

- Die erste Fassung von METAFONT wurde von Donald E. Knuth 1979 entwickelt, 1984 erschien eine revidierte, sehr viel elegantere Fassung.
- METAFONT erzeugt ein Raster mit einer einstellbaren Auflösung, das nur schwarze und weiße Pixel kennt.
- Das bedeutet, dass Farben nicht unterstützt werden und die Ausgabe von METAFONT nicht mehr skalierbar ist. Da Drucker METAFONT-Programme nicht ausführen können, muss im Vorfeld über die Auflösung entschieden werden.
- METAPOST wurde 1989-1994 von John D. Hobby entwickelt und bietet einen ähnlichen Sprachumfang wie METAFONT an, generiert aber PostScript.
- Das bedeutet, dass die Ausgabe skalierbar bleibt und direkt an einen PostScript-Drucker weitergereicht werden kann. Farben und Clipping werden unterstützt, rasterbezogene Operatoren fallen jedoch weg.

- Donald E. Knuth im METAFONTbook:

The 'META-' part is more interesting: It indicates that we are interested in making high-level descriptions that transcend any of the individual fonts being described.

[...]

Meta-design is much more difficult than design; it's easier to draw something than to explain how to draw it. One of the problems is that different sets of potential specifications can't be easily be envisioned all at once. Another is that a computer has to be told absolutely everything. However, once we have successfully explained how to draw something in a sufficiently general manner, the same explanation will work for related shapes, in different circumstances; so the time spent in formulating a precise explanation turns out to be worth it.

- Der erste Namensteil META von METAFONT und METAPOST steht also dafür, dass eine Quelle eine ganze Familie von Schriftschnitten (oder Diagrammen) generieren kann.
- Im Unterschied zu PostScript sind METAFONT und METAPOST primär Quellformate für die Definition von Schriftschnitten und Zeichnungen.
- Viele PostScript-Schriftschnitte werden mit speziellen Werkzeugen entwickelt, die dann Type-1-Schriftschnitte generieren.
- Die Motivation von Donald E. Knuth für einen anderen Ansatz ergibt sich daraus, dass Schriftschnitte nicht mit akzeptablen Ergebnis auf triviale Weise skalierbar sind. Zahlreiche Parameter sind zu verändern, wenn besonders kleine oder große Schriftschnitte zu generieren sind.
- Bei PostScript gibt es eine ähnliche Möglichkeit nur über die sogenannten *hints* in den Type-1-Schriftschnitten.

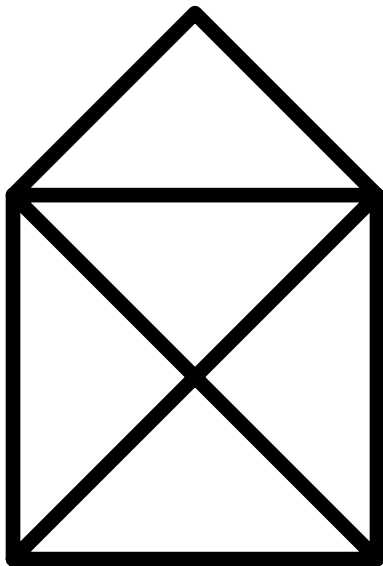
- Donald E. Knuth, *The METAFONTbook*, Addison-Wesley, 1986, ISBN 0-201-13445-4
- John D. Hobby, *A User's Manual for METAPOST*, <http://www.tug.org/docs/metapost/mpman.pdf>
- Alan Hoenig, *T_EX Unbound*, Kapitel 13, *Using METAFONT and METAPOST*, 1998, ISBN 0-19-509686-X
- Michael Goossens et al, *The L^AT_EX Graphics Companion*, Second Edition, Kapitel 3 und 4: *METAFONT and METAPOST: T_EX's Mates, METAPOST Applications*, 2007, ISBN 0-321-50892-0

