Softwareprojekt II - „Untersuchung zu den Begriffen Gültigkeit und Beweisbarkeit in einer mathematischen Theorie"

Reto Hablützel, Max Schrimpf

28. März 2013

Inhaltsverzeichnis

1 Einleitung

1.1 Aufgabenstellung

> Es soll augehend von einem gegebenen einfachen Axiomensystem (z.B "Gruppe") ein einfacher Satz einereits bewiesen werden und anderseits seine "Gültigkeit" illustriert werden. Ein Beweis soll entlang von gegebenen formalisierten Schlussregeln erfolgen, von der Gültigkeit soll man sich mittels "durchrechnen" überzeugen, bei unendlichen Strukturen mit einer zufallsgenerierten Auswahl von Fällen..

Ausgehend von dieser vorgegebenen Aufgabenstellung soll in dieser Projektarbeit eine Implementation erarbeitet werden mit der geforderten Fähigkeit die Gültigkeit eines Axiomsystemes in den Grundsätzen zu bewisen. Durch das Enorme Wachstum der Ausgangswerte sind dabei schon relativ kleine Systeme mit bedeutendem Rechenaufwand verbunden. Die genaue Schwerpunktsetzung bei der Umsetzung und das Vorgehen war dabei durch die offene Aufgabenstellung dem Projektteam überlassen. Vorgegeben war mur die Implementation eines automatisierten Gültigkeitsbeweises des Satzes zur Rechtsauslöschung in Gruppen.

1.2 Ausgangslage

1.2.1 MapReduce

MapReduce ist ein Framework, welches die parallele Verarbeitung grosser Datenmengen vereinfachen soll und erstmals von Google beschrieben und implementiert wurde. Google war mit dem Problem konfrontiert, dass sie sehr grossen Datenmengen verarbeiten mussten. Um ein Programm so zu schreiben, dass es auf einer verteilten Architektur gut skaliert, muss sehr schnell viel Code geschrieben werden, der nichts mit dem eigentlichen Problem zu tun hat. Deshalb hat Google ein Framework entwickelt, welches es dem Benutzer erlaubt sein Problem in zwei Phasen zu definieren und der Rest wird vom Framework übernommen. Dabei kümmert sich das Framework zum Beispiel darum, was passiert, wenn eine Maschine im ausfällt oder wie die einzelnen Arbeittsschritte auf die verschiedenen Maschinen aufgeteilt werden.

Die zwei Phasen sind, wie der Name suggeriert, Map und Reduce (und eine optionale Combiner Phase, aber diese ist zum Verständnis unwichtig). In diesen Phasen werden im Wesentlichen die beiden Funktionen Map und Reduce aufgerufen, welche hier kurz erklärt werden sollen.

[1] $map : (a \to b) \to A \to B$

Die Map Funktion bildet eine Menge von Elementen elementweise auf eine neue Menge ab. Dazu muss ihr zwei Parameter übergeben werden – eine Funktion, welche eine Element aus der Menge A auf ein Element in der Menge B abbildet und eine Menge von Elementen A. Dann wird die Funktion sukzessive

[1] Schreibweise: a ist ein Element aus einer Menge A

auf die Elemente der Liste A angewendet und somit die Menge B konstruiert, welche das Resultat ist.

$reduce : (b \to c \to c) \to c \to B \to c$

Die Reduce Funktion bildet eine Menge B auf einen einzelnen Wert ab. Dafür nimmt sie als ersten Parameter eine Funktion, welche einen Wert aus der Menge B und einen Akkumulator als Parameter nimmt und einen Wert b zurückgibt. Der zweite Parameter der Reduce Funktion ist der initiale Wert für den Akkumulator und der dritte ist die Menge B. Dann wird die übergebene Funktion mit jedem Element aus der Menge B und dem Akkumulator aufgerufen. Dabei ist das Resulat der Funktion jeweils der neue Wert für den Akkumulator. Sobald die ganze Liste traversiert wurde, ist der Wert vom Akkumulator das Resultat.

