

Computer Algebra Systeme

Simon Weckler

9. September 2022

Inhaltsverzeichnis

1	Einleitung	2
2	Umgang mit Benutzereingaben	2
2.1	Darstellung eines Ausdrucks im Programm	3
2.1.1	Darstellung als String	3
2.1.2	Darstellung als Baum	3
2.2	Parsen von Benutzereingaben	5
2.2.1	Konventionelle Struktur eines Ausdrucks	5
2.2.2	Grammatik eines konventionellen Ausdrucks	6
3	Automatische Vereinfachung	9
3.1	Transformationen	10
3.1.1	Distributive Transformation	10
3.1.2	Assoziative Transformation	11
3.1.3	Kommutative Transformation	11
3.1.4	Potenz-Transformationen	11
3.1.5	Quotienten-Transformation	11
3.1.6	Differenzen-Transformation	11
3.1.7	Identitäten-Transformationen	12
3.1.8	Numerische Transformationen	12
3.2	Algorithmen	12
3.2.1	Vereinfachung einer Differenz	12
3.2.2	Vereinfachung einer Division	13
3.2.3	Vereinfachung einer Potenz	13
3.2.4	Vereinfachung eines Produkts	15
3.2.5	Vereinfachung einer Summe	16
3.2.6	Vereinfachung eines Ausdrucks	17
4	Schlusswort	17
5	Bibliografie	18

1 Einleitung

Die Nachfrage nach Rechenhilfsmitteln ist fast so alt wie die Mathematik. Was lange Zeit der Abakus war, sind heute Computeralgebrasysteme. Diese Systeme sind für Studenten, Forschung und Industrie ein unerlässliches Werkzeug und bilden ein eigenes Marktsegment.

Aus der Definition von Computeralgebra der Gesellschaft für Informatik [5] lässt sich folgende Definition für ein Computeralgebrasystem ableiten:

Definition 1. *Ein Computeralgebrasystem ist ein Programm, welches Methoden zum Lösen mathematisch formulierter Probleme durch Algorithmen bietet. Es beruht auf der exakten endlichen Darstellung mathematischer Objekte und Strukturen.*

Das Rechnen mit beliebig großen Zahlen, Symbolen, Unbestimmten und Polynomen, das Faktorisieren von Zahlen und Polynomen, das Differenzieren von Polynomen und anderen Funktionen und das exakte Lösen von Differenzialgleichungen sind einige konkrete Beispiele für Problemstellungen, die die meisten Computeralgebrasysteme lösen können.

Damit ein Computeralgebrasystem nützlich ist, sollte es folgende Eigenschaften haben:

1. Eine Benutzeroberfläche, über die der Benutzer mathematische Ausdrücke eingeben kann.
2. Einen Vereinfachungsalgorithmus, welcher mathematische Ausdrücke vereinfachen kann.
3. Eine große Menge an mathematischen Algorithmen und Funktionen. Dazu gehören zum Beispiel Algorithmen zur Bestimmung der Ableitung eines Terms.
4. Eine eigene Programmierumgebung, insbesondere eine eigene Programmiersprache.

In dieser Arbeit soll nachvollzogen werden, wie Computeralgebrasysteme mit den Texteingaben von Benutzern umgehen und diese intern darstellen. Außerdem soll die Funktionsweise eines Computeralgebrasystem anhand des Algorithmus der automatischen Vereinfachung dargestellt werden. Für diese Arbeit wurde die Website <https://cas.dotenv.de> erstellt. In dieser Website wurden die Algorithmen, welche in dieser Arbeit vorgestellt werden, implementiert und können visuell nachvollzogen werden.

2 Umgang mit 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 die in Tabelle 1 dargestellten Beziehungen zwischen den Zeichen und ihrer mathematischen Bedeutung.

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

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

Tabelle 1: Mathematische Bedeutung der Zeichen

2.1 Darstellung eines Ausdrucks im Programm

2.1.1 Darstellung als String

Der Benutzer kann einen mathematischen Ausdruck in Form eines Strings eingeben. Um zu beurteilen, ob eine Darstellung als String auch für einen Computer passend ist oder nicht, muss zuerst beschrieben werden, welche Informationen das Programm benötigt, um mit dem Term arbeiten zu können. Diese sind die Bedeutung der einzelnen Zeichen sowie die Operatorrangfolge (Def. 2).

