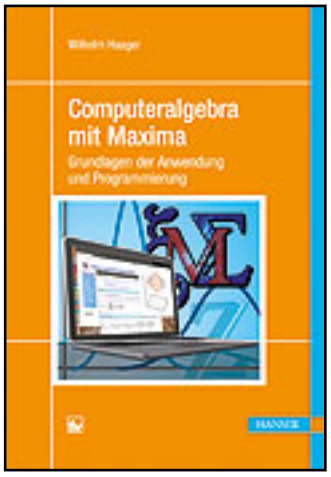
HANSER



Leseprobe

Wilhelm Haager

Computeralgebra mit Maxima

Grundlagen der Anwendung und Programmierung

ISBN (Buch): 978-3-446-44203-0

ISBN (E-Book): 978-3-446-43730-2

Weitere Informationen oder Bestellungen unter

http://www.hanser-fachbuch.de/978-3-446-44203-0

sowie im Buchhandel.

2 Ausdrücke

Jede Eingabe in Maxima ist ein *Ausdruck*, der von Maxima in bestimmer Weise interpretiert wird und immer ein *Ergebnis* liefert, das – wenn die Eingabe nicht mit einem Dollarzeichen abgeschlossen wurde – ausgegeben wird. Formal besteht kein Unterschied zwischen

- *Befehlen* oder Anweisungen, die bestimmte Aktionen veranlassen (wie zum Beispiel die Umformung mathematischer Ausdrücke oder das Lesen und Schreiben von Files),
- *mathematischen Ausdrücken*, bestehend aus Zahlen, Symbolen, Operatoren und Funktionen sowie
- *Funktionen*, die Parameter übernehmen und ein Ergebnis zurückliefern.

Zusätzlich zur Rückgabe eines Ergebnisses, das unmittelbar weiterverwendet werden kann (beispielsweise als Teil eines größeren Ausdrucks oder in einer Zuweisung), können Ausdrücke auch *Nebeneffekte* produzieren, etwa das Schreiben von Werten in ein File, das Produzieren einer Grafik oder die Ausgabe von informativen Meldungen.

2.1 Grundlegende Datentypen

Die kleinsten Bestandteile von Maxima-Ausdrücken heißen *Atome*. Sie können nicht mehr weiter zerlegt, aber durch *Operatoren* zu komplexeren Ausdrücken verknüpft werden oder Parameter von Funktionen sein, die wiederum komplexere Ausdrücke zurückliefern.

Die unterschiedlichen Arten von Atomen können als die grundlegenden Datentypen von Maxima angesehen werden; das sind:

- Ganzzahlen
- Gleitkommazahlen
- lange Gleitkommazahlen („bigfloat“)
- Zeichenketten (sowie einzelne Zeichen)
- symbolische Namen (Namen von Variablen und Funktionen)

2.1.1 Prädikatsfunktionen

Der Typ eines Ausdrucks kann mit sogenannten *Prädikatsfunktionen* (oder Abfragefunktionen) abgefragt werden, die als Ergebnis die logischen Werte *true* oder *false* liefern. Diese Funktionen werden häufig in Programmen benötigt, bei denen Entscheidungen in Abhängigkeit von Datentypen getroffen werden müssen.

Tabelle 2.1 Prädikatsfunktionen für Datentypen

`atom(`*x*`)`	Überprüft, ob der Ausdruck *x* ein Atom ist
Überprüfung auf …	
`symbolp(`*x*`)`	Symbolischer Name
`integerp(`*x*`)`	Ganzzahl
`evenp(`*x*`)`	Gerade Ganzzahl
`oddp(`*x*`)`	Ungerade Ganzzahl
`primep(`*x*`)`	Primzahl
`floatp(`*x*`)`	Gleitkommazahl
`bfloatp(`*x*`)`	Gleitkommazahl mit (beliebig) hoher Genauigkeit („bigfloat“)
`numberp(`*x*`)`	Beliebige Zahl
`ratnump(`*x*`)`	Rationale Zahl
`constantp(`*x*`)`	Konstante
`stringp(`*x*`)`	Zeichenkette

Bei Zahlen wird zwischen Ganzzahlen, Gleitkommazahlen mit *bestimmter* Genauigkeit und Gleitkommazahlen mit *beliebiger* Genauigkeit („bigfloat“) unterschieden. Rationale Zahlen werden von Gleitkommazahlen unterschieden; sie sind aber keine Atome, da sie in Zähler und Nenner zerlegt werden können.

