Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Visualisierung von Softwarebeweisen mithilfe von Visualisierungsbibliotheken und Frameworks in JavaScript

Abschlussarbeit

zur Erlangung des akademischen Grades

Master of Engineering

Verfasst von: Bennet Leo Jacobsen
Geboren am: 07.03.1993 in Berlin
Matrikelnummer: 551756

Abgabedatum: 08.09.2022
Studiengang: Computer Engineering
Erstprüfer: Prof. Dr. Thomas Baar
Zweitprüfer: Prof. Dr.-Ing. Johann Schmidek

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

SMT Solver	Satisfiability Modulo Theories Solver
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
MVC	Model View Controller
SVG	Scalable Vector Graphics
JSON	JavaScript Object Notation
CLI	Command Line Interface

Erklärung der Urheberschaft

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum Unterschrift

Einleitung

In dieser schriftlichen Ausarbeitung werden die Recherche, der Vergleich und die Auswahl von Frameworks zur Visualisierung von Softwarebeweisen abgebildet, mithilfe derer eine anschließende Implementierung für das grafische Darstellen des Ergebnisses eines solchen Beweises möglich sein soll.
Die Thematik wurde in Absprache mit dem Professor für Softwaretechnik und Datenbanken Herrn Prof. Dr. Thomas Baar an der Hochschule für Technik und Wirtschaft Berlin festgelegt und an dieser Hochschule ebenfalls angefertigt. In den folgenden Abschnitten dieses Kapitels sollen die Problemstellung und das thematische Umfeld, in dessen Bezug diese Arbeit entstanden ist, dargestellt werden.

1.1 Softwarebeweise

Seit mehreren Jahrzehnten schreitet die Digitalisierung der Welt rasant voran. Die sich daraus ergebenden Veränderungen wirken sich auf weite Teile des Lebens aus, welches zunehmend digital vernetzt wird, wie von der Bundeszentrale für politische Bildung beschrieben (Hesse, et al. 2020). Im Zuge dessen steigen ebenfalls die Komplexität der verwendeten Software und gleichzeitig die Abhängigkeit der Menschen von dieser immer weiter an. Um sicherzustellen, dass ein fehlerfreier Einsatz von Software gewährleistet werden kann, ist es notwendig, diese mithilfe sogenannter Softwarebeweise zu validieren.
Als Softwarebeweis im Sinne des Software-Engineerings bezeichnet man Resultat des Verfahrens der Sicherstellung, dass eine Software alle an sie gestellten Anforderungen erfüllt. Da Software im Laufe der letzten Jahrzehnte immer komplexer und umfangreicher geworden ist, ist es heutzutage notwendig, für das Überprüfen und Beweisen ihrer Funktionalitäten ebenfalls Software einzusetzen, die in der Lage ist, diese Aufgaben zu übernehmen.
Besonders im Hinblick auf den kommerziellen Einsatz von Software ist es wichtig, die Fehlerfreiheit der Software mit Softwarebeweisen zu garantieren, da sie aufgrund der ohne sie nicht erreichbaren Ausfallsicherheit von Systemen, Flexibilität bzgl. der Anforderungen und Funktionen, Kostenreduktion und schnellen Verfügbarkeit einen treibenden Technologiekatalysator für die Wirtschaft darstellt (Beyer, et al. 2017).

1.2 Problemstellung und Zielsetzung dieser Arbeit

Um sich die Problematik zu vergegenwärtigen, aus der heraus diese Arbeit entstanden ist, muss man sich einmal den Prozess des Lernens und Verstehens eines Sachverhaltes vor Augen führen und dabei hinterfragen, was genau die Schwierigkeit dieses Vorganges ausmacht.

Im Wesentlichen geht es dabei um zwei Dinge. Zum einen spielt der Aufwand an Zeit, die eine Person sich mit einem Sachverhalt auseinandersetzen muss, um ein Verständnis für diesen zu erlangen, eine Rolle und zu anderen ist die kognitive Anstrengung, die dafür aufgebracht werden muss, ein wesentlicher Aspekt.

Da die kognitive Kapazität des Gehirns mit Blick auf gleichzeitig durchführbare Aufgaben, wie zum Beispiel Berechnungen begrenzt ist, kann gesagt werden, dass die kognitive Anstrengung steigt, je mehr Aufgaben gleichzeitig erledigt werden müssen. So zum Beispiel bei der Multiplikation großer Zahlen ohne Hilfsmittel, was bedeutet, dass meist ein Zwischenergebnis gespeichert werden muss, während eine weitere Berechnung durchgeführt wird.

Abbildung 1: Stroop Effekt

Wie im Buch „Schnelles Denken, langsames Denken“ (Kahnemann 2016) beschrieben, ist ein weiterer Faktor, der berücksichtigt werden muss, das Hinwegsetzen über unterbewusste Prozesse, wie zum Beispiel Gewohnheiten oder den sogenannten Stroop Effekt. Dieser besagt, dass antrainierte Handlungen fast automatisiert ablaufen, während es erhöhte Konzentration braucht, um ungewohnte gedankliche Prozesse durchzuführen. So ist es, wie in Abbildung 1 zu erkennen, schwerer, die Schriftfarbe in der ersten Zeile zu sagen, als in der zweiten Zeile, da ein Mensch automatisch an eine Farbe denkt, wenn deren Name gelesen wird und dies im Gegensatz zu der visuell erkannten Farbe der Buchstaben steht.

Als letzten wesentlichen Faktor zählt Kahneman den gedanklichen Speicher auf, der beim Denken eines Gedankens oder Verstehen eines Sachverhaltes benötigt wird.

Diese drei Faktoren sind im Wesentlichen das, was die kognitive Kapazität beim Denken und Verstehen abrufen und machen letzten Endes den Schwierigkeitsgrad des Verstehens einer Sache aus. Je mehr das Lernmaterial einen dieser Faktoren hervorruft oder zum Verstehen benötigt, desto schwerer ist es, diesen Inhalt zu erlernen oder zu verstehen.

Gerade im Hinblick auf Softwarebeweise und deren Ergebnisse ist beachtliche kognitive Leistung aufzubringen, um ein gutes Verständnis für das zu erlangen, was diese aussagen und um ausmachen zu können, welche Faktoren in der Software oder bei der Festlegung der Parameter zur Überprüfung dieser zu welchen Teilergebnissen geführt haben.
Das Erzeugen von Hilfsmitteln zum Lernen und Verstehen in Form von Visualisierungen kann jedoch mit erheblichem Aufwand verbunden sein und wird deshalb oft vernachlässigt.
Deshalb soll im Zuge dieser Arbeit die These überprüft werden, dass sich durch die Auswahl des richtigen JavaScript-Frameworks ressourcenschonend Visualisierungen passend zu Beweisen umsetzen lassen.

1.3 Aufbau der Arbeit

Im anschließenden Kapitel werden eingangs die dieser Arbeit zugrunde liegenden Grundlagen erklärt. Dafür wird zunächst ein kurzer Einblick in das Beweisen von Software gegeben, um ein Überblick über die zu visualisierenden Inhalte zu schaffen. Parallel dazu werden die theoretischen Grundlagen erläutert, die für die Programmierung von Visualisierungen nötig sind. Im dritten Kapitel folgt eine Diskussion von Lösungsansätzen, wobei auf die Randbedingungen, gegeben durch die Eingliederung dieser Arbeit in eine Reihe von Arbeiten, eingegangen wird. Das vierte Kapitel wird sich mit der Auswahl und der beispielhaften Umsetzung eines Lösungsansatzes befassen, welcher im Anschluss daran hinsichtlich seiner Tauglichkeit analysiert wird. Zum Abschluss wird das Ergebnis der Arbeit mit einem möglichen Ausblick in die Zukunft des Projektes dargestellt.

2. Grundlagen

2.1 Softwarebeweise mit VeriFast

Zum Beweisen und Verifizieren von Software gibt es mehrere Verfahren, die in zwei übergeordnete Kategorien aufgeteilt werden können, wie im Artikel „Verification/Validation/Certification“ (Tran 1999) beschrieben. Zum einen die Verifikation durch dynamisches Testen, bei der anhand einer Anzahl von Tests mithilfe von Testdaten durch das Ausführen des Systems oder der Komponente Ein- und Ausgabewerte verifiziert werden. Zum anderen die Verifikation durch sogenanntes statisches Testen, bei der kein Ausführen des Systems oder der Komponente stattfindet, sondern der Code lediglich analysiert wird. Hierbei gibt es weitere Unterscheidungen, so kann beim statischen Testen auf Konsistenz des Codes hinsichtlich der Syntax, Typisierung oder korrekter Übergabeparameter getestet oder messbare Faktoren wie die Verständlichkeit oder Strukturierung des Codes geprüft werden. Wie von Tran ebenso weiter beschrieben, lässt sich das dynamische Testen ebenfalls weiter aufteilen. So wird zwischen funktionalem Testen, strukturellem Testen und zufälligem Testen unterschieden. Beim funktionalen Testen werden alle Funktionen des Systems hinsichtlich der Anforderungen an das System getestet. Diese Form des Testens wird auch als Blackbox-Test bezeichnet, da keine Kenntnis über die Implementierung des Systems erforderlich ist.

Beim strukturellen Testen hingegen wird das Wissen über die Implementierung des Systems vorausgesetzt, um gezielt das Ausführen der einzelnen Komponenten zu überprüfen. Das strukturelle Testen wird daher auch als Whitebox-Test angesehen. Beim funktionalen und strukturellen Testen werden jeweils bestimmte Eigenschaften des Systems mit Hilfe von ausgewählten Testszenarien überprüft.

Beim zufälligen Testen wird frei aus allen möglichen Testszenarien gewählt und diese mit zufällig gewählten Werten durchgeführt. Dadurch können Fehler aufgedeckt werden, die bei anderen systematischen Testverfahren unentdeckt bleiben. Erschöpfendes Testen, so Tran, ist ein Verfahren, welches dem zufälligen Testen zugeordnet werden kann. Allerdings ist, auch wenn hierbei die vollständige Funktionalität des Systems bewiesen werden könnte, indem alle möglichen Eingaben eines Systems an diesem getestet würden, die Umsetzung eines solchen Tests in der Realität aufgrund der Menge an möglichen Eingabekombinationen nicht möglich.

Die Funktionsweise des Werkzeugs, dessen Softwarebeweise dieser Arbeit zugrunde liegen, soll nun anhand eines simplen Beispiels dargestellt werden.

