Bernburg
Dessau
Köthen

Hochschule Anhalt
Anhalt University of Applied Sciences

emw
Fachbereich
Elektrotechnik, Maschinenbau
und Wirtschaftsingenieurwesen

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Engineering (B. Eng.)

Yang Zhang

Vorname Nachname

Elektro- und Informationstechnik, 2011, 4055535

Studiengang, Matrikel, Matrikelnummer

Thema: Mathematische Grundlagen der robusten Implementierung von Vektoroperationen

Prof. Dr. Marc Enzmann

1. Prüfer(in)

Prof. Dr. Andrea Jurisch

2. Prüfer(in)

13. 02. 2015

Abgabe am

Selbstständigkeitserklärung

Hiermit erkläre ich, dass die Arbeit selbstständig verfasst, in gleicher oder ähnlicher Fassung noch nicht in einem anderen Studiengang als Prüfungsleistung vorgelegt wurde und keine anderen als die angegebenen Hilfsmittel in Quellen, einschließlich der angegebenen oder beschriebenen Software, verwendet wurden.

Köthen, 10.02.2015

Ort, Datum | Unterschrift des Studierenden

Sperrvermerk

Sperrvermerk: ja [ ] nein [X]

Wenn ja: Der Inhalt der Arbeit darf Dritten ohne Genehmigung der/des (Bezeichnung des Unternehmens) nicht zugänglich gemacht werden. Dieser Sperrvermerk gilt für die Dauer von X Jahren.

Köthen, 10.02.2015

Ort, Datum | Unterschrift des Studierenden

Kurzfassung

Eine Vektoroperation ist die Berechnung von Vektoren. Zum Beispiel Addition, Subtraktion und skalares Produkt. Die Vektoroperation wird meistens für die Berechnung von Signalen, z.B. Faltung verwendet. Ein Programm ist für die Operationen entworfen worden. Mit welchen Methoden kann die Vektoroperation mit vielen Elementen einfach sein und wie kann die Robustheit des Programms stabil sein? Diese Probleme müssen in dieser Arbeit untersucht und gelöst werden.

Hierfür wurde die sichere Programmiersprache SPARK2014 gewählt, um die Robustheit des Programms realisieren zu können. Und das wichtigste Werkzeug, die Vor- und Nachbedingung von SPARK2014, wurde verwendet. Für das Programm ist ein mathematisches Konzept wichtig, es ist die Basis der dynamischen Programmierung. Am Anfang wurden mögliche Vor- und Nachbedingungen geschrieben. Anschließend durch mathematische Kenntnisse ‚Notwendig und Hinreichende Bedingung' wurden die repräsentativen Vorbedingungen gewählt und die Reihenfolge der Vorgehensweise festgelegt. Danach wurde die Methode durch mathematische Kenntnisse nachgewiesen. Zum Schluss wurden die entworfenen Vor- und Nachbedingungen durch die mathematischen Beispiele getestet.

Abstract

A vector operation is a calculation of vector. For example, addition, subtraction und scalar product. The vector is mostly on the calculation of signal, such as 'folding' used. A program has been designed for the operations. Which methods vector operation with many elements can be simple and can the robustness of the program to be stable. These issues need to be investigated and solved in this work.

In this work, we choose a secure programming language-SPARK2014 to the robustness of the program to realize. And the important tool pre- and post-condition of SPARK2014 is used. A mathematical concept of program is important; it is the basis of dynamic programming. At the beginning of the possible pre- and post-conditions are written. Then, by Mathematical knowledge-, Necessary and Sufficient Condition 'I choose the representative preconditions, and place an order. Thereafter, the method is proved by mathematical knowledge. Finally, the designed pre- and post-conditions are tested by the mathematical examples.

Inhaltsverzeichnis

1 Motivation und Zielsetzung

1.1 Einleitung in die Thematik

Heutzutage kann man die Gesellschaft als ‚Informationsgesellschaft' bezeichnen. In unserem Informationszeitalter spielt der Computer eine wichtige Rolle. Die Software von Computern bezieht sich auf alle Aspekte in der modernen Gesellschaft. Banksysteme, Transportsysteme, Mediensysteme, industrielle Kontrollsysteme und Officesysteme, alle sind abhängig von Software. Aus diesem Grund muss die Sicherheit der Menschen und das Geheimnis von Eigentum in einer korrekten Software gewährleistet sein. Software hat viele Formen. Die Korrektheit bei der Programmierung und die sofortige Reaktion sind die Schlüssel. Es gibt einige Softwares, z. B. Kalkulationsprogramme und Word Prozessoren. Wenn es Fehlern gibt, entstehen viele Schwierigkeiten. Dann bricht die Software nicht nur zusammen, sondern es gehen auch Daten verloren, schlimmer noch, die Zerstörung des Systems. Allgemein werden diese Schwierigkeiten in Programme für Sicherheit und Programme für Geheimnis aufgeteilt. Ein Programm, das Sicherheitsfehler aufweist, kann zum Verlust von Leben oder zu Umweltschäden führen. Wenn man numerische Berechnungen mit Software durchführt, entstehen gelegentliche Fehler. Normalerweise läuft ein Code in den problemlosen Fällen. Aber er verursacht sporadische Softwarefehler, die zu fehlerhaften Berechnungsergebnissen, schlimmer noch, zu Programmabstürzen führen können. Dabei liegt die Fehlerursache häufig nicht im Programm der Software selbst, sondern daran, dass numerische Zahlendarstellungen einen falschen Zustand aufweisen, z. B. keine Initialisierung, ein überschreiteter Wertebereich oder eine überschreitete Auflösung. Deshalb ist eine korrekte Software wichtig. Die Grundlage einer korrekten Software ist eine fehlerfreie Programmierung.

Die Vektoroperation spielt eine wichtige Rolle im heutigen Leben. In vielen Bereichen wird sie immer öfter genutzt, z. B. Regelung, Kommunikation oder Digitalsignalverarbeitung. Daraus ergibt sich die Notwendigkeit, ein sicheres und mit hoher Robustheit ausgestattetes Programm von Vektoroperationen zu entwickeln, das Fehler minimieren kann.

Zuerst wurde sich Gedanken über die fehlerfreie Software gemacht. Warum ist es für den Softwareentwurf sehr wichtig? Dann wurde das Buch ‚Software Engnieering'(Ian Sommerville, 2001) gelesen und sich die erforderlichen Kenntnisse angeeignet und verstanden.
Ein guter Softwareprozess sollte die Entwicklung fehlerfreier Software zum Ziel haben. Fehlerfreie Software ist Software, die genau ihrer Spezifikation entspricht. Das bedeutet jedoch

nicht unbedingt, dass sich die Software immer so verhalten wird, wie die Benutzer das erwarten. Die Spezifikation kann Fehler enthalten, die in der Software widergespiegelt werden, und die Benutzt könnten das System missverstehen oder falsch verwenden. Fehlerfreie Software ist daher für die Anzahl der Systemausfälle bedeutend und sollte in der Entwicklung kritischer Systeme stets angestrebt werden. [1]

Wenn man einen Softwareprozess entwerfen möchte, was sollte man tun? Mit Hilfe des Buches ‚Hybride Softwareentwicklung (Björn Berg, Philip Knott, Gregor Sandhaus)' kann man diesen Prozess besser verstehen.

Nach Sommerville gibt es den idealen Softwareentwicklungsprozess nicht. Daher haben viele Unternehmen ihre eigene Herangehensweise an die Softwareentwicklung erarbeitet. Trotz vieler unterschiedlicher Softwareentwicklungsprozesse gibt es einige grundlegende Aktivitäten, die alle Softwareentwicklungsprozesse gemeinsam haben:

1. *Spezifikation: Die Funktionalität einer Software und ihre Grenzen werden definiert.*
2. *Entwurf und Implementierung: Die Software wird gemäß ihrer Spezifikation erstellt.*
3. *Validation: Die Software wird validiert, damit sichergestellt ist, dass sie den Kundenanforderungen entspricht.*
4. *Evolution: Die Software wird weiterentwickelt, um sich ändernde Kundenanforderungen zu erfüllen. [2]*

Dann kann man sagen, der Entwurf ist Basis der Softwareentwicklung, und die Implementierung ist der hauptsächliche Teil der Softwareentwicklung. Bei dem Prozess des Entwurfs wird das Ziel in ein mathematisches Modell verwandelt.

Um ein mit hoher Robustheit und wenig fehlerhaftes Programm entwerfen zu können, sollte man ein perfektes mathematisches Modell zuerst entwickeln. Durch das perfekte mathematische Modell kann man die so genannte Programmierung gut implementieren. Danach, bei der Implementierung, soll eine robuste Umgebung entworfen werden, das heißt, man muss eine stark typisierte und mit hoher Sicherheit ausgestattete Programmiersprache auswählen. Wenn man über die Sicherheitsstufen von Programmiersprachen diskutiert, kann man drei Sicherheitsstufen zusammenfassen. Die erste Stufe ist die Programmiersprache Spark. Diese Programmiersprache nutzt eine ‚Defensive Programmierung', sie kann durch die Definition von Vor- und Nachbedingungen realisiert werden, die statisch und dynamisch Fehler während der Compilezeit und Laufzeit vermeidet. Dann ist die Sprache Ada, sie kann die Fehler während die Compliezeit überprüfen und statische Fehler vermeiden. Zum Schluss ist

die Sprache. Es gibt keine Prüfung vor dem Lauf des Programms. Das bedeutet, während des Laufs des Programms können statische und dynamische Fehler passieren und dann wird das ganze Programm zerstört.
Mit einer sicheren Sprache kann man ein robustes Programm aufbauen. In dieser Arbeit wurde die Programmiersprache SPARK ausgewählt. Sie kann durch die Begrenzung der Vor- und Nachbedingung Fehler exakt kontrollieren und Prüfleistung einsparen. Durch die Robuste Bibliothek kann man auch Fehlerzustände bei der Vektoroperation vermeiden, um eine hohe Robustheit und Sicherheit von Vektoroperationen und eine exakte Kontrolle der eingegebenen und ausgegebenen Werte zu realisieren.

1.2 Zielsetzung der Arbeit

Das Thema ‚Mathematische Grundlagen der robusten Implementierung von Vektoroperationen' ist Gegenstand des Berufspraktikums. Ein Entwurf der Vor- und Nachbedingungen durch mathematische Grundlagen ist die hier gestellte Aufgabe.
Um eine robuste Bibliothek der Vektoroperation entwerfen zu können, wurde in der Praktikumsarbeit die Programmiersprache Ada gewählt. Es wurde über die drei Sicherheitsstufen von Programmiersprachen diskutiert. Wie bereits zuvor erwähnt, gehört Ada zur zweiten Sicherheitsstufe. Deshalb ist Ada nicht die beste Programmiersprache.
In dieser Bachelorarbeit wurde die Programmiersprache SPARK gewählt, um die Robustheit der Vektoroperationen zu implementieren Die Programmiersprache SPARK nutzt eine ‚Defensive Programmierung'. Es kann durch die Definition von Vor- und Nachbedingung realisiert werden, statische und dynamische Fehler während der Compilezeit und Laufzeit zu vermeiden.
Die Bachelorarbeit wurde in zwei Teile aufgeteilt: „Teil des Entwurfs der Methode der Vor- und Nachbedingungen für die Programmierung und Teil der Programmierung". Ein Entwurf einer vollständigen und durchführbaren Methode ist die erste Aufgabe. Hierbei soll eine gute und nicht überflüssige Methode von Vor- und Nachbedingungen untersucht werden. Die Grundlage der Untersuchungen ist die Mathematik. Außerdem soll mit Hilfe der Mathematik die Methode bewiesen werden, weil die Programmierung nur den grammatischen Fehler testet. Sie kann nicht die logischen Fehler testen. Bei der Programmierung kann der falsche Entwurf von Vor- und Nachbedingung nicht die reale Wirkung realisieren, trotzdem kann sie ausgeführt werden. Danach sollen die Vor- und Nachbedingungen getestet werden. Das ist die zweite Aufgabe. Es soll eine Testmethode für Vor- und Nachbedingungen entworfen

werden. Anschließend können die korrekten Testfunktionen innerhalb einer anderen Bachelorarbeit programmiert werden.
Die Aufgaben in dieser Arbeit sind folgende:

- Entwurf der Methode von Vor- und Nachbedingungen für die Programmierung in SPARK
- Entwurf der Methode der Testfunktion für Vor- und Nachbedingungen

Lösungskonzept

Innerhalb dieser Aufgabe sollen die Methoden für die Vor- und Nachbedingung für die Programmierung in SPARK entworfen werden. Zuerst muss man alle Vektoroperationen kennen. Der nächste Schritt ist der, dass man nach der Forderung des Hinweises der Vektoroperationen die Bereiche der Vektoren sucht. Der dritte Schritt ist die Vermeidung möglicher Fehler von Vektoroperationen durch die Untersuchung aller Vor- und Nachbedingungen. Danach durch mathematische Kenntnisse ‚Notwendige und Hinreichende Bedingung' wählt man die repräsentativen Vor- und Nachbedingungen und ordnet eine Reihenfolge an. Um die Korrektheit der Vor- und Nachbedingungen festzulegen, werden die mathematischen Kenntnisse ‚Logische äquivalente Erklärung' gewählt. Zum Schluss werden mit Hilfe der Aktivitätsdiagramme und mathematischen Beispiele die Methoden der Testfunktionen von Vor- und Nachbedingungen entworfen.

2 Stand von Wissenschaft und Technik

In dieser Bachelorarbeit ist die Arbeitsumgebung die Programmiersprache SPARK, deshalb wird sie hier vorgestellt. Weil die Themen dieser Arbeit die Vektoroperationen sind, werden vier Vektoroperationen erklärt. Um die Korrektheit des Entwurfs nachzuweisen, werden in diesem Kapital noch die Beweismethoden, Variation von Aussage und Implikation, bedingte und gleichwertige Aussagen dargestellt. Außerdem wird die Norm und das UML erklärt.

2.1 Beweismethoden

In SPARK wird das Programm durch die Benutzung der ‚Proof Funktion' wirksam eingeschränkt und vereinfacht, sodass das Programm robust ausgeführt werden kann. Die ‚Proof Funktion' kann durch die Vor- und Nachbedingung und die behauptete Anmerkung realisiert werden. Vor der Benutzung der Vor- und Nachbedingung sollte sichergestellt werden, dass diese korrekt sind. Die Korrektheit der Vor- und Nachbedingung muss durch unterschiedliche Beweismethoden nachgewiesen werden.

In diesem Abschnitt werden die Beweismethoden angezeigt. In der Mathematik gibt es einen Begriff ‚Voraussetzungen'. Von der gewissen Voraussetzung V wird ein anderer Begriff formuliert. Das ist die ‚Behauptung'. Voraussetzungen und Behauptungen sind Aussagen, ihre Verknüpfung im mathematischen Satz sind Implikationen: $V \rightarrow B$. Der Nachweis ist für den Wahrheitswert w die Behauptung B. Beim Nachweis wird die Behauptung B durch die nachgewiesenen Voraussetzungen geschlussfolgert. Und als Nächstes werden die verschiedenen Methoden vorgestellt

2.1.1 Der direkte Beweis

Beim direkten Beweis geht man von einer Aussage A aus, deren Wahrheitswert bekannt ist, und folgert daraus die Aussage B. Die Aussage B ist dann ebenfalls wahr, denn aus einer wahren Prämisse kann man nur eine wahre Konklusion folgern. Eine falsche Konklusion kann aus einer wahren Prämisse nicht gefolgert werden. [3]

Beispiel:

V: a, b reell, a≥0, b≥0 B: $\sqrt{a \cdot b} \leq \frac{a+b}{2}$

Beweis: Zuerst werden die wahren Aussagen gefunden.

A: $(a-b)^2 \geq 0$

Daraus wird gefolgert:

$$a^2 - 2ab + b^2 \geq 0 \rightarrow a^2 + 2ab + b^2 \geq 4ab \rightarrow (a+b)^2 \geq 4ab$$

$$\rightarrow a + b \geq 2\sqrt{ab} \rightarrow \frac{a+b}{2} \geq \sqrt{a \cdot b}$$

2.1.2 Der indirekte Beweis

Beim indirekten Beweis setzt man zuerst eine Negation der Behauptung: A =, nicht Behauptung' oder A =, die Behauptung ist falsch', dann kann man durch die Negation der Bahauptung A eine Aussage B folgern. Dann kann man durch Kenntnisse beurteilen, dass die Aussage B falsch ist, sodass die negierte Behauptung ebenfalls falsch ist. Wenn die Negation der Behauptung falsch ist, ist die Behauptung bestimmt wahr.

Beispiel:

V: a,b reell, a≥0, b≥0 B: $\sqrt{a \cdot b} \leq \frac{a+b}{2}$

Beweis: Zuerst schreibt man die Negation der Behauptung, dass Zahlen a≥0, b≥0 mit $\frac{a+b}{2} \leq \sqrt{a \cdot b}$ existieren. Dann folgert man durch Quadrieren $\frac{(a+b)^2}{4} \leq a \cdot b \rightarrow$,

durch Multiplizieren mit 4: $(a+b)^2 \leq 4ab$,

durch Berechnen des Quadrates: $a^2 + 2ab + b^2 \leq 4ab$ b,

durch Subtrahieren von 4ab: $a^2 - 2ab + b^2 \leq 0$,.

