Computer Algebra Systeme

Simon Weckler

30. 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)*

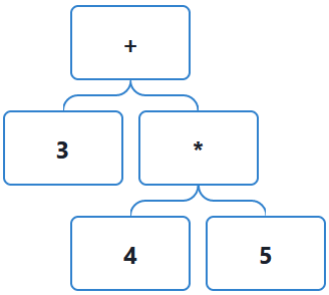

Abbildung 1: Der Baum für den Term $3 + 4 * 5$

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:
Hier wurden die Operatoren in Binäre und Nicht-Binäre Operatoren unterteilt.

Definition 6. ***Binäre Operatoren** sind Operatoren, welche immer zwei Operanden haben müssen*

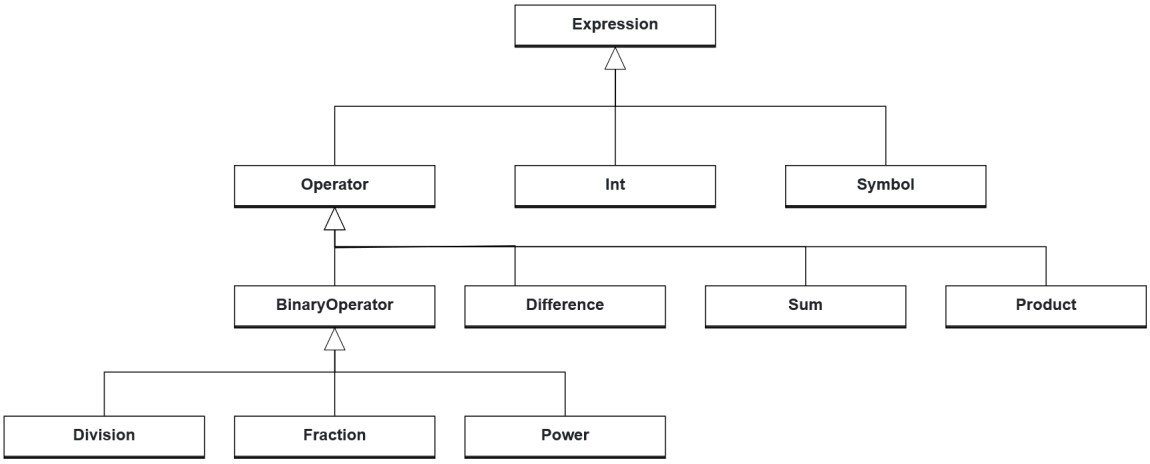

Abbildung 2: UML Diagramm aller Ausdrücke

Definition 7. ***Nicht-Binäre Operatoren*** *sind Operatoren, deren Operandenanzahl aufgrund des Assoziativgesetzes zwischen 1 und N variieren kann*

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:

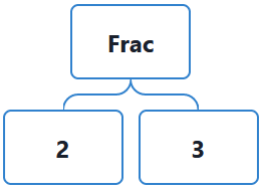

Abbildung 3: Ein Bruch im Baum mathematischer Ausdrücke

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 8. *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 9. *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 10. *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.

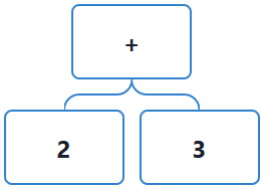

Abbildung 4: Parse Tree für den Input $2 + 3$

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. 8) 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.
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>
```

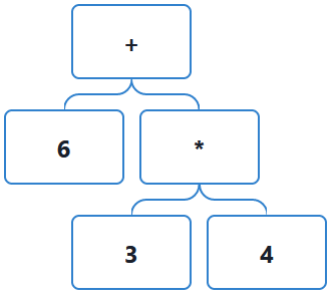

Abbildung 5: 1. Möglicher Baum für $6 + 3 * 4$

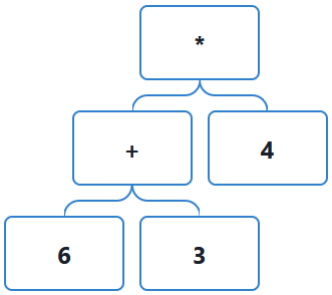

Abbildung 6: 2. Möglicher Baum für $6 + 3 * 4$

```
<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.

