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

Reto Hablützel, Max Schrimpf

30. Mai 2013

Inhaltsverzeichnis

Anzahl Elemente	Permutationen
1	1
2	16
3	19 683
4	4 294 967 296
5	298 023 223 876 953 125

Tabelle 1: Mögliche Permutationen einer zweistelligen Verknüpfung in einem n-Elementigen System

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 der vorliegenden Projektarbeit eine Implementatio,n mit der Fähigkeit die Gültigkeit eines Axiomsystemes in den Grundsätzen zu bewisen, erarbeitet werden. 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

Wir konnten uns im Rahmen einer Vertiefungsarbeit zum Thema Algorithmen und Datenstrukturen bereits mit dem Thema MapReduce beschäftigt. Damals hatten wir anhand des wohlbekannten Paradigma (1.2.1 MapReduce) unser eigenes Framework für MapReduce Berechnungen implementiert. Aufgrund der beschränkt zur Verfügung stehenden Zeit, mussten wir es damals jedoch bei der Ausführung auf einem einzelnen Rechner mit mehreren Threads belassen, anstatt ein ganzes Computernetz verwenden zu können. Ausserdem gab es weitere technische Limitierungen. Zum Beispiel konnte unser Framework nicht mit dem Szenario umgehen, dass ein Thread abnormal terminiert. Ein solches Konzept wäre insbesondere interessant im Ausblick auf eine echt verteilte Berechnung, da in einem Netzwerk immer wieder eine einzelne Maschine aussteigt.

1.2.1 MapReduce

MapReduce ist ein Entwurfsmuster, welches die parallele Verarbeitung grosser Datenmengen vereinfachen soll und erstmals von Google beschrieben und implementiert wurde. Google ist mit dem Problem konfrontiert, dass sie sehr grossen Datenmengen verarbeiten müssen. Um ein Programm so zu entwickeln, dass es auf einer verteilten Architektur gut skaliert, ist jedoch ein enormer Aufwand, welcher nichts mit dem eigentlich zu lösenden Problem zu tun hat, nötig. Daher entwickelte Google ein Framework, das es dem Benutzer erlaubt sein Problem in zwei Phasen zu definieren und die anschliessende Verteilung übernimmt. 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, für das Verständnis irrelevante, Combiner Phase). 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. Daraufhin wird die Funktion sukzessive 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 erhält sie als ersten Parameter eine Funktion, mit einen Wert aus der Menge B sowie einen Akkumulator, welcheeinen Wert b zurückgibt. Als zweiten Parameter erhält die Reduce Funktion einen initialen Wert für den Akkumulator und als dritten und letzten Parameter 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 des MapReduce Frameworks muss also für eine auszuführende Berechnung nur noch eine Funktion schrieben, welche ein Element aus der Menge A auf ein Element in der Menge B abbildet (Map Phase) anschliessend die Menge A bereitstellen. Auch 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). Für das MapReduce Framework ist es daraufhin möglich zum Beispiel die erste Funktion mit einem Element aus der Menge A an eine berechnende Maschine senden. Wie die Grafik zur Map Funktion illustriert, gibt es keine Datenabhängigkeiten (1:1 Mapping), weshalb jede

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

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. Nun kann das MapReduce Framework auf einer Maschine die Reduce-Phase durchführen. Hierbei ist wichtig zu bemerken, dass die Reduce Funktion nicht äquivalent 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.

1.3 Abgeleitete Aufgabenstellung

Basierend auf unserem Wissen aus den Algebra - Vorlesungen sowie unserem Interesse für verteilte Berechnungen und dem bestehenden MapReduce Framework, haben wir uns mit Herrn Heuberger so geeinigt, dass wir in einem ersten Teil die Prüfung eines Satzes für ein bestimmtes Axiomsystem als MapReduce Problem formulieren und dieses dann mit unserem Framework ausführen. Die konkrete Aufgabenstellung für den zweiten Teil haben wir zu beginn noch offen gelassen. Einen Variantenvergleich ist im Kapitel 3.1 „Möglichkeiten der Vertiefung der Aufgabenstellung" zu finden.

