Computer Algebra Systeme

Simon Weckler

29. August 2022

Inhaltsverzeichnis

1 Benutzereingaben

Die einfachste Form, einen mathematischen Ausdruck auf einem Computer einzugeben, ist unter der Verwendung der Symbole, welche einem auf der Tastatur schon zur Verfügung stehen. Dabei gelten folgende Beziehungen zwischen den Zeichen und ihrer mathematischen Bedeutung:

Zeichen	Bedeutung
0 - 9	Ziffer
+	Addition
-	Subtraktion
/	Division
*	Multiplikation
^	Potenz
a - z	Symbol (Variable)

Es ist mit nur diesen Zeichen möglich, eine Großzahl von mathematischen Termen darzustellen und ist deswegen ein simpler, aber effektiver Weg, um Benutzereingaben entgegenzunehmen.

1.1 Darstellung im Programm

1.1.1 Darstellung als String

Der Benutzer kann einen mathematischen Ausdruck also in Form eines Strings eingeben. Den String auch als Datenstruktur zur Darstellung des Ausdrucks im Programm zu verwenden hat allerdings Nachteile. Um zu beurteilen, ob eine Darstellung für einen Computer passend ist oder nicht, muss zuerst definiert werden, welche Informationen das Programm benötigt, um mit dem Term arbeiten zu können.

Definition 1. *Um mit einem mathematischen Term arbeiten zu können muss ein Programm die Bedeutung der einzelnen Zeichen sowie die Operatorrangfolge (Def. 2) kennen*

Definition 2. *Unter der **Operatorrangfolge** oder **Operatorpräzedenz** ist die Ordnung zu verstehen, in welcher die Operatoren eines Ausdrucks auszuwerten sind. In dem Term* $3 + 4 * 5$ *hat der Operator* $+$ *die geringste Priorität und der Operator* $*$ *die höchste. Folglich muss* $4 * 5$ *vor* $3 + 4$ *ausgerechnet werden.*

Für ein Programm existiert per se kein Unterschied zwischen dem Zeichen *3* und *+*. Beide haben denselben Datentyp. Deswegen ist es erforderlich, immer das Zeichen zu untersuchen und ihm seine jeweilige Bedeutung zuzuordnen.
Um die Operatorpräzedenz zu bestimmen, muss es über den String iterieren und ihn in Teile unterschiedlicher Priorität unterteilen. Sind diese beiden Schritte getan, kann das Programm Operationen auf dem Term ausführen.
Am Ende wird das Resultat der Operation wieder in einen String geschrieben, da dies ja der gewählte Datentyp ist. Das hat den Nachteil, dass alle gewonnen Informationen über die Bedeutung der Zeichen und die Priorität der Operatoren wieder verloren gehen. Will man eine weitere Operation auf dem Term durchführen, so muss man all diese Informationen noch einmal extrahieren. Das ist ineffizient und führt zu mehr Komplexität im Code.
Der String ist folglich keine passende Datenstruktur um einen Ausdruck abzuspeichern

1.1.2 Darstellung als Baum

Ideal wäre also eine Datenstruktur, welche sowohl die Bedeutung der Zeichen als auch die Operatorrangfolge speichert, sodass diese nicht neu berechnet werden müssen.
Dafür ist es wichtig zu verstehen, dass jeder mathematische Ausdruck entweder als atomarer oder zusammengesetzter Ausdruck klassifiziert werden kann.

Definition 3. *Ein **atomarer Ausdruck** ist jeder Ausdruck, der kein Operator ist, also ganze Zahlen und Symbole. Atomare Ausdrücke können nicht weiter vereinfacht werden.*

Definition 4. *Jeder **zusammengesetzte Ausdruck** besteht aus einem Operator und einem oder mehreren Operanden. Zu diesen Ausdrücken zählen Summen, Produkte, Differenzen, Divisionen und Potenzen*

Man kann nun atomare und zusammengesetzte Ausdrücke und ihre Beziehungen zueinander in einem Baum darstellen. Dabei gelten folgende Regeln:

Definition 5. *In einem **Baum mathematischer Ausdrücke** ist jeder Elternknoten ein zusammengesetzter Ausdruck. Die Kinder eines Elternknotens sind seine Operanden, welche Zusammengesetze (Def. 4) oder atomare (Def. 3) Ausdrücke sein können. Alle Blätter des Baumes sind atomare Ausdrücke. (Def. 3)*