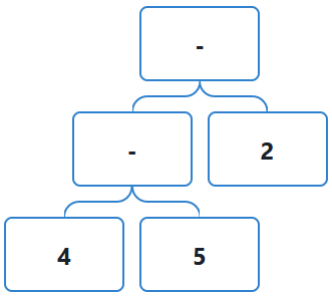

Abbildung 7: 1. Möglicher Baum für $4 - 5 - 2$

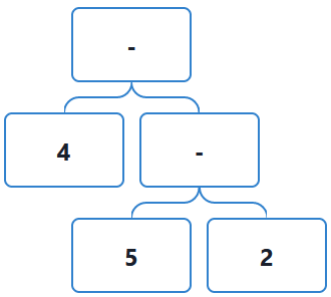

Abbildung 8: 2. Möglicher Baum für $4 - 5 - 2$

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
```

Der Algorithmus, der auf dieser Backus-Naur-Form basiert und zum Parsen eines Strings in einen Baum mathematischer Ausdrücke (Def. 5) dient kann im Source Code unter dem

Pfad /server/cas/parse.ts oder online über https://github.com/Flosi23/cas/blob/main/server/cas/parse.ts eingesehen werden.
Außerdem bietet sich die Möglichkeit das Verhalten des Algorithmus auf der website https://cas.dotenv.de direkt selbst auszuprobieren. Dafür muss ein mathematischer Ausdruck welcher der konventionellen Struktur entspricht (Def. 8) (Dabei sollte vor allem beachtet werden, dass zum Beispiel der Term $3x$ als $3 * x$ geschrieben werden muss) in das Eingabefeld eingeben und der Knopf „Calculate“ gedrückt werden. Der vom Algorithmus generierte Baum kann dann unter dem Reiter “Expression Tree“ eingesehen werden.

2 Automatische Vereinfachung

Jedes Computeralgebrasystem wendet mathematische Regeln an, um einen Ausdruck zu vereinfachen. Diesen Prozess nennt man automatische Vereinfachung. Jeder Ausdruck wird automatisch vereinfacht, bevor andere Operationen angewendet werden. Motivation hinter der automatischen Vereinfachung ist es, den Term in eine einheitliche und simplere Form zu bringen, damit man ihn mit anderen Termen vergleichen und damit rechnen kann.

Beispiel 2. *Nimmt man den Term $3 + x + y$ so gibt es aufgrund des Kommutativgesetzes 6 verschiedene Möglichkeiten diesen niederzuschreiben ohne, dass sich sein Ergebnis verändert*

1. $3 + x + y$
2. $3 + y + x$
3. $y + 3 + x$
4. $y + x + 3$
5. $x + y + 3$
6. $x + 3 + y$

Obwohl alle 6 Terme equivalent sind, sind sie syntaktisch nicht gleich und würden aufgrund dessen von einem Computeralgebrasystem vor der automatischen Vereinfachung nicht als gleich angesehen werden. Ohne eine Vereinheitlichung könnten die Terme also nicht verglichen werden. Wieso das problematisch ist, lässt sich an einem einfachen Beispiel zeigen.

Beispiel 3. *Nehme man den Term $6xy + 8yx$. Für einem Menschen ist klar ersichtlich, dass man diesen zu $14xy$ vereinfachen kann. Ein Computeralgebrasystem könnte diese Vereinfachung aber nicht vornhemen, da es die Terme xy und yx nicht als gleich ansehen und folglich nicht addieren würde.*

Neben der Sortierung ist ein weiterer wichtiger Schritt Operatoren wie Division und Differenz durch andere Operatoren zu ersetzen, um die Anzahl möglicher Operatoren zu reduzieren und den Baum mathematischer Ausdrücke so zu simplifizieren.

Zuletzt sollen auch alle Zahlenwerte so weit zusammengerechnet werden wie möglich und mathematische Gesetze (wie etwa das Distributivgesetz) angwendet werden um den Term zu vereinfachen.

Definition 11. *Ein Term gilt als **vollständig vereinfacht**, wenn die folgenden Bedingungen erfüllt sind:*