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

2.1 Ausgangslage

2.1.1 Gruppe

Eine Gruppe ist eine algebraische Struktur bestehend aus einer Menge S und einer Verknüpfung $\diamond$, welche die folgenden Axiome erfüllt:

- Assoziativität: $\forall a, b, c.\ (a \diamond b) \diamond c\ = a \diamond (b \diamond c)$
- Neutrales Element: $\exists e.\, \forall a.\ (a \diamond e = e \diamond a = a)$

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

– Inverses Element[3]: $\forall a.\ \exists b.\ (a \diamond b = e)$

Es gibt verschiedene Strukturen, die diese Eigenschaften haben. Ein Beispiel davon ist die Restklasse 3 mit einer Operation definiert als: $x \diamond y = (x + y)\,\%\,3$, also die gewöhnliche Addition in dieser Restklasse. Für diese Struktur kann man nun beweisen, dass sie eine Gruppe ist, indem man sich überzeugt, dass die Addition assoziativ ist, dass ein neutrales Element existiert (nämlich 0) und dass jedes Element ein inverses Element hat, mit welchem es auf die 0 abbildet (z.B. $1 \diamond 2 = 0$). Es gibt weit mehr Strukturen, die als Gruppe eingestuft werden und mit ausprobieren findet man einige. Diese mechanische Arbeit, also das Suchen solcher Strukturen, könnte uns ein Computer abnehmen und genau das ist ein Teil unserer Arbeit. Natürlich gibt es unendlich viele Möglichkeiten, weshalb wir diese Aufgabe zwar einschränken werden müssen, aber um trotzdem eine möglichst hohe Rechenpower zur Verfügung zu haben, haben wir uns entschieden, die Suche mit unserem MapReduce Framework auszuführen.

2.1.2 Satz

> Ein Satz oder Theorem ist in der Mathematik eine widerspruchsfreie logische Aussage, die mittels eines Beweises als wahr erkannt, das heißt, aus Axiomen und bereits bekannten Sätzen hergeleitet werden kann.[4]

Nachdem wir mittels dem MapReduce Framework irgendwelche Strukturen gefunden habe, welche (evt. per Zufall) die Axiome einer Gruppe erfüllen, können wir in einem zweiten Schritt noch prüfen, ob für diese auch bestimmte Sätze zutreffen. Im Gegensatz zu den Gruppen-Axiomen gibt es keine definierte Menge an Sätzen, welche für eine Gruppe gültig sind. Es werden ständig neue Sätze gefunden und ausserdem existieren Sätze, für die noch nicht allgemeingültig bewiesen werden konnte, dass sie für sämtliche Gruppen gelten - es wurde aber auch noch kein Gegenbespiel gefunden. Während wir mit dem MapReduce Framework zwar keine allgemeingültigen Beweise führen können, können wir aber mit sehr viel Rechenpower nach Gegenbespielen suchen, also bestimmte Konstellationen von Mengen und Verknüpfungen auf diesen, welche zwar eine Gruppe sind, aber einen bestimmten Satz nicht erfüllen. Ein Beispiel eines Satzes, ist der Satz der Rechtsauslöschung:

$$\forall a, b, c.\ b \diamond a = c \diamond a \Rightarrow b = c$$

Sämtliche Strukturen, welche die Axiome der Gruppe erfüllen, erfüllen auch diesen Satz. Den Beweis dazu findet man online[5].