Denn der Fehler ist offenbar. Weil die linke Seite der Ungleichung sich als Quadrat eines Binoms darstellen lässt und niemals negativ sein kann. Das bedeutet, dass $\frac{a+b}{2} \leq \sqrt{a \cdot b}$

Falsch ist, denn $\frac{a+b}{2} \geq \sqrt{a \cdot b}$ ist wahr.

2.2 Variation von Aussage und Implikation

Die Sprache mit elementarer Logik ist das Fundament der Sprache der Mathematik. Die elementare Logik bietet die Terminologie und Strukturen, die bei der Definition von mathematischen Begriffen und die Schaffung von mathematischen Argumenten verwendet werden. Logik spielt auch eine wichtige Rolle in der Entwicklung von Computerprogrammen und elektronischen Schaltungen.

Bei dem Entwurf des mathematichen Modells der Vor- und Nachbedingung ist jede Bedingung eine Aussage und die Variation von der Aussage. Anschließend werden Notwendigkeit, Aussage und seine Variation vorgestellt.

Aussage

Eine Aussage ist ein Satz, der wahr oder falsch ist, aber nicht beides. [4]

Es gibt drei verbundene Aussagen, die von Aussage p→q erzeugt wird. Es sind die Konverse, seine Inverse und seine Kontrapositive.

Konverse

Definition: Die Konverse der Aussage ist ‚wenn p, dann q' ist die Aussage ‚wenn q, dann p. 'sogenannt die Konverse von ‚p→q' ist ‚q→p'. [4]

Zum Beispiel: Wenn das Wetter kalt ist, dann schneit es, ist die Konverse von ‚Wenn es schneit, dann ist das Wetter kalt'. Das Beispiel zeigt die Wahrheit von der Aussage, aber die Wahrheit der Konverse kann nicht festgelegt werden.

Inverse

Definition: Die Inverse der Aussage ist die Aussage, dessen Ergebnis wird durch negative p und negative q ersetzt, sogenannt die Inverse von ‚p→q' ist ‚(~p)→(~q)'. [4]

Zum Beispiel, die Inverse von ‚Wenn das Polygon ein gleichschenkliches Dreieck ist, dann ist es ein Dreieck' ist ‚Wenn das Polygon nicht ein gleichschenkliches Dreieck ist, dann ist es nicht ein Dreieck'. Natürlich kann man durch die Wahrheit dem Koditional nicht die Wahrheit der Inverse festlegen.

Kontrapositive

Definition: Die Kontrapositive der Aussage ‚Wenn p, dann q' ist ‚Wenn –q, dann -p'. Sogenannt die Kontrapositive der ‚p→q' ist ‚(~q)→(~p)'. [4]

Zum Beispiel, die Kontrapositive von ‚Wenn es schneit, dann ist das Wetter kalt' ist ‚Wenn das Wetter nicht kalt ist, dann schneit es nicht'.

Alle Beziehungen werden in folgendem Bild illustriert.

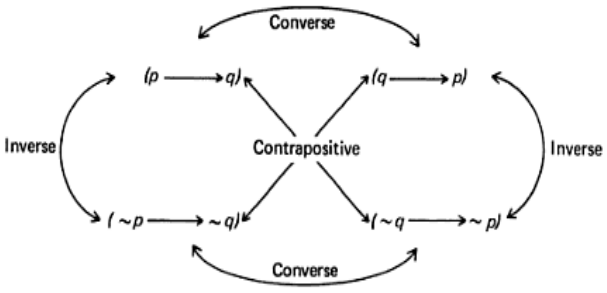

Abbildung 2.1: Alle Beziehungen [5]

2.3 Bedingte und gleichwertige Aussagen

In dieser Arbeit wird die Methode der Vor- und Nachbedingung durch die Benutzung der Kenntniss ‚Logische äquivalente Erklärung' und ‚Zwei logische Bedingungen' nachgewiesen. Durch die ‚Logische äquivalente Erklärung' kann man indirekt einige korrekte Aussagen bekommen, dann werden die Aussagen in die Vorbedingungen umgewandelt. So können die Vor- und Nachbedingungen einfach gefunden werden. Wenn man die repräsentativen Vor- und Nachbedingungen aus allen Vor- und Nachbedingungen wählt, muss man die ‚Zwei logischen Bedingungen' nutzen. Nachfolgend werden beide Kenntnisse näher erläutert.

2.3.1 Logische äquivalente Erklärung

Definition: In der Logik werden zwei Erklärungen r und a gleichwertige Erklärungen genannt, wenn sie identische Wahrheitswerte haben. [6]

Das bedeutet, wenn zwei Erklärungen r und a die gleiche Wahrheit in jeder möglichen Situation haben, kann gesagt werden, dass r logisch gleichwertig mit a ist.

a.) *Weil die Wahrheiten für ‚p→q' und ‚(~q)→(~p)' gleich sind, sind eine Aussage und seine Kontrapositive gleichwertig.* [7]

Zum Beispiel, ‚Wenn das Polygon ein gleichschenkliches Dreieck ist, dann ist es ein Dreieck' kann mit ‚Wenn das Polygon nicht ein Dreieck ist, dann ist es nicht ein gleichschenkliches Dreieck' ausgetauscht werden.

b.) *Weil die Wahrheiten für ‚q→p' und ‚(~p)→(~q)' gleich sind, sind die Konverse von Aussage und die Inverse von Aussage gleichwertig.* [7]

2.3.2 Zwei logische Bedingungen

In diesem Teil werden die zwei logischen Bedingungen erklärt. Wenn man die Vorbedingungen sucht, muss man die Kenntnis benutzen.
Häufig werden in der Mathematik die Wörter ‚notwendig' und ‚hinreichend' in der Aussageerklärung benutzt. Notwendige und hinreichende Bedingungen beschreiben, ob aus einer Aussage eine andere Aussage folgt. In diesem Entwurf von Vor- und Nachbedingungen existieren die logischen Bedingungen zwischen jeder Vorbedingung. Durch die logischen Bedingungen wird die Ordnung von Vorbedingungen abgestellt.

2.3.2.1 Notwendige Bedingung

‚p ist ausreichend für q' bedeutet, wenn p wirklich ist, will q auch wirklich sein. Deshalb ‚p ist ausreichend für q' ist gleichwertig mit ‚Wenn p, dann q'. [7]
Man hat zwei Aussagen A und B, wobei A notwendig für B sein soll. Dann bedeutet das: Wenn B eintritt, dann muss A auch eintreten. A ist also eine Art Eigenschaft von B. A ist aber nicht die einzige Eigenschaft von B. Der umgekehrte Fall muss nicht erfüllt sein.

2.3.2.2 Hinreichende Bedingung

Ähnlich bedeutet der Satz ‚q ist erforderlich für p', wenn q nicht passiert, wird p auch nicht passieren. Sogenannt ~q→~p. [7]
Man hat zwei Aussagen A und B, wobei A hinreichend für B sein soll. Dann bedeutet das: Wenn A eintritt, dann ist auch B erfüllt. A ist also eine Ursache für B. A ist aber nicht die einzige Ursache für B. B kann auch eintreten, ohne dass A eingetreten ist. Der umgekehrte Fall muss nicht erfüllt sein: Wenn B eintritt ist auch A erfüllt.

2.3.2.3 Notwendige und hinreichende Bedingung

Ist eine Bedingung *A* sowohl notwendig als auch hinreichend für eine Aussage *B*, also $A \Rightarrow B$ und $B \Rightarrow A$, so spricht man von einer äquivalenten Bedingung. Schreibt man eine äquivalente Bedingung in die "Wenn..., dann..."-Form, dann ist auch der Kehrsatz wahr.
Man schreibt dann: $A \Leftrightarrow B$.
Die übliche Sprechweise ist dann: "Aussage *A* genau dann, wenn Aussage *B*".

2.4 Vorstellung von SPARK

In doeser Bachelorarbeit sollen die Vor- und Nachbedingungen von Vektoroperationen entworfen werden. Der Entwurf ist für die Programmierung in SPARK. Die Vor- und Nachbedingung ist eine Besonderheit von SPARK. In diesem Abschnitt wird eine Kurzvorstellung von SPARK gegeben, und zwar, die wichtige Besonderheit der Vor- und Nachbedingung.
SPARK 2014 ist eine Programmiersprache, und eine Reihe von Verifikationswerkzeugen wird für die Anforderungen an eine Hochsicherungs - Softwareentwicklung entworfen. SPARK 2014 ist basiert auf Ada 2012. [14]

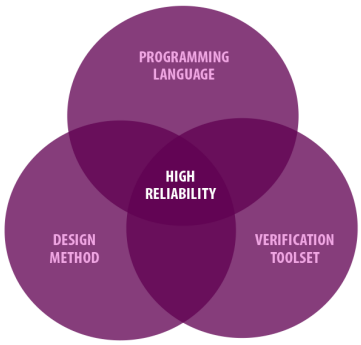

Abbildung 2.2:SPARK [15]

Sie besteht aus einer Programmiersprache, einem Überprüfungs-Toolset und einer Design-Methode. Die Kombinierung soll sicherstellen, dass ultra-niedrige Defekt-Software in Anwendungsbereichen, wo hohe Zuverlässigkeit gewährleistet sein muss, Anwendung findet. Z. B, Avionik im Flugzeugbau, medizinische Systeme, Prozesskontrollsoftware in Kernkraftwerken und Finanz-Software für Banken und Versicherungen.

2.4.1 Besonderheit von SPARK

SPARK zielt darauf ab, die Vorteile von Ada zu nutzen und gleichzeitig zu versuchen, alle potenziellen, unterschiedlichen Bedeutungen und Unsicherheiten zu beseitigen. Beim Design von SPARK ist die Forderung eindeutig, und es ist erforderlich, dass ihr Verhalten unabhängig von der Wahl des Ada Compilers ist. Diese Ziele werden einerseits durch Weglassen einiger Ada problematischer Funktionen und andererseits durch die Einführung von Annotationen oder ‚formaler Anmerkungen', die Absichten und Anforderung für bestimmte Komponenten eines Programms kodieren, realisiert.

2.4.2 Die Anmerkung der SPARK- Vor- und Nachbedingung

Die Aufgabe dieser Arbeit ist es, einen mathematischen Entwurf für Vor- und Nachbedingungen zu entwickeln. Daraus ergibt sich die Notwendigkeit, die Vor- und Nachbedingung von SPARK zu verstehen.
Vor- und Nachbedingung, das sind sehr wichtige Anmerkungen in Spark 2014. Sie können durch die Angabe der beabsichtigten Verhalten die Unterprogrammverträge stärken. Vor- und Nachbedingung definieren einen Vertrag dafür, welche Forderungen die Funktion erfüllen müssen.
Die Vorbedingung stellt dar, welche Voraussetzungen die Funktion erfüllen muss. Wenn sie gilt, so müssen nach der Ausführung der Funktion alle Nachbedingungen erfüllt sein, sonst ist das Programm nicht korrekt.
Die Nachbedingung einer Funktion oder eines Programms stellt dar, welche Aussagen nach der Ausführung gelten müssen, falls zuvor die Vorbedingungen erfüllt waren.

2.4.3 Wichtigkeit der Vor- und Nachbedingung

In SPARK spielt die Vor- und Nachbedingung eine sehr wichtige Rolle. Die erste Funktion der Vorbedingung ist die Vermeidung der Zerstörung des Programms. Z. B Man führt eine normale mathematische Rechnung: $Y=X^{1/2}$ durch. Anhand der mathematischen Grundlagen weiß man, dass der Wert von X größer als 0 sein muss. Wenn X ein negativer Wert ist, weist das Programm Fehler auf, schlimmer noch, es führt zur Zerstörung während der Laufzeit. Vorbedingung ist eine effektive Methode, die Situation zu vermeiden. Man schreibt eine Vorbedingung X>0. Dann kann das Programm bestimmt ausgeführt werden. Die andere Funktion der Vorbedingung ist die Vereinfachung des Tests. Mit der Verwendung der Vorbedingung können viele Tests vereinfacht werden. Zum Beispiel bei der Addition zwischen beiden Vektoren, jeder Vektor hat 1000 Elemente. Wenn der Befehl Anfangs ausgegeben wird, sollten alle Elemente implementiert werden. Die Implementierung ist kompliziert. Dann müssen einige Voraussetzungen erfüllt sein. Durch die Voraussetzungen werden nur wenige Elemente getestet. Die anderen Elemente brauchen nicht getestet zu werden. Das ist der Vorteil der Vorbedingung.
Mit Hilfe der Nachbedingung kann man feststellen, ob ein richtiges Ergebnis erzeilt wurde.

2.5 Vorstellung der Vektoroperation

In vielen Bereichen müssen numerische Berechnungen durchgeführt werden. Dann muss die Problematik bei der Laufzeit beachtet werden. Diese Problematik wird zur Zerstörung des Systems führen.
In dieser Bachelorarbeit wurden die Voraussetzungen von Vektoroperationen untersucht, um ihre Robustheit zu testen. Deshalb ergibt sich auch die Notwendigkeit, die verschiedenen Vektoroperationen zu verstehen.

2.5.1 Addition

Sollen zwei Vektoren $\vec{a}$ und $\vec{b}$ addiert werden, so bringt man durch Parallelverschiebung den Anfangspunkt des Vektors $\vec{b}$ in den Endpunkt des Vektors $\vec{a}$. Die Summe $\vec{a}+\vec{b}$ ist dann derjenige Vektor, der vom Anfangspunkt $\vec{a}$ zum Endpunkt von $\vec{b}$ führt.
Betrachte man nun das folgende Bild:

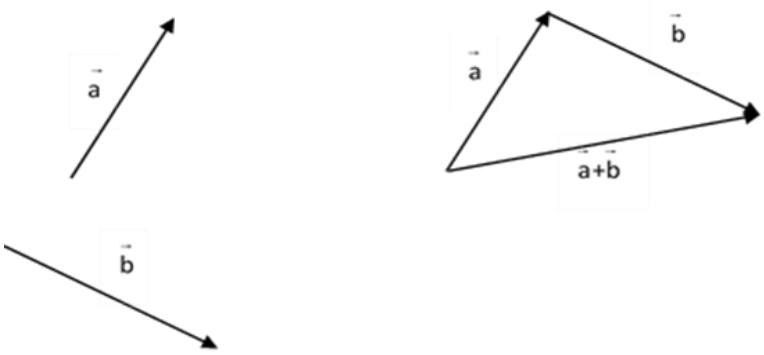

Abbildung 2.3: Addition

Die Summe bzw. die Addition zweier Vektoren ist folgendermaßen definiert:
Gleichung 2.1

$$\vec{a}+\vec{b}=(a_1,a_2,...,a_n)+(b_1,b_2,...,b_n)=(a_1+b_1,a_2+b_2,...,a_n+b_n) \quad \text{Gl. 2.1[8]}$$

Bei der Addition von Vektoren gibt es ein neutrales Element, den Nullvektor $\vec{0}=(0,0,...,0)$, dessen Komponenten alle Null sind und für die gilt
Gleichung 2.2

$$\vec{0}+\vec{a}=\vec{a} \quad \text{Gl. 2.2}$$

Zu jedem $\vec{a}=(a_1,a_2,...,a_n)$ gibt es auch ein inverses Element $-\vec{a}=(-a_1,-a_2,...,-a_n)$ mit

Gleichung 2.3

$$\vec{a} + -\vec{a} = \vec{0} \qquad \text{Gl. 2.3}$$

Damit kann man die Subtraktion von Vektoren definieren.

2.5.2 Subtraktion

Subtraktion von Vektoren

Gleichung 2.4

$$\vec{a} - \vec{b} = (a_1, a_2, ..., a_n) - (b_1, b_2, ..., b_n) = (a_1 - b_1, a_2 - b_2, ..., a_n - b_n) \qquad \text{Gl. 2.4[8]}$$

2.5.3 Multiplikation eines Vektors mit einer reellen Zahl

Für Vektor $\vec{a}$ in der Ebene bzw. im Raum hat diese Multiplikation folgende Bedeutung: man multipliziert einen Vektor mit einer reellen Zahl, dann erhält man einen Vektor $\gamma\vec{a}$ mit dem Betrag $|\gamma\vec{a}| = |\gamma| \cdot |\vec{a}|$. Schauen Sie das folgende Bild an.

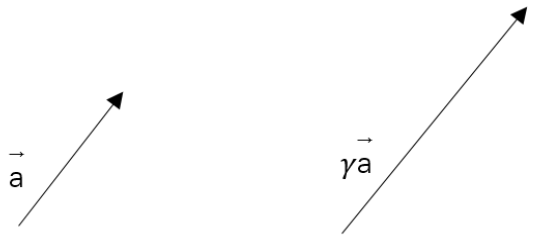

Abbildung 2.4: Multiplikation

Die Multiplikation eines Vektors mit einer Zahl ist folgendermaßen definiert:

Gleichung 2.5

$$\gamma\vec{a} = \gamma(a_1, a_2, ..., a_n) = (\gamma a_1, \gamma a_2, ..., \gamma a_n) \qquad \gamma \in R \qquad \text{Gl. 2.5[8]}$$

Dann für $\gamma\vec{a} > 0$, gleichsinnig parallel zum Vektor $\vec{a}$ ist.

für $\gamma\vec{a} < 0$, gegensinnig parallel zum Vektor $\vec{a}$ verläuft und

für $\gamma\vec{a} = 0$, gleich dem Nullvektor $\vec{0}$ ist.

2.5.4 Das Skalarprodukt von Vektoren

Ein Produkt aus zwei Vektoren, dessen Ergebnis eine skalare Größe ist, wird skalares Produkt genannt.