Definition 2. Unter der **Operatorrangfolge** oder **Operatorpräzedenz** ist die Ordnung zu verstehen, in welcher die Operatoren eines Ausdrucks auszuwerten sind.

Beispiel 1. 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.

Ein String bietet sich folglich nicht als Datenstruktur für einen mathematischen Ausdruck an, da weder die Operatorpräzedenz noch die Bedeutung der Zeichen klar ersichtlich ist. Für ein Programm existiert schließlich per se kein Unterschied zwischen dem Zeichen 3 und $+$, da beide denselben Datentyp (*char*) haben.

2.1.2 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. Ein Baum, bestehend aus Ausdrücken, ist dafür passend, da er diese Bedingungen, wie später erläutert, erfüllt.

Laut Cohen [1, S. 81] kann ein Ausdruck entweder atomar oder zusammengesetzt sein.

Definition 3. Ein **atomarer Ausdruck** ist eine ganze Zahl oder ein Symbol. Atomare Ausdrücke können nicht weiter vereinfacht werden.

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

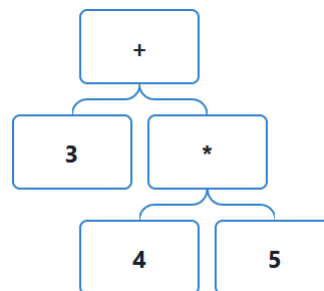


Abbildung 1: Der Baum für den Term $3 + 4 * 5$.

Beispiel 2. An dem Baum für den Term $3 + 4 * 5$ kann man erkennen, dass die Operatorrangfolge 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 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.

Von welchem Datentyp die Blätter und Knoten sind, ist ganz von der jeweiligen Implementierung abhängig. In dem für diese Arbeit geschriebenen Programm können folgende Ausdrücke Teil des Baumes sein: Welche Ausdrücke Teil des Baumes, in dem für diese Arbeit geschriebenen Programm sein können, ist in Abbildung 2 zu sehen.

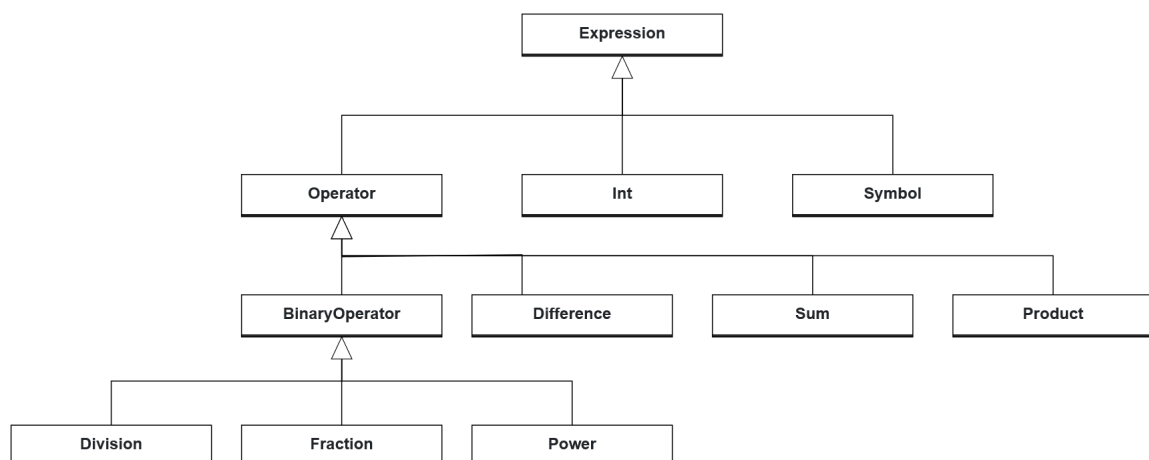


Abbildung 2: UML Diagramm aller Ausdrücke.