2.2 Generieren von Gruppen mittels Computer

Da es unendlich viele mögliche Mengen und Verknüpfungen gibt, haben wir uns auf die Restklassen als Mengen beschränkt. Eine bestimmte Restklasse kann

[3] Hierbei ist e das neutrale Element
[4] Aus Wikipedia, der freien Enzyklopädie: http://de.wikipedia.org/wiki/Satz_(Mathematik)
[5] http://www.proofwiki.org/wiki/Cancellation_Laws

beim Programmstart angegeben werden. Um mögliche Verknüpfungen auf einer Restklasse zu finden, haben wir einen Algorithmus implementiert, welcher alle möglichen zweistelligen Verknüpfungen auf einer endlichen Menge generiert. Grafisch können diese Verknüpfungen als Wertetabellen interpreiert werden. Folgend sind drei Beispiele für mögliche Verknüpfungen auf der Restklasse 2.

$\diamond$	0	1
0	0	0
1	1	1

$\diamond$	0	1
0	0	1
1	1	0

$\diamond$	0	1
0	1	1
1	1	1

Die mittlere Tabelle ist die intuitive Addition auf dieser Restklasse und die beiden anderen sind mit zufälligen Werten gefüllt. Insgesamt gibt es für die Restklasse 2 also 16 mögliche Wertetabellen, welche mit dem Computer generiert werden können. Aus diesem Beispiel wird ersichtlich, dass es für die Restklasse 2 16 mögliche Verknüpfungstabellen gibt, welche wir mit dem Computer simuliert haben. Für die Restklasse 3 sind es schon deren 19683. Eine Restklasse n hat allgemein fomuliert n^{n*n} mögliche Verknüpfungen.

Ausserdem gibt es in jeder Gruppe ein inverses Element. Um alle möglichen inversen Elemente zu finden, haben wir auch für diese einstellige Verknüpfung Wertetabellen generiert. Folgend zwei Beispiele für Verknüpfungen auf der Restklasse 2.

a	0	1
a^{-1}	1	0

a	0	1
a^{-1}	0	0

Für die Restklasse 2 gibt es also 4 mögliche Wertetabellen und für eine Restklasse n gibt es 2^n Möglichkeiten. Auch diese Tabellen können wir mit unserem Algorithmus simulieren.

Das neutrale Element ist einfach zu finden, weil jedes Element als neutrales gelten kann.

Schlussendlich suchen wir Gruppen indem wir eine zweistellige Verknüpfung, eine einstellige Verknüpfung und ein neutrales Element zusammen auf die Axiome überprüfen. Dazu prüfen wir jede Kombination dieser drei Komponenten in der Map Phase. Die genaue Formulierung dieses Problems als Map Funktion ist im nächsten Abschnitt beschrieben.

2.2.1 Implementationsdetails zur Generierung sämtlicher Wertetabellen

Bereits für die Restklasse 3 gibt es bei der zweistelligen Verknüpfung tausende von Möglichkeiten. Die Restklasse 4 passt wahrscheinlich schon nicht mehr in den Hauptspeicher eines Computers. Da wir diese Berechnung verteilen müssen wir also einerseits einen Algorithmus finden, der sich auf verschiedene Rechner aufteilen lässt, sodass jeder Rechner eine Teilmenge der Wertetabellen überprüfen kann, und diese Teilmenge sollte nicht zu viel Speicher verbrauchen. Auf dem Master alle Wertetabellen zu generieren und diese dann an die verschiedenen Worker (Rechner) über das Netz zu schicken ist also nicht praktikabel.

Nehmen wir als Beispiel die Restklasse 2. Bei genauerer Betrachtung einer Wertetablle sieht man, dass diese anstatt zweidimensional auch eindimensional interpretiert werden können.

