

Entwicklung eines Termersetzungssystems für
assoziative und kommutative Ausdrücke zum
vereinfachen arithmetischer Terme

Version 0.0.1

Bruno Borchardt

heute

Inhaltsverzeichnis

1	Einleitung	2
1.1	Definitionen	2
1.1.1	Funktionsauswertung	2
1.1.2	Variadische Funktionen	3
2	Datenstruktur	3
3	Patternmatching	6
4	Vereinfachen von Arithmetischen Termen	6
4.1	Vergleich meiner Features mit anderen Computeralgebrasystemen	7
5	Zusammenfassung	7

1 Einleitung

Hier kommt hin, was halt in eine Einleitung soll:

- was ist ein Termersetzungssystem?
- Einsatz von Termersetzungssystemen
 - Beweisprüfer /Beweisassistenten
 - Arithmetikausdrücke vereinfachen
 - Optimierender Compiler
 - bestimmt noch mehr
- was soll die in dieser Arbeit umgesetzte Variante (gut) können?

1.1 Definitionen

Eine Menge von Termen T ist in dieser Arbeit in Abhängigkeit der Mengen F und Σ , sowie der Funktion $\text{rank} : F \rightarrow \mathbb{N}_0$, definiert.

$$T(F, \Sigma) := \Sigma \cup \{(f, t_1, \dots, t_n) \mid f \in F, \text{rank}(f) = n, t_i \in T(F, \Sigma)\}$$

Elemente $f \in F$ werden *Funktionssymbole* genannt, Elemente $s \in \Sigma$ werden *Konstantensymbole* genannt. $(f, t_1, \dots, t_n) \in T \setminus \Sigma$ wird auch als $f(t_1, \dots, t_n)$ geschrieben und *Funktionsanwendung* genannt.

Wichtig ist, dass auf der Definitionsebene nicht zwischen einem Konstantensymbol $s \in \Sigma$ unterschieden wird, dessen Wert eindeutig ist, etwa der konkreten Komplexen Zahl $s = 3 - 5i$ und einem Symbol, welches vom Kontext abhängt, etwa dem Zeichen $s = x$ (vorrausgesetzt $3 - 5i, x \in \Sigma$).

1.1.1 Funktionsauswertung

Ein Funktionssymbol ist in der allgemeinen Definition eines Terms noch keine Funktion, da die Abbildungsvorschrift, sowie Definitionsmenge und Bildmenge nicht definiert sind. Ein Funktionssymbol ohne diese Eigenschaften kann daher eher als Datenstruktur gesehen werden, etwa die Funktionsanwendung des Funktionssymbols *pair* auf die zwei Parameter a und b als ein Paar: $\text{pair}(a, b)$. Die erste Erweiterung der Funktionssymbolmenge zur Menge von Funktionen folgt mittels der eval Funktion.

$$\tilde{T}(F, \Sigma) := \{(f, s_1, \dots, s_n) \mid f \in F, \text{rank}(f) = n, s_i \in \Sigma\}$$

$$\text{eval}: (\tilde{T} \rightarrow \Sigma) \times T \rightarrow \Sigma$$

$$\text{eval}(v, t) = \begin{cases} t & , t \in \Sigma \\ v(s_1, \dots, s_n) & , t = (f, t_1, \dots, t_n), s_i = \text{eval}(v, t_i) \end{cases}$$

Als Beispiel ist $\Sigma = \mathbb{N}_0$, $F = \{sum, product, power\}$ und v definiert als

$$v(f, s_1, \dots, s_n) = \begin{cases} s_1 + s_2 & , f = sum \\ s_1 \cdot s_2 & , f = product \\ (s_1)^{s_2} & , f = power \end{cases}$$

Der Term $t = sum(3, product(2, 4))$ kann dann ausgewertet werden zu

$$\begin{aligned} eval(v, t) &= eval(v, sum(3, product(2, 4))) \\ &= v(sum, eval(v, 3), eval(v, product(2, 4))) \\ &= v(sum, 3, v(product, eval(v, 2), eval(v, 4))) \\ &= v(sum, 3, v(product, 2, 4)) \\ &= v(sum, 3, 8) \\ &= 11 \end{aligned} \tag{1}$$

Nicht mehr ausreichend ist `eval`, wenn auch Symbole unbekannten Wertes in Σ aufgenommen werden oder wenn manche Funktionen nicht von bzw. nach Σ abbilden.