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

Definition 6. Binäre Operatoren sind Operatoren, die immer zwei Operanden haben.

Definition 7. Nicht-Binäre Operatoren sind Operatoren, deren Operandenzahl zwischen 1 und N variieren kann.

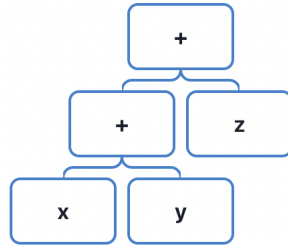


Abbildung 3: $x + y + z$ in binären Summen

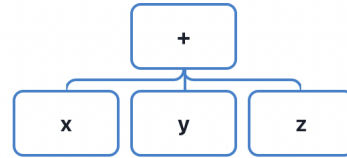


Abbildung 4: $x + y + z$ als eine Summe mit 3 Kindern

Beispiel 3. Die Summe $x + y + z$ wird von Menschen normalerweise interpretiert als: $(x + y) + z$. Die Operatorrangfolge ist hier aber nicht relevant, es gilt: $(x + y) + z = x + (y + z)$. Aus diesem Grund muss die Operatorrangfolge auch im Baum nicht berücksichtigt werden. Der Ausdruck kann also als Summe mit 3 Operanden dargestellt werden.

Da der Bruchstrich äquivalent zu einer Division ist, ist dieser auch ein Operator mit 2 Operanden und wird im Baum mathematischer Ausdrücke wie folgt dargestellt:

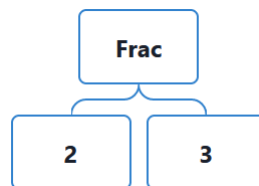


Abbildung 5: Ein Bruch im Baum mathematischer Ausdrücke.

2.2 Parsen von Benutzereingaben

2.2.1 Konventionelle Struktur eines Ausdrucks

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. Um die Struktur der Eingaben zu kennen, ist es notwendig diese zu definieren. Hier wurde die Definition Cohens [1, S. 86 – 87] leicht abgeändert. Insbesondere wurde die Anzahl möglicher Operatoren reduziert.

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 Operanden.
2. die Operatoren $*$, $/$, und $^$ haben immer 2 Operanden.
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

^
* /
+ -

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.

2.2.2 Grammatik eines konventionellen Ausdrucks

Um aus diesen Regeln einen Algorithmus zu bauen, ist es hilfreich, eine Grammatik für einen mathematischen Ausdruck unter Beachtung der Definition der konventionellen Struktur in Form einer sogenannten Backus-Naur-Form nach Astbury [3] zu definieren.

Definition 9. Eine **Backus-Naur-Form** (BNF) ist eine Notationsform, um die Grammatik von Sprachen zu definieren. Die Syntax einer Backus-Naur-Form besteht aus 3 Symbolen.

Zeichen	Bedeutung
$::=$	Ist definiert als
	Oder
$\langle X \rangle$	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. Ist die Definition abhängig von einer weiteren Definition, so überprüfe ob der String oder Teilstring auch der Kinddefinition entspricht.
3. Entspricht der String der Definition, so erweitere den Baum mit dem passenden Knoten beziehungsweise 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.

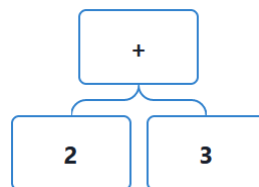


Abbildung 6: Parse Tree für den Input $2 + 3$.

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. Dabei dient die Definition einer Backus-Naur-Form für mathematische Ausdrücke von Gordon [4] als Vorlage.

```

<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
<symbol>   ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n |
               o | p | q | r | 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:

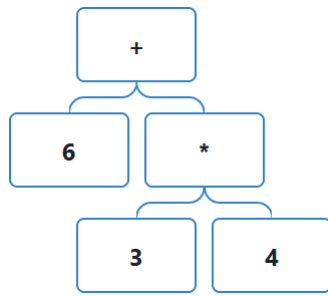


Abbildung 7: 1. Möglicher Baum für $6 + 3 * 4$.

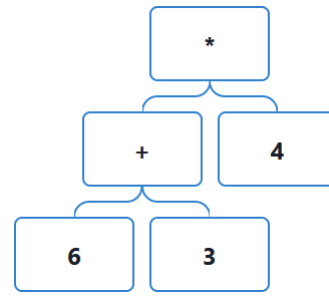


Abbildung 8: 2. Möglicher Baum für $6 + 3 * 4$.