- METAFONT dient in erster Linie dazu, Schriftschnitte zu definieren.
- METAPOST unterstützt auch diesen Einsatzzweck (und erlaubt somit auch die skalierbare Einbettung von in METAFONT formulierten Schriftschnitten in PostScript).
- Im Gegensatz zu METAFONT unterstützt aber METAPOST die Verwendung von Schriften und ermöglicht damit die elegante Spezifikation von Diagrammen mit eingebetteten Texten.
- Als Vorbild diente hier auch das 1982 von Brian W. Kernighan entwickelte *pic*, das als Filter die Generierung von Diagrammen für *troff* unterstützte.
- Da METAPOST EPS-Dateien erzeugt, lässt es sich von jedem Text-System aus verwenden, das die Integration von EPS-Dateien ermöglicht.

house.mp

```
% Ein Haeusle mit 8 Strichen
prologues := 1;
beginfig(1);
  l := 1in; % Masseinheit fuer das Haeusle
  pickup pencircle scaled 3pt;
  draw (0,0) -- (0,1l) -- (0.5l,1.5l) -- (1l,1l) --
        (0,1l) -- (1l,0) -- (0,0) -- (1l,1l) -- (1l,0);
endfig;
end.
```

- Die Syntax erinnert an Pascal. Kommentare beginnen mit „%“.
- Die Zuweisung `prologues := 1;` sorgt dafür, dass die speziellen PostScript-Kommentare erzeugt werden, die im Einklang mit den PostScript-Strukturierungskonventionen stehen.
- Mit einem METAPOST-Programm lässt sich eine Vielzahl von EPS-Dateien erzeugen. `beginfig(1)` .. `endfig` umschließt das erste zu generierende Diagramm.
- Mit `end` wird das METAPOST-Programm beendet.



house.mp

```
l := 1in; % Masseinheit fuer das Haeussle  
pickup pencircle scaled 3pt;
```

- Die Maßeinheiten entsprechen denen von PostScript. Es können aber auch diverse vordefinierte Maßeinheiten verwendet werden wie *pt* ($\frac{1}{72.27}$ Inch), *in* (Inch) oder *cm* (Zentimeter). Zusätzlich können (wie hier mit *l*) eigene Maßeinheiten definiert werden.
- Mit *pickup* wird ein neuer Zeichenstift definiert. Anders als in PostScript kann METAPOST beliebige geschlossene konvexe Pfade dafür nehmen. Hier liefert *pencircle scaled 3pt* einen kreisförmigen Zeichenstift mit einem Durchmesser von 3 Punkt. (Allerdings können nicht direkt Pfade an *pickup* übergeben werden, sie müssen noch zuvor mit der *makepen*-Funktion in einen Stift konvertiert werden. Bei *pencircle* ist dies bereits geschehen.)

`house.mp`

```
draw (0,0) -- (0,11) -- (0.51,1.51) -- (11,11) --  
      (0,11) -- (11,0) -- (0,0) -- (11,11) -- (11,0);
```

- *draw* akzeptiert einen Pfad als Operanden und zeichnet diesen mit dem aktuellen Zeichenstift.
- Pfade können u.a. auch das Verbinden von Punkten mit geraden Strichen (unter Verwendung des Operators --) konstruiert werden.
- Die Idee eines aktuellen Pfades (wie bei PostScript) gibt es nicht. Stattdessen sind Pfade ganz normale Objekte, die in Variablen abgelegt werden können und auf die sich eine Reihe von Operatoren beziehen.

```
theon$ ls
house.mp
theon$ mpost house.mp
This is MetaPost, version 2.00 (TeX Live 2018) (kpathsea version 6.3.0)
(/opt/ulm/ballinrobe/texlive/texmf-dist/metapost/base/mpost.mp
(/opt/ulm/ballinrobe/texlive/texmf-dist/metapost/base/plain.mp
Preloading the plain mem file, version 1.005) ) (./house.mp [1{psfonts.map}] )
1 output file written: house.1
Transcript written on house.log.
theon$ ls
house.1   house.log  house.mp
theon$
```

