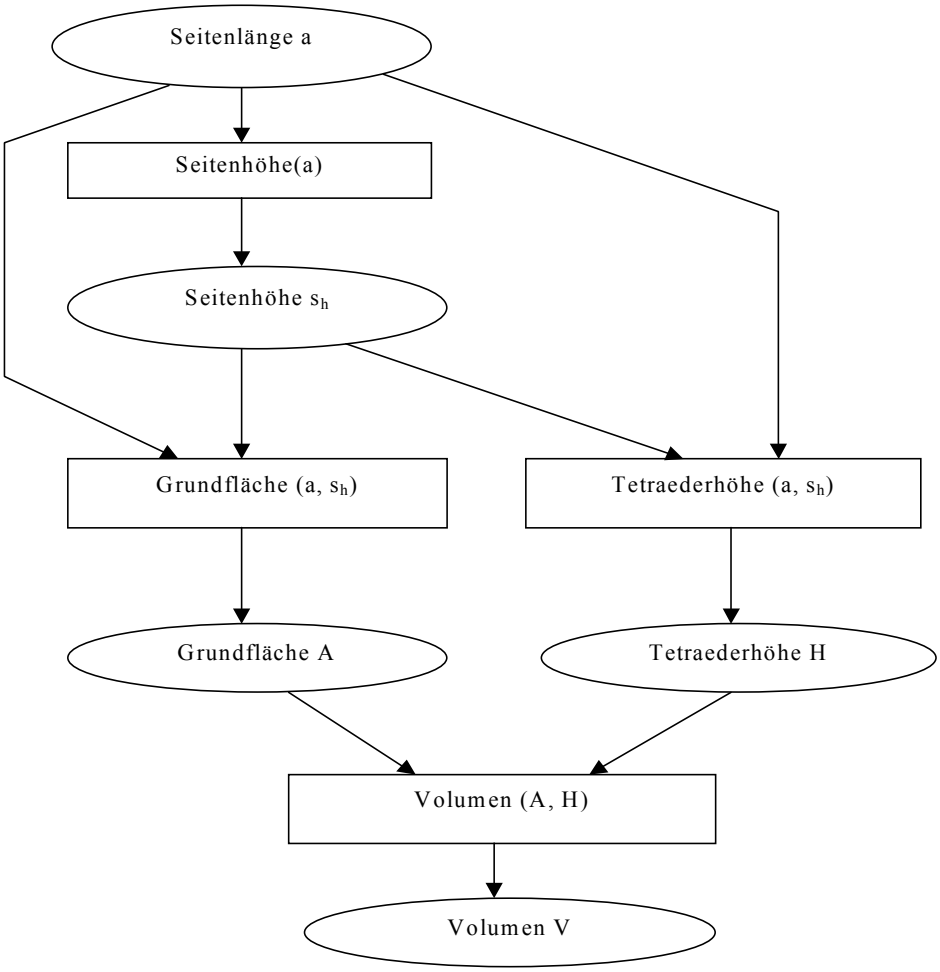
Modellierung:

Seitenhöhe s_h: $s_h = \frac{a}{2}\sqrt{3}$

Grundfläche A: $A = \frac{1}{2} a \bullet s_h = \frac{1}{4} a^2 \sqrt{3}$

Höhe Tetraeder H: $H = \sqrt{a^2 - \left(\frac{2}{3} s_h\right)^2} = a\sqrt{\frac{2}{3}}$

Volumen V: $V = \frac{1}{3} A \bullet H = \frac{1}{12} a^3 \sqrt{2}$



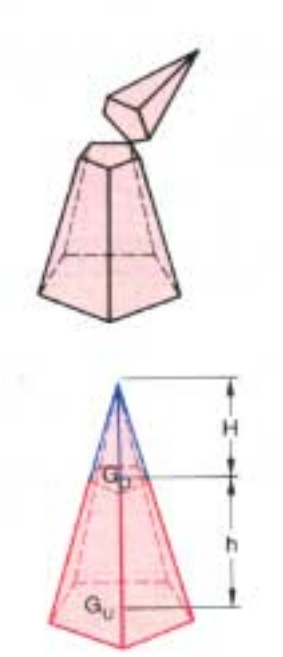
CAML-Programm:
```
let sqr = function
        x -> x *. x;;

let  seitenhoehe = function
        a -> a /. 2.0 *. sqrt(3.0);;

let flaeche = function
        a -> 0.5 *. a *. seitenhoehe(a);;

let tetraederhoehe = function
        a ->  sqrt(sqr(a) -. sqr(0.67 *. seitenhoehe(a)));;

let volumen = function
        a -> 0.33 *. flaeche(a) *. tetraederhoehe(a);;
```

Aufruf:
```
volumen(4.5);;
```

3.2. Pyramidenstumpf

Schneidet man von einer Pyramide die Spitze parallel zur Grundfläche ab, so erhält man einen Pyramidenstumpf.

3.2.1. Volumen Pyradmidenstumpf



Hieraus ergibt sich die Volumenformel für den Pyramidenstumpf.

