Heinrich C. Mayr, Martin Pinzger (Hrsg.): INFORMATIK 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 2065

Überführung von arithmetischen Ausdrücken in ein normalisiertes Polynom mittels Baumtransformation

Felix Knispel [1]

Abstract: In dieser Arbeit soll eine Technik vorgestellt werden, arithmetische Terme in eine normalisierte Polynomialform mittels Baumtransformationen zu überführen. Die Manipulation arithmetischer Terme ist eine Grundfunktion heutiger Computeralgebrasysteme. Diese verwenden dabei verschiedenste Ansätze. Ziel dieser Arbeit ist die Entwicklung eines Ansatzes zur Termumformung unter Verwendung von Übersetzerbautechnologien. Eine solche Technologie sind *bottom-up rewrite systems*, die die technologische Grundlage der Baumtransformation in dieser Arbeit darstellen. Es wird das Werkzeug `poly` vorgestellt, welches eine Technik implementiert, die Baumtransformation iterativ und phasenweise durchzuführen, um eine Aufteilung der Problematik in Teilprobleme zu ermöglichen.

Keywords: Baumtransformation, Termersetzung, Normalformen, Übersetzerbau

1 Problemstellung

In modernen Computeralgebrasystemen, wie *MATLAB* [Ma16a] und *Maxima* [Ma16b], ist die Transformation von arithmetischen Ausdrücken eine bedeutsame Grundfunktion. Es gibt eine Vielzahl von Techniken zur symbolischen Verarbeitung von arithmetischen Termen, wie beispielsweise die funktionale Programmierung oder Listen-Datenstrukturen. Insbesondere die Verwendung von Listen ist im Bereich der Computeralgebrasysteme ein häufiges Mittel [Fa01]. Die Wahl der zu verwendenden Technik ist anwendungsabhängig. Im Projekt „Entwicklung hocheffizienter Pumpensysteme“ [Pr16], in dessen Rahmen diese Arbeit entstand, werden unter Verwendung von Übersetzerbautechnologien aus einer entworfenen domänenspezifischen Sprache Beschreibungen von MILP-Problemen (engl. *mixed integer linear programming*) generiert. MILP-Problemlöser, wie z. B. *SCIP* [Ac04], geben dabei eine Syntax vor, in der das Problem formuliert werden muss. So wird im Falle von SCIP auch verlangt, dass Nebenbedingungen in einer Polynomialform aufgestellt werden müssen. Da die Integrierung von Computeralgebrasystem und deren Ergebnisse in Übersetzerbautechnologien sich als umständlich erwies, ergab sich in diesem Zusammenhang die Problemstellung arithmetische Ausdrücke in eine normalisierte Polynomialform unter der Anforderung der Verwendung von Übersetzerbautechnologien zu überführen.

Im Feld des Übersetzerbaus gibt es im Bereich der Codeerzeugung verschiedene Technologien zur Generierung von optimalem Code. Eine solche Technologie ist die BURS-Theorie (engl. *bottom-up rewrite systems*) von Pelegrí-Llopart und Graham [PLG88]. BURS bietet durch auf dynamischer Programmierung basierten Kostenanalyse eine Möglichkeit

[1] Martin-Luther-Universität Halle-Wittenberg, Institut für Informatik, Von-Seckendorff-Platz 1, 06120 Halle

durch Transformation von Ableitungsbäumen (kosten-)optimalem Code in einer Zielsprache zu generieren. Die Technologie wird in dieser Arbeit genutzt, um arithmetische Ausdrücke, die als Ableitungsbäume dargestellt sind, in eine normalisierte Polynomialform zu überführen, indem diese Bäume transformiert werden. Emmelmann [Em94] konnte bereits zeigen, dass Baumtransformation zur Manipulation arithmetischer Terme genutzt werden kann.

Das in dieser Arbeit entwickelte Werkzeug `poly` bekommt einen arithmetischen Ausdruck als Eingabe, überführt diesen Ausdruck in ein normalisiertes Polynom und gibt dieses aus. Mit einer gegebenen Bezeichnermenge $V = \{v_1, \ldots, v_k\}$ sei die Normalform eines gültigen Ergebnispolynoms P gegeben durch

$$P = \sum_{i=0}^{n} \sum_{\substack{j_1+\ldots+j_k=i \\ 0\leq j_1,\ldots,j_k\leq n}} c_{j_1,\ldots,j_k} \cdot v_1^{j_1} \cdot \ldots \cdot v_k^{j_k},\ c_{j_1,\ldots,j_k} \in \mathbb{R}. \tag{1}$$