$\diamond$	0	1
0	a	b
1	c	d

Eindimensional sähe die abgebildete Wertetabelle folgendermassen aus: $\diamond = \{a, b, c, d\}$. Vor allem in der Restklasse 2 kann man sich leicht überzeugen, dass sämtliche Kombinationen die Binärwerte der Zahlen zwischen 0 und 15 sind: z.B. $0 = \{0, 0, 0, 0\}$(alles bildet auf 0 ab) oder $6 = \{0, 1, 1, 0\}$ (die intuitive Addition der Restklasse 2). In der zweidimensionalen Wertetabelle konnten wir die Funktionswerte mithilfe der x und y Indizes ablesen. Diese x und y Werte können mit der folgenden Funktion auf den Index in der Binärrepräsentation abgebildet werden: $f_{trans2}(x, y) = y * 2 + x$

Wäre man als Beispiel in der sechsten Wertetabelle an der Abbildung $1 \diamond 0$ interessiert, würde das $x = 1$, $y = 0$ entsprechen. Transformiert auf die eindimensionale Repräsentation ergibt dies die Binärrepräsentation 0110 und für den Index $f_{trans2}(1, 0) = 1$. Also das zweite Element der Binärrepräsentation, was eine eins ist - wir in der Wertetabelle.

2.2.2 Implementierung von Quantoren in Java

Wie im Kapitel Ausgangslage beschrieben, werden Axiome einer Gruppe und Sätze einer beliebigen Struktur mittels Quantoren definiert. Als Beispiel soll das Axiom 'Es gibt ein neutrales Element' der Struktur Gruppe näher erläutert werden: $\exists e.\, \forall a.\, (a \diamond e = e \diamond a = a)$. Dieses Axiom verwendet zwei Quantoren: Einen Existenzquantor ($\exists$) und einen Allquantor ($\forall$). Mit dem Existenzquantor wird ausgesagt, dass es in einer bestimmten Struktur ein e gibt, für das das folgende Prädikat wahr ist, während mit dem Allquantor gesagt wird, dass das Prädikat für alle Elemente dieser Struktur gelten muss. In unserem Beispiel 'Es gibt ein neutrales Element' tauchen diese beiden Quantoren direkt nacheinander auf, wodurch dieses Axiom in Prosa folgendermassen formuliert werden kann: In einer Gruppe gibt es ein Element e, welches, wenn es mit irgendeinem Element a aus dieser Struktur verknüpft wird, immer auf das Element a abbildet. Dabei ist es gleichgültig, ob a mit e verknüpft wird oder e mit a - es muss immer auf a abbilden. Genau dann, wenn es ein solches Element e gibt, dann ist e das neutrale Element dieser Struktur und das Axiom ist erfüllt.

Da es in Java keine Quantoren gibt, haben wir uns Schleifen zu hilfe genommen um die Axiome zu implementieren. Mittels der folgenden Funktion wird für ein bestimmtes Element e geprüft, ob es das neutrale Element einer bestimmten Restklasse[6] für eine bestimmte Wertetabelle ist.

```
1  boolean isNeutral(int modulo, int aPerm, int e) {
2    for (int a = 0; a < modulo; a++) {
3      int b = map2d(a, e, aPerm, modulo);
4      if (a != b) { return false; }
5      int c = map2d(e, a, aPerm, modulo);
```

[6]Wir haben als Beispiel die Restklassen verwendet, aber dies würde genauso mit anderen Gruppen funktionieren.

```
6       if (a != c) { return false; }
7     }
8     return true;
9   }
```