Besitzen zwei Vektoren $\vec{a}$ *und* $\vec{b}$ *dieselbe Dimension n, also dieselbe Anzahl von Zeilen. So das Skalarprodukt-es wird mit SP abgekürzt- definiert durch*

Das Skalarprodukt $\vec{a} \cdot \vec{b}$ *zweier Vektoren* $\vec{a}$ *und* $\vec{b}$ *ist folgendermaßen definiert:*
Gleichung 2. 6

$$SP\vec{a},\vec{b} = \vec{a} \cdot \vec{b} = (a_1, a_2, ..., a_n) \cdot (b_1, b_2, ..., b_n) = a_1 b_1 + a_2 b_2 + ... + a_n b_n \qquad \text{Gl. 2.6}$$

Demzufolge ist das Skalarprdukt zweier Vektoren die Summe des Produkts der Komponenten derselben Zeilen. [9]
Das Skalarprodukt zweier Vektoren ist kein Vektor, sondern eine reelle Zahl, also ein Skalar.

2.6 Norm (Mathematik)

Norm spielt eine wichtige Rolle in der Berechnung von Vektoren. In diesem Programm wurde meistens Max_Norm, die so genannte unendliche Norm verwendet, um das maximale Element im Vektor zu finden. Nachfolgend wird der Begriff Norm vorgestellt.

2.6.1 Definition

Norm ist ein mathematisches Objekt. Bei der Bestimmung der Lösung eines linearen Gleichungssystems Ax=b erhält man normalerweise eine Näherungslösung x'. Um die Näherung zu messen, muss man die ‚Größe' eines Vektors x $\in C^n$ messen. Dies geschieht durch die Einführung einer Norm ||x|| im C^n. Sie ist eine reell wertige Funktion. Jeder Vektor x$\in C^n$ hat eine reelle Zahl.

Die genauen Definitionen werden folgendermaßen beschrieben (Jochen Werner, 1992):
‚Eine Abbildung ||·||: $C^n \rightarrow R$ *heißt eine Norm auf dem* C^n*, wenn gilt*

(a) $\|x\| > 0$ *für alle , x≠0 (Definiertheit,*

(b) $\|ax\| = |a| \cdot \|x\|$ *für alle a* $\in C$*, x* $\in C^n$*(Homogenität),*

(c) $\|x + y\| \leq \|x\| + \|y\|$ *für alle x, y*$\in C^n$ *(Dreiecksungleichung).'[10]*

Die Normeigenschaften a), b), c) bestätigt man hier leicht.

Für jde Norm || · || gilt

$\|x-y\| \geq \|\|x\|-\|y\|\|$ *für alle x, y* $\in C^n$.

Aus c) folgt nämlich:

$\|x\| = \|(x-y)+y\| \leq \|x-y\|+\|y\|$,

also $\|x-y\| \geq \|\|x\|-\|y\|\|$. *Durch Vertauschen von x und y und wegen b) folgt*

$\|x-y\| = \|y-x\| \geq \|y\|-\|x\|$ [9]

2.6.2 Wichtige Normen

Die für die numerische Mathematik bei weitem wichtigsten Normen sind: [10]

a) Euklidische Norm:

Gleichung 2.7

$$|x| = \left(\sum_{i=1}^{n} |x_i|^2\right)^{1/2} = \sqrt{|x_1|^2 + |x_2|^2 + ... + |x_n|^2} \qquad \text{Gl. 2.7}$$

b) Maximalnorm:

Gleichung 2.8

$$\|x\|_\infty = \max|x_i| = \max\{|x_1|, |x_2|, ..., |x_n|\} \qquad \text{Gl. 2.8}$$

c) Betragssummennorm:

Gleichung 2.9

$$\|x\|_1 = \sum_{i=1}^{n} |x_i| = |x_1| + |x_2| + ... + |x_n| \qquad \text{Gl. 2.9}$$

2.6.3 Vekttorräume mit Skalarprodukt

Skalarprodukt ist eine komplizierte Vektoroperation. Es gibt viele Berechnungsschritte. Wenn die einfachen und mittleren Vorbedingungen für das Skalarprodukt im dritten Teil entworfen werden, können nicht alle Elemente getestet werden. Deshalb muss eine geeignete Methode gefunden werden. Mit Hilfe dieser Methode kann das Ergebnis des Skalarprodukts dargestellt werden. Dann werden Formeln dargestellt. Sie können direkt oder indirekt das Ergebnis des Skalarprodukts zeigen.

a.) Wenn die Norm durch ein Skalarprodukt erzeugt wird, z. B. [11]

Gleichung 2.10

$$\|x\|_1 = \left\{ \sum_{n=1}^{m} |\delta_n|^2 \right\}^{1/2} \quad im\ K^m \qquad \text{Gl. 2.10}$$

b.) Ist $\langle \cdot, \cdot \rangle$ ein Skalarprodukt auf dem Vektorraum X, so ist durch $\|x\| = \langle x, x \rangle^{1/2}$ eine Norm auf X erklärt. Ist $\langle \cdot, \cdot \rangle$ ein Semiskalarprodukt, so ist $\|\cdot\|$ eine Halbnorm. [11]

c.) Ist $\langle \cdot, \cdot \rangle$ ein Skalarprodukt auf X und $\|\cdot\|$ die zugehörige Norm, so gilt[11]

Gleichung 2.11

$|\langle x, y \rangle| \le \|x\| \|y\|$ für alle $x, y \in X$, Schwarzsche Ungleich Gl. 2.11

d.) Eine Norm $\|\cdot\|$ auf einem Vektorraum X wird genau dann durch ein Skalarprodukt $\langle \cdot, \cdot \rangle$ erzeugt, wenn die Parallelogrammidentität gilt. Das Skalarprodukt ist dann gegeben durch [11]

Gleichung 2.12

$$\langle x, y \rangle = \frac{1}{4} \left(\|x+y\|^2 - \|x-y\|^2 \right) \qquad K \in R \qquad \text{Gl. 2.12}$$

2.7 Grundlagen der UML

Um die Aufgaben in dieser Arbeit lösen zu können, müssen viele Funktionen und Prozeduren programmiert werden. Um die Struktur der Codes einfach verstehen zu können, benutzt man die Aktivitätsdiagramme. Diese Diagramme gehören zur UML.

2.7.1 Vorstellung der UML

Anhand (Booch, Rumbaugh, & Jacobso, 2009) kann man erkennen:

‚Die vereinheitlichte Modellierungssprache UML (Unified Modeling Language) ist eine standardisierte Sprache, die zum Anfertigen von Softwarebauplänen dient. Mit Hilfe von UML lassen sich die Artefakte softwareintensiver Systeme visualisieren, spezifizieren, konstruieren und dokumentieren.'[12]

Die UML bietet eine standardisierte Möglichkeit, Blaupausen eines Systems zu schreiben, auch konzeptionelle Dinge wie Geschäftsprozesse und Datenbank-Schemata.

2.7.2 Aktivitätsdiagramm

2.7.2.1 Vorstellung des Aktivitätsdiagramms

Aktivitätsdiagramme beschreiben den dynamischen Prozess in einem System. Sie sind mit Flussdiagrammen verwandt. Die Aktivitäten bestehen auf Aktionen und zeitlichen Verknüpfungen. Die Abläufe werden durch Pfeile zwischen den Aktionen beschrieben. Wenn es sich gegenseitig ausschließende Bedingungen gibt, wird die Verzweigung ausgeführt. Es existiert auch die Möglichkeit, dass Aktionen parallel stattfinden können.
Mit der Anwendung der Aktivitätsdiagramme kann man die interne Logik komplexer Operationen, Modellierung und Algorithmen visualisieren. Aktivitätsdiagramme können in Verantwortungsbereiche gegliedert werden.

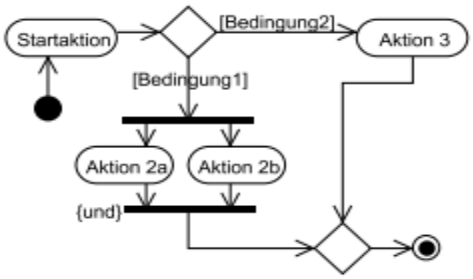

Abbildung 2.5: Aktivitätsdiagramme [13]

Im Beispiel finden nach der Startaktion entweder die Aktionen 2a und 2b statt, die beide beendet sein müssen, bevor der Ablauf terminiert, oder alternativ findet nach der Startaktion die Aktion 3 statt. Aktivitätsdiagramme können durch weitere sprachliche Elemente erweitert werden. [13]

2.7.2.2 Notation von Aktivitätsdiagrammen

Start-, Endknoten, Ablaufende

Ein Startknoten stellt den Beginn eines Ablaufes dar. Ein Endknoten beendet eine Aktivität vollständig. Ein Ablaufende terminiert einen Zweig einer Aktivität, die Aktivität selbst läuft weiter.

Abbildung 2.6: Symbol des Anfgangs-, Endzustands [16]

Aktion

Eine Aktion ist in UML das Basiselement, um Verhalten zu spezifieren.

Abbildung 2.7: Symbol der Aktion [16]

Pfeil

Ein Pfeil stellt den Fluss zwischen den verbundenen Elementen dar. In einer eckigen Klammer gibt es eine Bedingung. Wenn sie erfüllt ist, kann die Kante überquert werden.

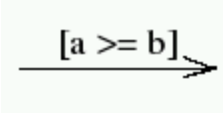

Abbildung 2.8: Symbol des Pfeils [16]

Entscheidung und Zusammenführung

a)Bei der Entscheidung wird nach Eintreffen des Tokens a entweder nach b oder nach c verzweigt

b)Bei der Zusammenführung wird nach Eintreffen der Token a oder b nach c verzweigt.

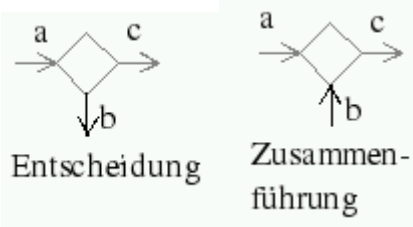

Abbildung 2.9: Symbol der Entscheidung und Zusammenführung [16]

3 Entwurf der Methode von Vor- und Nachbedingungen

Bei der Laufzeit entsteht bei den numerischen Rechnungen manche Problematik. Z. B. das Ergebnis ist überschritten. Durch die Benutzung der Vor- und Nachbedingung können die Fehler vor der Laufzeit entdeckt werden.

Im zweiten Teil wird die Aufgabe gestellt und Lösungskonzepte entworfen. Jetzt wird, gemäß dem Lösungskonzept, die erste Aufgabe gelöst.

3.1 Vektor mit einem Skalar

Skalar in Vektoroperationen bedeutet eine Multiplikation zwischen einem Vektor und einem Faktor, deshalb, wenn man die Vor- und Nachbedingung für Skalar programmiert, ist es ein bisschen einfach. Deshalb wird die Operation am Anfang erklärt.

Scale1: Result = Left × faktor

(Man benutzt das Alphabet, um die Formeln einfach darzustellen)

$\vec{a}$:bedeutet den Vektor ‚Left'.

$\vec{c}$:bedeutet den Vektor ‚Result'.

Scale2:Result = faktor × Right

$\vec{b}$:bedeutet den Vektor ‚Right'.

$\vec{c}$:bedeutet den Vektor ‚Result'.

Weil die Ideen, Methoden von Vorbedingungen und Beweise von Scale1 und Scale2 ähnlich sind, wird deshalb nur Scale1 erläutert.

3.1.1 Analysierung und Finden der Vorbedingung

Die Vektoroperation Skalar ist relativ einfacher als die Addition und Subtraktion. Diese Operation hat die einfachste Vorbedingung und eine Nachbedingung. Bei der Addition und Subtraktion müssen zwei Vektoren beachtet werden. Aber im Skalar beachtet man einen Vektor. Der Faktor braucht nicht getestet zu werden. Es wird die Methode des maximalen Wertes gewählt. Wenn der maximale Wert des Vektors die Forderung erfüllt, erfüllen die anderen Werte der Vektoren die Forderung.

3.1.2 Ordnung und Zusammenfassung der Vorbedingung

3.1.2.1 Einfache Vorbedingung

Wenn also $a_\infty \times Faktor \le Max\_\mathrm{Re}al$ und

Max_Real als der Grenzwert darstellbar bereit ist, dann kann jede Scale $a_n \times Faktor$ ausgeführt werden.

3.1.2.2 Komplexe Vorbedingung

Wenn also für jeder $n = 1, 2, ..., N$, gibt es

$\|a_n\| \times Faktor \le Max\_\mathrm{Re}al$ ($|a_n| \le Max\_\mathrm{Re}al$ und $|b_n| \le Max\_\mathrm{Re}al$)

dann kann jede Scale $a_n \times Faktor$ ausgeführt werden.

3.1.2.3 Nachbedingung

a.) $c_n = a_n \times Faktor$, für alle $n = 1, 2, ..., N$.

b.) $c_n \le Max\_\mathrm{Re}al$, für alle $n = 1, 2, ..., N$.

3.1.3 Nachweis von Vor- und Nachbedingungen

Gegeben: Vektor $\vec{a}$, $\vec{c}$ mit N Elemente

$\vec{a} = (a_1, a_2, ..., a_n)^T$;

$\vec{c} = (c_1, c_2, ..., c_n)^T$.

Definition:

Maximaler Absolutwert: $\|\vec{x}\|_\infty = \max\|x_n\|$, $n = 1, 2, ..., N$

Einfache Vorbedingung

Nach der Definition gibt es: $\|a\|_\infty \ge \|a_n\|$, für alle $n = 1, 2, ..., N$.

Damit ist: $\|a\|_\infty \times Faktor \ge a_n \times Faktor$, für alle $n = 1, 2, ..., N$.

Wenn also $\|a\|_\infty \times Faktor \le Max\_\mathrm{Re}al$ und

Max_Real als der Grenzwert darstellbar bereit ist, dann kann jede Scale $a_n \times Faktor$ ausgeführt werden.

Die komplexe Vorbedingung und Nachbedingung sind einfach zu verstehen. Deshalb werden sie hier nicht bewiesen.

3.2 Addition

Result=Left+Right

(Man benutzt das Alphabet, um die Formeln einfach zu darstellen)

$\vec{a}$:bedeutet Vektor ‚Left';

$\vec{b}$:bedeutet Vektor ‚Right';

$\vec{c}$:bedeutet Vektor ‚Result'.

3.2.1 Analysierung und Finden der Vorbedingung

In diesem Abschnitt sollen die Vorbedingungen der Addition gesucht werden. Im Gegensatz zu Skalar, die nur eine Vorbedingung hat, sollen hier mehrere Vorbedingungen und Nachbedingungen für die Addition entworfen werden. Weil es viele Vektoroperationen mit komplexen Berechnungsschritten gibt. Wenn eine einfache Vorbedingung benutzt wird, vielleicht kann man nicht beurteilen, ob das Ergebnis am Ende in Ordnung ist, oder es überläuft. Deshalb sollten mehrere Vorbedingungen und Nachbedingung entworfen werden. Dann werden die mittleren und komplexen Vorbedingungen entworfen. Die mittlere Vorbedingung ist komplexer als die einfache Vorbedingung. Die komplexe Vorbedingung ist am komplexesten. Für die mittleren und komplexen Vorbedingungen wird eine ausführliche Beurteilung erstellt.

Die Vorbedingungen und Nachbedingungen sollten Aussagen sein. In den Grundlagen sind die Aussage und ihre drei Variationen bekannt. Die Voraussetzungen brauchen nicht die notwendige und hinreichende Bedingung zu sein. Die hinreichende Bedingung ist genug für die Vorbedingung, weil eine Vorbedingung nicht die einzige Ursache für die Addition ist. Es gibt drei unterschiedliche Vorbedingungen. Die Operation der Addition kann vielleicht auch ausgeführt werden, ohne dass die einfache Vorbedingung eingetreten ist, sonst mit der mittleren oder komplexen Vorbedingung. Die Vorbedingungen haben folgende Form:
wenn Vorbedingung1, dann Addition;
wenn Vorbedingung2, dann Addition;
wenn Vorbedingung3, dann Addition;
Die Reihenfolge der Vorbedingungen wird durch die Menge der Elemente, die getestet wird, geordnet. Dann werden die Vorbedingungen in drei Teile geteilt, die einfache Vorebdingung, die mittlere Vorbedingung und die komplexe Vorbedingung. Nun werden die wichtigen und repräsentativen Vorbedingungen gesucht. Für die Addition kann man direkt eine Vorbedin-

gung zusammenfassen, dass jedes Ergebnis der Addition des Elements in Ordnung ist. Trotzdem die Vorbedingung richtig ist, ist es lästig und weitschweifig für die Programmierung. Für die Verringerung der Voraussetzung soll als Erstes eine einfache Vorbedingung entworfen werden, um die maximalen Werte von beiden Vektoren zu testen. Wenn die Grenzwerte der Vektoren die Forderungen erfüllen, erfüllen die anderen Werte der Vektoren die Forderung. Aber falls die Grenzwerte der Vektoren die Forderung nicht erfüllen, dann ist die einfache Vorbedingung nicht erfüllt. Dann wird noch eine ausführliche Beurteilung gemacht, bzw. die mittlere Vorbedingung. Die mittlere Vorbedingung ist auch nicht für jedes Element. Man muss auch einige repräsentative Elemente testen. Was sind die repräsentativen Elemente und wie findet man diese? Eine Idee ist es, dass das Ergebnis abhängig von zwei Werten ist. Wenn zwei Werte klein sind, ist das Ergebnis auch klein. Im Gegenteil ist das Ergebnis groß. Es soll vermieden werden, das Ergebnis zwischen beiden kleinen Werten zu testen. Deshalb werden die Elemente zum Testen gewählt, dessen Werte größer als die Hälfte der Grenzwerte sind. Wenn die einfache Vorbedingung und mittlere Vorbedingung ausgeführt, aber nicht erfüllt wurden, muss man eine komplexe Vorbedingung erstellen. In der komplexen Vorbedingung wird jedes Element getestet. Falls ein Vektor drei Vorbedingungen nicht erfüllt, kann man sagen, die Addition zwischen den Vektoren kann nicht ausgeführt werden.