Symbolische Namen können aus Buchstaben, Ziffern, dem Unterstrich und beliebigen Zeichen mit vorangestelltem Backslash („\“) bestehen, wobei das erste Zeichen keine Ziffer sein darf.

Symbole, Ganzzahlen und Zeichenketten sind Atome, rationale Zahlen aber nicht.

```
(%i1) [atom(a),atom(12),atom(12/7),atom("text")];
(%o1) [true , true , false , true]
```

Zahlen und Zeichenketten sind keine Symbole.

```
(%i2) [symbolp(a),symbolp("a"),symbolp(12)];
(%o2) [true , false , false]
```

Zuweisen einer Ganzzahl

```
(%i3) n1:20;
(%o3) 20
```

Gleitkommazahlen werden von Ganzzahlen unterschieden; Funktionsargumente werden ausgewertet.

```
(%i4) [integerp(12),integerp(12.0),integerp(n1)];
(%o4) [true , false , true]
```

Überprüfung auf eine gerade Zahl, ungerade Zahl und Primzahl

```
(%i5) [oddp(12),evenp(12),primep(12)];
(%o5) [false , true , false]
```

Überprüfung auf eine Gleitkommazahl

```
(%i6) [floatnump(12),floatnump(12.0),floatnump(12b0)];
(%o6) [false , true , false]
```

Überprüfung auf eine lange Gleitkommazahl („bigfloat“)

```
(%i7) [bfloatp(12),bfloatp(12.0),bfloatp(12b0)];
(%o7) [false,false,true]
```

Symbolische Konstanten und mathematische Funktionen mit Ganzzahlen bleiben unausgewertet, sie sind daher *keine* Gleitkommazahlen.

```
(%i8) [floatnump(%pi),floatnump(sqrt(2))];
(%o8) [false,false]
```

Überprüfung auf eine Zahl

```
(%i9) [numberp(12),numberp(n1/7)];
(%o9) [true,true]
```

Mathematische Funktionen werden, wenn möglich, *vereinfacht* und ergeben dann eine Zahl.

```
(%i10) [numberp(sqrt(3/9)),numberp(sqrt(4/9))];
(%o10) [false,true]
```

Überprüfung auf eine rationale Zahl; auch Ganzzahlen sind rationale Zahlen.

```
(%i11) [ratnump(1/2),ratnump(0.5),ratnump(7)];
(%o11) [true,false,true]
```

Überprüfung auf eine Konstante

```
(%i12) [constantp(%pi),constantp(sqrt(2))];
(%o12) [true,true]
```

Eine Zeichenkette ist *keine* Konstante.

```
(%i13) constantp("a");
declare: argument must be a symbol; found a
 -- an error. To debug this try: debugmode(true);
```

Durch Einschließen in doppelte Anführungszeichen wird ein Ausdruck zu einer Zeichenkette.

```
(%i14) [stringp(12/7),stringp("12/7")];
(%o14) [false,true]
```

2.1.2 Umwandlungsfunktionen

Mit Hilfe von Umwandlungsfunktionen können Ausdrücke eines Typs in einen anderen Typ umgewandelt werden. Die Umwandlungsfunktionen werten das Funktionsargument *vor* der Typumwandlung aus.

Tabelle 2.2 Umwandlungsfunktionen für Datentypen

`float(`*x*`)`	Umwandlung von *x* in eine Gleitkommazahl
`bfloat(`*x*`)`	Umwandlung von *x* in eine lange Gleitkommazahl („bigfloat“)
`round(`*x*`)`	Runden auf die nächstliegende Ganzzahl
`floor(`*x*`)`	Erzeugung der größten Ganzzahl kleiner oder gleich *x*
`ceiling(`*x*`)`	Erzeugung der kleinsten Ganzzahl größer oder gleich *x*
`string(`*x*`)`	Umwandlung eines Ausdrucks in eine Zeichenkette *x*
`eval_string(`*x*`)`	Auswertung einer Zeichenkette *x* als Ausdruck

`float` und `bfloat` wandeln alle Zahlen eines Ausdrucks in die Gleitkommadarstellung bzw. lange Gleitkommadarstellung um, `floor` und `ceiling` spalten ganzzahlige Anteile aus einem Ausdruck ab und lassen den Rest unausgewertet.

`string` wertet einen Ausdruck aus und wandelt das Ergebnis in eine Zeichenkette (engl.: „string“) um, und zwar in eine Form, in der sie wieder von Maxima als Ausdruck eingelesen werden könnte: als ASCII-Text ohne Bruchstriche sowie ohne Hoch- und Tiefstellungen. `eval_string` interpretiert eine Zeichenkette als Maxima-Ausdruck und wertet diesen aus; Näheres zu den Zeichenketten siehe Abschnitt 2.4.