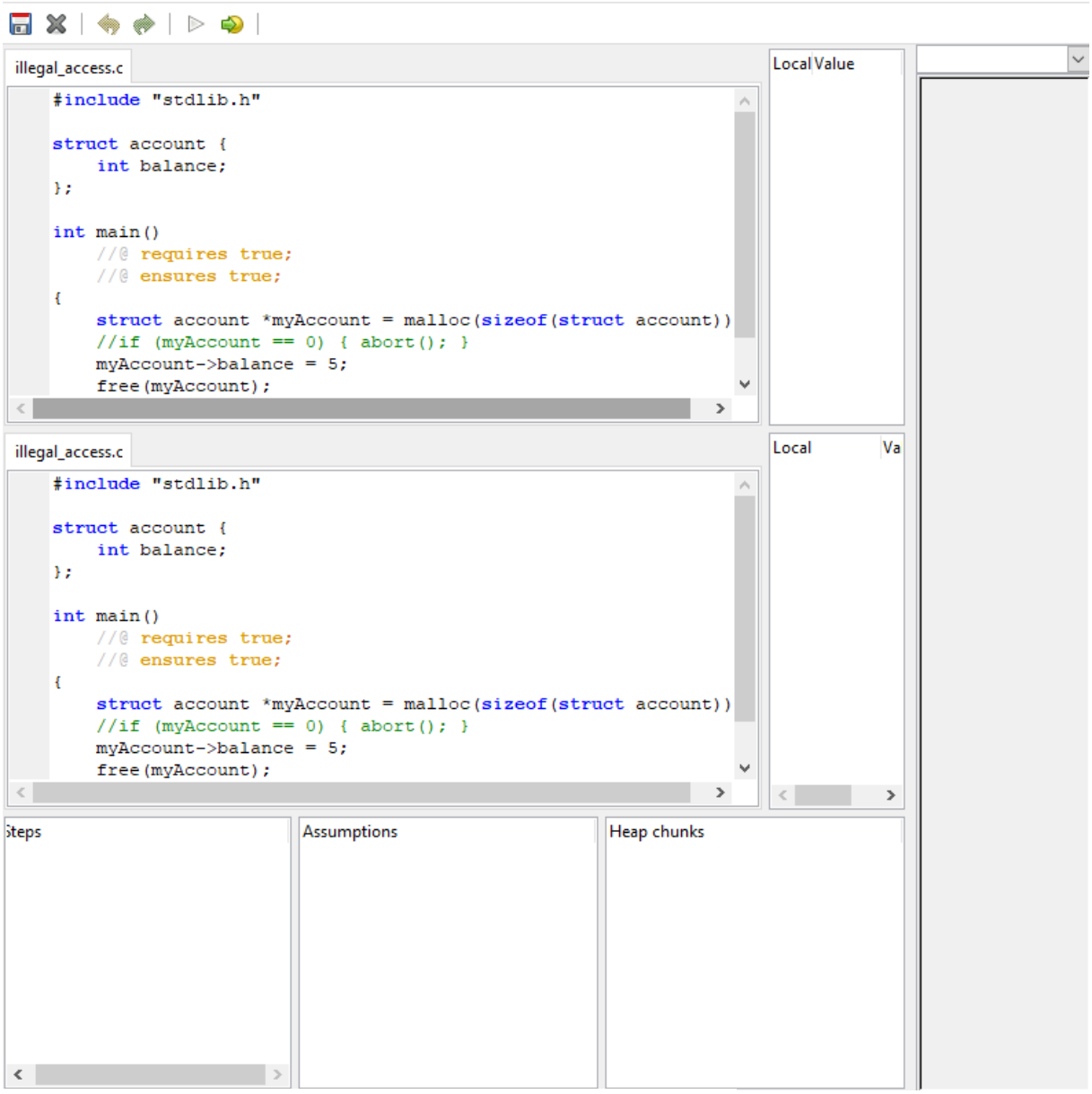

Abbildung 2: VeriFast nach dem Laden einer Code-Datei

Wie in Abbildung 2: VeriFast nach dem Laden einer Code-Datei zu sehen, können in der grafischen Benutzeroberfläche von VeriFast Dateien geladen werden. In diesem Fall soll eine Funktion verifiziert werden, die einem Datenstruct einen Wert zuweist.

Für das Verifizieren mit VeriFast wurde die Datei mit der notwendigen Syntax für Vor- und Nachbedingung vorbereitet. Diese drücken sogenannte Kontrakte aus, die für jede Funktion, die bewiesen werden soll, zu Beginn dieser neu definiert werden müssen. Die Vor- und

Nachbedingungen werden dabei in Kommentare zum Code hinzugefügt, damit sie vom C-Compiler ignoriert werden. Die Vorbedingung drückt aus, welche Bedingungen zum initialen Zustand der Funktion gelten und die Nachbedingung drückt aus, welche Bedingungen im finalen Zustand der Funktion gelten müssen. Die Vorbedingung wird nach „//@ requires“ angegeben und die Nachbedingung nach „//@ ensures“. In diesem Beispiel werden beide Bedingungen als „true“

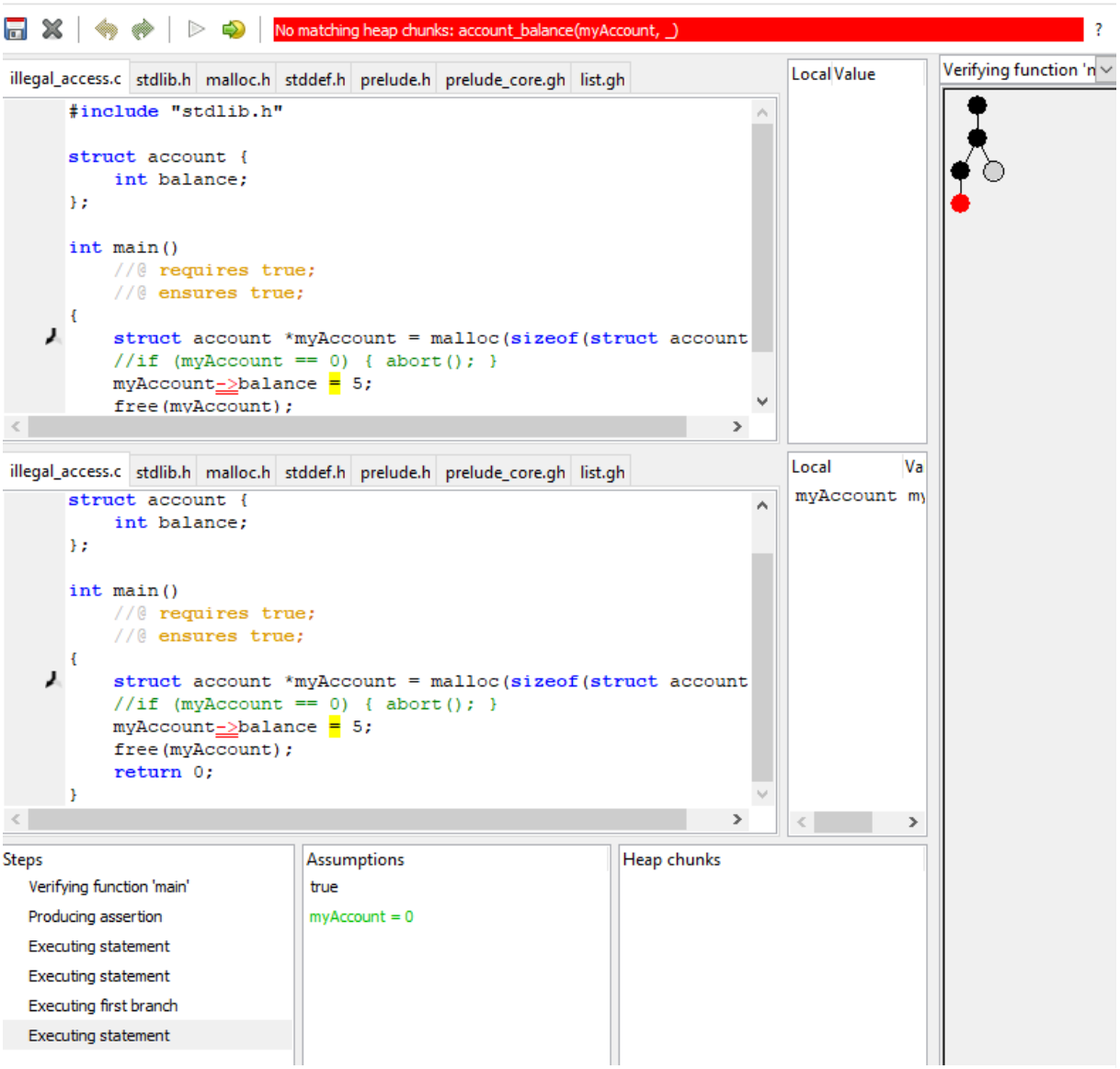

Abbildung 3: Beweis mit VeriFast schlägt fehl.

angegeben und somit nicht weiter definiert.

Startet man nun den Beweisvorgang mit VeriFast, so hat man die Wahl einen Halte-Punkt an der Stelle des Programms zu setzen, an dem sich der Cursor befindet, oder den gesamten Code zu verifizieren.

Wie in Abbildung 3: Beweis mit VeriFast schlägt fehl. zu erkennen, ist es in diesem Fall auf Anhieb

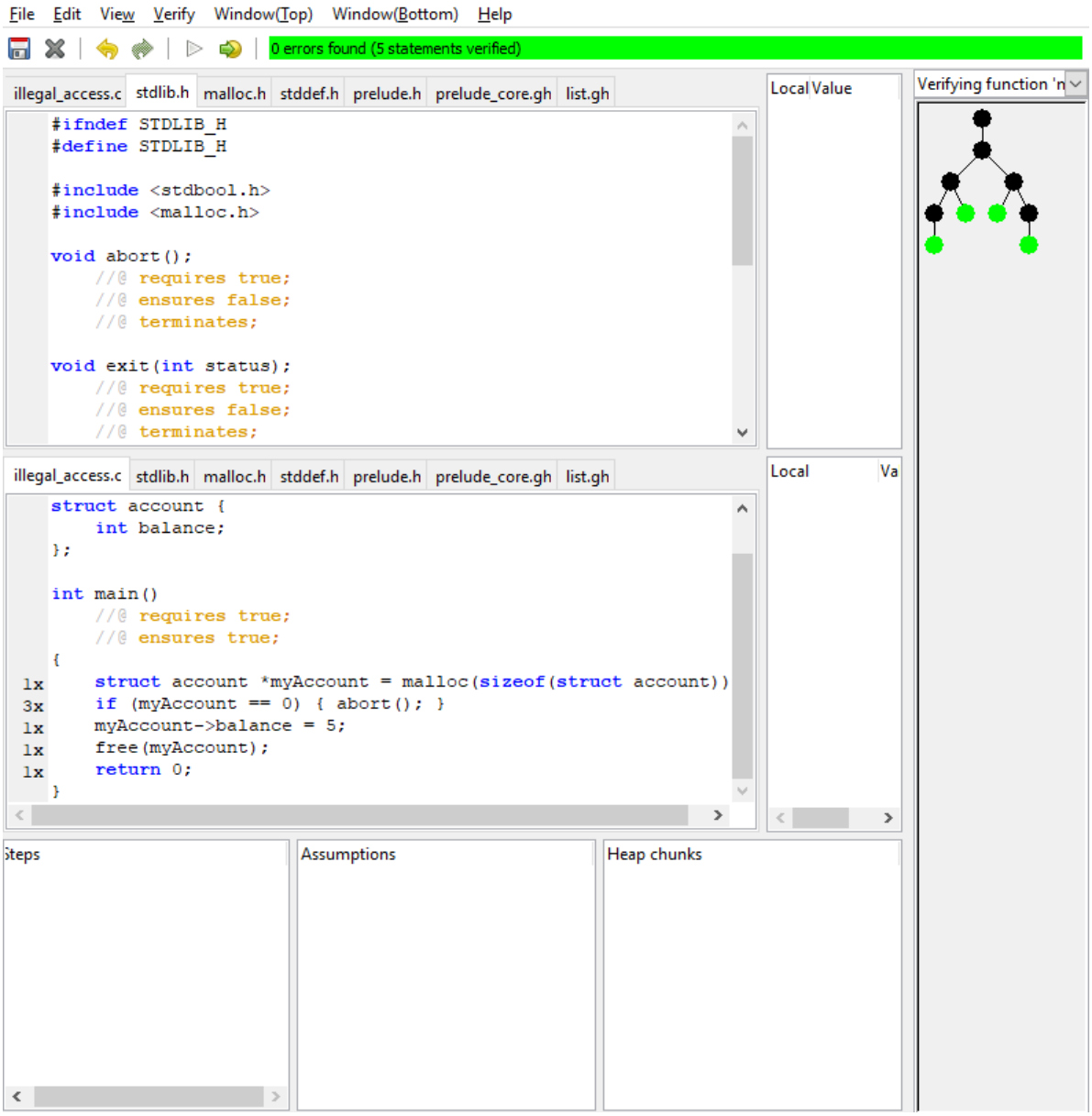

Abbildung 4: Erfolgreicher Beweis mit VeriFast

nicht möglich, den Code zu verifizieren, da nicht sichergestellt ist, dass der Speicher zugewiesen wurde. Somit gibt VeriFast die Fehlermeldung aus, dass für den Fall, dass die Speicherzuweisung nicht erfolgreich war, die anschließende Zuweisung des Wertes zu besagtem Speicher fehlschlägt. Dazu wird auf der rechten Seite des Programms ein Ablaufgraph dargestellt. In diesem Fall ist er noch unvollständig, da der Beweis schon für die erste Annahme fehlgeschlagen ist.