3.2.2 Ordnung und Zusammenfassung der Vorbedingung

3.2.2.1 Einfache Vorbedingungen

Wenn also $\|a\|_\infty + \|b\|_\infty \le Max\_\mathrm{Re}al$ und Max_Real als der darstellbare Grenzwert bereit ist, dann kann jede Addition $a_n + b_n$ ausgeführt werden.

3.2.2.2 Mittlere Vorbedingungen

Wenn also für $|a_n| \ge \frac{1}{2} \times Max\_\mathrm{Re}al$ oder $|b_n| \ge \frac{1}{2} \times Max\_\mathrm{Re}al$, gibt es alle $|a_n| + |b_n| \le Max\_\mathrm{Re}al$, dann kann jede Addition $a_n + b_n$ ausgeführt werden.

3.2.2.3 Komplexe Vorbedingungen

Wenn also für jeder n=1,2,3,..N, gibt es

$|a_n| + |b_n| \le Max\_\mathrm{Re}al$ ($|a_n| \le Max\_\mathrm{Re}al$ und $|b_n| \le Max\_\mathrm{Re}al$)

dann kann jede Addition $a_n + b_n$ ausgeführt werden.

3.2.2.4 Nachbedingungen

a.) $c_n = a_n + b_n$, für alle n=1,2,…N.

b.) $c_n \le Max\_\mathrm{Re}al$, für alle n=1,2,…N.

3.2.3 Nachweis von Vor- und Nachbedingungen

Gegeben: Vektor $\vec{a}$, $\vec{b}$ und $\vec{c}$ mit N Elemente

$\vec{a} = (a_1, a_2, ..., a_n)^T$;

$\vec{b} = (b_1, b_2, ..., b_n)^T$;

$\vec{c} = (c_1, c_2, ..., c_n)^T$.

Definition:

Gleichung 3.1

a.) Maximaler Absolutwert: $\|\vec{x}\|_\infty = \max\|x_n\|, n = 1, 2, ..., N$ (Gl. 3.1)

b.) Kontrapositive: Die Kontrapositive der Aussage ‚Wenn p, dann q' ist ‚Wenn –q, dann -p'. Sogenannt die Kontrapositive der ‚p→q' ist ‚(~q)→(~p)'.

c.) Eine Aussage ist mit seiner Kontrapositiven gleichwertig.

3.2.3.1 Einfache Vorbedingungen

Nach der Definition, folgt es $\|a\|_\infty \ge \|a_n\|$ ‚für alle $n = 1, 2, ..., N$ und $\|b\|_\infty \ge \|b_n\|$ ‚für alle $n = 1, 2, ..., N$.

Damit ist: $\|a\|_\infty + \|b\|_\infty \ge \|a_n\| + \|b_n\|$, für alle $n = 1, 2, ..., N$.

Wenn also $\|a\|_\infty + \|b\|_\infty \le Max\_\mathrm{Re}al$ l und Max_Real als der darstellbare Grenzwert bereit ist, dann kann jede Addition $a_n + b_n$ ausgeführt werden. Sonst geht man zu den mittleren und komplexen Vorbedingungen.

Wörtliche Erklärung:

Die einfache Voraussetzung ist eine Beurteilung, ob die Addition zwischen den beiden ‚MaxNorm' von beiden Vektoren, der so genannte maximale Wert, ausgeführt werden können. Weil die beiden ‚MaxNorm' größer als die anderen Werte in beiden Vektoren sind. Das Ergebnis der Addition zwischen den beiden ‚MaxNorm' ist auch größer als die anderen beiden Werte. Wenn die Ergebnisse der Addition zwischen beiden ‚MaxNorm' kleiner als der

Grenzwert sind, kann man folgern, dass die Ergebnisse der Additionen der anderen Elements bestimmt kleiner als der Grenzwert sind.

Aber wenn die Ergebnisse der Addition zwischen beiden ‚MaxNorm' größer als der Grenzwert sind, müssen noch ausführliche Beurteilungen gemacht werden. Weil die beiden ‚MaxNorm' vielleicht nicht im gleichen Element liegen. Folgende Hypothesen sollen aufgestellt werden, dass a_k und b_h die ‚MaxNorm' von beiden Vektoren sind, dass die Ergebnisse der Addition zwischen a_k und b_h größer als der Grenzwert ist, aber a_k und a_h und b_k und b_h erfüllen: $a_k + b_k < X_{\max}$ und $a_h + b_h < X_{\max}$. Deshalb geht man zu den mittleren Vorbedingungen, um eine ausführliche Beurteilung zu machen, ob das Ergebnis der Addition von dem anderen Element in Ordnung ist.

3.2.3.2 Mittlere Vorbedingungen

Nach der Definition folgt:

Wenn $|a_n| \le \frac{1}{2} \times Max\_\mathrm{Re}al$ und $|b_n| \le \frac{1}{2} \times Max\_\mathrm{Re}al$, dann $|a_n| + |b_n| \le Max\_\mathrm{Re}al$.

Die Aussage ist gleichwertig mit seinen Kontrapositiven:

Wenn $|a_n| + |b_n| > Max\_\mathrm{Re}al$, dann $|a_n| > \frac{1}{2} \times Max\_\mathrm{Re}al$ oder $|b_n| > \frac{1}{2} \times Max\_\mathrm{Re}al$.

Damit ist:

Für $|a_n| \le \frac{1}{2} \times Max\_\mathrm{Re}al$ und $|b_n| \le \frac{1}{2} \times Max\_\mathrm{Re}al$, gibt es bestimmt $|a_n| + |b_n| \le Max\_\mathrm{Re}al$.

Für $||a_n| > \frac{1}{2} \times Max\_\mathrm{Re}al$ oder $|b_n| > \frac{1}{2} \times Max\_\mathrm{Re}al$, gibt es möglich $|a_n| + |b_n| > Max\_\mathrm{Re}al$

.

Wenn also für $|a_n| > \frac{1}{2} \times Max\_\mathrm{Re}al$ oder $|b_n| > \frac{1}{2} \times Max\_\mathrm{Re}al$, gibt es alle $|a_n| + |b_n| \le Max\_\mathrm{Re}al$, dann kann jede Addition $a_n + b_n$ ausgeführt werden.

Wörtliche Erklärung:

Es gibt eine Aussage: Wenn a_n und b_n kleiner als die Hälfte des Grenzwertes sind, ist das Ergebnis der Addition zwischen a_n und b_n auch kleiner als der Grenzwert. Die Wahrheit ist echt.

In der Mathematik gibt es ein Theorem, sind eine Aussage und seine Kontrapositive gleichwertig. Deshalb ist die Kontrapositive der obigen Aussage echt: Wenn das Ergebnis der Addition zwischen a_n und b_n größer als der Grenzwert ist, existiert, $a_n > \frac{1}{2} X_{\max}$ oder

$b_n > \frac{1}{2} X_{\max}$ oder $a_n > \frac{1}{2} X_{\max}$, $b_n > \frac{1}{2} X_{\max}$. Durch die obigen Schlüsse wird die mittlere Vorbedingung gegeben. Man beurteilt die Ergebnisse der Additionen des Elements, dessen Wert größer als die Hälfte des Grenzwertes ist, ob die Ergebnisse in Ordnung sind. Wenn die mittleren Vorbedingungen nicht erfüllt sind, geht es zur komplexen Vorbedingung.

3.3 Subtraktion

Result = Left - Right

(Man benutzt das Alphabet, um die Formel einfach darzustellen.)

$\vec{a}$:bedeutet Vektor ‚Left'.

$\vec{b}$:bedeutet Vektor ‚Right.

$\vec{c}$:bedeutet Vektor ‚Result'.

3.3.1 Analysierung und Finden der Vorbedingung

Die Subtraktion ist ähnlich wie die Addition. Eine einfache Vorbedingung kann nicht erfüllt werden. Nun werden auch manche Vorbedingungen und Nachbedingungen für den Test entworfen. Die Methode des Entwurfs ist ähnlich wie die Addition.

Gemäß den Forderungen des Hinweises sollte erst eine einfache Vorbedingung entworfen werden. In beiden Vektoren ergibt der größte Vektor minus den minimalen Vektor das größte Ergebnis. Und der minimale Vektor minus größter Vektor ergibt das minimale Ergebnis. Die beiden Ergebnisse sind die repräsentativen Werte. Wenn die beiden, Subtraktion und Ergebnisse, in Ordnung sind, kann man einfach beweisen, dass die Subtraktionen zwischen anderen Elementen auch in Ordnung sind. Falls die beiden Ergebnisse den Rahmen des Grenzwertes sprengt, sollte man noch weiter beurteilen, ob die Subtraktion zwischen beiden Vektoren ausgeführt werden kann, weil die Situation gleich die der Addition ist. Der größte Vektor und der minimale Vektor liegen nicht im gleichen Element. Man kann nicht direkt die Subtraktion ausführen. Dann wählt man auch die Elemente aus, deren absolute Werte größer als die Hälfte der Grenzwerte sind, um sie zu testen. Die Subtraktion zwischen beiden Werten, die kleiner als die Hälfte der Grenzwerte sind, kann nicht ein den Rahmen gesprengtes Ergebnis ergeben. Wenn beide Voraussetzungen noch nicht erfüllt wurden, muss man den Test wie bei der Addition für jedes Element machen, bzw. die komplexe Vorbedingung. Falls ein Vektor drei Vorbedingungen nicht erfüllt, kann man sagen, die Subtraktion zwischen den Vektoren kann nicht ausgeführt werden.

3.3.2 Ordnung und Zusammenfassung der Vorbedingung

3.3.2.1 Einfache Vorbedingungen

Wenn also $\|a\|_\infty - b_{\min} \le Max\_\mathrm{Re}al$ und $\left|a_{\min} - \|b\|_\infty\right| \le Max\_\mathrm{Re}al$ und

Max_Real als der darstellbare Grenzwert bereit ist, dann kann jede Subtraktion $a_n - b_n$ ausgeführt werden.

3.3.2.2 Mittlere Vorbedingungen

Für $|a_n| \le \frac{1}{2} \times Max\_\mathrm{Re}al$ und $|b_n| \le \frac{1}{2} \times Max\_\mathrm{Re}al$, gibt es bestimmt $|a_n - b_n| \le Max\_\mathrm{Re}al$.

Für $|a_n| > \frac{1}{2} \times Max\_\mathrm{Re}al$ oder $|b_n| > \frac{1}{2} \times Max\_\mathrm{Re}al$, gibt es möglich $|a_n - b_n| > Max\_\mathrm{Re}al$.

Wenn also für $|a_n| > \frac{1}{2} \times Max\_\mathrm{Re}al$ oder $|b_n| > \frac{1}{2} \times Max\_\mathrm{Re}al$,

gibt es alle $|a_n - b_n| \le Max\_\mathrm{Re}al$, dann kann jede Subtraktion $a_n - b_n$ ausgeführt werden.

3.3.2.3 Komplexe Vorbedingungen

Wenn also für jede n=1,2,3,..N, gibt es

$|a_n - b_n| \le Max\_\mathrm{Re}al$ ($|a_n| \le Max\_\mathrm{Re}al$ und $|b_n| \le Max\_\mathrm{Re}al$)

dann kann jede Subtraktion $a_n - b_n$ ausgeführt werden.

3.3.2.4 Nachbedingungen

a.) $c_n = a_n - b_n$, für alle n=1,2,…N.

b.) $|c_n| \le Max\_\mathrm{Re}al$, für alle n=1,2,…N.

3.3.3 Nachweis von Vor- und Nachbedingungen

Gegeben: Vektor $\vec{a}$, $\vec{b}$ und $\vec{c}$ mit N Elemente

$\vec{a} = (a_1, a_2, ..., a_n)^T$;

$\vec{b} = (b_1, b_2, ..., b_n)^T$;

$\vec{c} = (c_1, c_2, ..., c_n)^T$.

Definition:

Gleichung 3.1

a.) Maximaler Absolutwert: $\|\vec{x}\|_\infty = \max\|x_n\|$, $n = 1, 2, ..., N$ (Gl. 3.1)

Gleichung 3.2

b.) Minimalwert: $x_{\min} = \min\{x_1, x_2, ..., x_N\}$; $n = 1, 2, ..., N$ (Gl. 3.2)

c.) Kontrapositive: Die Kontrapositive der Aussage ‚Wenn p, dann q' ist ‚Wenn –q, dann -p'. Die so genannte Kontrapositive von ‚p→q' ist ‚(~q)→(~p)'.

d.) Eine Aussage ist mit seiner Kontrapositiven gleichwertig.

3.3.3.1 Einfache Vorbedingungen

Es folgt:

$\|a\|_\infty \geq a_n$, für alle $n = 1, 2, ..., N$; $\|b\|_\infty \geq b_n$, für alle $n = 1, 2, ..., N$;

$a_{\min} \leq a_n$, für alle $n = 1, 2, ..., N$; $b_{\min} \leq b_n$, für alle $n = 1, 2, ..., N$.

Damit ist:

$\|a\|_\infty - b_{\min} \geq a_n - b_n$, für alle $n = 1, 2, ..., N$; $a_{\min} - \|b\|_\infty \leq a_n - b_n$, für alle $n = 1, 2, ..., N$.

Wenn also $\|a\|_\infty - b_{\min} \leq Max\_\mathrm{Re}al$ und $\left|a_{\min} - \|b\|_\infty\right| \leq Max\_\mathrm{Re}al$ und

Max_Real als der darstellbare Grenzwert bereit ist, dann kann jede Subtraktion $a_n - b_n$ ausgeführt werden.

Wörtliche Erklärung:

Die erste Voraussetzung ist eine Beurteilung, ob die beiden Subtraktionen zwischen ‚Left.Maxwert' und ‚Right.Minimalwert' und zwischen Left.Minimalwert' und ‚Right.Maxwert' ausgeführt werden können.

Weil in jedem Vektor alle Elemente kleiner als der ‚Maxwert' und größer als der ‚Minimalwert' sind. In der Subtraktion gibt es zwei Möglichkeiten, den Grenzwert zu erstellen. Die Subtraktion zwischen ‚Left.Maxwert' und ‚Right.Minimalwert' bildet den größten Wert. Die Subtraktion zwischen ‚LefMinimalwert' und ‚Right.Maxwert' bildet den kleinsten Wert. Wenn das Ergebnis der Subtraktion zwischen ‚Left.Maxwert' und ‚Right.Minimalwert' kleiner als der größte Grenzwert ist, und das Ergebnis der Subtraktion zwischen ‚LeftMiniamalwert' und ‚Right.Maxwert' größer als der kleinste Grenzwert ist, kann man folgern, dass die Ergebnisse der Subtraktionen der anderen Elemente bestimmt in Ordnung sind.

Aber wenn das Ergebnis der Subtraktion zwischen ‚Left.Maxwert' und ‚Right.Minimalwert' größer als der größte Grenzwert oder das Ergebnis der Subtraktion zwischen ‚Left.Minimalwert' und ‚Right.Maxmalwert' kleiner als der kleinste Grenzwert ist, muss man noch ausführliche Beurteilungen machen. Weil sie vielleicht nicht im gleichen Element lie-

gen. Folgende Hypothesen werden aufgestellt, dass a_k ‚Left.Maxwert' ist und b_h ‚Right.Minimalwert' ist. Das Ergebnis der Subtraktion zwischen a_k und b_h ist größer als der größte Grenzwert, aber a_k und a_h und b_k und b_h erfüllen: $a_k - b_k < X_{\max}$ und $a_h - b_h < X_{\max}$. Deshalb geht man zu den mittleren und komplexen Vorbedingungen, um eine ausführliche Beurteilung zu machen und festzustellen, ob das Ergebnis der Subtraktion von dem anderen Element in Ordnung ist.

3.3.3.2 Mittlere Vorbedingungen

Es folgt:

Wenn $|a_n| \le \frac{1}{2} \times Max\_\mathrm{Re}al$ und $|b_n| \le \frac{1}{2} \times Max\_\mathrm{Re}al$,dann $|a_n - b_n| \le Max\_\mathrm{Re}al$.

Die Aussage ist gleichwertig mit seinen Kontrapositiven:

Wenn $|a_n - b_n| > Max\_\mathrm{Re}al$, dann $|a_n| > \frac{1}{2} \times Max\_\mathrm{Re}al$ oder $|b_n| > \frac{1}{2} \times Max\_\mathrm{Re}al$.

Damit ist:

Für $|a_n| \le \frac{1}{2} \times Max\_\mathrm{Re}al$ und $|b_n| \le \frac{1}{2} \times Max\_\mathrm{Re}al$, gibt es bestimmt $|a_n - b_n| \le Max\_\mathrm{Re}al$

Für $|a_n| > \frac{1}{2} \times Max\_\mathrm{Re}al$ oder $|b_n| > \frac{1}{2} \times Max\_\mathrm{Re}al$, gibt es möglich $|a_n - b_n| > Max\_\mathrm{Re}al$.