Das Problem der schrittweisen Überführung in ein normalisiertes Polynom wird dabei in drei Teilprobleme unterteilt. Im ersten Abschnitt sollen alle Klammern im Ausdruck durch Ausmultiplizieren aufgelöst werden. Das Ergebnis ist für die meisten MILP-Problemlöser bereits ein gültiges Polynom. Dieses entspricht in der Regel jedoch nicht obiger Normalform, bestehend aus Monomen der Form $c_{j_1,\ldots,j_k} \cdot v_1^{j_1} \cdot \ldots \cdot v_k^{j_k}$, da das Ausmultiplizieren nicht zwangsläufig eine feste Bezeichnerordnung innerhalb der Monome gewährleistet und gegebenenfalls noch Konstanten zusammengefasst werden müssen. Die weitere Überführung des Polynoms in die Normalform erfüllt den Zweck, das nach dem Ausklammern resultierende Polynom weiter zu verkürzen, mit dem Ziel die Laufzeit des MILP-Problemlösers durch Linearisierung zu verbessern. Im folgenden Schritt sollen innerhalb der Monome gleiche Faktoren zu Potenzen zusammengefasst werden, sodass sich innerhalb der Monome keine Bezeichner wiederholen. Der letzte Schritt hat das Ziel das Polynom in ein gültiges Ergebnispolynom gemäß obiger Normalform zu überführen. Hierfür müssen Monome, die dieselben Bezeichner mit jeweils den gleichen Exponenten aufweisen, zusammengefasst werden.

Technische Grundlage für die Implementierung des Werkzeugs ist das Übersetzerbauwerkzeugkasten *Eli* [Gr92, El16]. Diese Entwicklungsumgebung stellt unter anderem ein Baumersetzungswerkzeug zur Verfügung.

2 Grundlagen

2.1 Bottom-up rewrite systems

Bottom-up rewrite systems (BURS) werden im Bereich der optimalen Codegenerierung verwendet. Ausgehend von einem als Ableitungsbaum dargestellten Programm, wird in diesem Baum nach Mustern gesucht, um optimal Code in einer Zielsprache zu generieren. Zu optimierende Eigenschaften können beispielsweise die Anzahl der Anweisungen oder die Anzahl der Prozessorinstruktionen sein. BURS bietet dabei die Möglichkeit Ableitungsbäume zu *transformieren*, d. h. Teilbäume durch andere Bäume zu ersetzen. In dieser

Arbeit wird diese Technik genutzt, um arithmetische Terme, dargestellt in einer Baumstruktur, durch Transformation ihrer Ableitungsbäume umzustellen.

Bäume werden in Präfixnotation oder als Graph, wie im Beispiel unten, dargestellt. So stellt beispielsweise $o(t_1,t_2)$ einen Baum dar, der in der Wurzel mit einem Symbol o beschriftet ist, deren Kinder zwei Unterbäume t_1 und t_2 sind. Die Knotenbeschriftungen werden einem *Operatoralphabet* Σ entnommen. Symbole weisen eine Stelligkeit $n \in \mathbb{N}$ auf, die der Anzahl der Kinder entspricht. *Baummuster* sind Bäume über diesem Alphabet, erweitert um Symbole der Stelligkeit 0, die *Variablen* genannt werden. Variablen einer Variablenmenge *Var* sind Statthalter für gültige Bäume aus dem Operatoralphabet. In einem Baummuster dürfen ausschließlich Variablen als Blätter vorkommen. $\sigma(m)$ beschreibt die Zuweisung von Bäumen an Variablen eines Baummusters m. Ein Baummuster *m passt* auf einen Baum t, wenn es eine Variablenzuweisung σ gibt, sodass $\sigma(m) = t$. *Baumersetzungs-* oder *Baumtransformationsregeln* überführen Baummuster und haben die Form $\alpha \to \beta$, wobei alle Variablen aus β in α vorkommen müssen. Ein *Baumersetzungssystem* ist eine Menge von Baumersetzungsregeln. Die *Position* eines Knotens in einem Baum wird als Folge von natürlichen Zahlen dargestellt. Die Position der Wurzel eines Baumes t wird durch die leere Folge ε angegeben. Der Unterbaum von t an einer Position l wird durch $t_{@l}$ notiert. Jede natürliche Zahl in einer Folge beschreibt den Index eines Kindknotens von links nach rechts, beginnend bei 1. Ist ks eine Folge bestehend aus $k \in \mathbb{N}$ und einer Folge s und $=$ ist zu verstehen als „ist definiert als“, so hängen Positionen und Unterbäume folgendermaßen zusammen:

$$t_{@\varepsilon} = t$$
$$\mathrm{op}(t_1,\ldots,t_n)_{@ks} = (t_k)_{@s},\ 1 \le k \le n,\ op \in \Sigma_n.$$

Eine Baumersetzungsregel $r : \alpha \to \beta$ ist auf einen Baum t an der Position p *anwendbar*, wenn α auf $t_{@p}$ passt. Die *Anwendung* von r auf t ergibt einen neuen Baum, der t gleicht, nur dass der Unterbaum $t_{@p}$ durch $\sigma(\beta)$ ersetzt wird.

Beispiel:
Gegeben sei ein Operatoralphabet $\Sigma = \Sigma_0 \cup \Sigma_2$. Es seien $+, * \in \Sigma_2$ zweistellige Symbole für die entsprechenden arithmetischen Operationen. $a, 0, 5 \in \Sigma$ sind nullstellige Symbole und damit gültige Blätter im Baum. Der Baum $t =$ `*(+(5,0),a)` ist somit ein zulässiger Baum und soll den Term $(5+0) \cdot a$ darstellen.