Entfernt man nun den Kommentar im Programmcode, sodass vor der Zuweisung des Wertes eine Kontrolle durchgeführt wird, ob es zugewiesenen Speicher für diese Zuweisung gibt, wie in Abbildung 4: Erfolgreicher Beweis mit VeriFast zu sehen, kann der Beweis erfolgreich

abgeschlossen werden.

Durch Anklicken der Knoten im Ablaufgraphen, wie in Abbildung 5: Wird einer der Knoten angewählt, sind die zugehörigen Bedingungen, Schritte und Speicherbelegungen ablesbar. erfolgt, kann man erkennen, an welcher Stelle im Code welche Annahmen gegolten haben und welche Speicherzuweisungen stattgefunden haben.

Somit lassen sich zwar die einzelnen Resultate nachvollziehen, jedoch ist es schwer zu erkennen, was die Ergebnisse der einzelnen Pfade in der Summe aussagen. An dieser Stelle wäre es für das Verständnis hilfreich, wenn es eine Möglichkeit geben würde, die Ergebnisse zusammenzufassen und zu visualisieren.

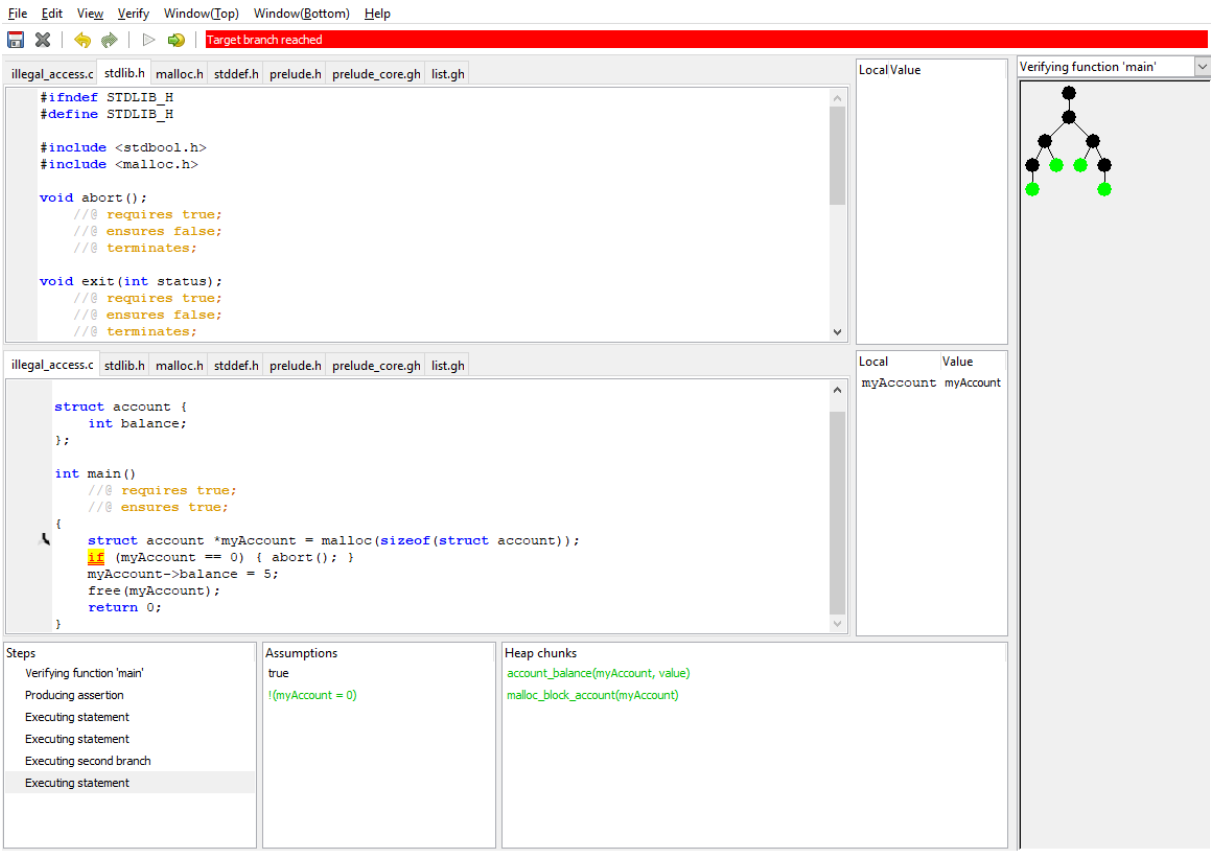

Abbildung 5: Wird einer der Knoten angewählt, sind die zugehörigen Bedingungen, Schritte und Speicherbelegungen ablesbar.

2.2 Visualisierungen

Aufgrund der Tatsache, dass die Informationen im Gedächtnis überwiegend in Bildern abgelegt werden (Sperling 1960), kann die Aufnahmefähigkeit, Verarbeitung und Speicherung von Informationen erheblich gesteigert werden, wenn die zu vermittelnden Inhalte visualisiert werden. Im Bereich der Softwareentwicklung hat es innerhalb der letzten Jahrzehnte maßgebliche Weiterentwicklungen gegeben, gerade im Hinblick auf Visualisierungen. So sind heute Python und JavaScript zwei der am weitesten verbreiteten Sprachen, wenn es um Visualisierungen geht. Die große Popularität von Python ist jedoch im Gegensatz zu JavaScript seiner großen Verbreitung im Bereich der Datenanalyse und des maschinellen Lernens geschuldet. Die Beliebtheit auf diesen Gebieten kommt daher, dass Python einen leichten Einstieg bietet und bei der Programmierung aufgrund der hohen Abstraktionsebene weniger auf den Code selbst geachtet werden muss. Somit kann auch von Einsteigern mehr Aufmerksamkeit auf das Finden einer Lösung gerichtet werden.

Für erfahrene Programmierer empfiehlt es sich aber, JavaScript zu verwenden. Nicht nur ist die Leistung von JavaScript der von Python überlegen, auch das Ausmaß an Bibliotheken gerade auf dem Gebiet der Visualisierungen ist seitens JavaScript von erheblich größerem Umfang. Ein weiterer Vorteil ist die Verfügbarkeit von TypeScript, womit eine strengere Typisierung möglich ist.

2.3 JavaScript

JavaScript ist eine Programmiersprache, die es in erster Linie erlaubt, komplexe Bestandteile von Webseiten zu programmieren. Sie kann als dritte der drei Standardtechnologien in der Webentwicklung gesehen werden, die anderen beiden sind HTML und CSS (MDN 2022). Bei HTML handelt es sich um eine Sprache zur strukturellen Gestaltung elektronischer Dokumente, so lassen sich Textblöcke, Überschriften, Tabellen, Bilder oder Verweise auf Internetseiten definieren.
CSS hingegen ist für die oberflächliche Gestaltung der Dokumente zuständig. Und kann das Aussehen der einzelnen Elemente, aus denen ein Dokument aufgebaut wird, beeinflussen. So lassen sich zum Beispiel Hintergrundfarben oder Schriftarten darüber festlegen.
JavaScript kommt zum Zuge, wenn die Daten eines Dokuments dynamisch angepasst werden

sollen. Darüber hinaus lassen sich mit JavaScript multimediale Inhalte darstellen und es öffnen sich viele neue Möglichkeiten, wie zum Beispiel die Entwicklung von Apps. Es erlaubt das Speichern, Verarbeiten und dynamische Anzeigen von Werten und vieles mehr.
Für die Entwicklung mit JavaScript sind viele Implementierungen von Funktionalitäten bereits als Bibliotheken vorhanden, sodass Entwickler diese Bibliotheken lediglich in ihre Projekte importieren müssen, um auf die Funktionalitäten zurückgreifen zu können, ohne sie selbst erneut implementieren zu müssen.

2.4 Frameworks

Bündel von Bibliotheken zu Paketen mit noch größerem Funktionsumfang werden Frameworks genannt. Diese Sammlungen von Bibliotheken unterstützen Programmierer bei der Entwicklung von Applikationen, indem sie je nach Anwendungsfall ein grobes Gerüst vorgeben, in welchem dann die Applikation entwickelt werden kann. Beispielsweise kann man sich einen Töpfer vorstellen, bei dem die Drehscheibe das Framework darstellt, mit welchem er alle möglichen Formen von Töpfen herstellen kann.
Die Abgrenzung von Frameworks zu Bibliotheken ist nicht immer eindeutig, da es Bibliotheken mit sehr großem, aber auch Frameworks mit kleinem Funktionsumfang geben kann. Generell lässt sich jedoch sagen, dass es sich bei Frameworks für gewöhnlich um eine Sammlung von Code handelt, welche das Entwickeln von Applikationen in einem bestimmten Gebiet unterstützt. Bibliotheken hingegen sind eher eine Ansammlung von Code, der dazu dient, bestimmte Teilaufgaben einer Applikation zu erledigen.
Moderne JavaScript Frameworks bedienen sich des sogenannten MVC-Designmusters. Es handelt sich dabei um ein dreigeteiltes Modell, bestehend aus „Model“ (zu Deutsch: Datenmodell), „View“ (zu Deutsch: Präsentation) und „Controller“ (zu Deutsch: Programmsteuerung). Das „Model“ steht dabei für den Teil der Applikation, welcher für die Verarbeitung und Bereitstellung der Daten zuständig ist. Als „View“ wird der Teil der Applikation bezeichnet, in welchem die grafische Aufbereitung und Darstellung der Daten für den Benutzer geregelt wird und der „Controller“ ist der Teil, der Eingaben des Nutzers entgegennimmt und in Anweisungen für „Model“ oder „View“ umwandelt.

Im anschließenden Abschnitt dieses Kapitels werden vier Frameworks, mit denen grafische Inhalte in JavaScript generiert werden können, vorgestellt. Zum Vergleich der Frameworks wurden

beispielhafte Implementierungen vorgenommen, die hier ebenfalls zur Veranschaulichung dargestellt werden sollen.