```

<exp>      ::= <exp> + <exp> | <exp> - <exp> | <term>
<term>     ::= <term> * <term> | <term> / <term> | <power>
<power>    ::= <power> ^ <power> | <factor>
<factor>   ::= <int> | <symbol>
<int>      ::= <digit> <int> | <digit>
<digit>    ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<symbol>   ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n |
              o | p | q | r | 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.

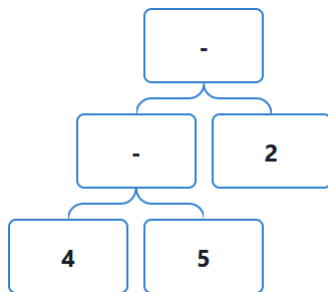


Abbildung 9: 1. Möglicher Baum für $4 - 5 - 2$.

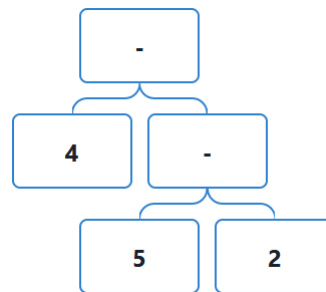


Abbildung 10: 2. Möglicher Baum für $4 - 5 - 2$.

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>      ::= + <term> | <exp> + <term> | - <term> | <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
<symbol> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n |
           o | p | q | r | 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 eingegeben und der Knopf „Calculate“ gedrückt werden. Der vom Algorithmus generierte Baum kann dann unter dem Reiter „Expression Tree“ eingesehen werden.

3 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 simple Form zu bringen, damit man ihn mit anderen Termen vergleichen und damit rechnen kann.

Beispiel 4. 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 äquivalent 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 5. 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 vornehmen, 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) angewendet werden, um den Term zu vereinfachen.

Definition 11. *Basierend auf der Definition Cohens [1, S. 90 – 92] gilt ein Term als **vollständig vereinfacht**, wenn die folgenden Bedingungen erfüllt sind:*

1. *$+$ ist ein Operator mit mindestens zwei Operanden von denen maximal 1 Operand eine rationale Zahl und keiner eine Summe ist.*
2. *$*$ ist ein Operator mit mindestens zwei Operanden, von denen keiner ein Produkt oder eine Summe ist. Maximal ein Operand ist eine rationale Zahl.*
3. *Rationale Zahlen sind in Summen und Produkten immer der 1. Operand.*
4. *Die Operatoren $-$ und $/$ kommen nicht vor.*
5. *Ein Quotient $\frac{u}{v}$, wobei $u \in \mathbb{N}$ und $v \in \mathbb{N} \setminus \{0\}$, wird als Bruch dargestellt. Für den Bruch $\frac{u}{v}$ gilt: $v > 1$, u und v teilerfremd.*
6. *Für einer Potenz v^n gilt: v ist keine Potenz oder Produkt.*

3.1 Transformationen

Folgende Transformationen Cohens [2, S. 64 – 79] sind Grundlage jedes Algorithmus zur automatischen Vereinfachung und können, wenn richtig angewandt, einen Term vollständig vereinfachen.

3.1.1 Distributive Transformation

Definition 12. *Bei der **distributiven Transformation** werden in Summen gleiche Terme mit konstanten Koeffizienten mit Hilfe des Distributivgesetzes vereinfacht, indem ihre Koeffizienten miteinander addiert werden. Ist einer der Operanden einer Multiplikation eine Summe, so werden die anderen Operanden der Multiplikation mit den Operanden der Summe multipliziert.*

Beispiel 6. Anwendung der distributiven Transformation bei einer Summe:

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

Beispiel 7. Anwendung der distributiven Transformation bei einer Multiplikation:

$$6 * x * (3 + y) = 6 * (3x + xy) = 18x + 6xy$$

3.1.2 Assoziative Transformation

Definition 13. Existiert ein Term U , der eine Summe oder ein Produkt ist, und einer der Operanden S ist auch eine Summe oder ein Produkt, so werden die Operanden von S zu U addiert. Diese Transformation wird immer vor der Distributiven Transformation angewendet.