- Der METAPOST-Interpreter wird mit *mpost* aufgerufen.
- Dieser wird interaktiv, falls kein *end* in der Quelle vorkommt und nicht beim Aufruf von *mpost* mit der Option „-interaction batchmode“ auf Interaktionen verzichtet wurde.
- Für jede in *beginfig..endfig* eingeschlossene Zeichnung wird eine Ausgabedatei erzeugt, deren Namen mit dem Basisnamen der Quelle beginnt, gefolgt von einem Punkt und der Nummer der Zeichnung. Die Datei-Endung „.eps“ wird nicht angefügt.

```

theon$ mpost error.mp
This is MetaPost, version 2.00 (TeX Live 2018) (kpathsea version 6.3.0)
(/opt/ulm/ballinrobe/texlive/texmf-dist/metapost/base/mpost.mp
(/opt/ulm/ballinrobe/texlive/texmf-dist/metapost/base/plain.mp
Preloading the plain mem file, version 1.005) ) (./error.mp
>> 0
! Improper 'addto'.
<to be read again>
                withpen
draw->...:also(EXPR0)else:doublepath(EXPR0)withpen
                                                .currentpen.fi._op_
<to be read again>
                {
--->{
        curl1}..{curl1}
1.6   draw (0) --
                (0,11) -- (0.51,1.51) -- (11,11) --
? X
Transcript written on error.log.
theon$

```

- Wenn $(0,0)$ innerhalb der Pfadkonstruktion durch (0) ersetzt wird, kommt es zu obigem Fehler.

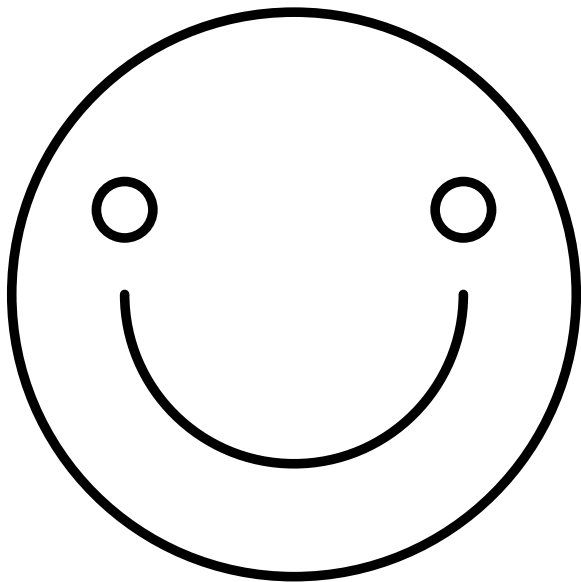
```
1.6      draw (0) --  
                (0,11) -- (0.51,1.51) -- (11,11) --  
? X  
Transcript written on error.log.  
theon$
```

- Die Fehlerstelle lässt sich ganz unten erkennen. „1.6“ steht für die Zeile 6 und die genaue Position lässt sich an dem Zeilenumbruch erkennen.
- METAPOST wird bei Fehlern interaktiv, es sei denn, dass dies von der Kommandozeile ausdrücklich ausgeschlossen wurde.
- Mit einer Eingabe von „X“ kann die METAPOST-Sitzung beendet werden. Ansonsten besteht die Möglichkeit, u.U. den Fehler interaktiv vorläufig zu korrigieren.

smiley.mp

```
% Ein Smiley
prologues := 1;
beginfig(1);
  l := 2in; % Radius des Gesichts
  pickup pencircle scaled 5pt;
  path circle;
  circle := (-1,0) .. (0,1) .. (1,0) .. (0,-1) .. cycle;
  draw circle scaled l; % Gesicht
  draw (-0.6l,0) .. (0,-0.6l) .. (0.6l,0); % Mund
  % linkes Auge
  draw circle scaled 0.1l shifted (-0.6l,0.3l);
  % rechtes Auge
  draw circle scaled 0.1l shifted (+0.6l,0.3l);
endfig;
end.
```

- Man beachte, dass sich *scaled 0.1l* nur auf den Pfad bezieht, nicht auf den Stift. (Bei PostScript ließ sich das nicht ohne weiteres trennen.)