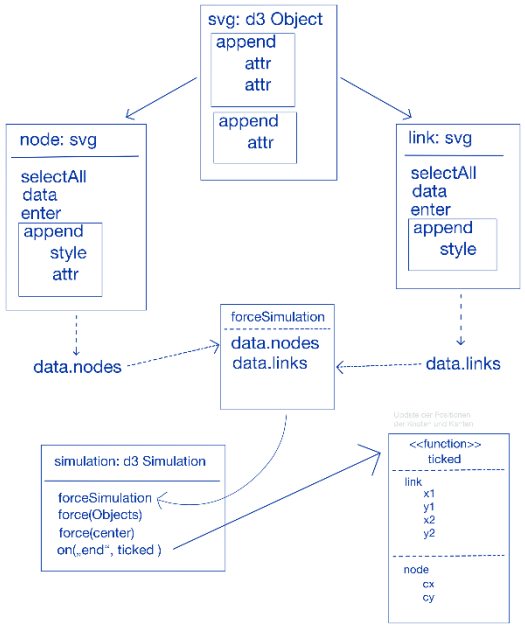

Abbildung 7: Modelldarstellung des D3-Beispiels

2.4.1 D3.js

D3 steht für „Data Driven Documents“ (zu Deutsch: Datengesteuerte Dokumente) und erlaubt es, Daten mit einem Dokument, dem sogenannten „Document Object Model“ in der Webentwicklung zu verknüpfen. Dieses Modell besagt, dass beim Laden einer HTML-Seite alle HTML-Tags in Objekte umgewandelt werden, die einer Eltern-Kind-Hierarchie unterliegen. Dadurch wird die logische Struktur des Dokuments vereinfacht und es ist leichter, die Struktur zu manipulieren und Objekte hinzuzufügen, oder zu entfernen. Es wurde ursprünglich von Mike Bostock und der dahinter stehenden Community entwickelt und am 18. Februar 2011 veröffentlicht. Wie in (Bostock, Data-Driven Documents 2021) beschrieben, ermöglicht D3 es, aus simplen Datenstrukturen wie Listen, oder Arrays Tabellen oder grafische Visualisierungen zu erstellen. Es ist eines der mächtigsten Frameworks, wenn es um Datenvisualisierungen in JavaScript geht und bietet eine Vielzahl an Möglichkeiten, diese umzusetzen. Dabei versucht D3 keineswegs eine Universallösung zu sein und sämtliche Funktionen für jeden möglichen Anwendungsfall bereitzustellen, sondern löst laut Bostock vielmehr auf eine effiziente Art und Weise den Kern des Problems der Manipulation von Dokumenten auf der Grundlage von Daten. Es erlaubt das Arbeiten mit Daten und Formen, ermöglicht das Aufbereiten linearer, hierarchischer, vernetzter oder geografischer Daten und gibt die Möglichkeit, saubere Übergänge zwischen Zuständen der Benutzeroberfläche zu realisieren.

Abbildung 6:Grafische Darstellung des D3-Beispiels

Darüber hinaus ist eine effektive Benutzer-Interaktion möglich, was im Hinblick auf moderne Datenvisualisierung ein großer Vorteil sein kann. Für die Verwendung von D3 ist ein Verständnis für HTML, CSS

```
var margin = {top: 10, right: 30, bottom: 30, left: 40},
  width = 400 - margin.left - margin.right,
  height = 400 - margin.top - margin.bottom;

//Anhängen des SVG-Objektes an den Body der Seite
var svg = d3.select("#flowchart")
.append("svg")
  .attr("width", width + margin.left + margin.right)
  .attr("height", height + margin.top + margin.bottom)
.append("g")
  .attr("transform",
        "translate(" + margin.left + "," + margin.top + ")");

d3.json("https://raw.githubusercontent.com/holtzy/D3-graph-gallery/master/DATA/data_network.json", function( data) {

  //Initialisierung der Verbindungen
  var link = svg
    .selectAll("line")
    .data(data.links)
    .enter()
    .append("line")
      .style("stroke", "#aaa")

  // Initialisierung der Knoten
  var node = svg
    .selectAll("circle")
    .data(data.nodes)
    .enter()
    .append("circle")
      .attr("r", 20)
      .style("fill", "#69b3a2")

  // Auflistung der "Forces", die auf das Netz wirken
  var simulation = d3.forceSimulation(data.nodes)                 // Force Algorithmus für die data.nodes
      .force("link", d3.forceLink()                               // Diese Force sorgt für Verbindungen zwischen den Knoten
            .id(function(d) { return d.id; })                     // Ausgabe der ID eines Knotens
            .links(data.links)                                    // und der Liste seiner Verbindungen
      )
      .force("charge", d3.forceManyBody().strength(-400))         // Hier wird die Abstoßung zwischen den Knoten festgelegt
      .force("center", d3.forceCenter(width / 2, height / 2))     // Anziehung der Knoten zur Mitte der Zeichenfläche
      .on("end", ticked);

  // bei jeder Iteration des FOrce Algorithmus wird diese Funktion aufgerufen und updatet die Positionen der Punkte
  function ticked() {
    link
        .attr("x1", function(d) { return d.source.x; })
        .attr("y1", function(d) { return d.source.y; })
        .attr("x2", function(d) { return d.target.x; })
        .attr("y2", function(d) { return d.target.y; });

    node
        .attr("cx", function (d) { return d.x+6; })
        .attr("cy", function(d) { return d.y-6; });
  }

})
```

Abbildung 8: Implementierung des D3-Beispiels

und JavaScript notwendig, sowie das besagte DOM. Darüber hinaus erfordern viele Visualisierungen ein grundlegendes Verständnis von SVG (zu Deutsch: skalierbare Vektor-Abbildungen). Die Möglichkeiten für Visualisierungen mit D3 sind beinahe grenzenlos, von Animationen über interaktive Grafiken zu Graphen für quantitative Analysen oder hierarchische Abbildungen, vernetzten Darstellungen oder Balken-, Linien-, Punkt- oder Flächendiagrammen oder Karten, mit D3.js lässt sich für nahezu sämtliche Daten eine passende Visualisierung erzeugen (Bostock, D3 Gallery 2020). Dazu ist die Dokumentation sehr umfangreich und es existiert aufgrund der weiten Verbreitung von D3 eine große Community, die bei Fragen über die Plattformen „Stack Overflow“, „Slack“, „Google Groups“ oder „Glitter“ aushilft. Das D3-eigene Wiki ist neben Englisch in 11 weiteren Sprachen verfügbar, darunter Spanisch, Russisch und Chinesisch. Es unterstützt in der aktuellen Version 5+ die Browser Chrome, Edge, Firefox und Safari.

In der in Abbildung 8 abgebildeten Implementierung wird mithilfe von Daten aus einer JSON-Datei[1], in welcher Knoten und Kanten definiert sind, ein Diagramm erstellt, in dem diese zusammenhängend abgebildet werden. Bei JSON handelt es sich um ein Datenformat, welches für die Speicherung und den Austausch von strukturierten Daten zwischen Anwendungen eingesetzt wird und insbesondere bei der Programmierung mit JavaScript viel Verwendung findet. Zum besseren Verständnis der Implementierung wurden die funktionalen Zusammenhänge in Abbildung 7 in einer Modelldarstellung zusammengefasst und die generierte Grafik in Abbildung 6 abgebildet.

In diesem Beispiel wird das Zusammenspiel von D3 und Daten dargestellt, aus denen grafische Inhalte generiert werden, welche im Dokument in dem Element mit der ID „flowchart“ (Zeile 6) dargestellt werden. Dieses Element befindet sich innerhalb einer HTML-Datei, von welcher aus das hier abgebildete JavaScript-Skript aufgerufen wird.

Zu Beginn der Ausführung in Zeile 1-3 werden die grundlegenden Parameter der Visualisierung festgelegt, wie der äußere Abstand zu anderen Elementen, die Breite und Höhe der Darstellung. Im Anschluss daran wird ein SVG-Element erstellt, welchem dann die Parameter übergeben werden, die nötig sind, um es auf der übergeordneten Seite mit den zuvor definierten Abmessungen am richtigen Element anzuzeigen.

Daraufhin werden für jedes „line“- und „circle“-Element der Daten aus der JSON-Datei Linien und Kreise zu dem SVG hinzugefügt. Mit Hilfe des d3-force Moduls werden die zuvor hinzugefügten Punkte um den Mittelpunkt der Grafik arrangiert, wobei sie sich gegenseitig abstoßen, sodass sie sich kreisförmig um die Mitte anordnen. Sobald die Funktion am Ende angekommen ist, wird die „ticked“-Funktion aufgerufen, über die die Start- und Endposition der Kanten und Positionen der Knoten festgelegt werden, sodass die Knoten durch die Kanten verbunden und die in Abbildung 6 dargestellte Grafik erzeugt werden kann.

Wie in Abbildung 8 zu erkennen, entsteht so ein SVG-Element, dem Attribute hinzugefügt wurden, zu dem „nodes“ und „links“ gehören, die sich aus den Daten der JSON-Datei ergeben. Diese erhalten über d3-force und die daraus aufgerufene „ticked“-Funktion die zugehörigen Koordinaten-Werte.

An dem Beispiel lässt sich erkennen, dass mit geringem Aufwand komplexe Zusammenhänge

[1] https://raw.githubusercontent.com/holtzy/D3-graph-gallery/master/DATA/data_network.json

zwischen dynamischen Daten visualisiert werden können, da auf D3-interne Funktionen zurückgegriffen werden kann.

2.4.2 Two.js

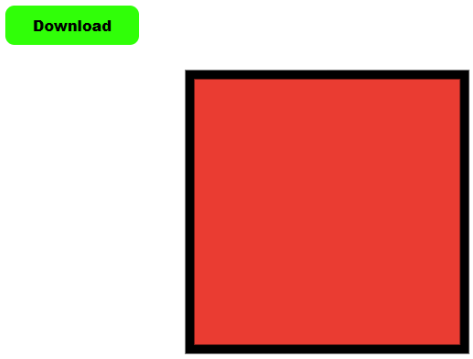

Abbildung 9: Grafische Darstellung des TWO.js-Beispiels

TWO (zu Deutsch: Zwei) ist, wie der Name vermuten lässt, spezialisiert auf das Erstellen und Animieren zweidimensionaler Objekte. Wie von (Chien 2022) beschrieben, setzt TWO dabei auf den sogenannten Szenengraphen, bei dem es sich um eine objektorientierte Datenstruktur handelt, die zur Darstellung zweidimensionaler Objekte handelt. Im Szenengraphen werden die zu zeichnenden Objekte gespeichert und können so nach ihrer Erstellung beliebig manipuliert werden, zum Beispiel durch Drehung, Verschiebung oder Skalierung. Darüber hinaus verfügt TWO über eine Animationsschleife, die nach Chien einfach zu automatisieren oder mit anderen Animationsbibliotheken kombiniert werden kann. Außerdem verfügt TWO über einen SVG-Interpreter, über den sich extern generierte SVG Elemente zu einer TWO-Szene hinzugefügt werden können. Obwohl sich TWO ursprünglich auf das vereinfachte Erzeugen und Animieren von SVG fokussierte, verfügt es heute ebenfalls über eine Palette an Funktionen, die das Erzeugen und die Arbeit mit Bitmap Bildern ermöglicht.