Beispiel 8. $2 * (x * y) = 2 * x * y$

3.1.3 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 9. $x + y + 2 = 2 + x + y$

3.1.4 Potenz-Transformationen

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

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

3.1.5 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 10. $u/v = u * v^{-1}$

3.1.6 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 11. $-v = -1 * v$

Beispiel 12. $u - v = u + (-1) * v$

3.1.7 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. $0^w = \begin{cases} 0 & w \in \mathbf{Q} \text{ and } w > 0 \\ \text{undefined} & \text{sonst} \end{cases}$
5. $1^w = 1$
6. $v^0 = \begin{cases} 1 & v \neq 0 \\ \text{undefined} & \text{sonst} \end{cases}$
7. $v^1 = v$

3.1.8 Numerische Transformationen

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

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

3.2 Algorithmen

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

3.2.1 Vereinfachung einer Differenz

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

Beispiel 13. In diesem Beispiel wird die Vereinfachung des Terms $2 - 3$ betrachtet. Dieser wird zuerst in eine Summe umgeschrieben und dann durch den Algorithmus zur Vereinfachung einer Summe zu -1 vereinfacht.

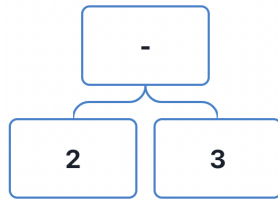


Abbildung 11: Baum von $2 - 3$.

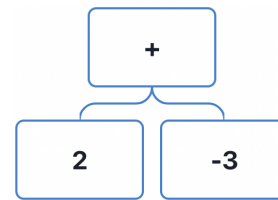


Abbildung 12: Anwendung der Differenzen-Transformation.



Abbildung 13: Die vollständig vereinfachte Summe.

3.2.2 Vereinfachung einer Division

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

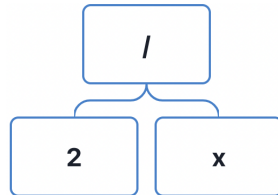


Abbildung 14: Baum von $\frac{2}{x}$.

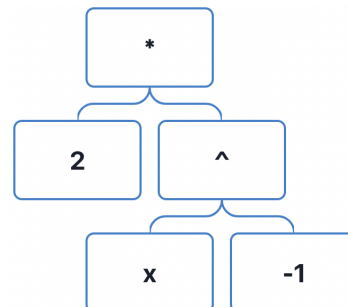


Abbildung 15: Anwendung der Quotienten-Transformation.

Beispiel 14. Der Term $\frac{2}{x}$ wird in ein Produkt umgeschrieben und dann so weit es geht vereinfacht. In diesem Fall ist das Produkt allerdings schon vollständig vereinfacht.

3.2.3 Vereinfachung einer Potenz

1. Ist die Basis 0, wird die 4. Identitäten-Transformation (Def. 18) angewandt.
2. Ist der Exponent eine ganze Zahl:
 - (a) Ist der Exponent 0 oder 1, wird die 6. beziehungsweise 7. Identitäten-Transformation (Def. 18) angewandt.
 - (b) Ist die Basis eine rationale Zahl, so wird die 3. numerische Transformation (Def. 19) und auf deren Ergebnis die numerische Transformation 4 (Def. 19) angewandt.

(c) Ist die Basis eine Potenz, wird die 2. Potenz-Transformation (Def. 15) angewandt.

(d) Ist die Basis ein Produkt, wird die 3. Potenz-Transformation (Def. 15) angewandt.