`smiley.mp`

```
path circle;
```

- In METAFONT gibt es die 8 Datentypen boolean, string, path pen, picture, transform, pair und numeric.
- In METAPOST kommt noch der Datentyp color hinzu.
- Variablen, die nicht vom Typ numeric sind, müssen explizit mit ihrem Typnamen definiert werden.
- Variablennamen bestehen typischerweise aus Klein- und Großbuchstaben einschließlich dem Unterstrich `_`.
- Ziffern können nicht Bestandteil eines Variablennamens sein.
- Vordefinierte Namen dürfen nicht überdefiniert werden.
- Variablen-Deklarationen sind normalerweise global.

smiley.mp

```
path circle;  
circle := (-1,0) .. (0,1) .. (1,0) .. (0,-1) .. cycle;
```

- In METAFONT/METAPOST können Pfade Variablen zugewiesen werden.
- Für Pfade gibt es zahlreiche Operatoren. In diesem Beispiel legt der Pfad-Operator `..` eine Bézier-Kurve zwischen einem Pfad und einem Punkt, so dass der Pfad ohne Ecken durch die beiden Punkte verläuft und vom Aussehen her optimiert wird.
- `cycle` ist ein spezieller Operator in einem Pfad-Ausdruck, der dem `closepath`-Operator in PostScript entspricht.

smiley.mp

```
draw circle scaled 1; % Gesicht
```

- Auf Pfade können diverse Transformations-Operatoren angewendet werden:

$$(x, y) \textit{ shifted } (a, b) = (x + a, y + b)$$

$$(x, y) \textit{ scaled } s = (sx, sy)$$

$$(x, y) \textit{ xscaled } s = (sx, y)$$

$$(x, y) \textit{ yscaled } s = (x, sy)$$

$$(x, y) \textit{ slanted } s = (x + sy, y)$$

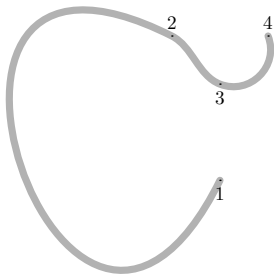
$$(x, y) \textit{ rotated } \theta = (x \cos \theta - y \sin \theta, x \sin \theta + y \cos \theta)$$

$$(x, y) \textit{ zscaled } (u, v) = (xu - yv, xv + yu)$$

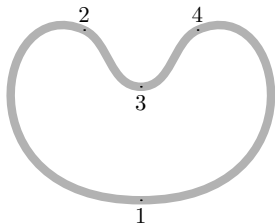
- Es gibt auch den generellen Operator *transformed*, der als rechten Operanden eine Variable oder einen Ausdruck vom Typ `transform` erwartet.

curves.mp

```
z1 = (0,-100); z2 = (-100,200);  
z3 = (0,100); z4 = (100,200);  
draw z1 .. z2 .. z3 .. z4;
```



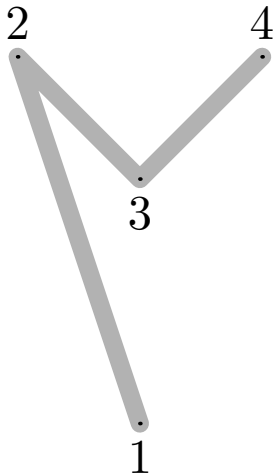
- z ist ein vordefiniertes Array von Punkten (Typ pair).
- $z1$ ist äquivalent zu $z[1]$.
- Es gilt $z1 = (x1,y1)$, d.h. x und y sind ebenfalls Arrays (vom Typ numeric), die mit dem Array z entsprechend als Aliase verknüpft sind.
- Der Operator `..` konstruiert eine Bézier-Kurve, wobei darauf geachtet wird, dass aufeinanderfolgende Bézier-Kurven ohne Kanten ineinander übergehen.



curves.mp

```
z1 = (0,-100); z2 = (-100,200);  
z3 = (0,100); z4 = (100,200);  
draw z1 .. z2 .. z3 .. z4 .. cycle;
```