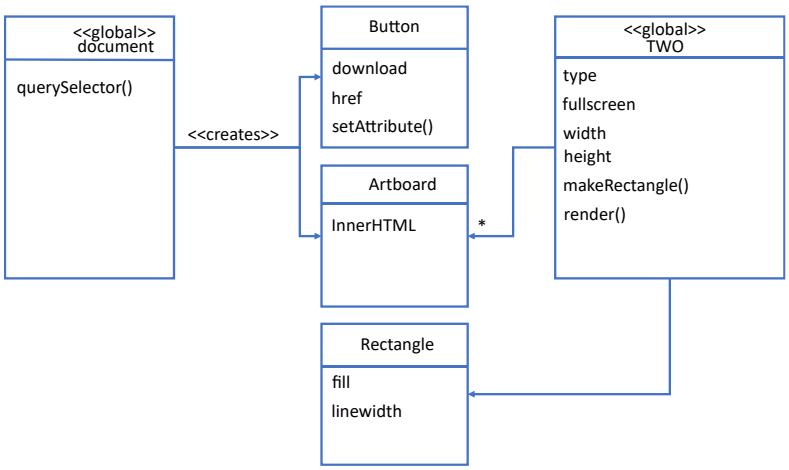

Abbildung 10: Modelldarstellung des TWO.js-Beispiels

```
import Two from 'https://cdn.skypack.dev/two.js@latest';

let artboard = document.querySelector('#artboard');
//Anhängen eines neuen TWO-Elements an das Dokument
let two  = new Two({
  type: Two.Types.svg,
  fullscreen: true
}).appendTo(artboard);
//Manipulation des Objektes nachd er erzeugung
let rectangle = two.makeRectangle(two.width / 2, two.height / 2, 300, 300);
rectangle.fill = 'rgb(234, 60, 50)';
rectangle.linewidth = 10;
//Zeichnen des TWO-Elements
two.render();

//Link aus Metadaten extrahieren und URL zum Herunterladen des Objektes generieren
var button = document.querySelector('#download-btn');
var blob = new Blob([artboard.innerHTML], { type: "image/svg+xml;charset=utf-8" });
var href = URL.createObjectURL(blob);
button.setAttribute('download', 'two-js-scene.svg');
button.setAttribute('href', href);
```

Abbildung 11: Implementierung des TWO.js-sBeispiels

In der in Abbildung 11 dargestellten Beispiel-Implementierung für eine mit TWO erzeugte Visualisierung wird ein SVG-Element erzeugt, welches im Anschluss daran über eine Schaltfläche heruntergeladen werden kann. Dazu werden in der übergeordneten HTML-Datei ein Element mit der ID „artboard“ und eine Schaltfläche mit der ID „download-btn“ benötigt, die mithilfe des hier abgebildeten JavaScripts manipuliert werden können. Wie bereits erwähnt, wird für das SVG-Element zunächst ein TWO Element erzeugt, dem der Typ „Two.Types.svg“ zugewiesen wird. Dieses Element wird dann an den Szenegraphen mit dem Namen „artboard“ angehängt. Anschließend wird das Element wie für TWO typisch manipuliert. Es wird als Rechteck mit Position, Höhe und Breite definiert und mittig auf der Zeichenfläche angeordnet, die Füllungsfarbe und die Breite der Begrenzungslinie wird festgelegt. Im Anschluss daran wird das Element gezeichnet und die Schaltfläche mit der Funktionalität versehen, die nötig ist, um die erzeugte Grafik herunterzuladen. Dafür wird aus den Metadaten des Szenegraphen eine URL generiert, über die sich die Grafik herunterladen lässt und der Name der herunterladbaren Datei wird festgelegt. Die beiden Parameter werden anschließend mit der Schaltfläche verknüpft. Das Beispiel zeigt, dass sich simple Visualisierungen mit geringem Aufwand realisieren lassen und ebenfalls eine Datenverarbeitung mit TWO möglich ist.

2.4.3 Zdog.js

Im Gegensatz zu den beiden vorangegangenen JavaScript-Bibliotheken, die sich hauptsächlich auf zweidimensionale Darstellung fokussieren, ist Zdog auf dreidimensionale Darstellungen ausgelegt. Dabei handelt es sich dahingegen nicht um dreidimensionale Darstellungen, sondern viel mehr um Pseudo-Dreidimensionalität. Die Geometrie der erzeugten Elemente ist im dreidimensionalen Raum abbildbar, jedoch werden sie als flache zweidimensionale Formen generiert. Zdog wurde entwickelt, um die Illustration von Vektoren im dreidimensionalen Raum zu ermöglichen und fügt zum Zeichnen von zweidimensionalen Kreisen und Vierecken eine weitere Dimension hinzu (DeSandro 2019).

Abbildung 13: Grafische Darstellung des Zdog-Beispiels

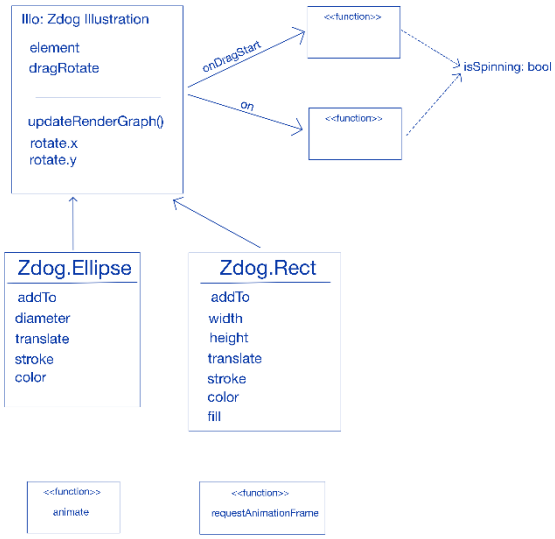

Abbildung 12: Modelldarstellung des Zdog-Beispiels

```
let isSpinning = true;

    let illo = new Zdog.Illustration({
      element: '.zdog-canvas',
      dragRotate: true,
      // Wenn das Objekt geklickt wird, die Animation unterbrechen
      onDragStart: function() {
        isSpinning = false;
      },
      on: function() {
        isSpinning = true;
      },
    });

    // Kreis
    new Zdog.Ellipse({
        addTo: illo,
        diameter: 80,
        // Den Kreis vor dem Quadrat anordnen
        translate: { z: 40 },
        stroke: 20,
        color: '#636',
      });

      // Quadrat
      new Zdog.Rect({
        addTo: illo,
        width: 80,
        height: 80,
        // Das Quadrat hinter dem Kreis anordnen
        translate: { z: -40 },
        stroke: 12,
        color: '#E62',
        fill: true,
      });

      illo.updateRenderGraph();

      function animate() {
        // für jeden Frame weiterdrehen
      if ( isSpinning ) {
        illo.rotate.y += 0.03;
      }
        illo.updateRenderGraph();
        // den nächsten Frame animieren
        requestAnimationFrame( animate );
      }
     //Animation starten
      animate();
```

Abbildung 14: Implementierung des Zdog-Beispiels

Die in Abbildung 14 dargestellte beispielhafte Implementierung von Zdog erzeugt zwei Körper, abgebildet in Abbildung 13, die sich entlang einer vertikalen Achse umkreisen. Um das Verständnis zu erleichtern, wurde auch hier mit Abbildung 12 eine Modellansicht parallel zum Code geschaffen.

Zunächst wird eine Zdog Illustration erstellt, die an ein HTML-Dokument, welches über ein Element mit der ID „zdog-canvas“ verfügt, angehängt werden kann. Der Illustration werden beim Erstellen ebenfalls Parameter zugewiesen, die das Drehen der Illustration durch Anklicken und Bewegen des Mauszeigers erlauben. Gleichzeitig wird ein Flag, welches die Drehung der

Illustration erlaubt oder verbietet, beim Laden ebendieser gesetzt, sodass sich der Inhalt der Illustration nach dem Laden dreht, jedoch anhält, sobald sie durch Nutzereingaben gedreht wird.

Im Anschluss daran werden der Illustration zwei Objekte hinzugefügt, ein Kreis in Form einer „Zdog.Ellipse“ und ein Quadrat in Form eines „Zdog.Rect“. Der Kreis wird entlang der z-Achse um 40 Einheiten verschoben, während das Quadrat um -40 Einheiten auf der z-Achse verschoben wird, wodurch die Objekte zu Beginn der Animation hintereinander erscheinen. Die Drehung wird in diesem Fall durch die Funktion „animate()“ in Zeile 39 realisiert, die am Ende des Scripts in Zeile 49 aufgerufen wird und sich am Ende eines Durchlaufs wieder selbst aufruft. In ihr wird die gesamte Illustration in jedem Frame um 0,03 Einheiten entlang der y-Achse rotiert, solange das „isSpinning“-Flag gesetzt ist, bevor die Grafik erneut gezeichnet wird und die „animate()“-Funktion sich für das Errechnen des nächsten Bildes erneut aufruft. Anhand des Beispiels ist zu erkennen, dass sich mit Zdog mit wenig Aufwand simple Animationen von einfachen dreidimensionalen Körpern umsetzen lassen.

2.4.4 Pts.js

Bei Pts handelt es sich ebenfalls um eine Bibliothek zur Umsetzung von Visualisierungen, jedoch legt Pts den Fokus auf Punkte, wie der Name nahelegt („Pts“ kurz für „Points“ zu Deutsch: Punkte). Pts baut auf der Abstraktion von Raum, Formen und Punkten auf und stellt Bezüge dazwischen grafisch dar. Es stellt die essenziellen Bausteine für kreatives Programmieren und Visualisieren zur Verfügung und eignet sich in erster Linie dazu, aus Ansammlungen von Punkten grafische Darstellungen zu erstellen. Hierbei unterstützt zum Beispiel das Pt Modul, welches Pt- und Group-Klassen bereitstellt, die die Manipulation von Punkten ermöglichen. Zwei weitere Module sind die „Op“- und „Num“-Module, welche eine Vielzahl von Algorithmen für numerische und geometrische Operationen bereitstellen. Darüber hinaus stellen das „Canvas“- und „SVG“-Modul die Klassen für das

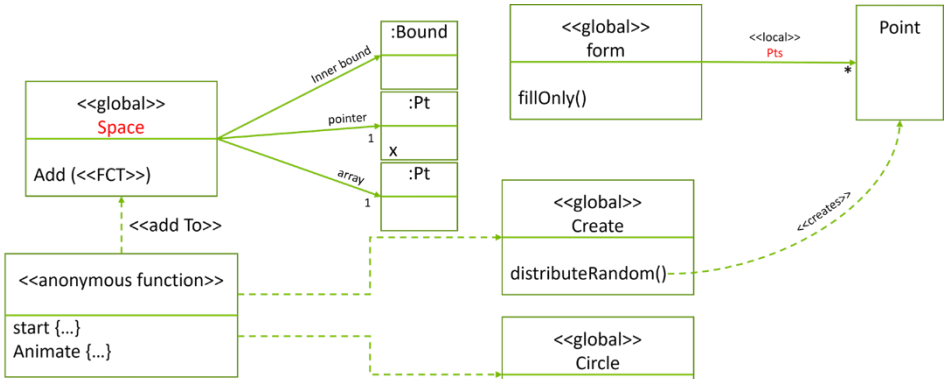

Abbildung 15: Modelldarstellung des Pts-Beispiels

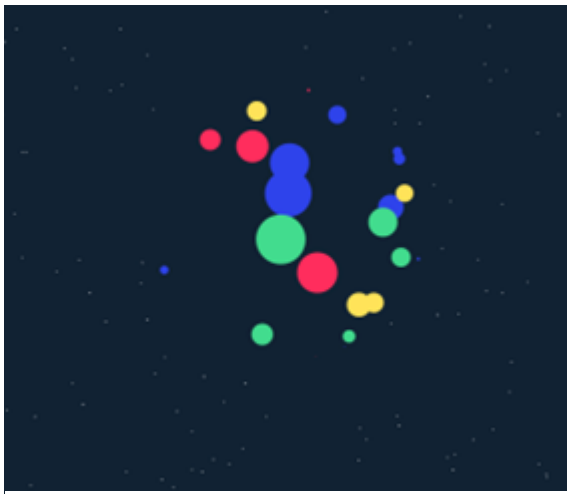

Abbildung 17: Grafische Darstellung des Pts-Beispiels

Arbeiten mit Canvas oder SVG zur Verfügung, wie in (Ngan 2021) beschrieben.