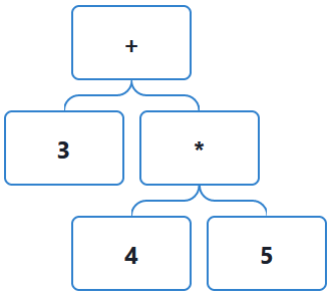

Abbildung 1: Der Baum für den Term $3 + 4 * 5$

Beispiel 1. *An dem Baum für den Term $3 + 4 * 5$ kann man erkennen, dass die Operatorrangfolge (Def. 2) in der Datenstruktur abgebildet wird. Da der Ausdruck $4 * 5$ ein Kind des Operators $+$ ist, muss dieser zuerst ausgewertet werden. In einem Baum mathematischer Ausdrücke (Def. 5) ist die Wurzel folglich immer der Operator mit der geringsten Priorität. Wie man außerdem erkennen kann, sind alle Blätter des Baumes ganze Zahlen, also atomare Ausdrücke (Def. 3).*

Von welchem Datentyp die Blätter und Knoten sind, ist ganz von der eigenen Implementierung abhängig. In diesem Computer Algebra System können folgende Ausdrücke Teil des Baumes sein:

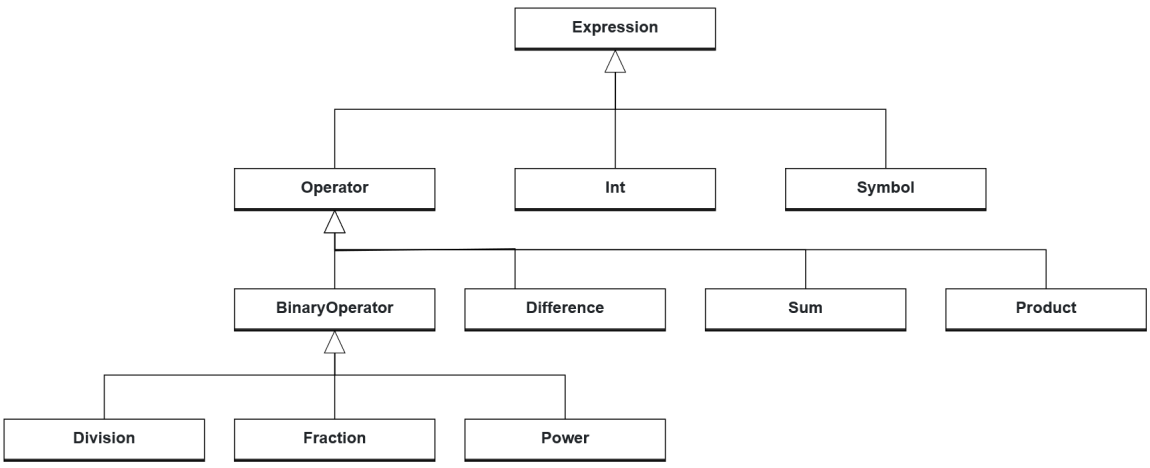

Abbildung 2: UML Diagramm aller Ausdrücke

Hier wurden die Operatoren in Binäre und Nicht-Binäre Operatoren unterteilt. Wie einem vielleicht auffällt, ist auch der Bruch ein Operator. Das ist darin begründet, dass ein Bruch

genau genommen auch eine Operation (Division) ist, allerdings eine, die nicht zu einer ganzen Zahl vereinfacht werden kann. Im Baum wird ein Bruch also wie folgt dargestellt:

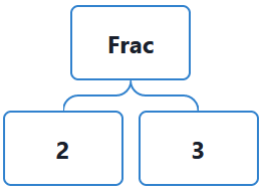

Abbildung 3: Ein Bruch im Baum mathematischer Ausdrücke

1.2 Parsen

In einem Computeralgebrasystem wird ein Term also in Form eines Baumes dargestellt. Für den Benutzer wäre es allerdings weder intuitiv noch einfach, einen Term als Baum anzugeben. Die Eingabe als String sollte also beibehalten werden. Folglich ist es erforderlich, die Zeichenkette nach Eingabe in einen Baum umzuwandeln. Dafür ist es notwendig, die Struktur einer Eingabe genau zu definieren.