Umwandlung einer rationalen Zahl in Gleitkommazahlen

```
(%i15) [7/20,bfloat(7/20),float(7/20)];
(%o15) [7/20, 3.5b-1 , 0.35]
```

Mathematische Funktionen werden, wenn möglich, *vereinfacht* aber nicht zu Gleitkommazahlen ausgewertet.

```
(%i16) [sin(%pi/6),sin(%pi/5)];
(%o16) [1/2, sin(%pi/5)]
```

Mit der Funktion `float` werden mathematische Funktionen zu Gleitkommazahlen ausgewertet.

```
(%i17) float(%);
(%o17) [0.5 , 0.58778525229247]
```

Auswertung eines Wurzelausdrucks zu einer Gleitkommazahl

```
(%i18) [sqrt(2),float(sqrt(2))];
(%o18) [sqrt(2) , 1.414213562373095]
```

Enthält ein arithmetischer Ausdruck eine Gleitkommazahl, so wird der gesamte Ausdruck in Gleitkommadarstellung ausgewertet.

```
(%i19) [2/3+7,2/3+7.0];
(%o19) [23/3, 7.666666666666667]
```

Anwendung der Funktion `float` auf einen Ausdruck

```
(%i20) float((1+3*a)/sqrt(1+b));
(%o20) (3.0 a+1.0)/sqrt(b+1.0)
```

Umwandlung eines Ausdrucks in eine Zeichenkette

```
(%i21) str:string((1+3*a)/sqrt(1+b));
(%o21) (3*a+1)/sqrt(b+1)
```

Eine Zeichenkette ist kein mathematischer Ausdruck, daher werden Werte von Variablen nicht eingesetzt.

```
(%i22) [a:5,b:3];
(%o22) [5 , 3]
(%i23) str;
(%o23) (3*a+1)/sqrt(b+1)
```

Umwandlung der Zeichenkette in einen Ausdruck; die Werte der Variablen werden eingesetzt.

```
(%i24) eval_string(str);
(%o24) 8
```

Ermittlung der nächstkleineren und nächstgrößeren Ganzzahl; ein ganzzahliger Anteil wird abgespalten.

```
(%i25) [floor((40/7+x)),ceiling(40/7-x)];
(%o25) [floor(x+5/7)+5 , 6 -floor(x+2/7)]
```

2.1.3 Konstanten

Eine Reihe mathematischer Konstanten besitzt symbolische Namen. Als Bestandteil von Ausdrücken bleiben sie im Allgemeinen unausgewertet, es sei denn, ihre Auswertung wird – beispielsweise durch die Funktion `float` – erzwungen.

Die logischen Konstanten *true* und *false* werden kaum in Benutzereingaben verwendet werden, sie sind vielmehr das Ergebnis von logischen Ausdrücken und Vergleichsausdrücken (siehe Abschnitt 2.2.2). Kann ein solcher Ausdruck nicht eindeutig zu *true* oder *false* ausgewertet werden, so liefert er den besonderen Wert *unknown*, der aber von Maxima *nicht* als Konstante angesehen wird.

Die Werte für „unendlich“, `inf`, `minv` und `infinity` werden in erster Linie als Integrationsgrenzen für bestimmte Integrale (Abschnitt 6.4) und bei der Berechnung von Grenzwerten (Abschnitt 6.1) verwendet.

Tabelle 2.3 Mathematische Konstanten

`%i`	Imaginäre Einheit i (in der Elektrotechnik meist mit j bezeichnet)
`%pi`	Kreiszahl π
`%e`	Eulersche Zahl e
`%phi`	Goldener Schnitt $\Phi = (1+\sqrt{5})/2$
`%gamma`	Euler-Mascheronische Konstante γ
`true`	Logischer Wert für „wahr“
`false`	Logischer Wert für „falsch“
`infinity`	Komplex unendlich
`inf`	Positiv unendlich
`minf`	Negativ unendlich
`und`	Unbestimmt

Die Wurzel einer negativen Zahl ergibt eine imaginäre Zahl.

```
(%i26) sqrt(-7);
(%o26) sqrt(7) %i
```

Die Kreiszahl π

```
(%i27) float(%pi);
(%o27) 3.141592653589793
```

Definition der eulerschen Zahl e

```
(%i28) limit((1+1/n)^n,n,inf);
(%o28) %e
```