Die Beispielimplementierung für Pts in Abbildung 16 erstellt die in Abbildung 17 dargestellte Visualisierung. Dabei wird eine Ebene mit Punkten gefüllt, welche beim Überfahren der Ebene mit dem Mauszeiger kreisförmig um den Zeiger ihre Größe ändern. Um diesen Effekt zu erzielen, wird zunächst, wie ebenfalls in der Modelldarstellung der

```
var pts = [];
var colors = ["#ff2d5d", "#42dc8e", "#2e43eb", "#ffe359"];

space.add( {

    // Mit 100 Punkten initialisieren
    start: (bound) => { pts = Create.distributeRandom( space.innerBound, 100 ); },

    animate: (time, ftime) => {

    let r = Math.abs( space.pointer.x-space.center.x )/space.center.x * 150 + 70;
    let range = Circle.fromCenter( space.pointer, r );

    // kontrolliere, ob der Punkt in der Reichweite des Kreises liegt
    for (let i=0, len=pts.length; i<len; i++) {

        if ( Circle.withinBound( range, pts[i] ) ) {

        // Größe der einzelnen Kreise berechnen
        let dist = (r - pts[i].$subtract(space.pointer).magnitude() ) / r;
        let p = pts[i].$subtract( space.pointer ).scale( 1+dist ).add( space.pointer );
        form.fillOnly( colors[i%4] ).point( p, dist*25, "circle" );

        } else {
        form.fillOnly("#fff").point(pts[i], 0.5);
        }

    }
    }

});


space.bindMouse().bindTouch().play();
```

Abbildung 16: Implementierung des Pts-Beispiels

Implementierung in Abbildung 15 zu erkennen, auf ein globales „Space“-Element zurückgegriffen, welchem über die anonyme Start-Funktion eine Menge an zufällig generierten Punkten hinzugefügt wird, die sich über die Fläche der Grafik verteilen. Die zweite anonyme Funktion zur Animation des zuvor beschriebenen Effektes errechnet für jeden einzelnen Punkt je nach Distanz zum Mauszeiger die Größe und Farbe der Punkte.

Über die Funktionen „bindMouse()“ und „bindTouch()“ wird im Anschluss die Position des Mauszeigers verfolgt und mit der „play()“ Funktion in einem verketteten Funktionsaufruf die in „space“ abgebildete Visualisierung dargestellt. An dieser Implementierung kann man erkennen, wie mit Pts.js Visualisierungen auf der Grundlage von Punkten erstellt werden können und Bezüge zwischen Punkten und hier dem Mauszeiger dargestellt werden können.

2.5 Zusammenfassung und Vergleich

	D3.js	Two.js	Zdog.js	Pts.js
Schwerpunkt	Datenvisualisierung allgemein	Erstellen und Animieren von zweidimensionalen Formen	Dreidimensionale grafische Darstellungen	Punkte, Formen, Farben, Bewegungen und Interaktionen
Popularität auf GitHub	103.000 Sterne 23.200 Forks 123 Contributors	7.800 Sterne 470 Forks 43 Contributors	9.300 Sterne 366 Forks 5 Contributors	4.800 Sterne 174 Forks 18 Contributors
Release: Erstes – Letztes Anzahl Releases	17.07.2013 - 03.07.2022 186	08.01.2014 - 01.06.2022 28	04.06.2019 - 23.10.2019 5	21.08.2017 - 19.02.2021 13
Vorschläge für mögliche Visualisierungen	168	46	22	38
Umsetzung komplexer Visualisierungen	Ja, vergleichsweise geringer Aufwand durch zahlreiche vorhandene Funktionen	Ja, allerdings aufwändig, da Funktionen meist nicht vorgegeben werden	Nicht ohne größeren Aufwand möglich	Ja, allerdings aufwändig, da Funktionen meist nicht vorgegeben werden
Lernkurve	Steil, durch die Menge an Möglichkeiten, Visualisierungen umzusetzen	Moderat, da stufenweise erst eine Begrenzte Zahl an Funktionalitäten umgesetzt werden kann	Flach, da der Funktionsumfang gering ausfällt	Moderat, da Funktionsumfang begrenzt, aber erweiterbar

Tabelle 1: Zusammenfassung der Visualisierungsbibliotheken

Anhand der vorangegangenen Abschnitte dieses Kapitels wurden die Hauptmerkmale der Visualisierungsbibliotheken in Tabelle 1 zusammengefasst. Zusätzlich zu den Merkmalen der Bibliotheken, die zuvor beschrieben wurden, wurden Daten bezüglich der Verbreitung der Bibliotheken hinzugefügt, damit abgeschätzt werden kann, wie umfangreich die Dokumentation jeweils ausfällt und wie groß die jeweiligen Communitys sind, die hinter den Bibliotheken stehen. Mit Abstand am weitesten verbreitet ist D3. Es befindet sich am längsten in der Entwicklung und verfügt über eine große Anzahl von Entwicklern, die Softwareentwicklung an oder mit D3 durchführen. Wie bereits in Kapitel 2.4.1 beschrieben, ist der Support bei D3 umfangreich und die Dokumentation ausführlich. Bei der Dokumentation sieht es glücklicherweise auch bei den weniger

populären Bibliotheken gut aus und so gibt es für die meisten umsetzbaren Visualisierungen für jede der Bibliotheken gut dokumentierte Beispiele und Anleitungen, die bei der Entwicklung unterstützen. Bei einem Blick auf die Zeile „Vorschläge für mögliche Visualisierungen“ fällt allerdings wieder auf, dass D3 mit Abstand vorne liegt. Das liegt vor allem daran, dass D3 im Gegensatz zu den anderen drei Bibliotheken aufgrund seines enormem Funktionsumfanges flexibel bei der Umsetzung von Visualisierungen ist. Die Beispielseiten der einzelnen Bibliotheken geben jeweils einen guten Überblick darüber, was für Visualisierungen sich damit umsetzen lassen und bieten gute Einstiegspunkte, um die eine oder andere Visualisierungsform für das umzusetzende Projekt auszuprobieren, um dann eine Entscheidung treffen zu können, welche Visualisierung sich am besten eignet. In der obersten Zeile der Tabelle sind die Schwerpunkte der jeweiligen Bibliotheken aufgelistet, damit man auf den ersten Blick einschätzen kann, ob eine der Bibliotheken für den gegenwärtigen Anwendungsfall geeignet ist oder nicht, um dann zu überlegen, wie wichtig Popularität oder das Umsetzen komplexer Visualisierungen für das Projekt ist. D3 ist hier allgemein gehalten, da sich alle möglichen Arten von Visualisierungen damit umsetzen lassen, wohingegen die anderen drei Bibliotheken einen stärker fokussierten Schwerpunkt haben. Two.js ist besonders für das Erstellen und Animieren zweidimensionaler Formen geeignet, wohingegen Zdog.js den Schwerpunkt auf dreidimensionale Darstellungen legt und Pts.js wieder etwas universeller einsetzbar, jedoch mit Fokus auf das Erstellen von Visualisierungen anhand von Punkten und deren Zusammenhängen. Bei der Umsetzung komplexer Visualisierungen tut Zdog hervor, da die Bibliothek dies nicht wirklich vorsieht und erheblicher Aufwand unternommen werden muss, um komplexe dreidimensionale Visualisierungen zu erzeugen, die nicht nur aufwändig zu programmieren, sondern auch ressourcenhungrig bei der Generierung sind. Mit Two.js und Pts.js lassen sich vergleichsweise komplizierte Visualisierungen umsetzen, wenn bei der Programmierung entsprechender Aufwand betrieben wird. Mit dem geringsten Aufwand lassen sich komplexe Visualisierungen mit D3 umsetzen, da es über eine große Anzahl von Hilfsfunktionen verfügt, die bei der Softwareentwicklung unterstützen. Zum Abschluss des Vergleichs kann der Aufwand verglichen werden, der betrieben werden muss, um den Umgang mit den einzelnen Bibliotheken zu erlernen. Dieser ist für Zdog.js am geringsten, da durch die begrenzte Menge an Funktionen der Zugang einfach ausfällt. Das Mittelfeld bilden Two.js und Pts.js, die zu Beginn einfach zu erlernen sind, aber durch das Hinzufügen von Animationen und komplexeren Visualisierungen einen erhöhten Anspruch beim Erlernen stellen. Das Schlusslicht bildet D3.js, da hier aufgrund der Menge an Funktionalitäten und der damit einhergehenden

Möglichkeiten, Visualisierungen umzusetzen, die Lernkurve am steilsten ausfällt. Es bedarf daher aufgrund der Menge des zu Erlernenden größeren Aufwands die Funktionen von D3 zu erlernen.

3. Lösungsansatz

Um mithilfe der Auswahl des korrekten Frameworks oder der richtigen Bibliotheken ressourcenschonend Visualisierungen passend zu Beweisen umzusetzen zu können, muss der Prozess des Umsetzens und Bereitstellens einer Visualisierung betrachtet werden. Nachdem man sich für eine grafische Visualisierung für eine Datenmenge oder einen Sachverhalt entschieden hat, ist es wichtig, das richtige Werkzeug, in diesem Fall die richtige Bibliothek auszuwählen, um so effizient wie möglich das gewünschte Resultat in Form der programmierten Visualisierung zu erzielen.
Wird ein Framework oder eine Bibliothek gewählt, die nicht zur gewünschten Visualisierung passt, muss erheblicher Aufwand betrieben werden, mit den zur Verfügung stehenden Mitteln das angestrebte Resultat zu erzielen.
Darüber hinaus ist es ebenfalls wichtig, die richtige Form der Bereitstellung der Ergebnisse zu wählen, sodass für den Fall, dass ein Projekt im Team durchgeführt wird, eine Kollaboration zwischen den einzelnen Mitgliedern der Gruppe möglich ist. So ist es wichtig, dass soweit möglich, auf Entwicklungsumgebungen, Frameworks und Bibliotheken zurückgegriffen wird, mit denen die Mehrheit des Teams vertraut ist, sodass der Aufwand für die Einarbeitung der einzelnen Mitglieder minimiert werden kann.
In diesem Kapitel soll der Weg der Entscheidungsfindung für ein Beispiel erläutert und der Entwurf einer Lösung unter Berücksichtigung der gegebenen Rahmenbedingungen erklärt und begründet werden.

3.1 Vorbereitung

Zu Beginn der Arbeit wurde eine Sammlung von JavaScript Bibliotheken erstellt, für die dann Testimplementierungen umgesetzt wurden, um die Komplexität der einzelnen Pakete besser einschätzen zu können. Zusätzlich dazu wurde eine Seite mit Implementierungen simpler Formen bereitgestellt, aus denen sich Visualisierungen zusammensetzen konnten. Diese Seiten wurden auf einen Webserver geladen, sodass stets darauf zugegriffen werden konnte (Jacobsen 2022). So

entstand eine Übersicht über die Bibliotheken und zusätzlich zu den Beispielen wurde der kommentierte Quellcode samt einer Beschreibung und Modelldarstellung bereitgestellt. Die Implementierung fand überwiegend in HTML statt, weshalb mit wachsendem Umfang und Komplexität der Beispiele klar wurde, dass für die Umsetzung der Visualisierung, die im Rahmen dieser Arbeit geschehen sollte, auf ein JavaScript-Framework zurückgegriffen werden muss.