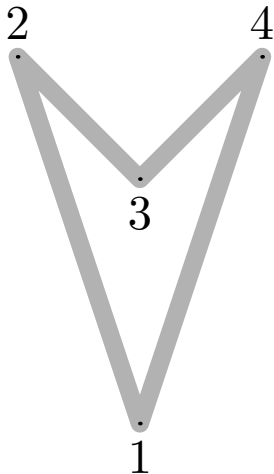
- Mit *cycle* wird der Pfad geschlossen.
- Das bedeutet in Kombination mit dem *..*-Operator, dass die gesamte Kurve ohne Kanten gestaltet wird.



curves.mp

```
z1 = (0,-100); z2 = (-100,200);  
z3 = (0,100); z4 = (100,200);  
draw z1 -- z2 -- z3 -- z4;
```

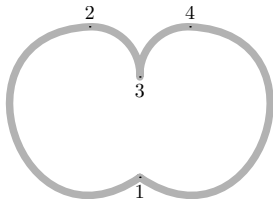
- Mit -- werden gerade Linien gezogen.



curves.mp

```
z1 = (0,-100); z2 = (-100,200);  
z3 = (0,100); z4 = (100,200);  
draw z1 -- z2 -- z3 -- z4 -- cycle;
```

- Auch hier führt ein *cycle* dazu, dass die Kurve geschlossen wird.



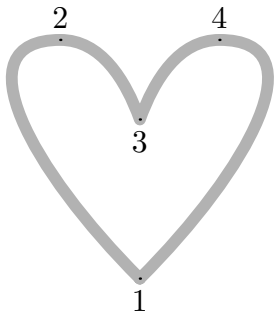
curves.mp

```
z1 = (0,-100); z2 = (-100,200);  
z3 = (0,100); z4 = (100,200);  
draw z1 .. z2 .. z3 & z3 .. z4 .. z1;
```

- Der Operator `&` verknüpft zwei Pfade miteinander, wobei keine Glättung der Kurve erzwungen wird.

curves.mp

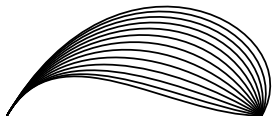
```
z1 = (0,-100); z2 = (-100,200);  
z3 = (0,100); z4 = (100,200);  
draw z1{dir 135} .. z2{right} .. {dir -70}z3 &  
z3{dir 70} .. z4{right} .. z1{dir 225};
```



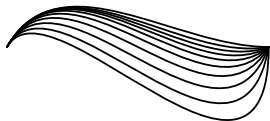
- Unmittelbar vor oder hinter einem Punkt kann eine Richtung angegeben werden, die dann jeweils diese Richtung in Richtung des Punktes bzw. ausgehend von dem Punkt erzwingt.
- Richtungen können als *left*, *right*, *up* and *down* spezifiziert werden. Alternativ kann die Richtung auch in Grad mit dem *dir*-Operator angegeben werden. Ferner ist es auch möglich, die Differenz zweier Punkte anzugeben.

`dcurves.mp`

```
for d = 240 step 10 until 360:  
  draw (0,0){dir 60} .. {dir d}(1in,0);  
endfor
```



- Zu beachten ist hier, dass die unteren Kurven einen Wendepunkt haben, d.h. dass sie auf dem Wege von links nach rechts sich nicht immer nur nach rechts drehen, sondern ab einem Punkt in der Mitte nach links.
- Dieses und die folgenden zwei Diagramme wurden (in etwas modifizierter Form) dem METAFONTbook von Donald E. Knuth übernommen aus den Seiten 18 und 19.