3. Trifft keiner der oberen Fälle zu, wird nichts vereinfacht.

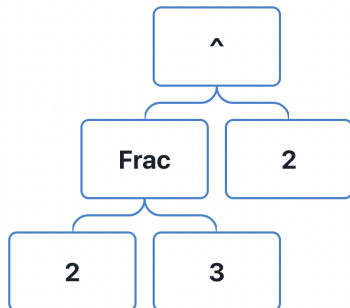


Abbildung 16: Baum des Ausdruckes $\frac{2}{3^2}$.

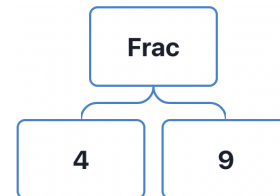


Abbildung 17: Anwendung der 3. numerische Transformation.

Beispiel 15. Da der Exponent im Term $\frac{2}{3^2}$ eine ganze Zahl ist, wird hier die 3. numerische Transformation angewandt. Der resultierende Bruch wird nicht weiter durch die 4. numerische Transformation vereinfacht, da er schon vollständig vereinfacht ist.

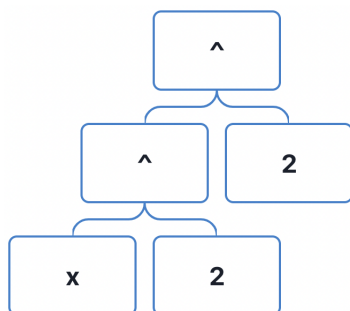


Abbildung 18: Baum von x^{2^2} .

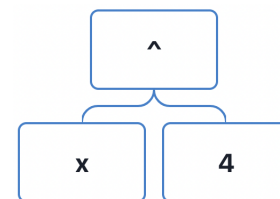


Abbildung 19: Anwendung der 3. Potenz-Transformation.

Beispiel 16. Der Term x^{2^2} wird mithilfe der 3. Potenz-Transformation vereinfacht.

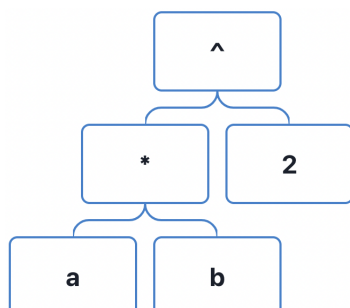


Abbildung 20: Baum von $(a * b)^2$.

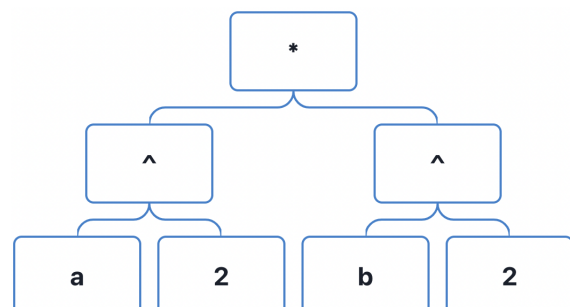


Abbildung 21: Anwendung 2. Potenz-Transformation.

Beispiel 17. Ist die Basis einer Potenz ein Produkt, wie etwa bei dem Term $(a * b)^2$, so wird die 2. Potenz-Transformation angewendet.

3.2.4 Vereinfachung eines Produkts

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

1. Assoziative Transformation (Def. 13).
2. Numerische Transformation 1 und 4 (Def. 19). Im selben Schritt wird die distributive Transformation angewandt und gleiche Terme miteinander multipliziert: Anwendung der 1. Potenz-Transformation (Def. 15), wenn beide Terme Potenzen mit gleicher Basis sind.
3. Identitäten-Transformation 2 und 3 (Def. 18).
4. Kommutative Transformation (Def. 14).

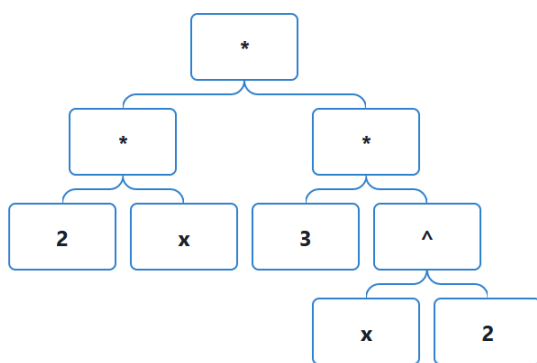


Abbildung 22: Baum von $(2 * x) * (3 * x^2)$.