Die Exponentialfunktion kann sowohl mit der Zahl `%e` als auch mit der Funktion `exp` realisiert werden.

```
(%i29) %e^x-exp(x);
(%o29) 0
```

Die eleganteste Formel der Mathematik, sie enthält die grundlegenden Konstanten e, i, π, 1 und als Ergebnis 0.

```
(%i30) %e^(%i*%pi)+1;
(%o30) 0
```

Goldener Schnitt

```
(%i31) float(%phi);
(%o31) 1.618033988749895
```

Euler-Mascheronische Konstante

```
(%i32) float(%gamma);
(%o32) 0.57721566490153
```

Vergleichsausdrücke werden mit der Funktion `is` zu logischen Werten ausgewertet.

```
(%i33) [is(1=1.0),is(3<4),is(x<y)];
(%o33) [false , true , unknown]
```

Bestimmtes Integral mit Integrationsgrenzen im Unendlichen

```
(%i34) integrate(exp(-x**2),x,minf,inf);
(%o34) √π
```

Grenzwert, der *positiv unendlich* ergibt

```
(%i35) limit((x**2-1)/x,x,inf);
(%o35) ∞
```

Grenzwert, der *komplex unendlich* ergibt

```
(%i36) limit(1/x,x,0);
(%o36) infinity
```

2.1.4 Systemvariablen

Maxima kennt eine Reihe von Systemvariablen. Manche dieser Variablen dienen zum Festlegen von Grundeinstellungen und können vom Benutzer verändert werden. Andere Systemvariablen liefern Informationen über den Zustand einer Maxima-Sitzung und können nur von Maxima selbst, aber nicht vom Benutzer verändert werden. Im Folgenden werden grundlegede Systemvariablen erläutert, mit denen Rechengenauigkeit und Ausgabe gesteuert werden können und die Informationen über benutzte Variablen liefern. Weitere Systemvariablen werden in entsprechenden, nachfolgenden Abschnitten vorgestellt.

Tabelle 2.4 Systemvariablen für Datentypen

`values`	Liste mit allen mit einem Wert belegten Benutzervariablen
`fpprec`	Rechengenauigkeit für lange Gleitkommazahlen („bigfloat“), Default: 16
`fpprintprec`	Anzahl der signifikanten Stellen für die *Ausgabe* von Gleitkommazahlen, Default: 0 (volle Genauigkeit)
`ratepsilon`	Toleranz für die Umwandlung von Gleitkommazahlen in rationale Zahlen, Default: 2.0e−8
`ibase`	Basis des Zahlensystems für die *Ein*gabe von Ganzzahlen, Default: 10
`obase`	Basis des Zahlensystems für die *Aus*gabe von Ganzzahlen, Default: 10

`fpprec` gibt die *Rechen*genauigkeit in Stellen für lange Gleitkommazahlen („bigfloat“) an, für normale Gleitkommazahlen hat dieser Wert keine Bedeutung. `fpprintprec` gibt die *Ausgabe*genauigkeit in signifikanten Stellen für Gleitkommazahlen an; auf die Rechengenauigkeit hat dieser Wert keinen Einfluss.

`ratepsilon` ist die Toleranzgrenze für die Umwandlung von Gleitkommazahlen in rationale Zahlen, beispielsweise mit dem Befehl `ratsimp` (siehe Abschnitt 5.2).

Liste aller benutzerdefinierten Variablen, die mit einem Wert belegt sind

```
(%i37) values;
(%o37) [n1,str,a,b]
```

Festlegen der *Rechen*genauigkeit für lange Gleitkommazahlen

```
(%i38) fpprec:30;
(%o38) 30
```

Die Zahl π auf 30 Stellen genau

```
(%i39) bfloat(%pi);
(%o39) 3.14159265358979323846264338328b0
```

Festlegen der *Ausgabe*genauigkeit für Gleitkommazahlen

```
(%i40) fpprintprec:5;
(%o40) 5
```

Entsprechend verkürzte Ausgabe einer Gleitkommazahl

```
(%i41) x1:(1/7+1e-5);
(%o41) 0.143
```

Umwandlung in eine rationale Zahl

```
(%i42) ratsimp(x1);
rat: replaced 0.143 by 100007/700000 = 0.143
(%o42) 100007/700000
```

Erhöhung der Toleranz für die Umwandlung in eine rationale Zahl

```
(%i43) ratepsilon:1e-4;
(%o43) 1.0 10^-4
```

Die rationale Zahl ist jetzt einfacher, aber weniger genau.

```
(%i44) ratsimp(x1);
rat: replaced 0.143 by 1/7 = 0.143
(%o44) 1/7
```

Mit den Variablen `ibase` und `obase` kann die Basis des Zahlensystems für die Eingabe bzw. Ausgabe von Ganzzahlen zwischen den Werten 2 und 36 eingestellt werden. Die Eingabe und Darstellung von Gleitkommazahlen bleibt davon unbeeinflusst.