1. $+$ *ist ein Operator mit mindestens zwei Operanden (aufgrund des Kommutativgesetzes ist es möglich, dass + n viele Operanden haben kann) von denen maximal 1 Operand eine Zahl oder ein Bruch sein darf (andernfalls könnte man den Ausdruck noch weiter zusammenfassen)*
2. $*$ *ist ein Operator mit mindestens zwei Operanden (Kommutativgesetz), von denen keiner ein Produkt sein darf. Außerdem darf nur maximal ein Operand eine Zahl oder ein Bruch sein.*
3. *Der Operator* $-$ *darf nicht vorkommen, da er immer als Produkt mit -1 dargestellt werden kann.*
4. *Auch der Operator* $/$ *darf nicht vorkommen, da man einen Quotienten immer als Produkt oder numerischen Bruch darstellen kann.*
5. *Ein Quotient, der einen Bruch bei dem sowohl Zähler, und Nenner nicht Null und natürliche Zahlen sind aufweist, muss als Bruch dargestellt werden. Außerdem muss der Nenner größer als eins und der größte gemeinsame Teiler eins sein.*
6. *Bei einer Potenz* v^n *muss* n *eine natürliche Zahl sein. Außerdem darf* v *keine eine Zahl, Bruch, Potenz oder ein Produkt sein*

2.1 Transformationen

Folgende Transformationen stellen die Grundlage eines jeden Algorithmus der automatischen Vereinfachung dar und können, wenn richtig angewandt einen Term vollständig vereinfachen

2.1.1 Distributive Transformation

Definition 12. *Bei der **distributiven Transformation** werden gleiche Terme mit konstanten Koeffizienten in einer Summe basierend auf dem Distributivgesetz vereinfacht, indem ihre Koeffizienten miteinander addiert werden.*

Beispiel 4. $2x + y + \frac{3}{2}x = \frac{7}{2}x + y$

2.1.2 Assoziative Transformation

Definition 13. *Existiert ein Term U, der eine Summe bzw. ein Produkt ist, und einer der Operanten S ist auch eine Summe bzw. ein Produkt, so werden die Operatoren von S zu U addiert. Diese Transformation wird immer vor der Distributiven Transformation ausgeführt.*

Beispiel 5. $2 * (x * y) = 2 * x * y$

2.1.3 Kommutative Transformation

Definition 14. *Die **kommutative Transformation**, basierend auf dem Kommutativgesetz, dient der Sortierung von Termen in Summen und Produkten, um eine einheitliche Form zu erreichen.*

Beispiel 6. $x + y + 2 = 2 + x + y$

2.1.4 Potenz-Transformationen

Definition 15. *Die **Potenz-Transforamtionen** basieren auf den 3 Potenzgesetzen:*

1. *P1* $a^m * a^n = a^{m+n}$
2. *P2* $(a * b)^n = a^n * b^n$
3. *P3* $(a^m)^n = a^{m*n}$

2.1.5 Quotienten-Transformation

Definition 16. *Quotienten werden komplett entfernt und stattdessen durch ein Produkt ersetzt. Ein Faktor des Produkts (der Nenner des Quotienten) ist eine Potenz mit der Basis des Nenners und dem Exponenten* -1

Beispiel 7. $u/v = u * v^{-1}$

2.1.6 Differenzen-Transformation

Definition 17. *Jede unäre Differenz wird durch eine Multiplikation mit -1 ersetzt. Jede binäre Differenz $u - v$ wird durch eine Summe ersetzt*

Beispiel 8. $-v = -1 * v$

Beispiel 9. $u - v = u + (-1) * v$