Das Baummuster $m =$ `*(+(X,Y),Z)` passt auf t mit $\sigma = \{X = 5, Y = 0, Z = a\}$ für Variablen $X, Y, Z \in Var$ mit $Var \subset \Sigma_0$. Die Baumersetzungsregel $r :$ `+(X,Y)` $\to$ `+(Y,X)` ist auf t anwendbar, da das Baummuster der linken Seite auf $t_{@1}$ passt. Die Anwendung von r auf t ergibt $t =$ `*(+(0,5),a)`.

2.2 Baumtransformationsregeln

In dem folgenden Kapitel werden die verwendeten Baumtransformationsregeln aufgezeigt. Auf der linken Seite einer Regel steht das zu ersetzende Baummuster. Auf der rechten Seite steht das Baummuster, welches das Muster auf der linken Seite ersetzt. Die Muster bestehen aus den Symbolen `Add` (Addition), `Sub` (Subtraktion), `Mul` (Multiplikation), `Div` (Division), `Pot` (Potenzierung) und `Neg` (Vorzeichen-Minus). Die Klammerung ist implizit durch den Aufbau des Baumes gegeben. Nichtterminale beginnen mit einem `c` bei Konstanten, einem `v` bei Bezeichnern und einem `t` bei arithmetischen Termen. In den Transformationsregeln sind t_i somit Variablen im Sinne von BURS. Konstanten und Bezeichner sind gültige Terme und können somit auch auf Terme passen, aber nicht umgekehrt. Abbildung 1 zeigt eine Baumtransformationsregel für das Distributivgesetz und den zu ersetzenden sowie den resultierenden Unterbaum.

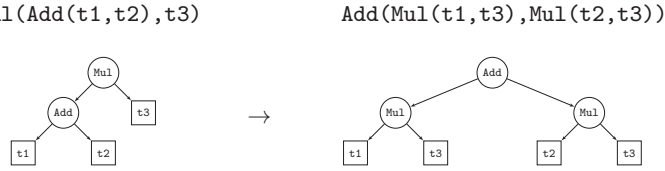

Abb. 1: Beispiel einer Baumtransformationsregel mit entsprechendem zu ersetzenden und resultierenden Baum

Die Operatoren „+“, „*“ und „/“ in Baumtransformationsregeln ergeben konstante Ergebnisse. Beispielsweise ergibt `c1+c2` das Ergebnis der Addition zweier Konstanten `c1` und `c2` als Konstante.

3 poly

3.1 Eingabeausdrücke

`poly` erwartet, dass der einzugebende arithmetische Term sich an eine bestimmte Form hält. Diese Form wird durch folgende kontextfreie Grammatik festgelegt. Kleingeschriebene Worte sind Nichtterminale. Operatoren sind Terminale und werden in einfachen Anführungszeichen angegeben. `ID` und `CONST` sind Terminale, die Bezeichner bzw. Konstanten repräsentieren. „|“ grenzt Auswahlmöglichkeiten voneinander ab.

```
expr     : sum .
sum      : sum '+' term |
           sum '-' term |
           term .
term     : term '*' factor |
           term '/' constant |
           factor .
```

```
factor   : ident '^' CONST |
           constant '^' CONST |
           primary .
primary  : ID | CONST | '-' primary | '(' sum ')' .
ident    : ID | '-' ident .
constant : CONST | '-' constant .
```

Es darf nur durch Konstanten dividiert werden. Potenzen dürfen entweder eine Konstante oder einen Bezeichner zur Basis haben. Als Exponenten sind ausschließlich natürliche Konstanten zulässig. Erlaubt sind somit folgende Operatoren mit der angegebenen Priorität.

	Klammerung `( )`				
$>$	Vorzeichen-Minus `-`	(Symbol: `Neg`)			
$>$	Potenzierung `^`	(Symbol: `Pot`)			
$>$	Multiplikation `*`	(Symbol: `Mul`)	$=$	Division `/`	(Symbol: `Div`)
$>$	Addition `+`	(Symbol: `Add`)	$=$	Subtraktion `-`	(Symbol: `Sub`)

3.2 Iterative Baumtransformation in Phasen

Für die Implementierung von `poly` wurde der Übersetzerbauwerkzeugkasten *Eli* verwendet. Eli stellt unter anderem ein Werkzeug zur Baumtransformation zur Verfügung. Nach der Angabe von Baumersetzungsregeln kann man diese auf Bäume einer internen Datenstruktur anwenden. Im Kontext der Transformation von arithmetischen Termen kann es aber vorkommen, dass die Anwendung einer Regel Muster im Baum hervorbringen kann, auf die wiederum Regeln anwendbar sind. Aus diesem Grund ist es nötig die Transformation der Terme in Iterationen ablaufen zu lassen, bis ein Fixpunkt erreicht ist, in dem der Term sich nicht mehr ändert. Folgender Algorithmus zeigt dieses Vorgehen.

```
repeat
  vergleich = term
  term = Baumtransformation(term)
until vergleich = term
return term
```