Für die Ziffern größer als 9 werden (wie bei Hexadezimalzahlen üblich) Buchstaben für die *Ausgabe* verwendet, `A` für 10, `B` für 11 etc. Bei der *Eingabe* einer Zahl muss das erste Zeichen jedenfalls eine *Ziffer* sein. Ist das erste Zeichen ein Buchstabe, interpretiert Maxima die Eingabe nicht als *Zahl*, sondern als *Variablenname*. Einem Buchstaben als erstem Zeichen einer Zahl ist daher die Ziffer 0 voranzustellen.

Achtung: Wurde `ibase` verändert, so ist bei der Wiederherstellung des ursprünglichen Wertes die Zahl im *neuen* (geänderten) Zahlensystem anzugeben.

Umschalten der Eingabe und Ausgabe auf das Hexadezimalsystem

```
(%i45) [ibase:16,obase:16];
(%o2D) [10,10]
```

Eine Rechnung mit Ganzzahlen im Hexadezimalsystem, es gibt auch Buchstaben als Ziffern.

```
(%i46) 64*5-45;
(%o2E) 1AF
```

AA wird nicht als Zahl, sondern als Variable angenommen.

```
(%i47) AA+5;
(%o2F) AA+5
```

Voranstellen einer *Null* lässt die Eingabe als *Zahl* erkennen.

```
(%i48) 0AA+5;
(%o30) 0AF
```

Gleitkommazahlen werden *immer* im Dezimalsystem ausgegeben.

```
(%i49) float(100);
(%o31) 256.0
```

2.2 Operatoren

Operatoren sind Elemente von Ausdrücken, die Atome oder Teilausdrücke – *Operanden* genannt – entweder miteinander verknüpfen oder auf sie angewendet werden (wenn es nur einen einzigen Operanden gibt). Sie können Namen besitzen (z. B. and, not) oder durch ein Symbol dargestellt werden, das aus einem oder mehreren Zeichen bestehen kann (z. B. +,\,*,\,<=).

Genaugenommen ist ein Operator nichts anderes als eine besondere Darstellungsform einer *Funktion* (siehe Abschnitt 2.3): Eine Funktion verarbeitet *Parameter* (oder *Argumente*) zu einem *Funktionswert*, ein Operator verknüpft *Operanden* zu einem *Verknüpfungsergebnis* – beides ist im Prinzip dasselbe, was in Abschnitt 2.6.1 genauer ausgeführt wird.

2.2.1 Arithmetische Operatoren

Die arithmetischen Operatoren für die Grundrechnungsarten entsprechen den in allen Programmiersprachen üblichen Operatoren, für die Potenz existieren zwei gleichwertige Operatoren: „**“ und „^“.

Die *Fakultät* einer Ganzzahl n ist definiert als:

$$n! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot n$$

Die *doppelte Fakultät* einer Ganzzahl n ist definiert als:

$$n!! = \begin{cases} n \cdot (n-2) \cdot (n-4) \cdot \ldots 2 & \text{für } n \text{ gerade} \\ n \cdot (n-2) \cdot (n-4) \cdot \ldots 1 & \text{für } n \text{ ungerade} \end{cases}$$

Mit dem „normalen“ Zuweisungsoperator „:“ wird ein Ausdruck (auf der rechten Seite) einem Symbol (auf der linken Seite) zugewiesen; das Symbol auf der linken Seite wird *nicht* ausgewertet. Mit dem Zuweisungsoperator „::“ wird auch die linke Seite ausgewertet, wobei diese Auswertung ein *Symbol* als Ergebnis liefern muss. Mit dem Operator „:=“ wird eine *Funktion* definiert (siehe Abschnitt 2.3).

Die vier Grundrechnungsarten

```
(%i1) (a+b)*(c-d)/e;
(%o1) (b+a)(c-d)/e
```

Für die Potenz gibt es zwei gleichwertige Operatoren.

```
(%i2) [a**b, a^b];
(%o2) [a^b, a^b]
```