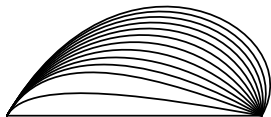
dcurves.mp

```
for d = 0 step 10 until 90:  
  draw (0,0){dir 60} .. {dir d}(1in,0);  
endfor
```

- Hier sind die Wendepunkte noch deutlicher zu sehen.

dcurves.mp

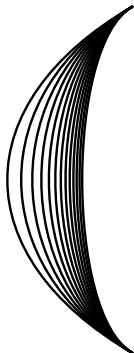
```
for d = 240 step 10 until 360:  
  draw (0,0){dir 60} ... {dir d}(1in,0);  
endfor
```



- Sei z_0 der linke Ausgangspunkt und z_1 der rechte Endpunkt.
- Dann gibt es möglicherweise einen Punkt z als Schnittpunkt der von z_0 und z_1 ausgehenden Strahlen entsprechend der angegebenen Richtungen.
- Falls es z gibt, dann liegt beim Operator „...“ die Kurve innerhalb des von z_0 , z_1 und z gebildeten Dreiecks, d.h. ein Wendepunkt wird vermieden.
- Falls es kein z gibt, entspricht dieser Operator dem „...“.

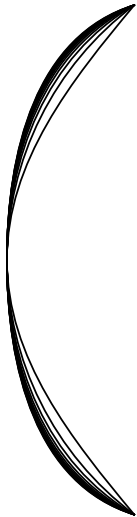
tension.mp

```
for t = -2 step 1 until 10:  
  draw (0,0){dir 150} .. tension (1+t/10) .. {dir 30}(0,1in);  
endfor
```



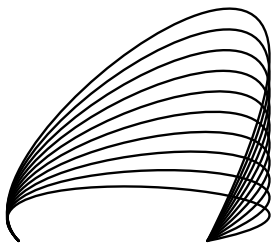
- Es ist möglich, die Spannung einer Kurve anzugeben. Per Voreinstellung wird eine *tension* von 1 gewählt.
- Je höher die Spannung wird, umso weiter nach rechts wird die Kurve in diesem Beispiel gedrückt.
- Die Spannung kann auch kleiner als 1 sein, muss aber mindestens den Wert $\frac{3}{4}$ haben. In diesen Fällen darf der Bogen sich noch weiter nach links ausstrecken.
- Das bedeutet, dass die Kontrollpunkte einer Bézier-Kurve umso näher zu den Ausgangs- und Endpunkten rücken, je höher die Spannung gewählt wird.

curl.mp



```
for c = 0 step 1 until 10:  
  draw (0,0){curl c} .. (-0.5in,1in) .. {curl c}(0,2in);  
endfor
```

- Wenn nichts anderes angegeben wird, entstehen annäherungsweise Kreisbögen.
- Mit *curl* kann an den Endpunkten ein Biegungsfaktor ausgewählt werden. Je höher dieser ist, umso steiler wird der Winkel, mit dem der Ausgangspunkt verlassen bzw. der Endpunkt erreicht wird.
- Ein Biegungsfaktor von 0 ist zulässig. In diesem Falle geht die Kurve fast direkt auf den Endpunkt zu. Per Voreinstellung liegt der Wert an den Endpunkten bei 1.
- Wenn *curl* inmitten einer Kurve angegeben wird, entsteht eine Knickstelle.



controls.mp

```
for y = 10 step 10 until 100:  
  draw (0,0) .. controls (-20, 20) and (90, y) .. (40, 0);  
endfor
```

- Die beiden inneren Kontrollpunkte einer Bézier-Kurve können auch mit *controls* explizit angegeben werden.

ABCDEFGHIJKLMNOPQRSTUVWXYZ_abcdefghijklmnopqrstuvwxyz

<=>:|

' ,

+ -

/ * \

! ?

& @ \$

^ ~

[

]

{ }

.

Bestandteil von Gleitkommazahlen; wird ansonsten ignoriert

, ; ()

stehen jeweils für sich alleine

"

Beginn einer Zeichenkette

0123456789

Numerische Konstante

%

Kommentare (bis zum Ende der Zeile)

- METAFONT und METAPOST konvertieren den Programmtext in eine Sequenz von Symbolen.
- Leerzeichen, Zeilentrenner, Kommentare und Punkte trennen Symbole.
- Solange aufeinanderfolgende Zeichen der gleichen Zeichenklasse angehören, werden sie zu einem Symbol zusammengefasst.
- Eine besondere Behandlung erfahren numerische Konstanten, die mit einem Punkt (gefolgt von mindestens einer Ziffer) oder mit einer Ziffer beginnen. Es werden soviele Zeichen verschlungen, wie maximal zu einer numerischen Konstante gehören können.
- Zeichenketten gehen von " bis zum darauffolgenden ".
- Beispiele:

Eingabe	Symbole
<i>circle.radius</i>	„circle“, „radius“
<i>x2p</i>	„x“, „2“, „p“
<i>grmb!\$!#@</i>	„grmb!“, „\$“, „!“, „#@“

- In METAFONT und METAPOST werden alle eingelesenen Symbole, die keine Zeichenketten oder numerischen Konstanten sind, als Namen bezeichnet.
- Alle Namen fallen in eine von zwei Kategorien: Funken (*sparks*) und Etiketten (*tags*).
- Zu den Funken gehören alle vordefinierten Operatoren und die zusätzlich definierten Makros. Vordefinierte Variablen gehören **nicht** dazu.

- Beispiele für Funken:

beginfig *draw* *path* := + () ; [

- Beispiele für Etiketten:

c *circle* *d* *l* *prologues* *x*

- Variablenamen bestehen aus einer Sequenz von Etiketten, bei denen auch numerische Konstanten enthalten sein können (jedoch nicht am Anfang).
- Mit einer Sequenz von Etiketten sind hierarchische Variablenamen möglich. Beispiele:
 a a' $a'b$ $a'b::c$
- Wenn aufeinanderfolgende Etiketten der gleichen Zeichenklasse angehören, empfiehlt sich die Verwendung eines Punktes als Trenner.
Beispiel: $a.b$
- Numerische Konstanten werden als Index eines Arrays interpretiert.
Beispiel: $x2p$ ist äquivalent zu $x[2].p$ und
 $a1.5b$ ist äquivalent zu $a[3/2]b$
- Die eckigen Klammern [...] sind nur notwendig, wenn für den Index ein Ausdruck verwendet wird.

- Obwohl eine Notation wie *a.b* Verbundsstrukturen nahelegt, handelt es sich dabei um Hierarchien. Das bedeutet, dass *a* nicht für das Gesamtobjekt steht. Stattdessen kann *a* vom Datentyp *numeric* sein, während *a.s* vom Datentyp *string* ist und *a.p* den Datentyp *path* hat, wenn dies zuvor so deklariert wurde: *string a.s; path a.p*
- Es ist aber möglich, einen Teil eines hierarchischen Namens als Parameter bei einem Makro oder als zu durchlaufenden Wert bei einer *forsuffixes*-Schleife anzugeben. Entsprechend können hierarchische Namen genutzt werden, um verbundartige Strukturen zu repräsentieren.

- Neben dem Zuweisungsoperator $:=$ gibt es auch den Operator $=$, der nicht nur dem Vergleich dient, sondern auch die Spezifikation linearer Gleichungssysteme erlaubt.
- Beispiel:
 $a = 2b+c; 7b = c/2 - a; 4a = 5c+b - 3;$
ist äquivalent zu:
 $a=1.92; b=-0.12; c=2.16;$
- Somit gibt es Variablen mit einem unbekanntem Wert und ein System partiell aufgelöster Gleichungssysteme.

- Folgende Operationen mit unbekanntem Werten sind zulässig:
 - $\langle \text{unknown} \rangle$
 - $\langle \text{unknown} \rangle + \langle \text{unknown} \rangle$
 - $\langle \text{unknown} \rangle - \langle \text{unknown} \rangle$
 - $\langle \text{unknown} \rangle * \langle \text{known} \rangle$
 - $\langle \text{known} \rangle * \langle \text{unknown} \rangle$
 - $\langle \text{unknown} \rangle / \langle \text{known} \rangle$
 - $\langle \text{known} \rangle [\langle \text{unknown} \rangle, \langle \text{unknown} \rangle]$
 - $\langle \text{unknown} \rangle [\langle \text{known} \rangle, \langle \text{known} \rangle]$
- Diese Restriktionen stellen sicher, dass das Gleichungssystem linear bleibt.