Der Benutzer vom MapReduce Framework, welches etwas berechnen will, muss also eine Funktion schrieben, welche ein Element aus der Menge A auf ein Element in der Menge B abbildet (Map Phase) und die Menge A bereitstellen. Ausserdem muss eine Funktion geschrieben werden, welche für ein Element aus der Menge B kombiniert mit einem Akkumulator auf ein Element aus der Menge C abbildet (Reduce Phase). Das MapReduce Framework kann dann zum Beispiel die erste Funktion mit einem Element aus der Menge A zusammen an eine Maschine schicken. Wie die Grafik zur Map Funktion illustriert, gibt es keine Datenabhängigkeiten (1:1 Mapping), weshalb jede Funktionsanwendung sehr gut auf verschiedenen Maschinen ausgeführt werden kann. Die Map Phase gilt als abgeschlossen, wenn sämtliche Elemente aus der Menge A auf ein Element in der Menge B abgebildet wurden. Dann kann das MapReduce Framework auf einer Maschine die Reduce-Phase durchführen. Hierbei ist wichtig zu bemerken, dass die Reduce Funktion nicht ganz so optimal parallelisiert werden dann, da es Datenabhängigkeiten gibt. Eine parallele Verarbeitung ist nur möglich, wenn die Reduce Funktion assoziativ ist. Dann könnte die Reduce-Phase in einer Art Baum-Struktur ausgeführt werden – darauf soll aber nicht weiter eingegangen werden.

Folgendendes Beispiel soll die Idee von den zwei Phasen verdeutlichen. Dafür definieren wir ein initiale Menge an Elementen A und je eine Funktion für die Map Phase f_m und eine Funktion für die Reduce Phase f_r.

$A = 1, 2, 3, 4$

$f_m\ x = x^2$

$f_r\ x\ a = x + a$

In diesem Beispiel werden in der Map Phase die Elemente aus der Menge A quadriert. Somit entsteht eine Menge B = {1, 4, 9, 16}. In der Reduce Phase werden alle Elemente aus der Menge B aufaddiert[2]. Das Endresultat ist also 30.

[2]Der initiale Akkumulator ist in diesem Fall das erste Element aus der Menge B.

2 Umsetzung - Teil 1 Gültigkeit des Satzes zur Rechtsauslöschung in Gruppen

2.1 Map und der Reduce Funktion für die Aufgabenstellung

2.1.1 Prüfen derAxiome als Map-Funktion

Das CPU-intensive Prüfen der Axiome wurde als MAP-Funktion definiert. Dabei werden den einzelnen Workern jeweils ein Bereich an Permutationen der Additionstabelle und ein n im Sinne der derzeit vorhandenen Anzahl an verschiedenen Objekten als Input gegeben. Die Worker führen dann die Überprüfung der Axiome durch indem sie jeweils für alle Additionstabellen in ihrem Bereich alle möglichen Inversentabellen mit allen möglichen neutralen Elementen prüfen.

Hierbei gibt es Vereinfachungen, die inhaltlich korrekt sind jedoch die benötigte Rechenleistungen massiv reduzieren:

- Sowohl die Additions- wie auch die Permutationstabelle werden nicht immer vollständig vor der Prüfung der Axiome aufgestellt, sondern es wird eine Funktion implementiert mit der zu einer gegebenen Permutationsnummer eines beliebigen n das Ergebnis für ein bestimmtes Element zurückgibt. Das heisst der Rechenaufwand wird substantiell reduziert da bei einer Prüfung der Axiome im Normalfall nur ein quasi vernachlässigbarer Teil der Additionstabelle und, abhängig von Erfolg oder Misserfolg der Überprüfungen, auch nur ein minimer Teil der Inversentabelle benötigt wird.

- Für eine gegebene Additions- und Inversen-Tabellenkombination wird jeweils nur das neutrale Element, das für den ersten Eintrag in der Inversentabelle berechnet wurde, geprüft. Da logischerweise nur ein neutrales Element für eine spezifische Kombination aus Tabellen existiert, kann das Ergebnis der ersten Rechnung verwendet werden, da dieses bereits alle anderen möglichen neutralen Elemente wiederlegt. Auch diese Annahme reduziert substantiell die durchgeführten, nicht zielführenden Rechnungen.