Wenn also für $|a_n| > \frac{1}{2} \times Max\_\mathrm{Re}al$ oder $|b_n| > \frac{1}{2} \times Max\_\mathrm{Re}al$,

gibt es alle $|a_n - b_n| \le Max\_\mathrm{Re}al$, dann kann jede Subtraktion $a_h - b_h$ ausgeführt werden.

Wörtliche Erklärung:

Es gibt auch eine Aussage: Wenn die absoluten Werte von a_n und b_n kleiner als die Hälfte des Grenzwertes sind, ist das Ergebnis der Subtraktion zwischen a_n und b_n auch kleiner als die Grenzwerte. Die Wahrheit ist echt.

Seine Aussage und seine Kontrapositive sind gleichwertig. Deshalb ist die Kontrapositive der obigen Aussage echt: Wenn das Ergebnis der Addition zwischen a_n und b_n größer als der Grenzwert ist, existiert, $|a_n| > \frac{1}{2} X_{\max}$ oder $|b_n| > \frac{1}{2} X_{\max}$ oder $|a_n| > \frac{1}{2} X_{\max}$, $|b_n| > \frac{1}{2} X_{\max}$.

Durch die obigen Schlüsse erhält man die mittlere Vorbedingung. Man beurteilt die Ergebnisse der Subtraktion des Elements, dessen absoluter Wert größer als die Hälfte des Grenzwertes ist, ob diese in Ordnung sind. Wenn sie die mittleren Vorbedingungen nicht erfüllen, geht man zur komplexen Vorbedingung.

3.4 Scalar_Produkt

Result = <Left • Right>

(Man benutzt das Alphabet, um die Formeln einfach darzustellen)

$\vec{a}$:bedeutet Vektor ‚Left';

$\vec{b}$:bedeutet Vektor ‚Right';

$\vec{c}$:bedeutet Real ‚Result'.

Scalar_Produkt ist eine komplizierte Vektoroperation, sie ist der schwerste Teil in dieser Arbeit, weil die Vektoroperation aus zwei Teilen besteht, dem Teil der Multiplikation und dem Teil der Addition. Die Vorbedingungen und Nachbedingungen müssen die beiden Anteile beinhalten.

3.4.1 Analysierung und Finden der Vorbedingung

Hier muss eine wichtige Sache beachtet werden. Man kann direkt die mathematische Formel von Scalar_Produkt benutzen. Aber dadurch können vielleicht einige Fehler auftreten. Weil das Ergebnis der Addition zwischen einem kleinen Wert und einem größeren Wert ein korrekter Wert sein kann. Und der kleinere Wert und der größere Wert vielleicht nicht in Ordnung sind. So muss man das Produkt von jedem Element beurteilen. Erst muss die einfache Vorbedingung entworfen werden. Man wählt zwei repräsentative Werte: die beiden maximalen Werte von beiden Vektoren. Wenn das Produkt von beiden Vektoren in Ordnung ist, kann das Produkt von dem anderen Element auch korrekt sein. Nach der Beurteilung des Produkts muss man die Addition aller Elemente beachten. Eine Methode ist die Akkumulation von dem ersten Element bis zum letzten Element. Aber das ist nicht sehr einfach, dass jedes Element implementiert werden muss.

Dann benutzt man eine mathematische Formel:

Gleichung 3.3

$$\left|\langle a,b\rangle\right| \le \|a\|\|b\| = \sqrt{\left(a_1^{\,2}+a_2^{\,2}+\ldots+a_n^{\,2}\right)\left(b_1^{\,2}+b_2^{\,2}+\ldots+b_n^{\,2}\right)} \le \sqrt{n^2\left(\|a\|_\infty \|b\|_\infty\right)^2} \qquad \text{(Gl. 3.3)}$$

Die Vektoroperation Scalar_Produkt ist gleich oder kleiner als das Produkt von beiden euklidischen Normen. Das Produkt von beiden euklidischen Normen ist auch kleiner als ein Ergebnis, das aus Maximalnorm von beiden Vektoren besteht. Dann bekommt man direkt das Ergebnis. Wenn das Ergebnis oder das Produkt von beiden Maximalwerten nicht in Ordnung sind, muss noch weiter beurteilt werden, ob das Scalar_Produkt zwischen beiden Vektoren

ausgeführt werden kann, weil die Situation gleich mit der der Addition ist. Die beiden größten Vektoren liegen vielleicht nicht im gleichen Element. Dann ist der Test nur für das Produkt von den beiden größten Vektoren nicht ausreichend. Man muss andere repräsentative Werte und Elemente finden. Die Quadratwurzel des Grenzwertes ist eine gute Entscheidung.

Wenn beide Elemente mit diesen Werten kleiner als die Quadratwurzel der Grenzwerte sind, ist das Produkt von beiden Elementen bestimmt kleiner als der Grenzwert. Dann wählt man die Elemente, deren absolute Werte größer als die Quadratwurzel der Grenzwerte sind, um sie zu testen. Wenn die Produkte dieser Elemente in Ordnung sind, testet man die Addition von allen Produkten. Hier verwendet man folgende mathematische Formel:

Gleichung 3.4

$$\langle a,b\rangle = \frac{1}{4}\left(\|a+b\|^2 - \|a-b\|^2\right) = \frac{1}{4}\left(\sum_{n=1}^{N}|a_n+b_n|^2 - \sum_{n=1}^{N}|a_n-b_n|^2\right) \le \frac{1}{4}\left[\left(\|a\|_\infty + \|b\|_\infty\right)^2 \times N - 0\right]$$

(Gl. 3.4)

Wenn das Ergebnis von ganz rechts der Formel kleiner als der Grenzwert ist, bedeutet es, dass das Scalar_Produkt in Ordnung ist. Falls die beiden Voraussetzungen noch nicht erfüllt sind, muss die Operation wie die bei der Addition für jedes Element gemacht werden. Das ist die komplexe Vorbedingung. Die Multiplikation wird zwischen allen Elementen ausgeführt. Und die Addition wird zwischen allen Produkten gemacht. Falls beide Vektoren drei Vorbedingungen nicht erfüllen, kann man sagen, das Scalar_Produkt zwischen den Vektoren kann nicht ausgeführt werden.

3.4.2 Ordnung und Zusammenfassung der Vorbedingung

3.4.2.1 Einfache Vorbedingungen

Wenn a.) $\|a\|_\infty \times \|b\|_\infty \le Max\_\mathrm{Re}al$, und Max_Real als der darstellbare Grenzwert bereit ist,

dann kann jeder Scale $a_n \times b_n$ ausgeführt werden.

b.) $\sqrt{n^2\left(\|a\|_\infty \|b\|_\infty\right)^2} \le Max\_\mathrm{Re}al$

dann kann das Scalar_Produkt $|\langle a,b\rangle|$ ausgeführt werden.

3.4.2.2 Mittlere Vorbedingungen

Wenn also

a.) für $|a_n| > \sqrt{Max\_\mathrm{Re}al}$ oder $|b_n| > \sqrt{Max\_\mathrm{Re}al}$, gibt es alle $|a_n| \times |b_n| \le Max\_\mathrm{Re}al$,

dann kann jeder Scale $a_n \times b_n$ ausgeführt werden.

b.) Wenn $\frac{1}{4}\left[\left(a_{\max}+b_{\max}\right)^2 \times N - 0\right] \le Max\_\mathrm{Re}al$,

dann kann das Scalar_Produkt $\left|\langle a,b\rangle\right|$ ausgeführt werden.

3.4.2.3 Komplexe Vorbedingungen

a.) für jede $n=1,2,...,N$, $\left|a_n\right| \times \left|b_n\right| \le Max\_\mathrm{Re}al$,

dann kann jeder Scale $a_n \times b_n$ ausgeführt werden.

b.) $a_1 \times b_1 + a_2 \times b_2 + ... + a_N \times b_N = \sum_{n=1}^{N}\left(a_n \times b_n\right) \le Max\_\mathrm{Re}al$

dann kann das Scalar_Produkt $\left|\langle a,b\rangle\right|$ ausgeführt werden.

3.4.2.4 Nachbedingungen

a.) $c = a_1b_1 + a_2b_2 + ... + a_Nb_N = \sum_{n=1}^{N}\left(a_n \times b_n\right)$, für alle $\left|c_n\right| \le Max\_\mathrm{Re}al$.

b.) $\left|c_n\right| \le Max\_\mathrm{Re}al$, für alle $\left|c_n\right| \le Max\_\mathrm{Re}al$.

3.4.3 Nachweis von Vor- und Nachbedingungen

Gegeben: Vektor ***a, b*** mit N Elementen

$\vec{a} = \left(a_1, a_2, ..., a_n\right)^T$;

$\vec{b} = \left(b_1, b_2, ..., b_n\right)^T$.

Definition:

a.) Maximaler Absolutwert: $\left\|\vec{x}\right\|_\infty = \max\left\|x_n\right\|$, $n=1,2,...,N$

b.) $x_{\max}$: der Maximalwert von Verktor

c.) Ist $\langle\cdot,\cdot\rangle$ ein Skalarprodukt auf N und $\langle x,y\rangle \le \|x\|\|y\|$

Gleichung 3.5

d.) $\|x\| = \left\{\sum_{n=1}^{m}\left|\delta_n\right|^2\right\}^{1/2}$ (Gl. 3.5)

e.) Kontrapositive: Die Kontrapositive der Aussage ‚Wenn p, dann q' ist ‚Wenn –q, dann –p'. So genannte Kontrapositive von ‚p→q' ist ‚(~q)→(~p)'.

f.). Eine Aussage ist mit seiner Kontrapositiven gleichwertig.

Gleichung 3.6

g.) $\langle x, y\rangle = \frac{1}{4}\left(\|x+y\|^2 - \|x-y\|^2\right)$ (Gl. 3.6)

3.4.3.1 Einfache Vorbedingungen

Es folgt:

$\|a\|_\infty \geq \|a_n\|$, für alle $n = 1, 2, ..., N$.

$\|b\|_\infty \geq \|b_n\|$ |, für alle $n = 1, 2, ..., N$.

$\|a\| = \left\{\sum_{n=1}^{m} |a_n|^2\right\}^{1/2}$

$\|b\| = \left\{\sum_{n=1}^{m} |b_n|^2\right\}^{1/2}$

$\langle a, b\rangle \leq \|a\|\|b\|$

Damit ist:

a.) $\|a\|_\infty \times \|b\|_\infty \geq \|a_n\| \times \|b_n\|$, für alle $n = 1, 2, ..., N$.

b.) $|\langle a, b\rangle| \leq \|a\|\|b\| = \sqrt{\left(a_1^2 + a_2^2 + ... + a_n^2\right)\left(b_1^2 + b_2^2 + ... + b_n^2\right)} \leq \sqrt{n^2\left(\|a\|_\infty \|b\|_\infty\right)^2}$

Wenn a.) $\|a\|_\infty \times \|b\|_\infty \leq Max\_\mathrm{Re}al$, und Max_Real als der darstellbare Grenzwert bereit ist,

dann kann jeder Scale $a_n \times b_n$ ausgeführt werden.

b.) $\sqrt{n^2\left(\|a\|_\infty \|b\|_\infty\right)^2} \leq Max\_\mathrm{Re}al$

dann kann das Scalar_Produkt $|\langle a, b\rangle|$ ausgeführt werden.

Wörtliche Erklärung:

Die einfachen Vorbedingungen haben zwei Teile. Die erste Voraussetzung ist eine Beurteilung für den Additionsteil, ob die Multiplikation zwischen beiden ‚MaxNorm' von beiden Vektoren, dem so genannten maximalen Wert, ausgeführt werden können. Weil alle Elemente kleiner als ‚MaxNorm' sind, wenn das Ergebnis der Multiplikation zwischen beiden ‚MaxNorm' kleiner als der Grenzwert ist, kann man folgern, dass die Ergebnisse der Multiplikationen der anderen Elemente bestimmt kleiner als der Grenzwert sind. Nach dem Anteil der Multiplikation macht man nun die Beurteilung des Anteils der Addition. Gemäß der Beziehung

$|\langle a, b\rangle| \leq \|a\|\|b\| = \sqrt{\left(a_1^2 + a_2^2 + ... + a_n^2\right)\left(b_1^2 + b_2^2 + ... + b_n^2\right)} \leq \sqrt{n^2\left(\|a\|_\infty \|b\|_\infty\right)^2}$ (Gl. 3.3)

weiß man, wenn $\sqrt{n^2\left(\|a\|_\infty \|b\|_\infty\right)^2} \le Max\_\mathrm{Re}al$, ist das Ergebnis des Skalar_Produkts auch kleiner als der Grenzwert.

Aber wenn das Ergebnis der Multiplikation zwischen beiden ‚MaxNorm' größer als der Grenzwert ist, muss noch eine ausführliche Beurteilung gemacht werden. Weil die beiden ‚MaxNorm' vielleicht nicht im gleichen Element liegen. Es wird folgende Hypothese aufgestellt, dass a_k und b_h die ‚MaxNorm' von beiden Vektoren sind, das die Ergebnisse der Multiplikation zwischen a_k und b_h ist größer als der Grenzwert sind, aber a_k und a_h und b_k und b_h erfüllen: $a_k \times b_k \le Max\_\mathrm{Re}al$ und $a_h \times b_h \le Max\_\mathrm{Re}al$ x. Deshalb geht man zur mittleren und komplexen Vorbedingung, um eine ausführliche Beurteilung zu machen, ob das Ergebnis der Multiplikation von dem anderen Element in Ordnung ist.

3.4.3.2 Mittlere Vorbedingungen

Es folgt:

a.) $\|a\|_\infty \ge \|a_n\|$ ||a||∞ >||an||, für alle $n = 1, 2, ..., N$.

$\|b\|_\infty \ge \|b_n\|$, für alle $n = 1, 2, ..., N$.

b.) Wenn $|a_n| \le \sqrt{Max\_\mathrm{Re}al}$ und $|b_n| \le \sqrt{Max\_\mathrm{Re}al}$, dann $|a_n| \times |b_n| \le Max\_\mathrm{Re}al$.

Die Aussage ist gleichwertig mit seinen Kontrapositiven:

Wenn $|a_n| \times |b_n| > Max\_\mathrm{Re}al$, dann $|a_n| > \sqrt{Max\_\mathrm{Re}al}$ oder $|b_n| > \sqrt{Max\_\mathrm{Re}al}$.

Damit ist:

a.) Für $|a_n| \le \sqrt{Max\_\mathrm{Re}al}$ und $|b_n| \le \sqrt{Max\_\mathrm{Re}al}$, gibt es bestimmt $|a_n| \times |b_n| \le Max\_\mathrm{Re}al$

Für $|a_n| > \sqrt{Max\_\mathrm{Re}al}$ oder $|b_n| > \sqrt{Max\_\mathrm{Re}al}$, gibt es möglich $||a_n| \times |b_n| > Max\_\mathrm{Re}al$.

b.)

$$\langle a,b\rangle = \frac{1}{4}\left(\|a+b\|^2 - \|a-b\|^2\right) = \frac{1}{4}\left(\sum_{n=1}^{N}|a_n+b_n|^2 - \sum_{n=1}^{N}|a_n-b_n|^2\right) \le \frac{1}{4}\left[\left(a_{\max}+b_{\max}\right)^2 \times N - 0\right]$$

Wenn also

a.) für $|a_n| > \sqrt{Max\_\mathrm{Re}al}$ oder $|b_n| > \sqrt{Max\_\mathrm{Re}al}$, gibt es alle $|a_n| \times |b_n| \le Max\_\mathrm{Re}al$,

dann kann jeder Scale $a_n \times b_n$ ausgeführt werden.

b.) Wenn $\frac{1}{4}\left[\left(a_{\max}+b_{\max}\right)^2 \times N - 0\right] \le Max\_\mathrm{Re}al$,

dann kann das Scalar_Produkt $|\langle a,b\rangle|$ ausgeführt werden.

Wörtliche Erklärung:

Wenn a_n und b_n kleiner als die Quadratwurzel von dem Grenzwert sind, ist das Ergebnis der Multiplikation zwischen a_n und b_n auch kleiner als der Grenzwert. Es wird das gleiche Theorem benutzt wie bei der Addition, eine Aussage und seine Kontrapositive sind gleichwertig. Deshalb, wenn das Ergebnis der Multiplikation zwischen a_n und b_n größer als der Grenzwert ist, existiert, $a_n > X_{\max}^{1/2}$ oder $b_n > X_{\max}^{1/2}$ oder $a_n > X_{\max}^{1/2}$, $b_n > X_{\max}^{1/2}$. Durch die obigen Schlüsse werden die mittleren Vorbedingungen gegeben. Nun beurteilt man die Ergebnisse der Multiplikation des Elements, dessen Wert größer als die Quadratwurzel des Grenzweres ist, ob die Ergebnisse in Ordnung sind. Danach, gemäß der Beziehung:

$$\langle a,b\rangle = \frac{1}{4}\left(\|a+b\|^2 - \|a-b\|^2\right) = \frac{1}{4}\left(\sum_{n=1}^{N}|a_n+b_n|^2 - \sum_{n=1}^{N}|a_n-b_n|^2\right) \le \frac{1}{4}\left[\left(a_{\max}+b_{\max}\right)^2 \times N - 0\right]$$

(Gl. 3.4)

Wenn $\frac{1}{4}\left[\left(a_{\max}+b_{\max}\right)^2 \times N - 0\right] \le Max\_\mathrm{Re}al$, dann ist das Ergebnis des Skalar_Produkts in Ordnung. Wenn die mittleren Vorbedingungen nicht erfüllt wurden, werden die komplexen Vorbedingungen gemacht.