Diese Funktion nimmt drei Parameter: Der erste ist ein Wert vom Typ Integer, also eine Ganzzahl, welcher die Restklasse repräsentiert. Der zweite Parameter ist der Identifikator für die zweistellige Wertetabelle, welcher auch als Ganzzahl modelliert ist. Der letzte ist schliesslichen das vermeindlich neutrale Element, für das diese Funktion prüft, ob es wirklich das neutrale Element ist. In der zweiten Zeile wird durch alle Elemente der Restklasse iteriert, wodurch die Variable a den Wert jedes Elementes einmal annimmt (beginnend bei 0). In der dritten Zeile wird die Funktion map2d aufgerufen, welche das Element a (also jedes Element der Restklasse einmal) mit dem neutralen Element e verknüpft. Diese Funktion map2d funktioniert nach unserem eigents erfundenen Algorithmus, den wir im Kapitel 'Generieren von Gruppen mittels Computer' beschrieben haben. Der Rückgabewert dieser Funktion wird der Variable b zugewiesen um in der nächsten Zeile mit der Variable a verglichen zu werden. Wenn diese Bedingung zutrifft, bedeutet das, dass $a \diamond e = e$ gilt, also ist das e schonmal linksneutral. Wenn es nicht linksneutral ist, dann wir die ganze Funktion sofort abgebrochen, weil das Prädikat einen Allquantor beinhaltet - es muss also für alle Elemente gelten. Auf den folgenden Zeilen fünf und sechs wird die Variable a nochmals mit dem vermeindlich neutralen Element verknüft - diesmal aber um die Rechtsneutralität zu überprüfen (deshalb sind hier die Parameter a und e für die Funktion map2d vertauscht). Wenn sämtliche Elemente dieser Restklasse links- und rechtsneutral sind, dann wird die Schleife verlassen und die Funktion gerät zur Zeile acht, wo der boolsche Wert 'wahr' zurückgegeben wird. Dies bedeutet, dass das Element e tatsächlich das neutrale Element in dieser Struktur ist.

2.3 Map und Reduce Funktionen für das Finden von Gruppen und Prüfen von Sätzen

2.3.1 Prüfen derAxiome als Map-Funktion

Das CPU-intensive Suchen generieren der Strukturen, welche möglicherweise die Axiome erfüllen, wurde als MAP-Funktion definiert.

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 im-

mer 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.

2.3.2 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.

4 Umsetzung - Teil 2 MapReduce Framework zur verteilten Berechnung

4.1 Master als zentrale Kontrollinstanz

Der Master ist ein Server, welcher die Aufgaben (Tasks) und Worker verwaltet. Ein Task ist eine abstrakte Einheit, welche von einem Worker erledigt werden kann, also entweder ein Teil einer Map- oder Reduce-Berechnung. Im klassischen MapReduce, wie es von Google beschrieben wurde, ist ein Worker eine Maschine. Somit verteilt eine Hauptmaschine (Master) die verschiedenen Aufgaben an verschiedene Rechner in einem Netzwerk. In unserer Implementation ist aber ein Worker nicht zwingend eine anderen Maschine, sondern wir haben diese Einheit abstrahiert. In der ersten Version unseres Frameworks gab es nur eine Art von Workern - Threads - aber mittlerweile kann via Plugin irgendeine Art von Worker dem Master übergeben werden. Die konkrete Verwaltung wird dabei vom Pool übernommen, welcher den Status der Worker kennt (z.B. beschäftigt, frei, unauffindbar, etc) und weiss, welche Tasks gerade zu erledigen sind.Ein Plugin ist Stück Code, welches eine Worker-Technologie zur Verfügung stellt. Somit gibt es also zum Beispiel ein Plugin, welches Threads als Worker zur Verfügung

stellt und ein Plugin, welches andere Rechner über eine Socket-Verbindung zur Verfügung stellt. Dies wurde so abstrahiert, dass in Zukunft auch weitere Arten von Workern hinzugeügt werden können. Denkbar wäre zum Beispiel CUDA (NVidia Grafikkarten) oder Android Smartphones.