Die Ausgabe der Map-Funktion sind schlussendlich alle Gruppen auf die die gegebenen Axiome zutreffen. Eine Gruppe wird ausreichend durch 3 Zahlen beschrieben:

- Die Nummer der Additionstabelle. Aus Ihr lassen sich mit der oben beschriebenen Funktion alle Additionen, die in der Gruppe möglich sind, herleiten.

- Die Nummer der Inversentabelle. Aus Ihr lassen sich mit der oben beschriebenen Funktion die Inversen zu allen Elementen der Gruppe herleiten

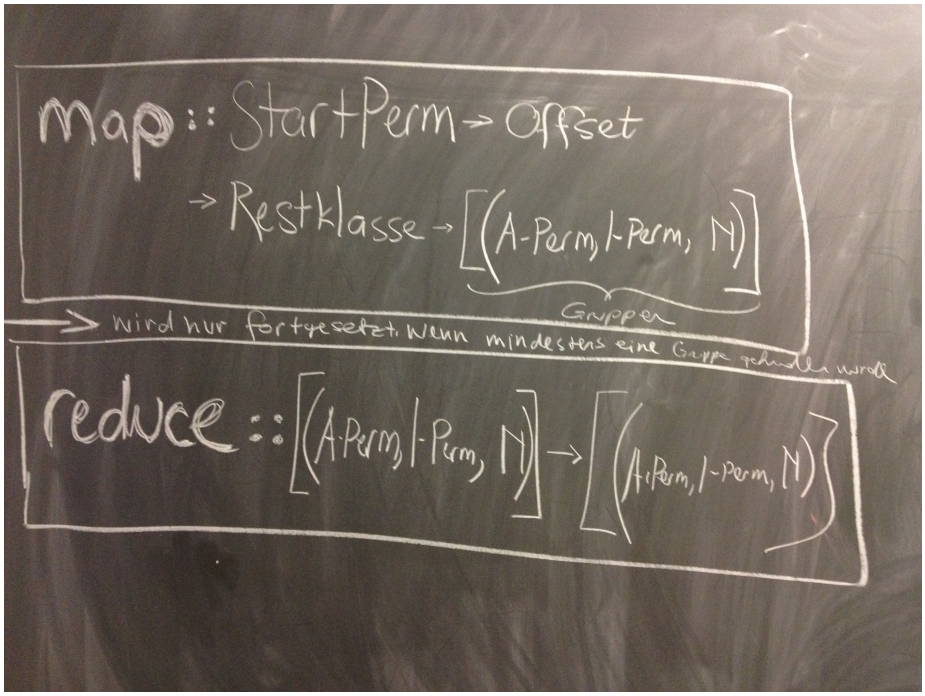
- Das neutrale Element der Gruppe

Diese effiziente Speicherung ermöglicht das Vorhalten von enorm vielen Gruppen ohne einen hohen Speicherbedarf.

2.1.2 Prüfen der Sätze als Reduce-Funktion

In der Reduce-Phase wird für die, relativ zur Menge der möglichen Gruppen, enorm kleine Menge an effektiven Gruppen der Satz jeweils geprüft. Als Output gibt sie die Gruppen für die der Satz nicht zutrifft. Da anzunehmen ist, dass somit der Output immer leer ist, kann er auch als booleanscher Wahrheitswert interpretiert werden:

- Wenn der Output leer ist, ist der Satz für die geprüften Gruppen wahr.
- Wenn der Output nicht leer ist, ist der Satz für die geprüften Gruppen falsch (was einer mathematischen Sensation gleich käme).



3 Weiteres Vorgehen

3.1 Möglichkeiten der Vertiefung der Aufgabenstellung

3.1.1 Erweiterung des MapReduce Frameworks

Bei diesem Schwerpunkt wäre die Zielsetzung eine möglichst grosse Kapazität Gültigkeiten zu beweisen mit dem MapReduce Framework und verteiltem Rechnen zu erzielen. Die enormen Datenmengen, die bereits bei kleinen Eingabewerten entstehen sind auf einem einzelnen System quasi nicht zu bewältigen dabei ist primär die Rechenzeit relevant. Daher müssten Möglichkeiten gefunden werden um eine Menge an Inputwerten zu anderen Computern zu senden,

dort berechnen zu lassen und für das weitere Vorgehen relevante Daten zu returnieren.