Die Grundidee zur Lösung des Problems der Überführung von arithmetischen Ausdrücken in ein normalisiertes Polynom ist die Unterteilung in Teilprobleme, die nacheinander gelöst werden sollen. Es ist also nötig verschiedene Baumersetzungssysteme zu implementieren und getrennt von einander in Phasen ablaufen zu lassen. In Eli kann man jedoch nur ein Baumersetzungssystem angeben. Dabei kann jedoch das Problem auftreten, dass mehrere Baumtransformationsregeln verschiedener Phasen um ein Muster im Baum konkurrieren. Somit musste eine Methode entwickelt werden die Baumtransformation in aufeinanderfolgenden Phasen unter Angabe eines Baumersetzungssystems ablaufen zu lassen. Damit Muster verschiedener Transformationsphasen nicht um dieselben Teilbäume konkurrieren,

werden zusätzliche Operatorsymbole eingeführt. Es werden die Symbole der entsprechenden Operationen mit einer Ziffer der zugehörigen Transformationsphase versehen. Für die Addition ergeben sich somit die Operationssymbole `Add1` für die erste Transformationsphase, `Add2` für die zweite Transformationsphase und `Add3` für die dritte Transformationsphase. Obiger Algorithmus muss für alle drei Transformationsphasen nacheinander angewandt werden. Die Knoten des Baumes werden zunächst mit den Operatorsymbolen der ersten Transformationsphase versehen und das Baumersetzungssystem bis zum Erreichen eines Fixpunktes angewandt. Ist der Fixpunkt erreicht, wird der resultierende Baum mit den Operatorsymbolen der zweiten Transformationsphase versehen und das Verfahren wird erneut angewandt. Selbiges schließlich auch im dritten Transformationsschritt.

3.3 Überführung in ein normalisiertes Polynom

Der Ablauf der kompletten Umformung in ein Polynom wird in drei Schritte unterteilt. Jeder dieser Schritte hat eine bestimmte Polynomialform als Ergebnis, welche für den folgenden Schritt als Ausgang dient. Diese Unterteilung hat zudem den Effekt, dass das Verfahren übersichtlich geordnet abläuft und die Auswahl der anzuwendenden Transformationsregeln in jedem Schritt eingeschränkt wird. Abbildung 2 stellt das strategische Vorgehen beim Aufruf einer Transformation dar.

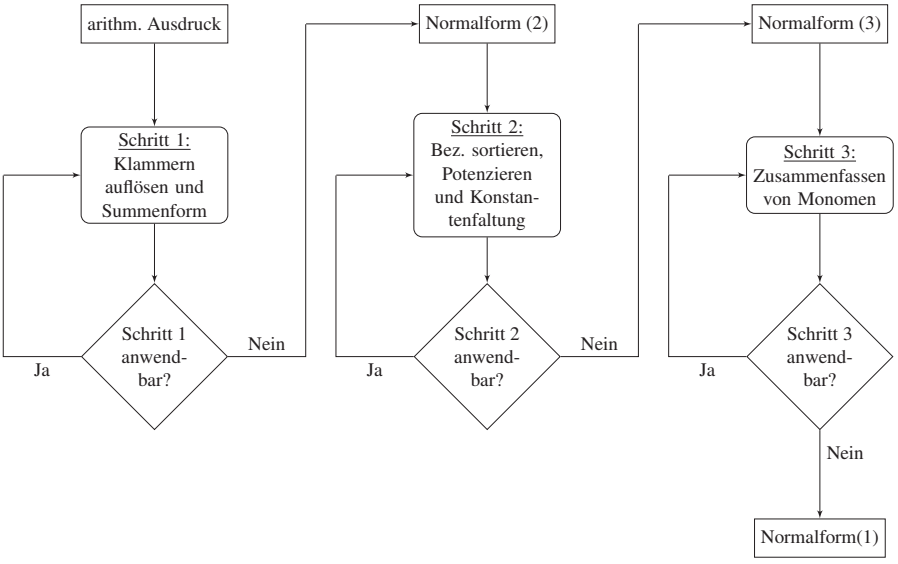

Abb. 2: Schematische Darstellung der Überführung von arithmetischen Ausdrücken in ein normalisiertes Polynom

Klammern auflösen und Überführung in Summenform

Im ersten Schritt soll der eingegebene arithmetische Term in eine klammerfreie Summe von Monomen überführt werden. Das Ergebnis dieses Schrittes soll mit einer Bezeichner-

menge V folgender Polynomialform genügen.

$$\sum v_1 \cdot \ldots \cdot v_k,\ k \in \mathbb{N},\ v_1, \ldots, v_k \in V \cup \mathbb{R} \tag{2}$$

Die Monome des Ergebnispolynoms dieses Schritts sind ein beliebiges Produkt von Konstanten und Bezeichnern. Konstanten und Bezeichner können in einem Monom mehrfach vorkommen. Abbildung 3 listet alle hierfür nötigen Transformationsregeln auf.

(A) Assoziativgesetz			
`Add1(t1,Add1(t2,t3))`	→	`Add1(Add1(t1,t2),t3)`	(1)
(B) Distributivgesetz			
`Mul1(Add1(t1,t2),t3)`	→	`Add1(Mul1(t1,t3),Mul1(t2,t3))`	(2)
`Mul1(t1,Add1(t2,t3))`	→	`Add1(Mul1(t1,t2),Mul1(t1,t3))`	(3)
(C) Eliminierung von Subtraktion und Division			
`Div1(c1,c2)`	→	`c1/c2`	(4)
`Div1(t,c)`	→	`Mul1(1/c,t)`	(5)
`Neg1(t1)`	→	`Mul1(-1,t1)`	(6)
`Sub1(t1,t2)`	→	`Add1(t1,Mul1(-1,t2))`	(7)

Abb. 3: Baumtransformationsregeln, um einen arithmetischen Ausdruck in einen Ausdruck ohne Klammerung und Subtraktion zu überführen.