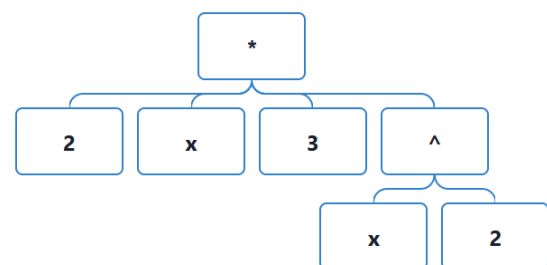


Abbildung 23: Anwendung der assoziativen Transformation.

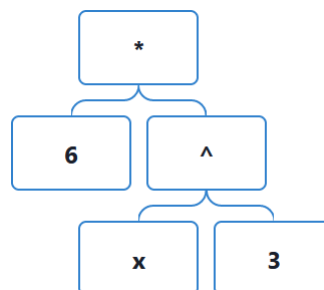


Abbildung 24: Anwendung der 1. numerischen Transformation und der 1. Potenz-Transformation

Beispiel 18. In diesem Beispiel wird der Term $(2 * x) * (3 * x^2)$ betrachtet. Zuerst wird die assoziative Transformation angewandt und die Operanden des Produkts werden mit den Operanden der Operanden ersetzt. Im nächsten Schritt wird die 1. numerische Transformation sowie die 1. Potenz-Transformation angewendet, da die Terme x und x^2 dieselbe Basis x haben. Es wird keine Identitäten-Transformation angewandt, da weder 0 noch 1 Operanden des Produkts sind. Auch die kommutative Transformation verändert nichts an der Struktur des Terms, da alle Operanden schon in der richtigen Reihenfolge sind.

3.2.5 Vereinfachung einer Summe

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

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).

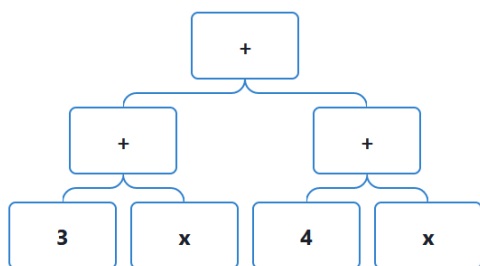


Abbildung 25: Baum von $(3 + x) + (4 + x)$

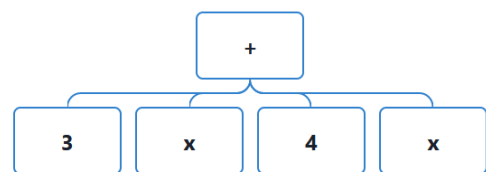


Abbildung 26: Anwendung der assoziativen Transformation.

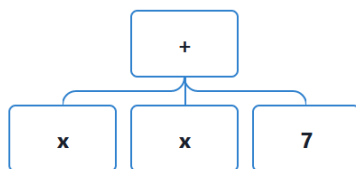


Abbildung 27: Anwendung der 2. numerischen Transformation.

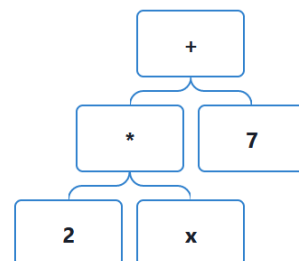


Abbildung 28: Anwendung der distributiven Transformation.

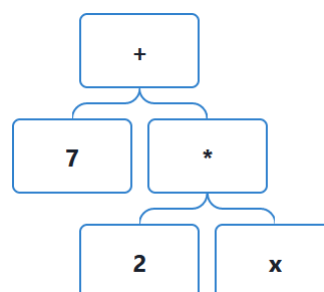


Abbildung 29: Anwendung der kommutativen Transformation.