2.1.7 Identitäten-Transformationen

Definition 18. *Die Identitäten-Transformation definiert folgende Regeln welchen im Umgang mit Termen, die Einsen oder Nullen enthalten angewandt werden können*

1. $u + 0 = u$
2. $u * 0 = 0$
3. $u * 1 = u$
4. $0^w = \begin{cases} 0 & w \in \mathbf{Q} \ and \ w > 0 \\ undefined \end{cases}$
5. $1^w = 1$
6. $v^0 = \begin{cases} 1 & v \neq 0 \\ undefined & v = 0 \end{cases}$
7. $v^1 = v$

2.1.8 Numerische Transformationen

Definition 19. *Die numerischen Transformationen definieren, wie rationale Zahlen in Termen vereinfacht werden sollen.*

1. *Konstanten in einem Produkt werden miteinander multipliziert.* $3 * 2 * x = 6 * x$
2. *Konstanten in einer Summe werden miteinander addiert.* $3 + 3 + x = 6 + x$
3. *Potenzen, deren Basis sowie Exponent Konstanten sind, werden berechnet.* $3^2 = 9$
4. *Brücke werden so weit wie möglich gekürzt.* $\frac{12}{8} = \frac{3}{4}$

2.2 Algorithmen

Basierend auf diesen Transformationen lassen sich nun Algorithmen für die Vereinfachung von Differenzen, Divisionen, Potenzen, Additionen, Summen und Produkten formulieren.

2.2.1 Vereinfachung einer Differenz

Bei der Vereinfachung einer Differenz wird nur die Differenzen-Transforamtion (Def. 17) angewandt.

2.2.2 Vereinfachung einer Division

Sind Zähler und Nenner rationale Zahlen, wird die Division zu einem Bruch umgeschrieben. Sonst wird die Quotienten Transformation (Def. 16) angewandt.

2.2.3 Vereinfachung einer Potenz

1. Ist die Basis 0, wird die Identitäten-Transformation 4 (Def. 18) angewandt.
2. Ist der Exponent eine ganze Zahl:
   (a) Ist der Exponent 0 oder 1, wird die Identitäten-Transformation 6 oder 7 (Def. 18) angewandt.
   (b) Ist die Basis eine rationale Nummer, so wird die numerische Transformation 3 (Def. 19) und auf deren Ergebnis die numerische Transformation 4 (Def. 19) angewandt.
   (c) Ist die Basis eine Potenz, wird die Potenztransformation 2 (Def. 15) angewandt.
   (d) Ist die Basis ein Produkt, wird die Potenztransformation 3 (Def. 15) angewandt.
3. Trifft keiner der oberen Fälle zu, wird nichts vereinfacht.

2.2.4 Vereinfachung eines Produkts

Ein Produkt wird vereinfacht, indem die folgenden Transformationen in genau dieser Reihenfolge angwendetwerden.

1. Assoziative Transformation (Def. 13).
2. Numerische Transformation 1 und 4 (Def. 19).
3. Gleiche Terme werden miteinander multipliziert: Anwendung der Potenz-Transformation 1 (Def. 15), wenn beide Terme Potenzen mit gleicher Basis sind
4. Identitäten-Transformation 2 und 3 (Def. 18).
5. Kommutative-Transformation (Def. 14)

2.2.5 Vereinfachung einer Summe

Eine Summe wird vereinfacht, indem die folgenden Transformationen in genau dieser Reihenfolge angwendetwerden.

1. Assoziative Transformation (Def. 13).
2. Numerische Transformation 2 und 4 (Def. 19).
3. Distributive Transformation (Def. 12)
4. Identitäten-Transformation 1 und 3 (Def. 18).
5. Kommutative-Transformation (Def. 14)