Das Distributivgesetz findet Anwendung, um alle Summen innerhalb von Multiplikationen auszuklammern. Um die in der Normalform für Ergebnispolynome geforderte Summenform zu gewährleisten, werden Subtraktionen eliminiert, indem sie in eine Addition und eine Multiplikation des zweiten Operanden mit der Konstante -1 umgeformt werden. Die Anwendung des Distributivgesetzes und der Eliminierung der Subtraktion ermöglichen zusammen eine resultierende Summenform nach diesem Schritt. Vorbereitend, um den dritten Schritt zu erleichtern, wird das Assoziativgesetz für die Addition angewandt, um den Baum der Summe nach links abzuleiten.

Sortierung der Bezeichner, Potenzieren und Konstantenfaltung

Der erste Transformationsschritt stellt die klammerfreie Summenform der Ergebnispolynome her. Der folgende zweite Schritt überführt die Monome innerhalb der Summe in die gewünschte Form. Das Ergebnis dieses Schrittes genügt mit einer Bezeichnermenge V folgender Polynomialform.

$$\sum_{i=0}^{n} \sum_{\substack{j_1+\ldots+j_k=i \\ 0 \le j_1,\ldots,j_k \le n}} c \cdot v_1^{j_1} \cdot \ldots \cdot v_k^{j_k},\ c \in \mathbb{R},\ v_1, \ldots, v_k \in V \tag{3}$$

Um die Transformation zu erleichtern, werden die Monome nach links abgeleitet. Dies verringert die benötigten Transformationsregeln. Hierfür ist eine Realisierung des Assoziativgesetzes für die Multiplikation nötig. Eine Anforderung der Normalform für Ergebnispolynome ist, dass höchstens eine Konstante in einem Monom vorkommt, die zudem im

Monom ganz links stehen soll. Dies wird in diesem Schritt realisiert, indem Konstanten im nach links abgeleiteten Unterbaum des Terms nach unten gereicht werden, bis diese im unteren linken Blatt des Unterbaums stehen. Kommen mehrere Konstanten in einem Monom vor, so werden sie solange nach unten gereicht, bis zwei Konstanten Kinder des untersten Multiplikationsknotens sind. Dieses Muster wird durch einen einzelnen Konstantenknoten mit dem Ergebnis der Multiplikation beider Konstanten ersetzt.

(A) Assoziativgesetz			
`Mul2(t1,Mul2(t2,t3))`	→	`Mul2(Mul2(t1,t2),t3)`	(1)
(B) Konstantenfaltung			
`Mul2(c1,c2)`	→	`c1*c2`	(2)
`Mul2(t,c)`	→	`Mul2(c,t)`	(3)
(C) Bezeichner sortieren und Potenzieren			
`Mul2(c,v)`	→	`Mul2(c,Pot2(v,1))`	(4)
`Mul2(v1,v2)`	→	*wenn v1 = v2:* `Pot2(v1,2)` *wenn v1 < v2:* `Mul2(Pot2(v1,1),Pot2(v2,1))` *wenn v2 < v1:* `Mul2(Pot2(v2,1),Pot2(v1,1))`	(5)
`Mul2(Pot2(v1,c),v2)`	→	*wenn v1 = v2:* `Pot2(v1,c+1)` *wenn v1 < v2:* `Mul2(Pot2(v1,c),Pot2(v2,1))` *wenn v2 < v1:* `Mul2(Pot2(v2,1),Pot2(v1,c))`	(6)
`Mul2(Mul2(t,Pot2(v1,c)),v2)`	→	*wenn v1 = v2:* `Mul2(t,Pot2(v1,c+1))` *wenn v1 < v2:* `Mul2(Mul2(t,Pot(v1,c)),Pot2(v2,1))` *wenn v2 < v1:* `Mul2(Mul2(t,v2),Pot2(v1,c))`	(7)

Abb. 4: Baumtransformationsregeln um arithmetische Ausdrücke weiter zu vereinfachen. Es werden Konstanten gefaltet sowie Bezeichner nach einer Bezeichnerordnung sortiert und potenziert. Die Vergleichsoperatoren $=$ und $<$ beziehen sich auf eine Bezeichnerordnung.

Weiterhin verlangt die Normalform, dass innerhalb der Monome gleiche Bezeichner zu Potenzen zusammengefasst und die Bezeichner nach einer Bezeichnerordnung sortiert sind. Die in Abbildung 4 aufgelisteten Baumtransformationsregeln realisieren die Potenzierung von Bezeichnern mittels einer zeitgleich stattfindenden Sortierung eben dieser. Damit das folgende Verfahren funktioniert, ist gegebenenfalls eine Vorverarbeitung des Termes nötig. So werden beim Einlesen der Eingabe alle Potenzen in Multiplikationen umgeformt. Ausgangspunkt für die Sortierung und Potenzierung ist nun ein nach links abgeleiteter Unterbaum, in dem gegebenenfalls im untersten linken Blatt eine Konstante steht. Bezeichner, die nun Kinder der untersten Multiplikation sind, werden durch Potenzen mit dem Exponenten 1 ersetzt. Potenzen befinden sich während dieses Verfahrens prinzipiell stets in den unteren Blättern des Baumes. Anschließend wird der Baum nach dem Muster `Mul(Mul(Term,Potenz),Bezeichner)` abgesucht. Ein Bezeichner wird nun entsprechend der Bezeichnerordnung im Baum nach unten gereicht, wenn der Bezeichner entsprechend der Ordnung kleiner als der Bezeichner der benachbarten Potenz