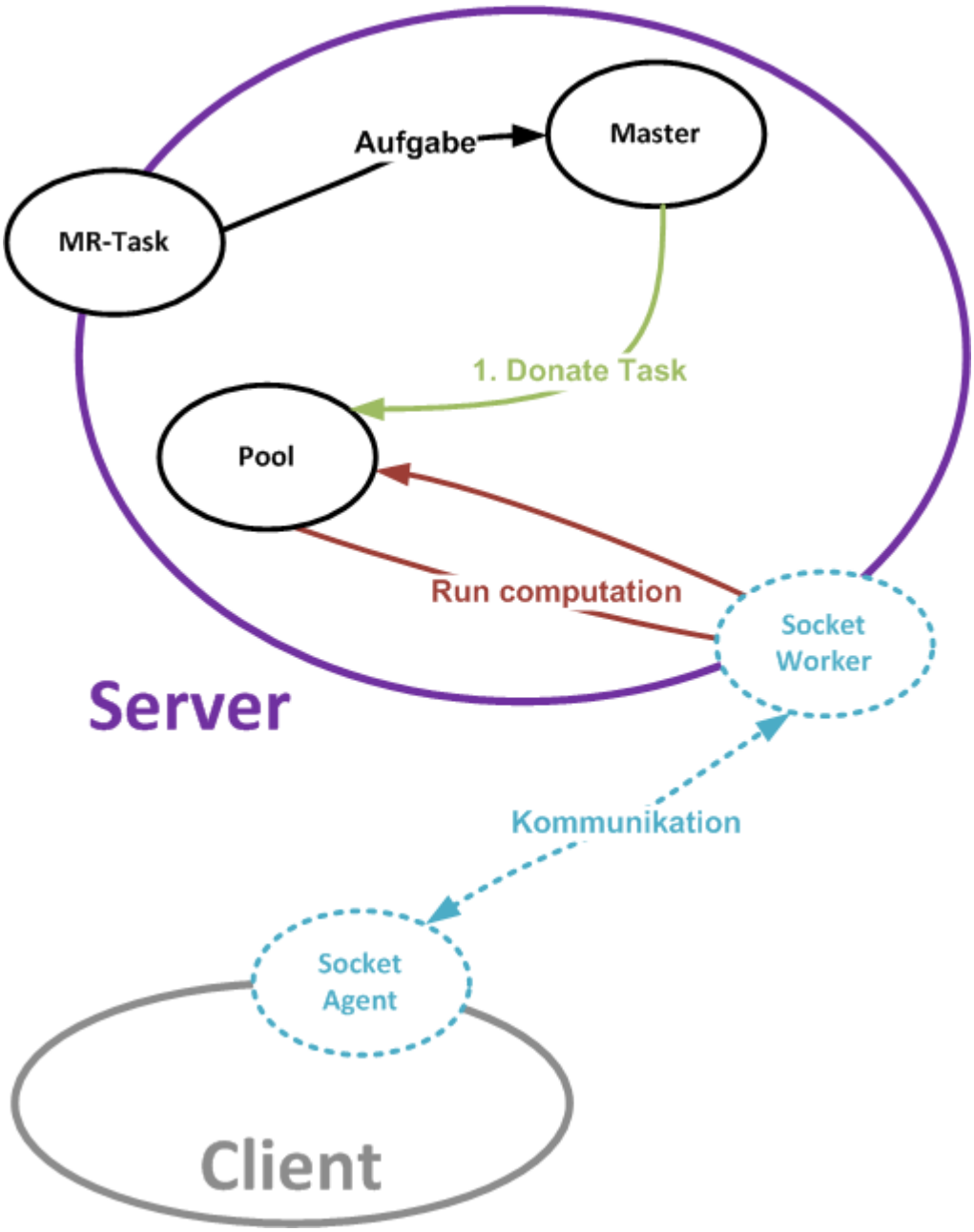
4.2 Ablauf

Wenn eine Berechnung mit dem MapReduce Framework ausgeführt werden soll, ist der Ablauf folgendermassen (vereinfacht): Der Benutzer implementiert die beiden Interfaces MapInstruction und ReduceInstruction (sowie optional eine CombinerInstruction, aber diese soll zur Vereinfachung weggelassen werden) und erstellt zusammen mit einer Menge an Input einen MapReduceTask. Dieser MapReduceTask ist die ursprüngliche Berechnungseinheit, welche dem Master zur Berechnung übergeben wird.Daraus kann dann der Master die Map- und Reduce Instruktionen extrahieren und erstellt zunächst aus je einem Teil vom Input und einer Instanz der MapInstruction einen MapTask. Somit entstehen für eine MapReduce Berechnung mehrere MapTasks, welche dem Pool vom Master zur Verarbeitung übergeben werden. Gleichzeitig wird vorausgesetzt, dass dem Pool durch ein oder mehrere Plugins bereits Worker zur Verfügung gestellt worden sind (donaten). Somit kann der Pool einen auszuführenden MapTask einem verfügbaren Worker übergeben. Wenn dieser Worker eine andere Maschine im Netz abstrahiert, würde das bedeuten, dass dieser MapTask auf einem anderen Rechner ausgeführt wird.Um diesen Task auszuführen, muss ein Kontext erstellt werden, in welchem der Task ausgeführt werden kann, denn ein Task (Map sowie Reduce) arbeitet jeweils gegen ein Emitter, in welchem die Zwischenresultate abgespeichert werden können. Aus der Sicht einer Instruktion (Map sowie Reduce) ist es also transparent, wo sie ausgeführt wird. Sie muss lediglich in den Kontext emitten.Sobald die Berechnung (z.B. auf einem anderen Rechner) ausgeführt wurde, meldet sich der Worker wieder beim Pool, dass er wieder bereit ist, Aufgaben anzunehmen. Also könnte eine ganze Berechnung entweder von einem einzelnen Worker in mehreren Schritten ausgeführt werden oder auf mehrere Worker aufgeteilt, je nach dem wie schnell ein einzelner ist. Gleichzeitig prüft der Master kontinuierlich, ob alle