Anforderungen bei diesem Schwerpunkt wären primär die Handhabung von verteilten Systemen, Design der Applikation und eine performante Umsetzung.

3.1.2 Freie Eingabe von auf Gültigkeit zu prüfenden Annahmen

Hierbei läge das Hauptaugenmerk darauf quasi eine Eingabesprache für ein automatisiertes Gültigkeitsbeweis-Programm zu erstellen. Es müsste eine Möglichkeit gefunden werden dem Programm mathematische Strukturen mit bestimmten Eigenschaften, darauf gültige Axiome und daraus resultierend zu prüfende Sätze zu übermitteln. Dabei müsste, gleich ob interaktiv oder durch die Auswertung von Eingabedateien, eine Eingabesyntax definiert werden und, was wohl die wirkliche Problemstellung des Schwerpunktes wäre, eine Auflösung der Syntax und eine darauf basierende dynamische Umsezuung in ausführbaren Code.

Anforderungen bei diesem Schwerpunkt lägen primär in der theoretischen Informatik beim Thema Compilerbau sowie der dynamischen Generierung einer Interpretation der Mathematischen Objekte, ihrem Einbinden und dem Ausführen in einer Applikation zur schlussendlichen Gültigkeitsprüfung.

3.1.3 Automatisierte Beweisbarkeit

In diesem Fall wäre das Ziel ein Framework zu erstellen welches mit einem gegebenen Set aus mathematischen Operationen (zugelassenen Schlussregeln) automatisch Sätze aus Eingaben (Sätzen und Axiomen) herleitet. Die Operationen, die das System vornehmen könnten entsprächen den im Kalkül zugelassenen Schlussregeln. Ziel wäre es somit automatisiert mehr oder weniger sinnvolle Aussagen mathematisch formell korrekt herzuleiten. Dabei müssten Kriterien definiert werden mit denen die Unterscheidung von formal korrekten aber irrelevanten und formal korrekten, relevanten Aussagen möglich macht.

Anforderungen bei diesem Schwerpunkt wäre wieder die Interpretation der in einer gegebenen Syntax vorliegenden Eingaben jedoch des weiteren das Erkennen von neuen relevanten Schlussfolgerungen.

3.2 Entscheidung

Der auf Anhieb spannendste Schwerpunkt stellte nach der ersten Bertrachtung die „Freie Eingabe von auf Gültigkeit zu prüfenden Annahmen" dar. Die in dem Schwerpunkt vorkommenden Themen wurdenweitestgehend im bisherigen Studium noch nicht behandelt und böten somit eine gute Möglichkeit neues kennenzulernen.

Die Automatisierte Beweisbarkeit erschien bereits bei der ersten Betrachtung als eine eher unwahrscheinliche Vertiefung, da der schlussendliche Nutzen massiv in Frage gestellt werdern kann. Es wäre zu befürchten, dass eine enorme Menge an Arbeitszeit und Herzblut investiert würde um Schlussendlich eine Applikation zu erstellen welche Mathematisch völlig irrelevante Sätze herleiten würde.

Eine Erweiterung des MapReduce Framework erschien bereits nach der ersten Analyse als überschaubares Problem im Sinne der Beherrschbarkeit der Problemstellung. Zwar ist der mit der Vertiefung verbundene Aufand alles andere als als klein zu erachten, jedoch sollten keine vollkommen unvorhergesehenen Fragestellungen, zu denen ohne eine sehr grosse Menge an Vertiefung keine Lösung gefunden werden könnte, auftreten.

Das Kriterium der Beherrschbarkeit gab schlussendlich auch den Ausschlag anstatt der Vertiefung Nr. 2 eine Erweiterung des Frameworkes anzustreben. Die Gefahr durch die enorme Komplexität am Ende nur eine mehr prototypisch als sinnvolle Implementation eines Programmes zu erstellen schreckte uns ab. Desweiteren stellte die Idee mit der gegebenen Infrastruktur der Fachhochschule eventuell eine grosse Menge an PCs für einen Feldversuch zu besitzen einen grossen Ansporn dar.