ist. Sobald dies nicht mehr möglich ist, wird der Bezeichner in eine Potenz umgeformt. Hier sind zwei Fälle zu beachten. (1) Der linke Nachbar ist eine Potenz, dessen Bezeichner gemäß der Ordnung kleiner ist. In diesem Falle wird der Bezeichner in eine Potenz mit dem Exponenten 1 umgeformt. (2) Der linke Nachbar ist eine Potenz, dessen Bezeichner der gleiche ist. In diesem Falle werden beide zu einer Potenz zusammengefasst, indem der Exponent des linken Nachbarn um eins erhöht wird. Das Verfahren arbeitet sich so Iteration für Iteration im Baum nach oben, solange bis alle Bezeichner in eine Potenz aufgenommen wurden. Am Ende dieses Schrittes bleibt die aus dem ersten Schritt resultierende Summenform erhalten. In diesem Schritt wird lediglich die Struktur der Monome verändert.

Zusammenfassen von Monomen

Im dritten und letzten Schritt der Transformation soll die Normalform für Ergebnispolynome

$$P = \sum_{i=0}^{n} \sum_{\substack{j_1+\ldots+j_k=i \\ 0\leq j_1,\ldots,j_k\leq n}} c_{j_1,\ldots,j_k} \cdot v_1^{j_1} \cdot \ldots \cdot v_k^{j_k},\ c_{j_1,\ldots,j_k} \in \mathbb{R},\ n \in \mathbb{N} \tag{1}$$

hergestellt werden. Ausgehend vom Ergebnis des zweiten Schritts ist es nun noch nötig Monome mit gleichen Bezeichnern zusammenzufassen, d. h. zwei Monome im Polynom weisen dieselben Bezeichner mit den gleichen Exponenten auf, so müssen sie zu einem Monom mit gleichen Bezeichnern zusammengefasst und die vorkommenden Konstanten aufsummiert werden. Grundlegende Idee dies zu realisieren ist es, im Baum nach benachbarten Monomen zu suchen, die die gleiche Bezeichnermenge aufweisen. Es ist also zunächst nötig, die Reihenfolge der Monome im Baum zu verändern. Hierfür wird die Vorgehensweise des zweiten Schritts, um Bezeichner in Monomen zu sortieren, aufgegriffen und auf die Sortierung von Monomen angepasst. Im zweiten Schritt wurden Bezeichner im Baum des entsprechenden Monoms entsprechend einer Bezeichnerordnung nach unten gereicht. Befanden sie sich an der richtigen Stelle, wurden sie „markiert“, indem sie mit einem Exponenten versehen wurden. Die Markierung war nötig, damit der Baum von unten nach oben in jeder Iteration abgesucht werden konnte, damit so sortierte Bezeichner von noch nicht einsortierten Bezeichnern unterschieden werden konnten. Für das Zusammenfassen von Monomen ist es somit ebenfalls nötig eine Ordnung auf Monome festzulegen, nach der sortiert werden kann. Sind zwei Monome gleich, so müssen sie zusammengefasst werden, d. h. ihre Konstanten müssen summiert werden. Für ein Monom m ermittle $\mathrm{k}(m)$ die darin vorkommende Konstante.

$$\mathrm{k}(m) = \begin{cases} c, & \exists c \in m,\ c \in \mathbb{R} \\ 1, & \text{sonst} \end{cases} \tag{4}$$

Für die Monome m_1 und m_2 mit gleicher Bezeichnermenge fasse $\mathrm{merge}(m_1, m_2)$ die Monome zusammen.

$$\mathrm{merge}(m_1, m_2) = (m_1 \setminus \mathrm{k}(m_1)) \cup \{\mathrm{k}(m_1) + \mathrm{k}(m_2)\} \tag{5}$$

Eine geschickte Markierung von bereits einsortierten Monomen ist nötig. Es empfiehlt sich den obersten Multiplikationsknoten im Unterbaum, an dem ein ganzes Monom hängt, zu markieren, damit dieser in der Mustersuche innerhalb der aus dem ersten Schritt der Transformation nach links abgeleiteten Summe berücksichtigt werden kann. Es wird ein neues Symbol `Mon3` (kurz für *Monom*) eingeführt, welches sich mathematisch nicht von dem Multiplikationssymbol `Mul3` unterscheidet. Ist der oberste Multiplikationsknoten eines Monoms mit `Mon3` markiert, so soll das bedeuten, dass das Monom an seiner endgültigen Stelle im Polynom einsortiert wurde. Am Ende dieses Transformationsschritts sind also nur Konstanten, `Add3`- oder `Mon3`-Knoten Kinder eines `Add3`-Knotens. Die in Abbildung 5 aufgelisteten Baumtransformationsregeln realisieren den letzten Transformationsschritt.