4 Test der Vor- und Nachbedingungen

Nach dem Entwurf sollte die Methode beurteilt werden, ob sie erfolgreich verlaufen ist. Deshalb wird in diesem Abschnitt erklärt, wie die Vor- und Nachbedingungen, die im dritten Teil entworfen werden, getestet werden.

4.1 Analysierung der Vor- und Nachbedingungen

In diesem Abschnitt wird der Zusammenhang zwischen Zuständen und Vektoroperationen analysiert. Es gibt vier Funktionen und Prozeduren mit Vor- und Nachbedingungen, die getestet werden müssen: ‚Add', ‚Subtract', ‚Scale', ‚Scalar_Product'.

a.) Add mit Vor- und Nachbedingungen, nimmt Vektoren ‚Left' und ‚Right' und reserviert die Gesamtsumme im ‚Result'.
b.) Subtract mit Vor- und Nachbedingungen, nimmt Vectoren ‚Left' und ‚Right' und reserviert den Rest im ‚Result'.
c.) Scale mit Vor- und Nachbedingungen, nimmt Vector ‚Left' und Factor und reserviert das Produkt von ‚Left' und Factor im ‚Result'.
d.) Scale_Product mit Vor- und Nachbedingungen, nimmt Werte von ‚Left' und ‚Right' und rechnet das ‚Scalar_Product'. Wenn der Wert überläuft oder irgendein zwischen ‚Left' und ‚Right' Fehler habt, wird ‚Error_Status' ‚Overflow_Error ' ausgegeben.

Bei Vektoroperationen hat das Ergebnis zwei mögliche Zustände, ‚No_Error' und ‚Overflow_Error'.

No_Error: **Der Wert hat keine Fehler.**

Overflow_Error: **Der Wert der Vektoren läuft über.**

Durch die obige Analysierung kann Folgendes zusammengefasst werden: Wenn die Vektoroperationen beginnen, müssen' die Vektoren initialisiert werden. Nach dem Prozess ‚Initialize' ist der Zustand ‚No_Error'. Dann können die Vor- und Nachbedingungen der Vektoroperationen implementiert werden. Die Operationen der Vorbedingungen sollten nacheinander gemacht werden. Zum Beispiel wird bei der Vektoroperation ‚Add' nach der Initialisierung erst die einfache Vorbedingung gemacht. Es entstehen zwei mögliche Zustände. Die erste Möglichkeit ist, dass die Operation keine Fehler hat und die Resultate in Ordnung sind. Das bedeutet, man kann direkt ‚Add' ausführen. Manchmal ergibt sich eine andere Möglichkeit, dass das Ergebnis der Operation die einfache Vorbedingung ‚Overflow_Error' ist, bzw. die Werte des Resultats laufen über. Dann muss die mittlere Vorbedingung, schlimmer noch,

die komplexe Vorbedingung gemacht werden. Falls die Operation von allen Vorbedingungen falsch ist, kann die Vektoroperation nicht durchgeführt werden.

4.2 UML-diagramme von Vor- und Nachbedingungen

4.2.1 Zustandsdiagramme der Vektoroperationen

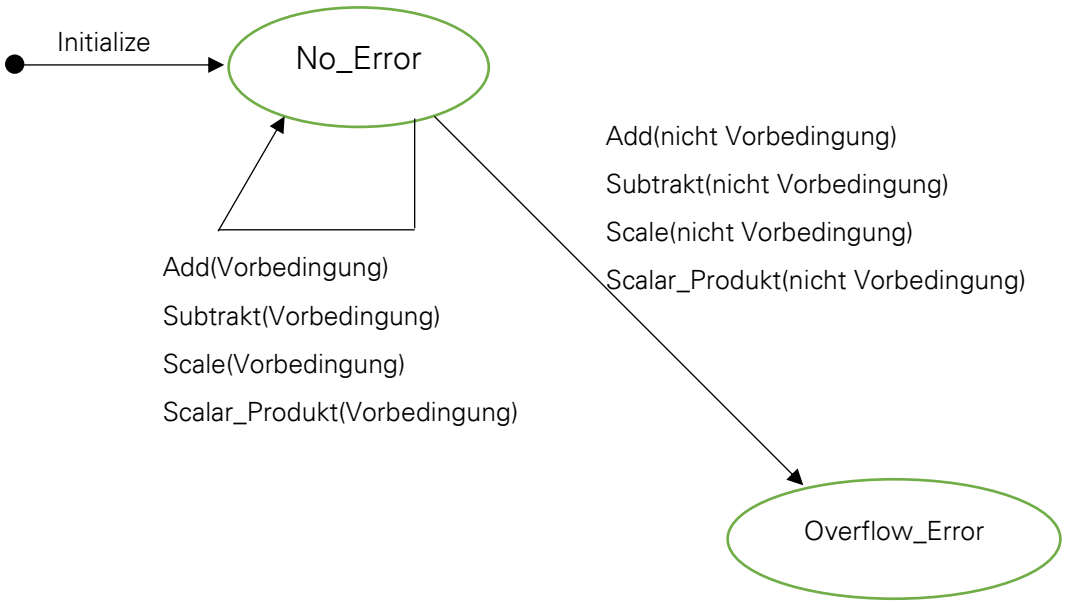

Abbildung 4. 1: Zustandsdiagramm

4.2.2 Aktivitätsdiagramme des Ablaufs der Vorbedingungen

Addition:

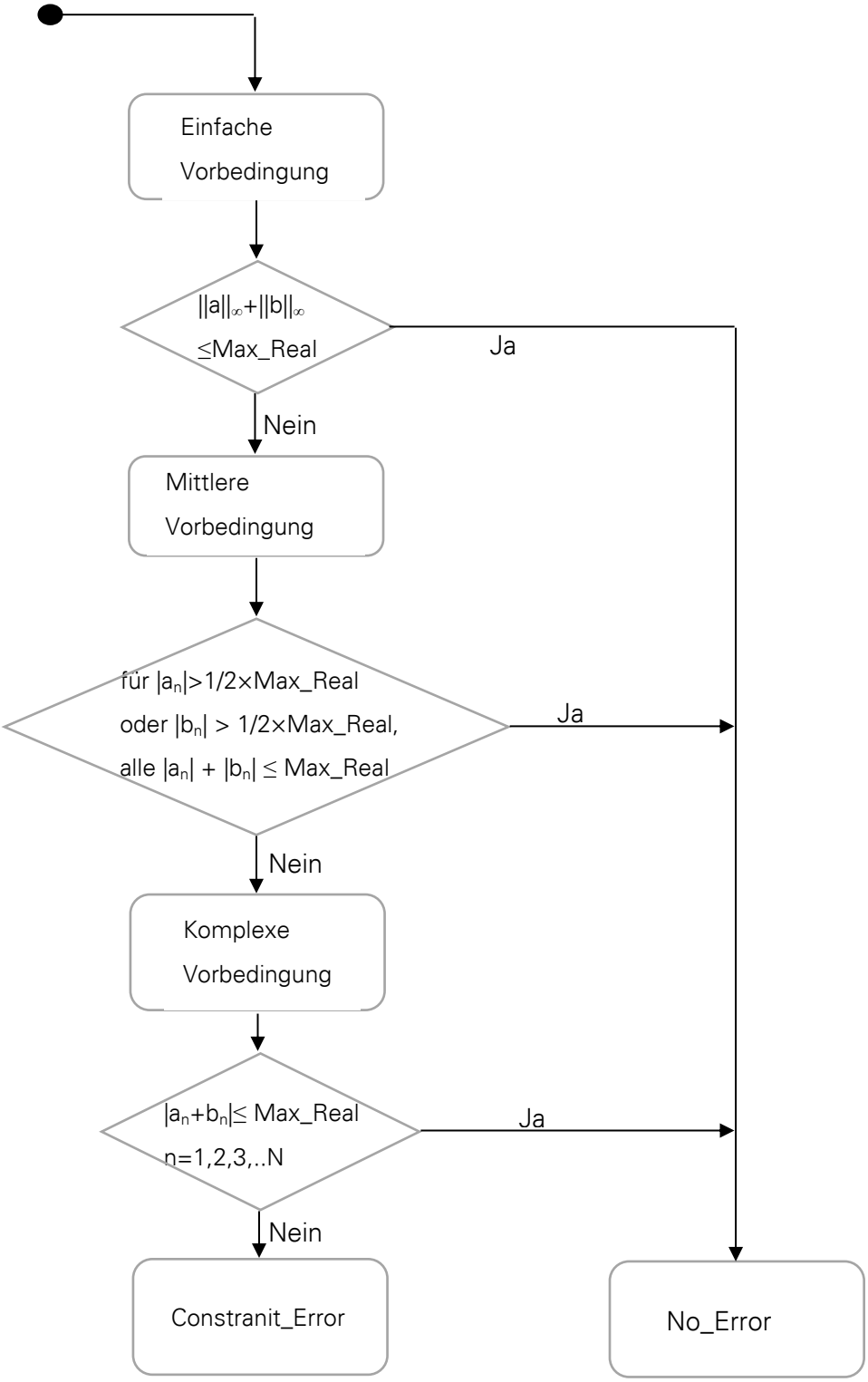

Abbildung 4. 2: Aktivitätsdiagramm der Addition

Subtraktion:

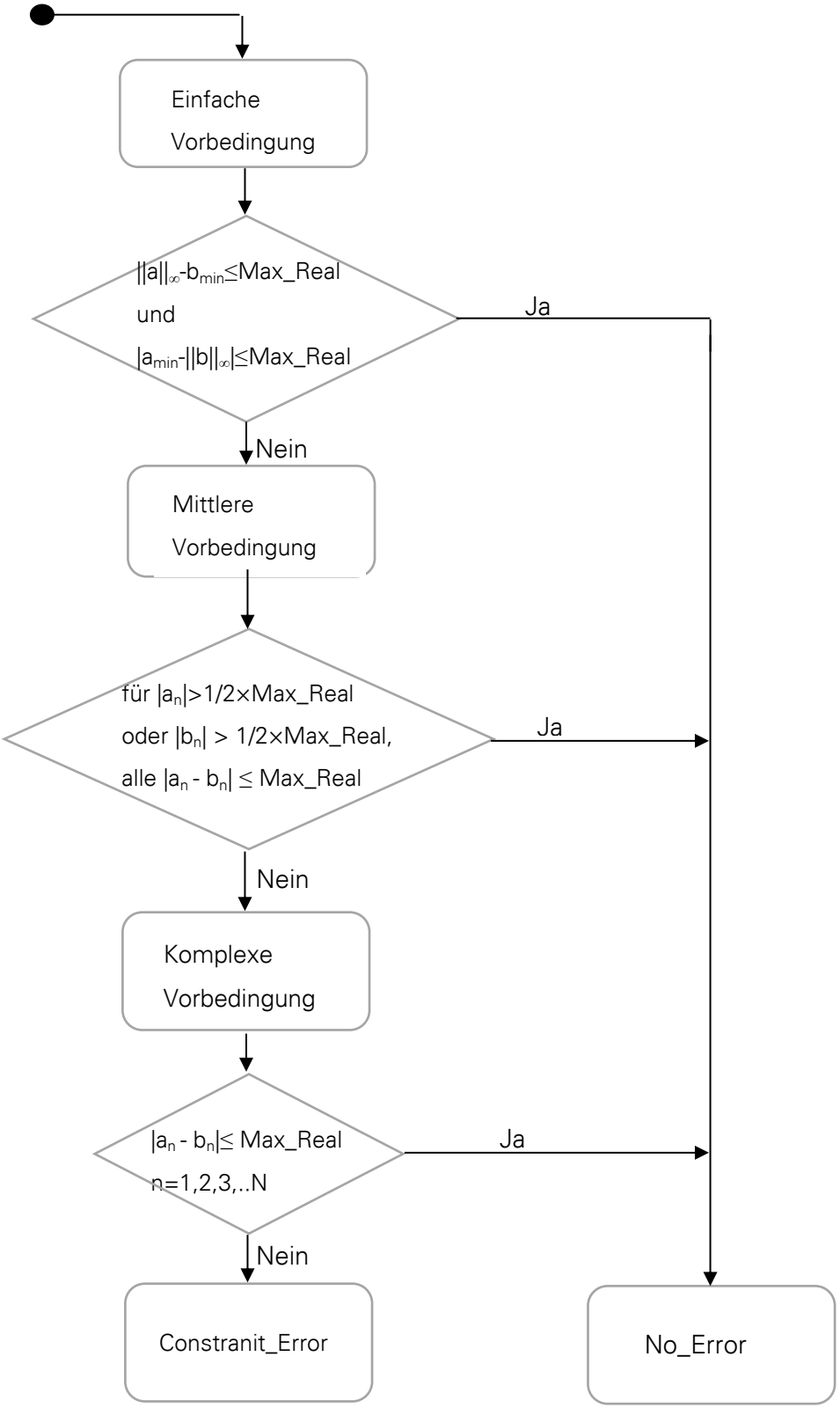

Abbildung 4. 3: Aktivitätsdiagramm der Subtraktion

Multiplikation:

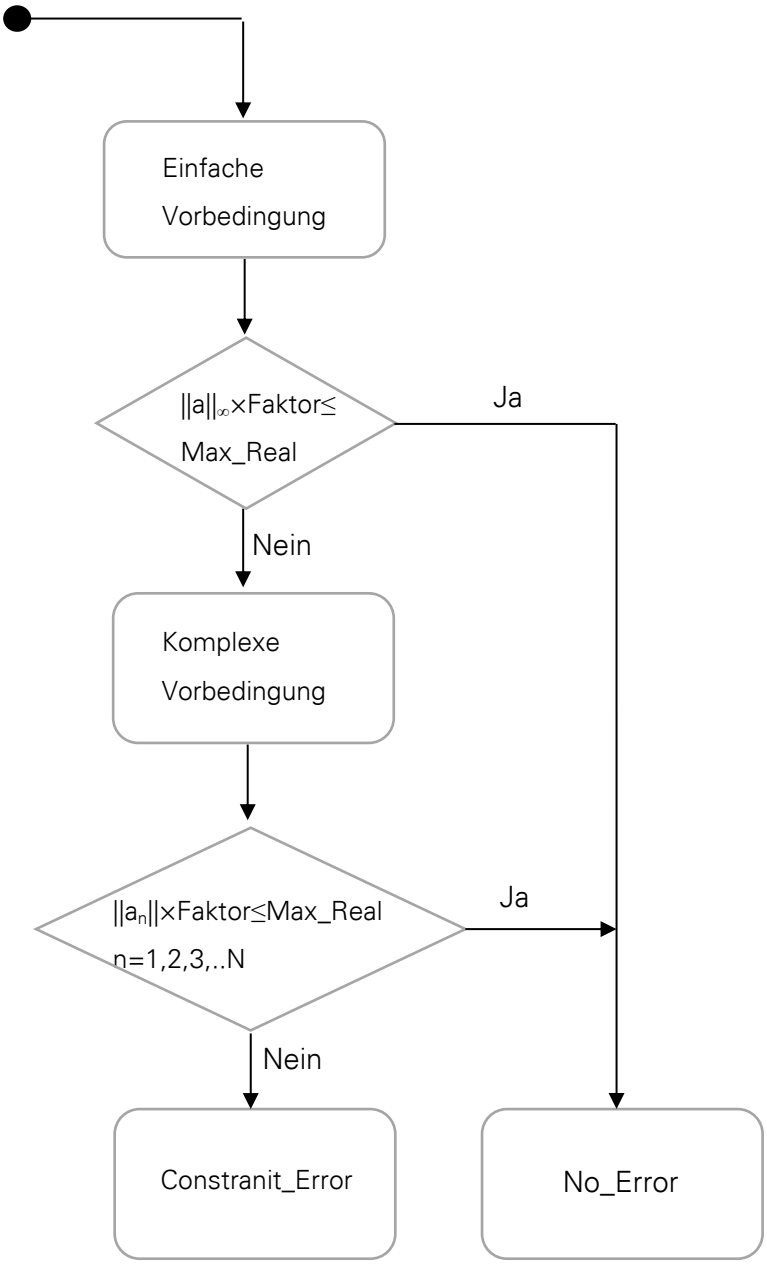

Abbildung 4. 4: Aktivitätsdiagramm der Multiplikation

Scalar_Produkt:

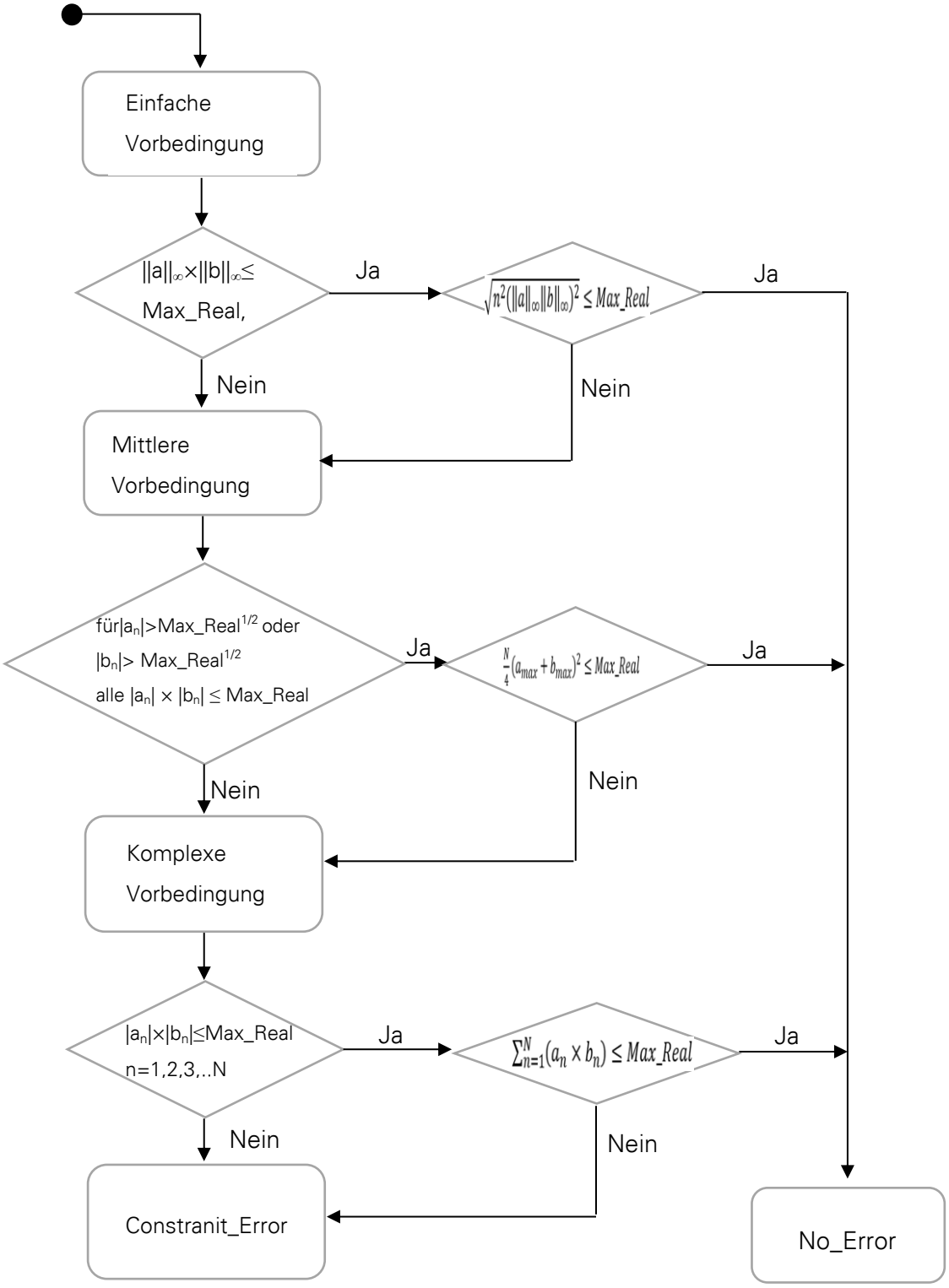

Abbildung 4. 5: Aktivitätsdiagramm von Scalar_Produkt

4.3 Test durch mathematisches Beispiel

In diesem Abschnitt sollen durch mathematische Beispiele die Vor- und Nachbedingungen getestet werden. Für jede Vektoroperation setzt man zuerst ein Beispiel, das die einfache Vorbedingung erfüllt. Wenn das Beispiel der einfachen Vorbedingung beim Programm bestanden hat, bedeutet es, dass der Test der einfachen Vorbedingung erfolgreich ist. Nach dem Test der einfachen Vorbedingung wird die mittlere Vorbedingung getestet. Man setzt das Beispiel, das die einfache Vorbedingung nicht erfüllen muss, aber es erfüllt die mittlere Vorbedingung. Weil nur die einfache Vorbedingung nicht bestanden hat, kann die mittlere Vorbedingung durchgeführt werden. Mit der gleichen Methode setzt man das Beispiel für den Test der komplexen Vorbedingung. Das Beispiel erfüllt die einfachen und mittleren Vorbedingungen nicht, zu gleicher Zeit erfüllt es die komplexe Vorbedingung. Zum Schluss setzt man auch ein Beispiel, dass die drei Vorbedingungen nicht erfüllt. Nach dem Ablauf der Vorbedingungen tritt ‚Found_Error' auf. Das bedeutet, die drei Vorbedingungen sind erfolgreich.

4.3.1 Addition

a.) Zuerst wird die einfache Vorbedingung getestet.

$$\begin{bmatrix} 0.1R \\ 0.2R \\ 0.4R \end{bmatrix} + \begin{bmatrix} 0.6R \\ 0.5R \\ 0.4R \end{bmatrix} = \begin{bmatrix} 0.7R \\ 0.7R \\ 0.8R \end{bmatrix}$$

Man setzt $||a||_\infty$=0.4R, $||b||_\infty$=0.6R, dann $||a||_\infty+||b||_\infty$=R, das Ergebnis ist in Ordnung, deshalb ist das Resultat des Vektors im Zustand ‚No_Error'. Der Test der einfachen Vorbedingung ist im Zustand ‚No_Error'.

b.)Als Nächstes soll die mittlere Vorbedingung getestet werden. Das Beispiel erfüllt die einfache Vorbedingung nicht, aber es erfüllt die mittlere Vorbedingung.

$$\begin{bmatrix} 0.1R \\ 0.2R \\ 0.5R \end{bmatrix} + \begin{bmatrix} 0.6R \\ 0.5R \\ 0.4R \end{bmatrix} = \begin{bmatrix} 0.7R \\ 0.7R \\ 0.9R \end{bmatrix}$$

Setzt man $||a||_\infty$=0.5R, $||b||_\infty$=0.6R, dann $||a||_\infty+||b||_\infty$=1.1R, ist das Ergebnis im Zustand ‚Overflow_Error'. Kommt man nun zum Test der mittleren Vorbedingung. Für jedes

$|a_n| > \frac{1}{2} Max\_\mathrm{Re}al$ oder $|b_n| > \frac{1}{2} Max\_\mathrm{Re}al$, ob gibt es alle $|a_n| + |b_n| \le Max\_\mathrm{Re}al$ gibt.

Für die Elemente, dessen Werte größer als die Hälfte der Grenzwerte sind, 0.1R+0.6R=0.7R und 0.5R+0.4R=0.9R, deshalb sind alle a_n + b_n in Ordnung. Der Test zeigt ‚No_Error'. Man braucht nicht die komplexe Vorbedingung zu benutzen.

c.) Dann soll die komplexe Vorbedingung getestet werden, deshalb setzt man ein Beispiel, das die einfache und mittlere Vorbedingung nicht erfüllt.

$$\begin{bmatrix} 0.1R \\ 0.2R \\ 0.7R \end{bmatrix} + \begin{bmatrix} 0.6R \\ 0.5R \\ 0.5R \end{bmatrix} = \begin{bmatrix} 0.7R \\ 0.7R \\ 1.2R \end{bmatrix}$$

Für die einfache Vorbedingung, gibt es 0.6R+0.7R=1.3R. Das Ergebnis ist der Zustand ‚Overflow_Error'. Danach wird die mittlere Vorbedingung getestet. Es gibt zwei Elemente, deren Werte größer als die Hälfte des Grenzwerts sind. 0.1R+0.6R=0.7R, das ist in Ordnung. Aber 0.7R+0.5R=1.2R, das ist zu groß, sodass man nun zum Test der komplexen Vorbedingung übergeht.

Nach dem Test der komplexen Vorbedingung wird der Zustand ‚Constranit_Error' gezeigt, weil es ein Ergebnis mit dem ‚Overflow_Error' Wert gibt.

d.) Zum Schluss soll ein Beispiel getestet werden, wo die drei Vorbedingungen nicht erfüllt werden.

$$\begin{bmatrix} 0.4R \\ 0.5R \\ 0.6R \end{bmatrix} + \begin{bmatrix} 0.9R \\ 0.5R \\ 0.8R \end{bmatrix} = \begin{bmatrix} 1.3R \\ 1.0R \\ 1.4R \end{bmatrix}$$

i.) $a_{\max} + b_{\max} = 1.5R > R$, die einfache Vorbedingung ist nicht erfüllt.

ii.) Für jedes $|a_n| > \frac{1}{2} Max\_\mathrm{Re}al$ oder $|b_n| > \frac{1}{2} Max\_\mathrm{Re}al$:

$$0.4R + 0.9R = 1.3R > R$$

$$0.6R + 0.8R = 1.4R > R$$

Die mittlere Vorbedingung ist auch nicht erfüllt.

iii.) Test für jedes Element:

$$0.4R + 0.9R = 1.3R > R$$

$$0.5R + 0.5R = 1.0R$$

$$0.6R + 0.8R = 1.4R > R$$

Die komplexe Vorbedingung ist auch nicht erfüllt.

Deshalb erfüllt das Beispiel diese drei Vorbedingungen nicht.

4.3.2 Subtraktion

a.) Zuerst soll die einfache Vorbedingungen getestet werden.

$$\begin{bmatrix} 0.9R \\ 0.8R \\ 0.2R \end{bmatrix} - \begin{bmatrix} 0.6R \\ 0.1R \\ 0.0R \end{bmatrix} = \begin{bmatrix} 0.3R \\ 0.7R \\ 0.2R \end{bmatrix}$$

Man setzt $\|a\|_\infty$=0.9R, a_{min}=0.2R, $\|b\|_\infty$=0.6R, b_{min}=0.0R. Dann $\|a\|_\infty$ - b_{min}= 0.9R - 0.0R = 0.9R, und a_{min} - $\|b\|_\infty$ = 0.2R - 0.6R = -0.4R, sind die beiden Ergebnisse in Ordnung, deshalb ist das Resultat in Ordnung. Der Test ist auch im Zustand ‚No_Error'.

b.) Danach gibt es zwei Beispiele, um die mittlere Vorbedingung zu testen.

I.)
$$\begin{bmatrix} 0.9R \\ 0.8R \\ 0.2R \end{bmatrix} - \begin{bmatrix} 0.6R \\ 0.1R \\ -0.3R \end{bmatrix} = \begin{bmatrix} 0.3R \\ 0.7R \\ 0.5R \end{bmatrix}$$

Man setzt $\|a\|_\infty$=0.9R, b_{min}=-0.3R, dann $\|a\|_\infty$- b_{min}=0.9R – (-0.3R)=1.2R, ist das Ergebnis im Zustand ‚Overflow_Error'. Deshalb erfüllen die beiden Vektoren die einfache Vorbedingung nicht. Nun zum Test der mittleren Vorbedingung.

II.)
$$\begin{bmatrix} 0.9R \\ 0.8R \\ -0.2R \end{bmatrix} - \begin{bmatrix} 0.9R \\ 0.1R \\ 0.1R \end{bmatrix} = \begin{bmatrix} 0.0R \\ 0.7R \\ -0.3R \end{bmatrix}$$

Man setzt $\|b\|_\infty$=0.9R, a_{min}=-0.2R, dann a_{min} - $\|b\|_\infty$ =-0.2R – 0.9R=-1.1R, das Ergebnis ist im Zustand ‚Overflow_Error'. Nun folgt der Test der mittleren Vorbedingung.

In den beiden obigen Beispielen ist die einfache Vorbedingung nicht erfüllt. Jetzt wird der Test der mittleren Vorbedingung ausgeführt. Für jede $|a_n| > \frac{1}{2} Max\_\mathrm{Re}al$ oder $|b_n| > \frac{1}{2} Max\_\mathrm{Re}al$, ob es alle $|a_n - b_n| \le Max\_\mathrm{Re}al$ gibt. Für die Elemente, deren Werte größer als die Hälfte der Grenzwerte sind, 0.9R - 0.6R=0.3R und 0.8R - 0.1R=0.7R und 0.9R-0.9R=0.0R, deshalb sind alle a_n - b_n in Ordnung. Der Test zeigt den Zustand ‚No_Error'. Man braucht die komplexe Vorbedingung nicht zu benutzen.

c.) Dann wird die komplexe Vorbedingung getestet, deshalb setzt man ein Beispiel, das die einfache und mittlere Vorbedingung nicht erfüllt.

$$\begin{bmatrix} 0.9R \\ 0.8R \\ 0.2R \end{bmatrix} - \begin{bmatrix} -0.1R \\ 0.1R \\ -0.3R \end{bmatrix} = \begin{bmatrix} 1.0R \\ 0.7R \\ 0.5R \end{bmatrix}$$

Für den Test der einfachen Vorbedingung gibt es 0.9R – (-0.3R)=1.2R. Das Ergebnis ist der Zustand ‚Overflow_Error'.

Danach testet man die mittlere Vorbedingung. Es gibt zwei Elemente, deren absolute Werte größer als die Hälfte des Grenzwerts sind. 0.9R- (-0.1R)=R, und 0.8R- 0.1R=0.7R die Werte sind alle in Ordnung, sodass man nicht zum Test der komplexen Vorbedingung kommen muss.

d.) Zum Schluss soll ein Beispiel gezeigt und getestet werden, wo die drei Vorbedingungen nicht erfüllt wurden.

$$\begin{bmatrix} 0.8R \\ 0.5R \\ 0.4R \end{bmatrix} - \begin{bmatrix} -0.3R \\ 0.1R \\ -0.8R \end{bmatrix} = \begin{bmatrix} 1.1R \\ 0.5R \\ 1.2R \end{bmatrix}$$

i.) $\|a\|_\infty - b_{\min} = 0.8R - (-0.8R) = 1.6R > R$, die einfache Vorbedingung ist nicht erfüllt.

ii.) Für jede $|a_n| > \frac{1}{2} Max\_\mathrm{Re}al$ oder $|b_n| > \frac{1}{2} Max\_\mathrm{Re}al$:

$$0.8R - (-0.3R) = 1.1R > R$$

$$0.4R - (-0.8R) = 1.2R > R$$

Die mittlere Vorbedingung ist auch nicht erfüllt.

iii.) Test für jedes Element:

$$0.8R - (-0.3R) = 1.1R > R$$

$$0.5R - 0.1R = 0.4R$$

$$0.4R - (-0.8R) = 1.2R > R$$

Die komplexe Vorbedingung ist auch nicht erfüllt.

Deshalb erfüllt das Beispiel diese drei Vorbedingungen nicht.

4.3.3 Multiplikation

a.) Zuerst soll die einfache Vorbedingung getestet werden.

$$\begin{bmatrix} 0.1R \\ 0.4R \\ 0.2R \end{bmatrix} \times 2 = \begin{bmatrix} 0.2R \\ 0.8R \\ 0.4R \end{bmatrix}$$

Man setzt $||a||_\infty$=0.4R, dann $\|a\|_\infty \times Faktor = 0.4R \times 2 = 0.8R$, das Ergebnis ist in Ordnung, deshalb ist das Resultat in Ordnung. Der Test der einfachen Vorbedingung ist auch ‚No_Error'.

b.) Zum Schluss soll die komplexe Vorbedingung getestet werden, deshalb setzt man ein Beispiel, das die einfache Vorbedingung nicht erfüllt.

$$\begin{bmatrix} 0.1R \\ 0.6R \\ 0.2R \end{bmatrix} \times 2 = \begin{bmatrix} 0.2R \\ 1.2R \\ 0.4R \end{bmatrix}$$

Für die einfache Vorbedingung, gibt es $\|a\|_\infty \times Faktor = 0.6R \times 2 = 1.2R$. Das Ergebnis ist im Zustand ‚Overflow_Error', sodass man zum Test der komplexen Vorbedingung übergeht.
Nach dem Test der komplexen Vorbedingung wird der Zustand ‚Consranit_Error' gezeigt, weil es im Ergebnis den ‚Overflow_Error' Wert gibt.

4.3.4 Scalar_Produkt

Die Tests der Vorbedingung von Scalar_Produkt sind relative kompliziert. Durch das obige Aktivitätsdiagramm kann man den Prozess der Vorbedingung verstehen. Anschließend wurde eine Tabelle erstellt, die die Möglichkeit der Prozesse anzeigt.