Definition 6. *Die **konventionelle Struktur** eines Ausdrucks entspricht der Struktur vor der automatischen Vereinfachung. Diese Struktur ist auch die, die in der Mathematik verwendet wird. Hier gilt:*

1. *die Operatoren $+$ und $-$ haben entweder 1 oder 2 Operanten*
2. *die Operatoren $*$, $/$, und $\hat{}$ haben immer 2 Operanten*
3. *die Operanden der Operatoren sind Algebraische Ausdrücke*

Jeder Ausdruck kann immer einem Klammerlevel zugeordnet werden. Standardmäßig ist das Klammerlevel 0. Befindet sich ein Ausdruck in runden Klammern, so erhöht sich sein Klammerlevel um 1. Der Ausdruck mit dem höheren Klammerlevel hat immer die höhere Priorität gegenüber einem Ausdruck mit niedrigerem Klammerlevel. Es gelten folgende Regeln für die Operatorrangfolge (Def. 2) bei gleichem Klammerlevel:

Höchste Priorität

$\hat{}$

$*$ $/$

$+$ -

Niedrigste Priorität

Falls 2 Operatoren dieselbe Priorität haben (z. B. + und -) hat der rechte Operator die höhere Priorität. Die einzige Ausnahme ist die Potenz, bei welcher der linke Operator die höhere Priorität hat

Um aus diesen Regeln einen Algorithmus zu bauen, ist es hilfreich, eine Grammatik für einen mathematischen Ausdruck unter Beachtung der eben definierten Regeln in Form einer Backus-Naur-Form zu definieren.

Definition 7. *Eine **Backus-Naur-Form** (BNF) ist eine Notationsform, um die Grammatik von Sprachen zu definieren. Der Syntax einer Backus-Naur-Form ist sehr einfach zu verstehen, da es nur 3 Symbole gibt.*

Zeichen	Bedeutung
::=	Ist definiert als
|	Oder
<X>	Name einer Kategorie X

```
<Addition>     ::=    <Ziffer> + <Ziffer>
<Ziffer>       ::=    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Diese Backus-Naur-Form kann gelesen werden als: Eine Addition besteht aus 2 Ziffern, welche durch ein Plus Symbol getrennt sind. Eine Ziffer ist eine Zahl von 0 bis 9. Wie einem vielleicht auffällt, erlaubt diese Definition nur die Addition von 2 Ziffern. Der String $32 + 2$ würde also nicht dieser Definition einer Addition entsprechen.

Um einen String nun in einen Baum auf Basis einer Backus-Naur-Form umzuwandeln, lässt sich folgender Algorithmus formulieren

Definition 8. *Algorithmus zum Parsen eines BNF Baums*

1. *Überprüfe, ob der gegebene String der 1. Option der jeweiligen Definition entspricht.*
2. *Enthält die Definition eine weitere Definition, so überprüfe ob der String oder Teilstring auch dieser Definition entspricht.*
3. *Entspricht der String der Definition, so erweitere den Baum mit dem passenden Knoten / Blatt.*
4. *Ist das nicht der Fall, so probiere die nächste Option. Gibt es keine weitere Option mehr, so muss es sich um einen invaliden String handeln.*

Wenden wir diesen Algorithmus an, entsteht ein Baum für den gegebenen Term. Die vorher definierte Backus-Naur-Form würde zum Beispiel für den Input $2 + 3$ folgenden Baum generieren.

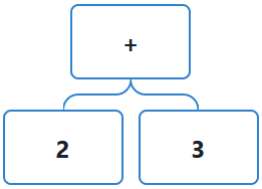

Abbildung 4: Parse Tree für den Input $2 + 3$

Ziel ist es nun, eine Backus-Naur-Form für einen mathematischen Ausdruck zu definieren, welche alle Aspekte der Definition eines mathematischen Ausdrucks in der konventionellen Struktur (Def. 6) berücksichtigt.