Beispiel 19. In diesem Beispiel wird der Term $(3+x)+(4+x)$ betrachtet. Im 1. Schritt wird die assoziative Transformation angewandt. Darauf folgt die 2. numerische Transformation, bei

der 4 und 3 zu 7 zusammengefasst werden. Im 3. Schritt wird $x + x$ im Zuge der distributiven Transformation zu $2 * x$ vereinfacht. Es wird keine Identitäten-Transformation angewandt, da 0 kein Operand der Summe ist. Durch die kommutative Transformation werden 7 und $2 * x$ miteinander vertauscht, da ganze Zahlen immer am Anfang von Summen stehen müssen.

3.2.6 Vereinfachung eines Ausdrucks

Alle einzelnen Algorithmen werden in einem Algorithmus kombiniert, der zuerst alle Operanden vereinfacht, und dann abhängig vom Typ des Ausdrucks die angemessene Vereinfachung vornimmt.

Ist der Ausdruck:

1. Undefined \rightarrow undefined.
2. Ein Symbol oder eine ganze Zahl \rightarrow Ausdruck wird nicht weiter vereinfacht.
3. Ein Bruch \rightarrow Anwenden der 4. numerischen Transformation.
4. Eine Potenz \rightarrow Vereinfachung einer Potenz.
5. Eine Division \rightarrow Vereinfachung einer Division.
6. Ein Produkt \rightarrow Vereinfachung eines Produkts.
7. Eine Summe \rightarrow Vereinfachung einer Summe.
8. Eine Differenz \rightarrow Vereinfachung einer Differenz

Die Algorithmen, welche einen mathematischen Ausdruck vereinfachen, können im Source Code unter dem Pfad /server/cas/simplify oder online über <https://github.com/Flosi23/cas/tree/main/server/cas/simplify> eingesehen werden. Wie auch in Abschnitt 2.2 bietet sich zudem die Möglichkeit das Verhalten der Algorithmen auf der Website <https://cas.dotenv.de> selbst zu untersuchen. Der vereinfachte Baum ist nach Eingabe des Ausdrucks unter dem Reiter „Simplified Expression Tree“ zu finden.

4 Schlusswort

Modern Computeralgebrasysteme sind kein Hexenwerk. Komplizierte Operationen werden in immer kleinere Teilprobleme unterteilt bis das Problem für einen Computer einfach lösbar ist. So kann man die Vereinfachung eines Ausdrucks unterteilen in die Vereinfachung einer Summe, Potenz, usw... Die Vereinfachung einer Summe lässt sich nun auf die Anwendung verschiedener Transformationen zurückführen.

Was Computeralgebrasysteme aber so faszinierend macht, ist ihre Fülle an Funktionen und die Geschwindigkeit mit der diese angewandt werden. In diesen beiden Punkten schneidet

das im Zuge dieser Arbeit geschriebene Programm schlecht ab. Da es weder eine eigene Programmiersprache noch vielfältige Algorithmen zur Manipulation von Termen anbietet, kann es kaum als Computeralgebrasystem bezeichnet werden. Vielmehr sollte es als ein Hilfsmittel zur Visualisierung der automatischen Vereinfachung und des Parsing der Benutzereingaben angesehen werden. Das Ziel des Programms und dieser Arbeit wurde somit erreicht.

5 Bibliografie

- [1] Joel S. Cohen. *Computer Algebra and Symbolic Computation - Elementary Algorithms*. A K Peters, 2002.
- [2] Joel S. Cohen. *Computer Algebra and Symbolic Computation - Mathematical Methods*. A K Peters, 2003.
- [3] Peter Astbury. *A-level Computing*. Hrsg. von Peter EJ Kemp. 2009.
- [4] Scott Gordon. *BNF (Backus-Naur Form) and Parse Trees*. 2015. URL: <https://athena.ecs.csus.edu/~gordonvs/135/resources/04bnfParseTrees.pdf>.
- [5] Fachgruppe der Gesellschaft für Informatik. *Was ist Computeralgebra?* Erscheinungsdatum unbekannt. URL: <https://fachgruppe-computeralgebra.de/computeralgebra/>.