$V_{Stumpf} = V_{gesamt} - V_{Spitze}$

$$V_{Stumpf} = \frac{1}{3} G_u (H+h) - \frac{1}{3} G_0 H$$

$$= \frac{1}{3}\left(G_u h + (G_u - G_0)H\right)$$

Nach den Sätzen über zentrische Streckungen gilt:

$$\frac{G_u}{G_0} = k^2 \quad \text{mit} \quad k = \frac{H+h}{H}$$

Also: $$\frac{G_u}{G_0} = \left(\frac{H+h}{H}\right)^2$$

Auflösung nach H: $$H = \frac{\sqrt{G_0}}{\sqrt{G_u} - \sqrt{G_0}} \bullet h$$

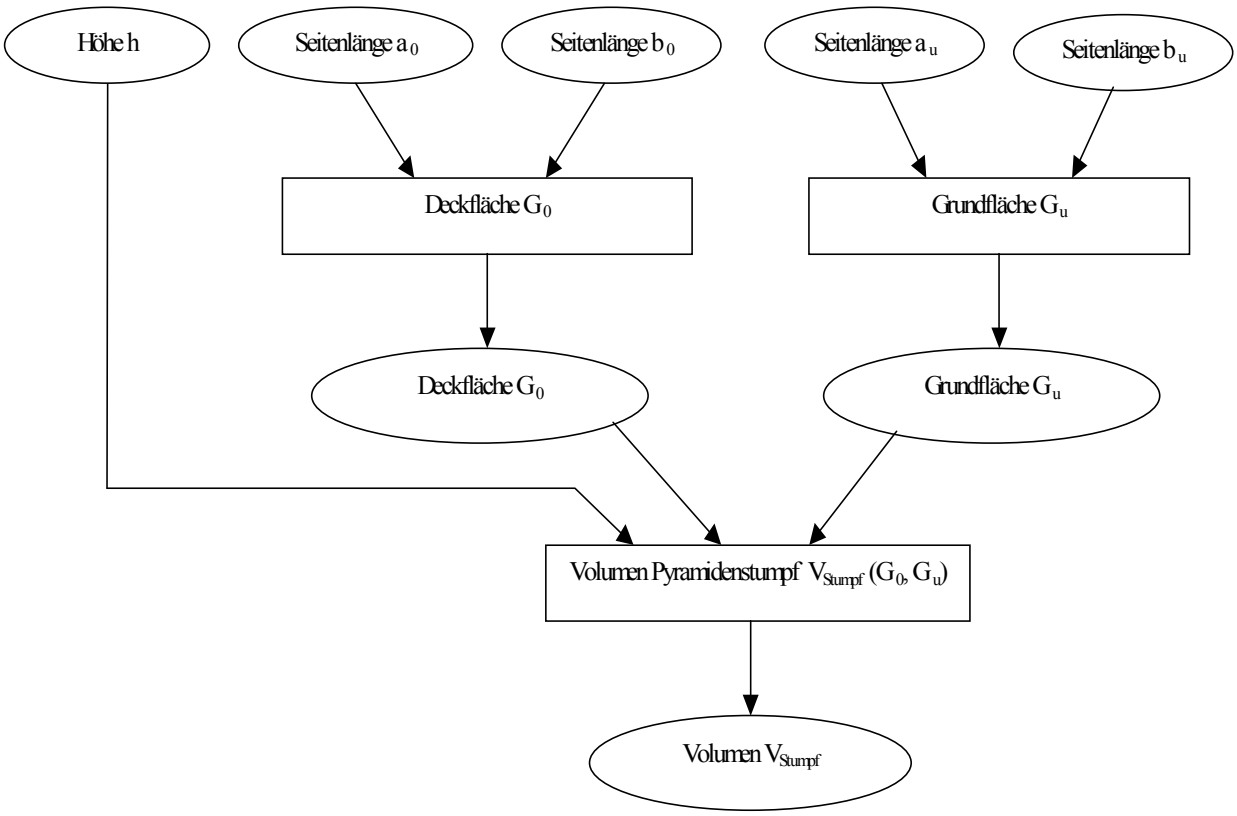
Diesen Term für H kann man in die Volumenformel einsetzen:

$$V_{Stumpf} = \frac{1}{3} \bullet \left[G_u \bullet h + (G_u - G_0) \bullet \frac{\sqrt{G_0}}{\sqrt{G_u} - \sqrt{G_0}} \bullet h \right]$$

$$= \frac{1}{3} \bullet \left[G_u \bullet h + \frac{\left(\sqrt{G_u} + \sqrt{G_0}\right) \bullet \left(\sqrt{G_u} - \sqrt{G_0}\right) \bullet \sqrt{G_0} \bullet h}{\sqrt{G_u} - \sqrt{G_0}} \right]$$

$$= \frac{1}{3} \bullet h \bullet \left[G_u + \sqrt{G_u} \bullet \sqrt{G_0} + G_0 \right]$$

Modellierung Volumen Pyramidenstumpf mit den Rechteckflächen G_u und G_0.



CAML-Programm:

```
let deckflaeche = function
        (a0, b0) -> a0 *. b0;;

let grundflaeche = function
        (au, bu) -> au *. bu;;

let pyramidenstumpf = function
      (au, bu, a0, b0, h) ->  0.33 *. h *.
                              (grundflaeche(au, bu) +.
                               sqrt(grundflaeche(au, bu)) *.
                               sqrt(deckflaeche(a0, b0)) +.
                               deckflaeche(a0, b0));;
```

Aufruf: pyramidenstumpf(4.0, 5.0, 2.0, 3.0, 10.0);;

Ergebnis: 121.499689

Mit diesem Programm können bei geringfügigen Änderungen der Funktionen zur Berechnung der Deck- und der Grundfläche das Kegelstumpfvolumen sowie das Pyramidenstumpfvolumen mit beliebiger Querschnittsfläche berechnet werden.

Literatur

(1) Becker
Materialien zum Informatikunterricht - Funktionale Programmierung
Landesmedienzentrum Rheinland-Pfalz, 1999

(2) Gudenberg
Algorithmen, Datenstrukturen, Funktionale Programmierung
Eine praktische Einführung mit CAML LIGHT
Bonn, Addison-Wesley 1996