1.1.2 Variadische Funktionen

2 Datenstruktur

Hier wird erläutert, wie meine konkrete Implementierung Terme speichert und verwaltet. Vielleicht werden auch ein paar grundlegende Algorithmen auf der Datenstruktur erklärt, sofern ich mein exaktes Vorgehen in keinem Paper wiederfinde.

Denkbare Algorithmen zum erklären:

- Falten eines Terms
- Sortieren eines Terms / ordnen von Teiltermen
- exaktes Ausrechnen von Teilbäumen ohne Variablen (und warum ich das mit dem Zusammenfassen von geschachtelten Summen und weiteren Sachen in einen Algorithmus zusammenwursten musste)
- Frage: aktuell parse ich einen String zum bauen eines Terms noch in $O(n^2)$. Das ist warscheinlich nicht optimal. Ergo würde ich diesen Teil wenn, dann eher am Rande erwähnen. Ist es für die Arbeit wichtig, das noch zu optimieren (unter der Vorraussetzung, dass die von mir gewählte Grammatik kontextfrei ist, was sie aber glaube ich ist)?

Mögliche Tangente: Meine Idee (und Umsetzung) einer Art Aufzählung (in C++ und Co. as enum in der Sprache enthalten), die Hirarchien erlaubt und damit Basis eines (wie ich finde) relativ eleganten Typsystems für die einzelnen Arten von Termknoten darstellt. (Tangente der Tangente: Das native C++

Typsystem mit Umsetzung von Polymorphie als Vererbung erschwert einem nicht nur das Speichern eines polymorphen Baums in einem Array, sondern verteilt auch den Code jedes Algorithmus über die verschiedenen Klassen, die die einzelnen Baumknotenarten modellieren. Deswegen habe ich mir was eingendes gebastelt)

Sowohl der zu vereinfachende Term, als auch die Vereinfachungsregeln liegen im Programm als Baumstruktur vor. Die Knoten des Baumes haben jeweils einen der folgenden Typen:

1. Komplexe Zahl
Jeder bekannte Wert wird intern als komplexe Zahl mit Realteil und Imaginärteil als Fließkommazahl doppelter Präzision aus der C++ Standardbibliothek gespeichert.
2. Variable
Eine Variable besteht nur aus ihrem Name, welcher als Array von ASCII Buchstaben vorliegt.
3. Bekannte Funktion
Im Speicher des einzelnen Terms ist eine dem Programm bekannte Funktion als Array von Referenzen auf ihre Parameter gespeichert. Auf der Datenebene ist dabei nicht direkt erkennbar, ob die Funktion eine feste Anzahl von Parametern hat, wie etwa die Exponentialfunktion, oder variadisch ist, wie zum Beispiel die Summe. Dies erlaubt es, oft nicht zwischen beiden Arten differenzieren zu müssen.
4. Unbekannte Funktion
Funktionen, deren Name dem Programm nicht bekannt sind, speichern diesen pro Instanz, analog zur Variable. Vor dem Namensarray befindet sich ein Parameterarray, identisch zu dem der bekannten Funktion. Damit muss in vielen Fällen auch nicht zwischen bekannten Funktionen und unbekannten Funktionen unterschieden werden.
5. Baum-Mustervariable
Während die normale Variable vor allem im zu vereinfachenden Term erwartet wird, ist die Baum-Mustervariable das Symbol, welches in einem Muster für einen Teilbaum des zu vereinfachenden Terms steht.
6. Wert-Mustervariable
Ähnlich wie die Baum-Mustervariable steht auch diese repräsentativ für einen Teil des zu vereinfachenden Baums, hier allerdings nicht für einen beliebigen Teilbaum, sondern einen bekannten Zahlenwert. Die Separierung von der Baum-Mustervariable ist notwendig, um mehr Arten von Werten erkennen zu können, ohne extra ein System für das Überprüfen von Logikbedingungen einbauen zu müssen (wenn auch dieses mächtiger wäre).