3.2 Randbedingungen

In diesem Abschnitt des dritten Kapitels werden die Randbedingungen, die durch die Eingliederung der Arbeit in eine Reihe von Arbeiten gegeben sind, dargestellt und eine Entscheidung für eine Visualisierungsbibliothek herbeigeführt.

3.2.1 Wahl des Frameworks

Da sich die Bereitstellung der Visualisierung über einen Webserver für die Eingliederung der Arbeit in eine Reihe von Arbeiten als nicht praktikabel erwies, wurde beschlossen, die Implementierung mithilfe eine JavaScript Frameworks umzusetzen. Der Grund dafür war, dass die Anwendung auch ohne die Verwendung eines Webservers genutzt und weiterentwickelt werden sollte. Daher wurde die Entwicklung auf Node.js umgestellt, wobei es sich um eine plattformübergreifende Open-Source-JavaScript-Laufzeitumgebung handelt, die das Ausführen von JavaScript Code außerhalb des Browsers ausgeführt werden und somit ein lokaler Webserver für die Entwicklung betrieben werden kann. Darauf aufbauend wird ein JavaScript-Framework benötigt, in welchem die Applikation, welche die Visualisierung umsetzen soll, entwickelt wird. Zur Wahl standen Angular, React und VUE, auf deren Set und Features im folgenden Abschnitt eingegangen werden soll.

Angular

Angular (Google 2022) ist ein von Google entwickeltes Framework, welches eine solche Fülle an Funktionalitäten umfasst, dass es eher eine Plattform als ein Framework ist. Das Angular-Ökosystem reicht mit seinem CLI und seiner progressiven Web App Unterstützung über den Code hinaus. Aufgrund seines Funktionsumfanges gibt es wenige Anwendungsfälle, für die Angular nicht über eine eingebaute Lösung verfügt. Der Nachteil dieser umfassenden Ausstattung ist allerdings, dass Angular sehr anspruchsvoll zu erlernen ist.

React

Als Gegensatz dazu gibt es React (Meta-Platforms 2022), welches von Meta entwickelt wird. Es ist eher minimalistisch ausgelegt und bezeichnet sich daher selbst als Bibliothek. Für viele Funktionalitäten muss man zusätzliche Pakete installieren, die von der Community hinter React bereitgestellt werden, da sie in Standard-React nicht enthalten sind. Der minimalistische Ansatz von React kann gerade mit Blick auf kleinere bis mittelgroße Projekte von Vorteil sein, da React nicht mit Funktionen überladen und so einfacher zu erlernen ist. Jedoch bringt er auch den Nachteil mit sich, dass Softwareprojekte davon abhängig sein können, dass die Entwickler von Zusatzpaketen für Kompatibilität mit neuen Versionen von React sorgen.

VUE

Bei VUE hingegen handelt es sich um ein alleinstehendes Open-Source-Projekt (You 2022). Es liegt in etwa in der Mitte zwischen React und Angular, da es viele Funktionalitäten enthält, aber nicht in dem Ausmaß, wie es bei Angular der Fall ist. Es fokussiert sich auf die Kern-Funktionalitäten, die zum Schreiben von Code benötigt werden, kommt mit einem Router und einer eingebauten Lösung für State-Management, die vom Kern-VUE-Team verwaltet wird. Es verfügt aber zum Beispiel nicht über eine standardmäßig verbaute Form-Validierung, die stattdessen wie bei React von der Community entwickelt und verwaltet wird. Daher umfasst es weniger Funktionen als Angular, aber etwas mehr Funktionalitäten als React.

Funktionalität	Angular	React	VUE
Benutzeroberflächen-Manipulation	Ja	Ja	Ja
State Management	Ja	(Ja)	Ja
Routing	Ja	Nein	Ja
Form Validation	Ja	Nein	Nein

Tabelle 2: Zusammenfassung der technischen Unterschiede und Gemeinsamkeiten von Angular, React und VUE

Zusammenfassung

Alle drei Frameworks haben gemeinsam, dass sie im Kern interaktive Benutzeroberflächen aus wiederverwendbaren Komponenten erstellen. Jedoch unterscheiden sie sich nicht nur in der Ausstattung mit Funktionalitäten, sondern auch in der Art und Weise, wie sich der Code von Applikationen, die mithilfe dieser Frameworks entwickelt werden, zusammensetzt.

In VUE wird Standard-JavaScript und HTML-Code mit einigen kleinen Markierungen verwendet, über die VUE die JavaScript- und HTML-Komponenten verknüpfen kann. Das bedeutet, dass es in den Dateien Passagen gibt, die in HTML geschrieben sind und in JavaScript geschriebene Abschnitte, die zu dem HTML-Code gehören. So lassen sich die aus dem HTML-Code generierten Komponenten durch den JavaScript-Code manipulieren, indem VUE die Zusammenhänge zwischen den aus dem HTML-Code resultierenden Komponenten und den dazugehörigen JavaScript-Komponenten herstellt.

Auch in Angular findet eine strikte Trennung zwischen HTML und JavaScript statt. Hier werden ebenfalls HTML-Masken verwendet, zu denen bestimmte Instruktionen hinzugefügt werden können, die daraufhin vom Framework interpretiert werden können, wie zum Beispiel das Warten auf einen Mausklick. Hinzu kommt dann der JavaScript- oder TypeScript-Code, der mit der HTML-Maske kommunizieren und diese manipulieren kann.

Gänzlich anders hingegen sieht die Syntax in React aus. Dort gibt es keine Trennung von JavaScript- und HTML-Code, stattdessen besteht die gesamte Anwendung aus JavaScript und Komponenten werden als JavaScript-Funktionen dargestellt. Man spricht dabei von der sogenannten JSX-Syntax, die ein besonderes Merkmal von React darstellt, welches erlaubt, im JavaScript-Code zu definieren, welche HTML-Komponenten unter welchen Bedingungen an welcher Stelle des zu generierenden Dokumentes angezeigt werden sollen. Dadurch entsteht in React eine Mischung von JavaScript- und HTML-Code.

Auch mit Blick auf die Komplexität einer Implementierung und dem Aufwand, der zum Erlernen der Frameworks betrieben werden muss, unterscheiden diese sich. Da Angular ein komplexes Set-up benötigt und wie bereits beschrieben umfangreich in seinen Funktionen ist, kann es zwischen den drei als das am schwersten zu erlernende Framework angesehen werden. Der Aufwand, der betrieben werden muss, um selbst ein simples Projekt mit Angular zu realisieren, ist daher größer als bei den anderen beiden Frameworks.
React benötigt zwar theoretisch kein komplexes Projekt-Set-up, erfordert für die Verwendung der JSX-Syntax jedoch eine komplexere Einrichtung als einen einfachen Import in ein Web-Projekt.

Beide Frameworks verfügen jedoch über Werkzeuge, die diese Einrichtung automatisiert durchführen können.

VUE ist im Gegensatz zu den anderen beiden Frameworks leichter zu erlernen und anzuwenden, da sich einfache Projekte ohne komplexes Einrichten umsetzen lassen. Für größere Projekte mit erhöhter Komplexität ist es jedoch auch möglich, wie bei den anderen beiden Frameworks mithilfe von Bordmitteln von VUE ein komplexes Set-up zu generieren.

Hinsichtlich ihrer Leistung unterscheiden sich die Frameworks nur unwesentlich, da ungenutzte Bestandteile der Frameworks in das schlussendlich gebaute Projekt nicht integriert werden. Daher macht es leistungstechnisch keinen großen Unterschied, ob React mit seiner minimalistischen Ausstattung verwendet wird, oder Angular oder VUE, die standardmäßig viele Funktionalitäten mitbringen.

Schaut man sich die Popularität der drei Frameworks an, so ergibt sich folgendes Bild.

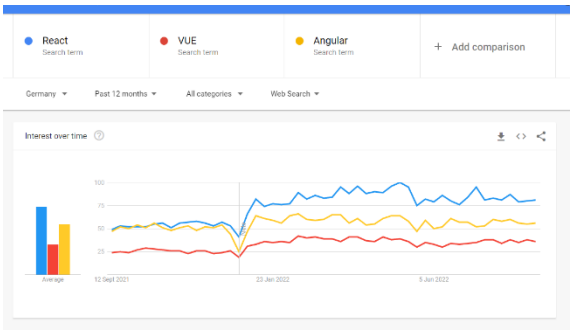

Abbildung 18: Google Trends der letzten 12 Monate - React (blau), VUE (rot), Angular (gelb)

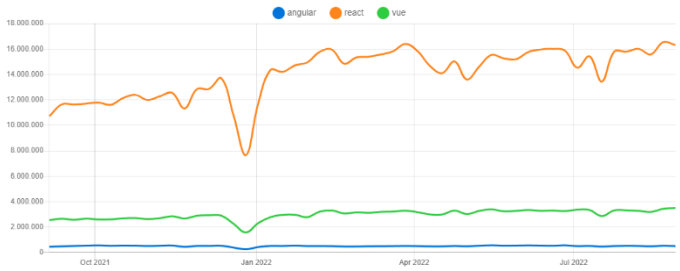

Abbildung 19: npm Downloads der letzten 12 Monate - Angular (blau), React (orange), VUE (grün)

Es ist zu erkennen, dass sich sowohl in den Google Trends für Deutschland, die die Menge an Suchanfragen zu den jeweiligen Suchbegriffen wiedergeben in Abbildung 18, als auch in den npm Downloads in Deutschland, die wiedergeben, wie oft ein Paket heruntergeladen wurde in Abbildung 19, React mit Abstand vor Angular und VUE liegt. Untermauert wird dies durch eine Suchanfrage zu Jobangeboten auf der Seite Indeed (Indeed 2022) (Zugriff am 3.9.2022), nach der für React 12.360 Jobangebote in Deutschland existieren, für Angular 9.754 Angebote und für VUE 5.478. So lässt sich erkennen, dass React das populärste Framework in Deutschland ist, während Angular auf Platz 2 liegt und VUE den Schluss in diesem Vergleich bildet.

3.3 Wahl der Umsetzung

In der Vorbereitung dieser Arbeit wurde ein Konzept für eine Visualisierung erarbeitet, die beim Verständnis des Ergebnisses von VeriFast unterstützen soll. In der Visualisierung sollen die Gültigkeitsbereiche in Intervallform übereinandergelegt werden, sodass am Ende ein Bereich auf

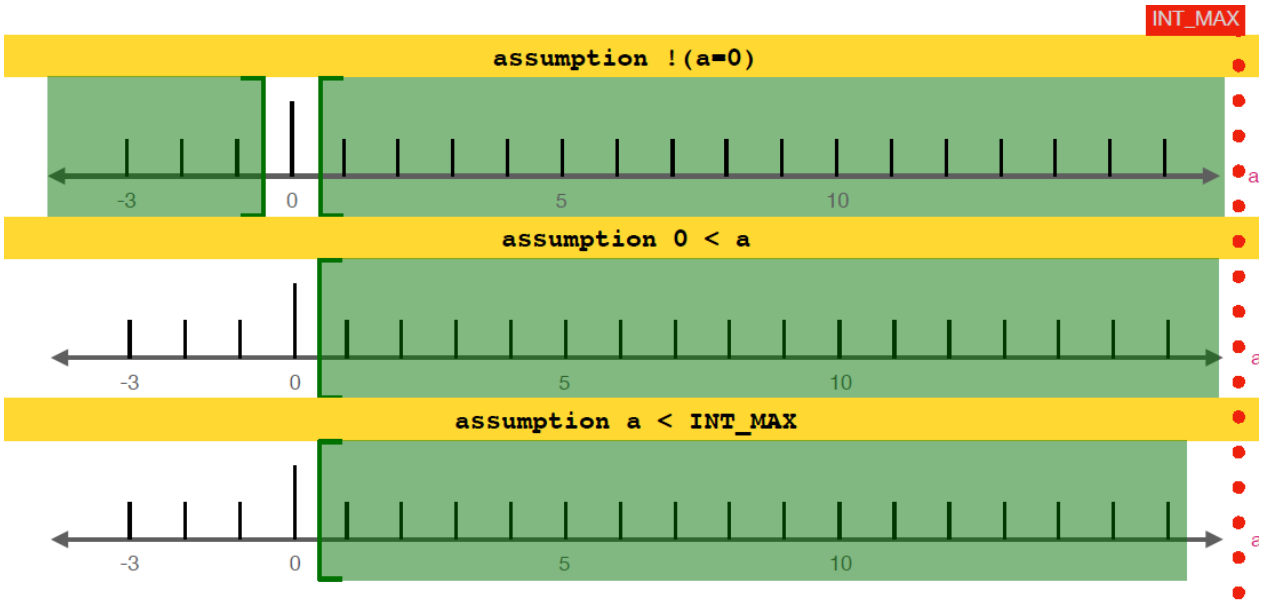

Abbildung 20: Visualisierung der einzelnen Gültigkeitsbereiche von a (by Judith Lippold)

einem Zahlenstrahl ausgegeben werden kann, in dem die zu verifizierende Funktion erfolgreich bewiesen werden kann.