```
<exp>       ::= <exp> + <exp> | <exp> * <exp> | <exp> - <exp> |
                <exp> / <exp> | <exp> ^ <exp> | <int> | <symbol>
<int>       ::= <digit> <int> | <digit>
<digit>     ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
<symbol>    ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n |
                o | p | q | s | t | u | v | w | x | y | z
```

Diese Grammatik ist in der Lage, eine große Anzahl von Ausdrücken zu repräsentieren. So zum Beispiel auch $6 + 3 * 4$. Bei genauerer Betrachtung fällt einem jedoch auf, dass diese Grammatik die Operatorpräzedenz (Def. 2) nicht berücksichtigt. Es gibt für den Term $6+3*4$ zwei verschiedene Bäume, die mit der Grammatik übereinstimmen. Allerdings ist nur der 1. Baum korrekt.

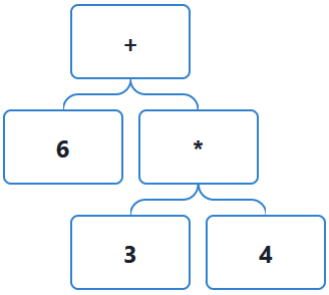

Abbildung 5: 1. Möglicher Baum für $6 + 3 * 4$

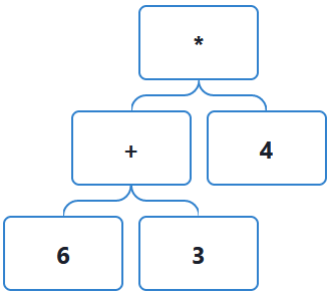

Abbildung 6: 2. Möglicher Baum für $6 + 3 * 4$

Dieses Problem kann dadurch gelöst werden, dass man eine Multiplikation als Operand einer Addition erlaubt, aber Addition nicht als Operand der Multiplikation zulässt. Diese Änderung resultiert in folgendem BNF:

```
<exp>       ::= <exp> + <exp> | <exp> - <exp> |
<term>      ::= <term> * <term> | <term> / <term> | <factor>
<power>     ::= <power> ^ <power>
<factor>    ::= <int> | <symbol
<int>       ::= <digit> <int> | <digit>
<digit>     ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
```

```
<symbol>  ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n |
              o | p | q | s | t | u | v | w | x | y | z
```

Für den Term $6 + 3 * 4$ gibt es nur noch einen möglichen Baum, allerdings gibt es noch ein weiteres Problem. Bei der Subtraktion von mehreren Zahlen, zum Beispiel $4 - 5 - 2$ gibt es wieder zwei mögliche Bäume mit zwei verschiedenen Ergebnissen.

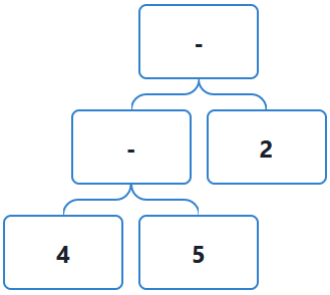

Abbildung 7: 1. Möglicher Baum für $4 - 5 - 2$

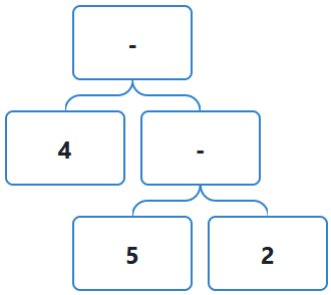

Abbildung 8: 2. Möglicher Baum für $4 - 5 - 2$

Das Problem entsteht dadurch, dass sowohl die linke als auch die rechte Seite der Differenz weitere Differenzen enthalten können. Das darf also nur noch auf der linken Seite erlaubt sein. Die einzige Ausnahme ist die Potenz, da sie ein rechts assoziativer Operator ist. Was außerdem noch fehlt, sind unäre Summen und Differenzen, sowie die Möglichkeit, mit Klammern die Operatorrangfolge zu manipulieren. Mit allen diesen Änderungen sieht die Backus-Naur-Form nun folgendermaßen aus:

```
<exp>     ::= + <exp> | <exp> + <term> | - <exp> | <exp> - <term> | <term>
<term>    ::= <term> * <power> | <term> / <power> | <power>
<power>   ::= <factor> ^ <power> | <factor>
<factor>  ::= ( <exp> ) | <int> | <symbol>
<int>     ::= <digit> <int> | <digit>
<digit>   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
<symbol>  ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n |
              o | p | q | s | t | u | v | w | x | y | z
```