EV1: die erste einfache Bedingung
EV2: die zweite einfache Bedingung
MV1: die erste mittlere Bedingung
MV2: die zweite mittlere Bedingung
KV: die komplexe Bedingung

	EV1	EV2	MV1	MV2	KV	Zustand
Möglichkeit1	Ja	Ja	—	—	—	No_Error
Möglichkeit2	Ja	Nein	Ja	Ja	—	No_Error
Möglichkeit3	Nein	—	Ja	Ja	—	No_Error
Möglichkeit4	Nein	—	Ja	Nein	Ja	No_Error
Möglichkeit5	Ja	Nein	Ja	Nein	Ja	No_Error
Möglichkeit6	Nein	—	Nein	—	Nein	Constranit_Error

Tabelle 4.1: Die Möglichkeit der Prozesse in den Tests

a.) (Möglichkeit 1) Zuerst soll die einfache Vorbedingungen getestet werden. Dann setzt man ein Beispiel, das die einfachen Vorbedingungen erfüllt.

$$\begin{bmatrix} 0.1\sqrt{R} \\ 0.2\sqrt{R} \\ 0.3\sqrt{R} \end{bmatrix}^T \cdot \begin{bmatrix} 0.4\sqrt{R} \\ 0.1\sqrt{R} \\ 0.1\sqrt{R} \end{bmatrix} = 0.04R + 0.02R + 0.03R = 0.09R$$

Man setzt $||a||_\infty=0.3R^{1/2}$, $||b||_\infty=0.4R^{1/2}$, dann $||a||_\infty \cdot ||b||_\infty=0.12R$, das Ergebnis ist in Ordnung, deshalb ist die erste einfache Vorbedingung bestanden. Dann berechnet man: $\sqrt{n^2\left(\|a\|_\infty \|b\|_\infty\right)^2} = \sqrt{3^2\left(0.3\sqrt{R} \times 0.4\sqrt{R}\right)^2} = 0.36R$, das Ergebnis ist auch in Ordnung. Deshalb ist der Test der einfachen Vorbedingung im Zustand ‚No_Error'.

b.) Danach führt man einige Beispiele an, um die mittlere Vorbedingung zu testen. Deshalb erfüllt das Beispiel die einfachen Vorbedingungen nicht. Es gibt zwei Möglichkeiten.
I.)(Möglichkeit 2)

$$\begin{bmatrix} 0.5\sqrt{R} \\ 0.3\sqrt{R} \\ 0.4\sqrt{R} \end{bmatrix}^T \cdot \begin{bmatrix} 0.1\sqrt{R} \\ -0.7\sqrt{R} \\ 0.5\sqrt{R} \end{bmatrix} = 0.05R - 0.21R + 0.2R = 0.04R$$

In dem Beispiel setzt man $||a||_\infty=0.5R^{1/2}$, $||b||_\infty=0.7R^{1/2}$, dann $||a||_\infty \cdot ||b||_\infty=0.35R$, ist das Ergebnis in Ordnung. Aber für die zweite einfache Vorbedingung gibt es: $\sqrt{n^2\left(\|a\|_\infty \|b\|_\infty\right)^2} = \sqrt{3^2\left(0.5\sqrt{R}\right)^2 \times \left(-0.7\sqrt{R}\right)^2} = 1.05R > R$. Deshalb muss man die mittlere Vorbedingung testen. Die erste mittlere Vorbedingung, für $|a_n| > \sqrt{Max\_\mathrm{Re}al}$ oder

$|b_n| > \sqrt{Max\_\mathrm{Re}al}$, ob es alle $|a_n| \times |b_n| \le Max\_\mathrm{Re}al$ gibt. Es gibt kein Element, dessen Wert größer als die Quadratwurzel des Grenzwertes ist. Das bedeutet, die beiden Vektoren erfüllen die erste mittlere Vorbedingung. Danach verwendet man die zweite mittlere Vorbedingung. Weil die Berechnung ist:

$$\frac{1}{4}\left[\left(\|a\|_\infty + \|b\|_\infty\right)^2 \times N - 0\right] = \frac{1}{4}\left(0.5\sqrt{R} + 0.5\sqrt{R}\right)^2 \times 3 = 0.75R < R$$

ist das Ergebnis im Zustand ‚No_Error'.

II.) (Möglichkeit 3)

$$\begin{bmatrix} 1.0\sqrt{R} \\ 0.1\sqrt{R} \\ \sqrt{R} \end{bmatrix}^T \cdot \begin{bmatrix} 0.1\sqrt{R} \\ -2.0\sqrt{R} \\ 0.1\sqrt{R} \end{bmatrix} = 0.1R - 0.2R + 0.1R = 0$$

Als zweites Beispiel setzt man $||a||_\infty = R^{1/2}$, $||b||_\infty = -2R^{1/2}$, dann $||a||_\infty \cdot ||b||_\infty = 2R$, ist das Ergebnis auch zu groß. Dann muss man die mittlere Vorbedingung testen. Die erste mittlere Vorbedingung, für $|a_n| > \sqrt{Max\_\mathrm{Re}al}$ oder $|b_n| > \sqrt{Max\_\mathrm{Re}al}$, ob es alle $|a_n| \times |b_n| \le Max\_\mathrm{Re}al$ gibt. Es gibt ein Element, dessen Werte größer als die Quadratwurzel des Grenzwertes sind. Aber das Ergebnis von beiden Elementen ist in Ordnung: $0.1\sqrt{R} \times -2\sqrt{R} = -0.2R$ Dann kann man die zweite mittlere Vorbedingung testen. Die Berechnung ist:

$$\frac{1}{4}\left[\left(a_{\max} + b_{\max}\right)^2 \times N - 0\right] = \frac{3}{4}\left(\sqrt{R} + 0.1\sqrt{R}\right)^2 = 0.9075R < R$$

Deshalb ist das Ergebnis in Ordnung. Das Beispiel erfüllt die mittleren Vorbedingungen.

c.) Als Nächstes sollen die komplexen Vorbedingungen getestet werden, deshalb führt man zwei Beispiele an, die die einfachen und mittleren Vorbedingungen nicht erfüllen.

I.) Das erste Beispiel ist die Möglichkeit 4 in obiger Tabelle.

$$\begin{bmatrix} 1.0\sqrt{R} \\ 0.2\sqrt{R} \\ \sqrt{R} \end{bmatrix}^T \cdot \begin{bmatrix} 1.0\sqrt{R} \\ 2.0\sqrt{R} \\ 0.5\sqrt{R} \end{bmatrix} = 0.1R + 0.4R + 0.5R = R$$

Das zweite Beispiel $||a||_\infty = R^{1/2}$, $||b||_\infty = 2R^{1/2}$, dann $||a||_\infty \cdot ||b||_\infty = 2R$, ist das Ergebnis auch zu groß. Dann muss die mittlere Vorbedingung getestet werden. Die erste mittlere Vorbedingung, für $|a_n| > \sqrt{Max\_\mathrm{Re}al}$ oder $|b_n| > \sqrt{Max\_\mathrm{Re}al}$,ob es alle

$|a_n| \times |b_n| \le Max\_\mathrm{Re}al$ gibt. Es gibt ein Element, dessen Werte größer als die Quadratwurzel des Grenzwertes sind. Aber das Ergebnis von beiden Elementen ist in Ordnung: $0.2\sqrt{R} \times 2\sqrt{R} = 0.4R$ Dann kann man die zweite mittlere Vorbedingung testen. Die Berechnung ist :

$$\frac{1}{4}\left[\left(a_{\max} + b_{\max} \right)^2 \times N - 0 \right] = \frac{3}{4}\left(\sqrt{R} + 2\sqrt{R} \right)^2 = 6.75R > R$$

Es bedeutet, die beiden Vektoren erfüllen die beiden mittleren Vorbedingungen nicht. Danach verwendet man die komplexe Vorbedingung. Weil die Summe 0.1R+0.4R+0.5R=R ist, ist das Ergebnis des Tests im Zustand ‚No_Error'.

II.) Das zweite Bespiel ist folgendes:(Möglichkeit 5)

$$\begin{bmatrix} 1.0\sqrt{R} \\ 0.3\sqrt{R} \\ \sqrt{R} \end{bmatrix}^T \cdot \begin{bmatrix} 0.5\sqrt{R} \\ 0.5\sqrt{R} \\ 0.3\sqrt{R} \end{bmatrix} = 0.5R + 0.15R + 0.3R = 0.95R$$

Das Beispiel ist die Möglichkeit 5 in der Tabelle. In dem Beispiel setzt man $||a||_\infty = R^{1/2}$, $||b||_\infty = 0.5R^{1/2}$, dann $||a||_\infty \cdot ||b||_\infty = 0.5R$, ist das Ergebnis in Ordnung. Dann testet man weiter die zweite einfache Vorbedingung. Man berechnet:

$$\sqrt{n^2 \left(\|a\|_\infty \|b\|_\infty \right)^2} = \sqrt{3^2 \left(\sqrt{R} \times 0.5\sqrt{R} \right)^2} = 1.5R,$$

Das Ergebnis ist zu groß. Die mittleren Vorbedingungen müssen getestet werden.

Die erste mittlere Vorbedingung, für $|a_n| > \sqrt{Max\_\mathrm{Re}al}$ oder $|b_n| > \sqrt{Max\_\mathrm{Re}al}$, ob es alle | $|a_n| \times |b_n| \le Max\_\mathrm{Re}al$ gibt. Es gibt keine Elemente, deren Werte größer als die Quadratwurzel des Grenzwertes sind. Dann kann direkt die zweite mittlere Vorbedingung getestet werden. Die Berechnung ist:

$$\frac{1}{4}\left[\left(a_{\max} + b_{\max} \right)^2 \times N - 0 \right] = \frac{3}{4}\left(\sqrt{R} + 0.5\sqrt{R} \right)^2 = 1.6875R > R$$

Es bedeutet, die beiden Vektoren erfüllen die beiden mittleren Vorbedingungen nicht. Danach verwendet man die komplexe Vorbedingung. Weil die Summe 0.5R + 0.15R + 0.3R=0.95R ist, ist das Ergebnis des Tests im Zustand ‚No_Error'.

d.) Zum Schluss soll ein Beispiel gezeigt und getestet werden, das drei Vorbedingungen nicht erfüllt. (Möglichkeit 6)

$$\begin{bmatrix} 0.1\sqrt{R} \\ 0.2\sqrt{R} \\ 2.0\sqrt{R} \end{bmatrix}^T \cdot \begin{bmatrix} 2.0\sqrt{R} \\ 0.5\sqrt{R} \\ 0.4\sqrt{R} \end{bmatrix} = 0.2R + 0.1R + 0.8R = 1.1R > R$$

i.) einfache Vorbedingungen: $\|a\|_\infty \times \|b\|_\infty = 2\sqrt{R} \times 2\sqrt{R} = 4R$ die einfache Vorbedingung ist nicht erfüllt.

ii.) mittlere Vorbedingungen: für $|a_n| > \sqrt{Max\_\mathrm{Re}al}$ oder $|b_n| > \sqrt{Max\_\mathrm{Re}al}$:

$$0.1\sqrt{R} \times 2\sqrt{R} = 0.2R$$

$$2\sqrt{R} \times 0.4\sqrt{R} = 0.8R$$

Dann $\frac{1}{4}\left[\left(a_{\max} + b_{\max}\right)^2 \times N - 0\right] = \frac{3}{4}\left(2\sqrt{R} + 2\sqrt{R}\right)^2 = 12R > R$

Die mittleren Vorbedingungen sind auch nicht erfüllt.

iii.) Test für jedes Element:

$$0.1\sqrt{R} \times 2\sqrt{R} = 0.2R$$

$$0.2\sqrt{R} \times 0.5\sqrt{R} = 0.1R$$

$$2\sqrt{R} \times 0.4\sqrt{R} = 0.8R$$

Dann ist die Summe aller Produkte:

$a_1 b_1 + a_2 b_2 + ... + a_N b_N = 0.2R + 0.1R + 0.8R = 1.1R > R$

Die komplexe Vorbedingung ist auch nicht erfüllt.

Deshalb erfüllt das Beispiel diese drei Vorbedingungen nicht.

5 Zusammenfassung und Ausblick

Die Vor- und Nachbedingungen von SPARK vermeiden nicht nur die Zerstörung des Programms, sondern vereinfachen auch die Tests. Mit der Verwendung der Vorbedingung können komplizierte Implementierungen vereinfacht werden. Die Vor- und Nachbedingungen beschreiben die Voraussetzungen und Aussagen nach der Ausführung. Sie spielen eine wichtige Rolle in Spark 2014.

In dieser Bachelorarbeit, mit der Hilfe von Prof. Enzmann, wurde der Entwurf der Vor- und Nachbedingungen der Operationen für die arithmetische Berechnung untersucht, um die robuste Implementierung von Vektoroperationen zu realisieren. Die erste Schwierigkeit bestand in der Benutzung der Notwendigen und Hinreichenden Bedingung beim Entwurf. Am Anfang wurden viele Vorbedingungen geschrieben. Aber es sah nicht gut aus und war auch überschüssig. Hier brauchte die Voraussetzung nicht die Notwendige und Hinreichende Bedingung zu sein. Die Hinreichende Bedingung ist genug für die Vorbedingung, weil eine Vorbedingung nicht die einzige Ursache für Operationen ist. Es gibt viele Vorbedingungen. Die Operation der Addition kann vielleicht auch ausgeführt werden, ohne dass die erste Vorbedingung eingetreten ist, sonst mit der zweiten Vorbedingung. Dann wurden nur die repräsentativen Vorbedingungen verwendet. Es wurden drei unterschiedliche Vorbedingungen gesetzt, die einfache Vorbedingung, mittlere Vorbedingung und komplexe Vorbedingung. Die Reihenfolge der Vorbedingungen wird durch die Menge der Elemente, die getestet wurden, geordnet. Und es gibt einen Schwerpunkt, die komplexe Vorbedingung ist notwendig. Anfangs wurde die komplexe Vorbedingung vernachlässigt. Die komplexe Vorbedingung ist für alle Elemente des Vektors. Sie lässt die ganzen Vorbedingungen vollständig sein. Die Methode der Vorbedingung von ‚Scalar_Produkt' ist relative schwer, weil die Vektoroperation zwei Anteile, ‚Scale' und ‚Addition' hat. Jede Vorbedingung muss diese zwei Anteile behalten. Der Anteil ‚Scale' ist einfach. Die Methode des Entwurfs ist ähnlich wie ‚Addition'. Aber der Anteil ‚Addition' von ‚Scalar_Produkt' ist schwer. Es sollte die Summe aller Produkte von ‚Scale' getestet werden. Bei der einfachen und mittleren Vorbedingung konnte die Addition zwischen allen Produkten durchgeführt werden. Hier wurde die Formel der Norm benutzt. Es wurde der Grenzwert der Formel berechnet. Dann wurden durch den Vergleich zwischen Grenzwert der Formel und Grenzwert von ‚Scalar_Produkt' die einfachen und mittleren Vorbedingungen entworfen. Am Anfang glaubte man, der Entwurf ist ok. Mit der Hilfe von Prof. Enzmann konnte festgestellt werden, dass für einige Zusammenhänge dieser Nachweis zu trivial ist. Für die Entwürfe mussten deutliche und ausführliche Erläuterungen gemacht wer-

den. Für einen Außenstehenden konnten diese sehr knappen Notizen für das Verständnis nicht genügen.

Nach dem Entwurf und dem Nachweis der Vorbedingung wurde der zweite Teil erledigt, die Test-Funktion. Dieser Anteil ist relative einfach. Es wurde das Aktivitätsdiagramm für jede Operation erstellt. Dadurch konnte man einfach verstehen, wie die Vorbedingungen laufen. Mit Hilfe des Aktivitätsdiagramms wurden die mathematischen Beispiele gesetzt, um jede Vorbedingung zu testen.

Durch die Benutzung von SPARK 2014 mit Vor- und Nachbedingungen kann der statische und dynamische Fehler effektiv während der Compilezeit und Laufzeit der Vektoroperation vermieden werden. Mit der Hilfe der Vor- und Nachbedingung konnten auch viele Prüfschritte gespart werden. Die Vektoroperationen mit höherer Sauberkeit und Sicherheiten wurden realisiert. Eine andere wichtigere Sache war die, dass die Flexibilität und Robustheit der Bibliothek des Programms stabiler war. Die Bibliothek wurde meistens auf der Berechnung von Signal, z. B. Faltung verwendet. Angesichts der Komplexität der Hardware-Umgebung und der nicht wirklich einheitlichen Standardbibliothek besteht auch das Risiko bei der Benutzung der Standardbibliothek. Die hier entworfene Bibliothek mit Vor- und Nachbedingung kann das Risiko möglichst vermeiden. Die Leistung der Prüfung wird auch durch die exakte Vor- und Nachbedingung erhöht.

SPARK lässt das Programm sichtbar sein. Unter SPARK ist das Werkzeug von ‚Vor- und Nachbedingung' schon vorhanden. Als nächstes Ziel wäre es wichtig, auf den vorhandenen Werkzeugen aufzubauen und sie weiterzuentwickeln.

Anhang 1: Quellen- und Literaturverzeichnis

Buchquellen

[1] Ian Sommerville: Software Engineerung, 6. Aufl,München : Pearson Studium, c2001;

[2] Björn Berg,Philip Knott,Gregor Sandhaus : Hybride Softwareentwicklung, S. 11-13;

[3] Wolfgan Schäfter, Kurt Georgi, Gisela Trippler: Mathematik-Vorkurs, 6. Auflage;

[4] Susanna Epp: Discrete Mathematics with Applications, S. 24, 43, 44;

[5] John M. Peterson: Finite Mathematics, S. 10;

[6] Ignacio Bello, Anton Kaul, Jack R. Britton: Mathematics, S. 107-108;

[7] Ed Wheeler, Jim Brawner: Discrete Mathematics for Teachers, S. 17-18;

[8] Christopher Dietmaie: Mathematik für Wirtschaftsingenieure: Lehr- und Übungsbuch S. 54-55;

[9] Karsten Kirchgessner,Marco Schreck : Vektor- und Matrizenrechnung für Dummies, S. 85;

[10] Jochen Werner: Numerische Mathematik/Lineare und nichtlineare Gleichungssystem, Interpolation, numerische Integration, S. 16, 17;

[11] Joachim Weidmann :Lineare Operatoren in Hilberträumen 1: Grundlagen, S. 17, 18, 23;

[12] Grady Booch, James Rumbaugh, Ivar Jacobson,Pearson Deutschland GmbH:Das UML-Benutzerhandbuch: aktuell zur Version 2.0, S. 37;

[13] Stephan Kleuker: Grundkurs Software-Engineering mit Uml, Kapital Überblick UML, S. 383, 386;

Onlinequellen

[14] http://docs.adacore.com/spark2014-docs/html/lrm/introduction.html (Abrufdatum 04.02.2015)

[15] http://www.spark-2014.org/about(Abrufdatum 04.02.2015)

[16] https://www.fbi.h-da.de/labore/case/uml/aktivitaetsdiagramm.html Abrufdatum 04.02.2015)

Anhang 2: Abbildungsverzeichnis

Anhang 3: Tabellenverzeichnis

Anhang 4: Gleichungsverzeichnis