MapTasks ausgeführt worden sind.Sobald alle MapTasks für eine Berechnung ausgeführt worden sind, beginnt der Master alle Resultate zu sammeln und schliesst somit die Map-Phase ab. Darauf folgt die Combiner-Phase, in welcher die Resultate der Map-Phase nach Schlüssel gruppiert werden und somit auf die Reduce-Phase vorbereitet. Da vor der Shuffle-Phase alle MapTasks ausgeführt werden sein müssen, gilt dies als Bottleneck von MapReduce.Nach der Shuffle-Phase beginnt der Master damit die einzelnen ReduceTasks, bestehend aus einer Instanz der ReduceInstruction und einem Teil der Shuffle Resultate, zu erstellen und diese wieder dem Pool zu übergeben. Dann beginnt der Pool wieder diese ReduceTasks den verfügbaren Worker zuzuteilen und der Master wartet wiederum auf deren Fertigstellung. Sobald auch diese fertig sind, ist die ganze MapReduce Berechnung fertig.

4.3 Ablauf bei verteilter Berechnung

Für das Ausführen von verteilen Berechnungen haben wir das Framework SIMON[7] verwendet. Mit Hilfe dieses Frameworks können Methoden auf einer entfernten Instanz irgendwo im Netzwerk gemacht werden. Damit sich der Master nicht um die Komplexitäten im Zusammenhang mit Remote Procedure Call (RPC) kümmern muss, wird ein einzelner Client, also eine entfernte Instanz, auf dem Server genauso wie ein gewöhnlicher Thread als Worker im Pool registriert. Die konkrete Implementation dieses Workers ruft aber, anstatt den Task direkt auszuführen, via RPC eine Methode auf dem Client auf und übermittelt im die Instruktion mit zugehörigen Eingabewerten.

[7] http://dev.root1.de/projects/simon



Auf dem Client wartet ein sogenannter SocketAgent ständig darauf, dass er Tasks vom Server zugeordnet bekommt. Sobald ein Task angekommen ist, wird dieser ausgeführt und die Resultate werden wieder an den SocketWorker geschickt, welcher auf dem Server ist. Der SocketWorker kümmert sich dann um das korrekte Speichern der Resultate und meldet dem Pool, dass er wieder

Arbeit aufnehmen kann.