(A) Assoziativgesetz

`Add3(t1,Add3(t2,t3))`	→	`Add3(Add3(t1,t2),t3)`	(1)

(B) Konstantenfaltung

`Add3(c1,c2)`	→	`c1+c2`	(2)
`Add3(t,c)`	→	`Add3(c,t)`	(3)

(C) Sortieren und Zusammenfassen von Monomen

`Add3(t,v)`	→	`Add3(t,Mul3(1,Pot3(v,1))`	(4)
`Add3(v,t)`	→	`Add3(Mul3(1,Pot3(v,1)),t)`	(5)
`Add3(t,Pot3(v,c))`	→	`Add3(t,Mul3(1,Pot3(v,c))`	(6)
`Add3(Pot3(v,c),t)`	→	`Add3(Mul3(1,Pot3(v,c)),t)`	(7)
`Add3(c,Mul3(t1,t2))`	→	`Add3(c,Mon3(t1,t2))`	(8)
`Add3(`$\underbrace{\texttt{Mul3(t1,t2)}}_{\texttt{m1}}$`,`$\underbrace{\texttt{Mul3(t3,t4)}}_{\texttt{m2}}$`)`	→	*wenn* `m1` = `m2` : merge(`m1`,`m2`) *wenn* `m1` < `m2` : `Add3(Mon3(t1,t2),Mon3(t3,t4))` *wenn* `m1` > `m2` : `Add3(Mon3(t3,t4),Mon3(t1,t2))`	(9)
`Add3(`$\underbrace{\texttt{Mon3(t1,t2)}}_{\texttt{m1}}$`,`$\underbrace{\texttt{Mul3(t3,t4)}}_{\texttt{m2}}$`)`	→	*wenn* `m1` = `m2` : merge(`m1`,`m2`) *wenn* `m1` < `m2` : `Add3(m1,Mon3(t3,t4))` *wenn* `m1` > `m2` : `Add3(Mon3(t3,t4),m1)`	(10)
`Add3(Add3(t1,`$\underbrace{\texttt{Mon3(t2,t3)}}_{\texttt{m1}}$`),` $\underbrace{\texttt{Mul3(t4,t5)}}_{\texttt{m2}}$`)`	→	*wenn* `m1` = `m2` : `Add3(t1,`merge(`m1`,`m2`)`)` *wenn* `m1` < `m2` : `Add3(Add3(t1,m1),Mon3(t4,t5))` *wenn* `m1` > `m2` : `Add3(Add3(t1,m2),m1)`	(11)

Abb. 5: Baumtransformationsregeln für den letzten Transformationsschritt. Es werden Konstanten gefaltet und Monome mit gleichen Bezeichnern und zugehörigen gleichen Exponenten zusammengefasst. `m1` und `m2` stehen für die gekennzeichneten Monome.

Bereits im ersten Transformationsschritt wird die Polynomsumme nach links abgeleitet. Dies hat, wie bereits im zweiten Transformationsschritt, den Vorteil, dass so die Anzahl der nötigen Transformationsregeln minimiert wird. In dem nach links abgeleiteten Baum werden durch Anwendung des Assoziativgesetzes Konstanten nach unten gereicht. Benachbarte Konstanten werden schließlich durch Addition zusammengefasst. Monome innerhalb des Polynoms sind immer mit einem Multiplikationsknoten `Mul3` gewurzelt. Durch Erkennen dieser Knoten bei der Baummustersuche lassen sich alle Monome innerhalb der

Summe finden. Die an diesen Knoten hängenden Terme werden nun entsprechend einer Ordnung auf Monomen verglichen, um die Monome zu sortieren. Ähnlich wie im zweiten Transformationsschritt werden so entsprechend der Ordnung kleinere Monome im Polynom nach links und somit im nach links abgeleiteten Baum der Summe nach unten gereicht. Werden dabei zwei benachbarte Monome als gleich befunden, indem sie dieselben Bezeichner mit gleichen Exponenten aufweisen, so werden sie durch merge() zusammengefasst. Kann ein Monom aufgrund der Ordnung nicht weiter im Baum nach unten gereicht werden, wird es als einsortiert markiert, indem es mit einem neuen Symbol `Mon3` versehen wird. Das Monom ist strukturell anschließend dasselbe wie zuvor, es ist lediglich in `Mon3` gewurzelt. Somit lassen sich in der Baummustererkennung bereits einsortierte Monome von noch nicht einsortierten Monomen unterscheiden, sodass die Sortierung sich Iteration für Iteration im Baum des Polynoms nach oben arbeitet. So wird jedes Monom einsoriert und gegebenenfalls mit anderen Monomen zusammengefasst. Einzelne Bezeichner oder potenzierte Bezeichner sind ebenfalls gültige Monome. Damit auch diese korrekt einsortiert werden können, sind die Regeln (3) bis (7) nötig. Hier werden diese Monome durch eine Multiplikation mit 1 mit `Mul3` gewurzelt.