3 Patternmatching

Hier stelle ich vor, was für Matchingalgorithmen (für Bäume mit Knoten beliebigen Grades) ich in der Literatur finde. (Hoffentlich finde ich nicht den von mir genutzten, aber den als erster entdeckt zu haben halte ich für unwahrscheinlich.) Den von mir genutzten Algorithmus würde ich hier im Detail erklären und die Laufzeit analysieren. Spoiler: ich meine gelesen zu haben, dass das matchen von Mustern mit möglichen Mehrfachnennungen von Mustervariablen in variadischen Ausdrücken NP-schwer ist, aber vielleicht kann ich für praxisrelevante Teilmengen aller Musterterme bessere Worst-Case Laufzeiten erzielen. (Das optimieren meines Algorithmus ist noch im Gange).

einfacher Matchingalgorithmus

4 Vereinfachen von Arithmetischen Termen

Ich habe begonnen das Termersetzungssystem zu entwickeln, um Arithmetische Ausdrücke zu vereinfachen (etwa $a + 2a \rightarrow 3a$). Wie genau ich das umsetze, wird in diesem Abschnitt erläutert.

Während die Datenstruktur und der Matchingalgorithmus schon benutzbar sind, ist dieser Teil von mir bisher so gut wie gar nicht implementiert worden. Der grobe Plan ist aber folgender:

1. Funktionen höherer Ordnung anwenden:
 - ableiten (Prototyp dafür steht schon)
 - vielleicht integrieren (soll für den allgemeinen Fall wohl schwer sein)
 - vielleicht fouriertransformieren
 - vielleicht laplacetransformieren
 - ganz ganz ganz ganz vielleicht Differentialgleichungen lösen
2. Normalform herstellen:
 - alles ausmultiplizieren ($a \cdot (b + c) \rightarrow a \cdot b + a \cdot c$)
 - Vorzeichen aus ungeraden Funktionen herausziehen ($\sin(-x) \rightarrow -\sin(x)$)
 - Vorzeichen in geraden Funktionen auf plus setzen ($\cos(-x) \rightarrow \cos(x)$)
 - Überlegen, wie man das selbe für Fälle mit Summen im Argument definiert ($\cos(a - b)$ vs. $\cos(b - a)$)
 - bekannte Faktoren aus Potenz ziehen ($(3x)^2 \rightarrow 9x^2$)
 - ...
3. Vereinfachen:

- manche Transformationen sollten immer angewendet werden (etwa $\sin^2(x) + \cos^2(x) \rightarrow 1$)
- andere Transformationen nur ausprobieren und mit einer passenden Metrik gucken, wie gut ein Term nach Anwendung noch weiter vereinfacht werden kann (etwa, wenn man aus verschiedenen Optionen des Ausklammersns wählen kann)
- vielleicht Linearfaktorzerlegung von Polynomen (schätze ich für den allgemeinen Fall schwierig ein, solange ich nur exakte Operationen zulasse)
- vielleicht Polynomdivision (schätze ich genau so schwierig ein, zumindest wenn ich nicht vorher schon Linearfaktoren habe)
- ...

Anmerkung 1: Die Normalform ist notwendig, um zu garantieren, dass mehrfaches Auftreten eines Teilbaums / Teilterms auch erkannt wird.

Anmerkung 2: es kann sein, dass ich manche Eigenschaften der Normalformauch während des Vereinfachungsschrittes immer wieder wiederherstellen muss.

4.1 Vergleich meiner Features mit anderen Computeralgebrasystemen

Ich bin ja nicht der erste, der auf die Idee kommt, Terme zusammenzufassen. Wolphram Alpha und Maple sind zwar nicht Open Source, aber andere Optionen, wie etwa SymPy aus der Python Standardbibliothek soweit ich weiß schon. Da lässt sich bestimmt ein bisschen vergleichen, wie andere Leute die selben Probleme lösen.

5 Zusammenfassung

Was halt in eine Zusammenfassung kommt