4 Zusammenfassung

Ergebnis dieser Arbeit ist, dass es im Bereich des Übersetzerbaus eine praktikable Möglichkeit ist Übersetzerbautechnologien zu verwenden, um arithmetische Terme umzuformen, insofern eine Normalform als Ziel vorgegeben wird. Die Überführung von arithmetischen Ausdrücken in ein normalisiertes Polynom mittels Baumtransformation konnte durch eine Zerlegung in kleinere Teilprobleme realisiert werden. Hierfür wurde eine Technik entwickelt, wie man durch Angabe eines einzigen Baumersetzungssystems die Baumtransformation in Phasen ablaufen lassen kann. Da bei der Termumformung nach einem Umformungsschritt nicht immer die gewünschte Normalform resultiert, wurde zudem eine Möglichkeit gezeigt die Baumtransformation iterativ durchzuführen.

Für die Korrektheit des dargelegten Baumersetzungssystems wird auf die Masterarbeit, in dessen Rahmen diese Arbeit entstand, verwiesen [Kn15]. Es war hierfür nötig zu zeigen, dass die Baumersetzungssysteme der jeweiligen Transformationsschritte sowohl noethersch als auch konfluent sind und stets in den angegebenen Normalformen resultieren.

Es stellt sich jedoch die Frage, ob die in dieser Arbeit entwickelte Technik verbessert werden kann. So ist beispielsweise die Konstantenfaltung zum Zusammenfassen von Bezeichern gemessen an ihrer mathematischen Einfachheit eine recht aufwendige Operation, da beispielsweise bei dem in dieser Arbeit vorgestellten Verfahren in bestimmten Fällen viele Konstanten anfallen können, die gefaltet werden müssen.

Die Laufzeit des entwickelten Programms war im Rahmen dieser Arbeit nicht von primärer Wichtigkeit. Laufzeituntersuchungen waren somit nicht Bestandteil dieser Arbeit. Deren Ergebnisse wären jedoch hilfreich zur Bewertung der Praktikabilität der verwendeten Technik der Baumtransformation hinsichtlich des Kontextes dieser Arbeit. Oben genannter Ansatzpunkt könnte die Laufzeit verbessern. Ein weiterer Ansatzpunkt hinsichtlich

der Laufzeit und Effizienz wären Untersuchungen, die den Umfang des zu verwendenden Transformationsregelsatzes betrachtet. Es zeigte sich, dass die nötigen Transformationsregeln für das Ziel der Arbeit hinsichtlich ihrer Anzahl überschaubar waren. Durch das Vergrößern des Regelsatzes wäre es möglich größere Muster abzufangen und somit Aktionen mehrerer Transformationsiterationen in einer Aktion zu vereinen. Der Preis wäre jedoch eine höhere Anzahl von im Baum zu suchenden Mustern. Es stellt sich also die Frage, ob die Verwendung weniger Regeln mit weniger abzusuchenden Mustern und dafür vielen Transformationsiterationen performanter ist, als die Einsparung von Iterationen durch die Verwendung mehrerer Regeln und somit vielen abzusuchenden Mustern.

Die BURS-Theorie bietet durch Kostenoptimierung die Möglichkeit die Anwendung von Tranformationsregeln z. B. in der Reihenfolge zu beeinflussen. Die Ziele dieser Arbeit konnten mit einheitlichen Kosten erreicht werden. Wiederum wären in Hinblick auf die Performanz Untersuchungen interessant, wie man durch Kostensteuerung das bestehende System gegebenenfalls verbessern könnte.

In dieser Arbeit sind Eingabeausdrücke noch eingeschränkt. Die entwickelte Technik müsste noch um Transformationsregeln oder gar Transformationsschritte erweitert werden, um beliebige arithmetische Ausdrücke als Eingabe zuzulassen.

Literaturverzeichnis

[Ac04] Achterberg, Tobias: SCIP - a framework to integrate Constraint and Mixed Integer Programming. Bericht 04-19, ZIB, 2004.

[El16] Eli. http://eli-project.sourceforge.net/, Stand: Feb. 2016.

[Em94] Emmelmann, H.: Codeselektion mit regulär gesteuerter Termersetzung. Berichte der Gesellschaft für Mathematik und Datenverarbeitung - 241, 1994.

[Fa01] Fateman, Richard J.: A review of Macsyma. IEEE Trans. on Knowledge and Data Eng, 1:133–145, 1982–1984, 2001.

[Gr92] Gray, Robert W.; Levi, Steven P.; Heuring, Vincent P.; Sloane, Anthony M.; Waite, William M.: Eli: A Complete, Flexible Compiler Construction System. Commun. ACM, 35(2):121–130, Februar 1992.

[Kn15] Knispel, F.: Untersuchungen von Baumersetzungsstrategien zur Überführung von arithmetischen Ausdrücken in ein Polynom. Masterarbeit, 2015.

[Ma16a] MathWorks. http://de.mathworks.com/products/matlab/, Stand: Feb. 2016.

[Ma16b] Maxima. http://maxima.sourceforge.net/, Stand: Feb. 2016.

[PLG88] Pelegrí-Llopart, E.; Graham, S. L.: Optimal Code Generation for Expression Trees: An Application BURS Theory. In: Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. POPL '88, ACM, New York, NY, USA, S. 294–308, 1988.

[Pr16] Projekt - Entwicklung hocheffizienter Pumpensysteme. http://swt.informatik.uni-halle.de/projekte/46534_2710967/, Stand: Feb. 2016.