Da es sich hier um eine zweidimensionale Datenvisualisierung mittels eines Zahlenstrahls handelt, liegt die Wahl von D3.js als Visualisierungsbibliothek nahe. Es wäre zwar auch eine Umsetzung mit Two.js möglich, jedoch kann bei D3 auf Hilfsfunktionen zum Erstellen des Zahlenstrahls zurückgegriffen werden, wodurch Ressourcen gespart werden können. Zudem lassen sich mit D3 die Gültigkeitsbereiche dynamisch generieren und über die Manipulation eines SVG anzeigen. Die umfangreiche Dokumentation und insbesondere die Vorschläge für Visualisierungen mit D3 glichen den Nachteil des Funktionsumfanges aus, sodass ein schneller Einstieg mit dieser Bibliothek möglich war.

Die Auswahl des Frameworks hing einerseits von der Komplexität der Anwendung ab, da die Visualisierung eines Zahlenstrahls kein Routing erforderte und voraussichtlich auch das State-Management keine große Rolle in der Implementierung spielen würde, musste sich zwischen React und VUE entschieden werden. Da die Ausstattung von React gut zum umzusetzenden Projekt passte, wurde das Framework für die Implementierung gewählt. Ein weiterer Vorteil war, dass für das an diese Arbeit anknüpfende Projekt voraussichtlich ebenfalls React genutzt werden sollte.

Da für die Visualisierung nur mit ganzzahligen Werten gearbeitet wurde, wird das Ausschließen eines Wertes vom Wertebereich durch das Legen der Grenze des Wertebereichs auf die Mitte

zwischen dem Wert, der ausgeschlossen werden soll, und seinem im Geltungsbereich enthaltenen nächsten Nachbarn verdeutlicht. Liegt die Grenze des Gültigkeitsbereiches genau auf einem Wert, so soll dies signalisieren, dass der Wert im Wertebereich enthalten ist.

4. Umsetzung und Implementierung

4.1 Technische Voraussetzungen

Die Entwicklung fand mit Visual Studio Code statt und der aktuelle Stand des Quellcodes wurde über ein GitHub Repository (Jacobsen, Zahlenstrahl 2022) bereitgestellt. Für das Deployment wurde Node.js genutzt, um während der Entwicklung eine lokale Instanz eines Webservers zu generieren, über die dynamische Änderungen im Quellcode kompiliert und im Browser angezeigt werden konnten. Versuchsweise wurde Next.js als Entwicklungsframework für React genutzt, da dies in der Dokumentation von React empfohlen wird, um serverseitig gerenderte Webseiten zu erstellen, sodass statische Seiten generiert werden können, die zum Beispiel das Anzeigen von Inhalten in Browsern, die das Ausführen von JavaScript deaktiviert haben, ermöglicht. Zusätzlich dazu wurde ein automatisches Deployment über die Plattform vercel.com eingerichtet, die den Code aus dem GitHub Repository automatisch baut.

4.2 Implementierung

4.2.1 Ablauf

In diesem Abschnitt wird der Ablauf beschrieben, der zum erfolgreichen Anzeigen einer Visualisierung führen soll und anhand dessen die Applikation entwickelt wurde. Wie im Ablaufgraphen in Abbildung 21 zu erkennen, beginnt der Vorgang mit der Auswahl einer Aktion. Es soll einerseits möglich sein, die gesetzten Parameter für den Gültigkeitsbereich anzuzeigen, andererseits soll es ebenfalls möglich sein, diese zu löschen. Um die Parameter bereitzustellen, müssen diese über die dritte mögliche Aktion hinzugefügt werden können und für den Fall, dass alle Parameter eingegeben worden sind, soll es möglich sein, die daraus resultierende Visualisierung anzuzeigen.

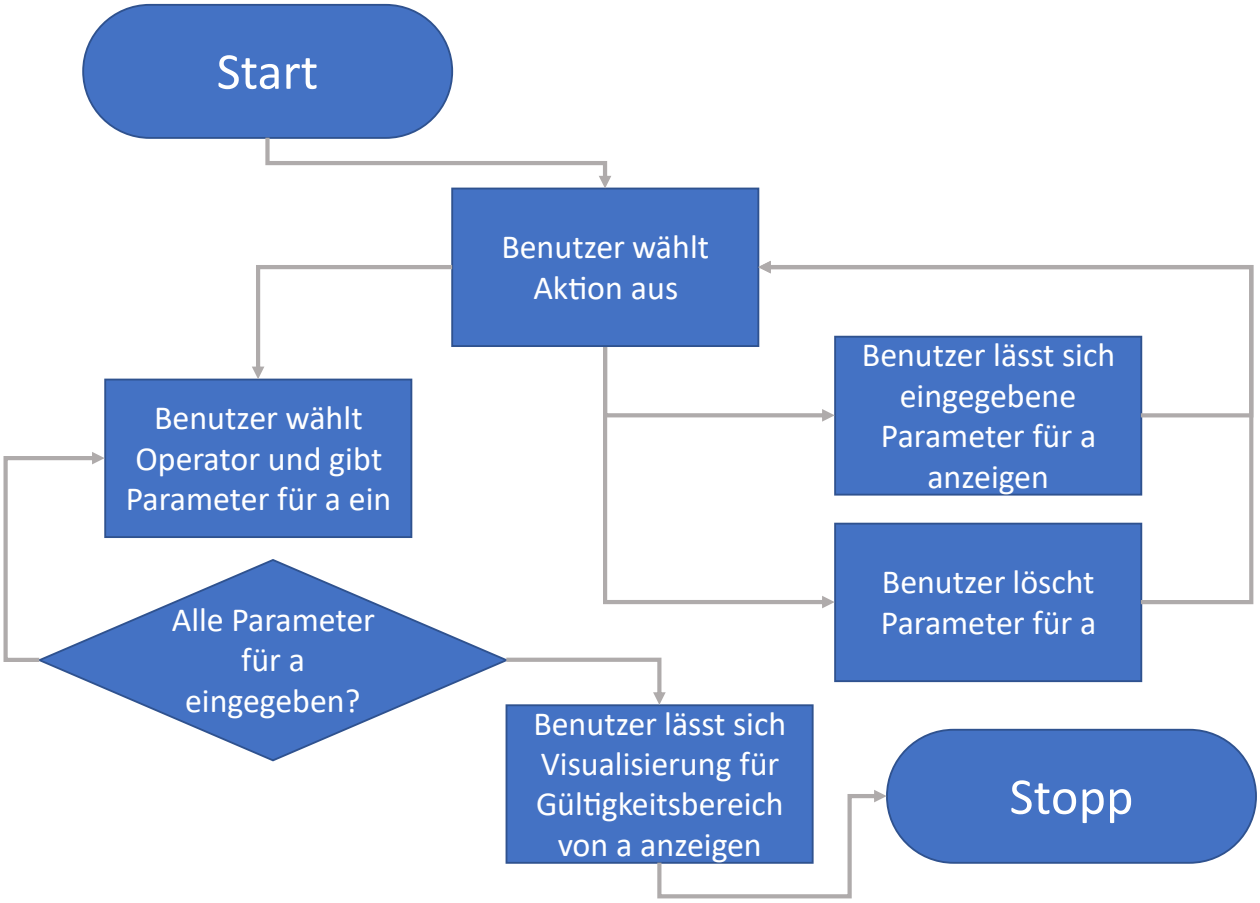

Abbildung 21: Ablaufgraph für den Fall des erfolgreichen Gernerierens einer Visualissierung

4.2.2 Frontend

Für die Eingabe der Parameter wurde, wie in Abbildung 22 zu erkennen, eine Mischung aus einem Dropdown-Menü und einem Textfeld vorgesehen. Durch das Dropdownmenü kann sichergestellt werden, dass keine unzulässigen Eingaben gemacht werden und dass bei der Anwendung der

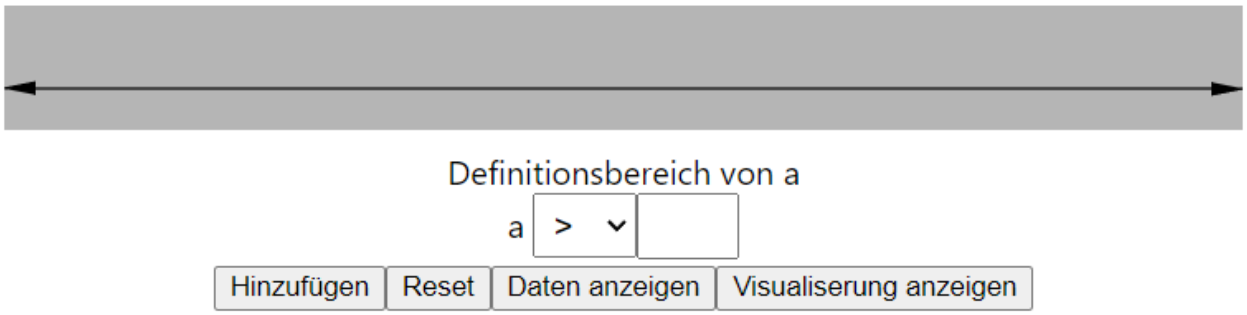

Abbildung 22: Benutzeroberfläche der Visualisierungs-Applikation

Applikation Operationen wie „≤“, „≥“ ausgewählt werden können, ohne dass Sonderzeichen eingegeben werden müssen. Das Textfeld wurde auf die Eingabe von ausschließlich Zahlen begrenzt, damit keine Fehler beim Ermitteln der Intervalle entstehen.

Über die Schaltfläche „Hinzufügen“ lässt sich der eingegebene Parameter zur Parameterliste hinzufügen. Die Schaltfläche „Reset“ ermöglicht das Löschen der Parameterliste und über die Schaltfläche „Daten anzeigen“ lassen sich die Parameter der Liste in der Konsole des Webbrowsers ausgeben. Beim Aktivieren der Schaltfläche „Visualisierung anzeigen“ wird anhand der eingegebenen Parameter ermittelt, in welchem Bereich a liegen kann und die Visualisierung dazu angezeigt. Dieser Teil der Applikation ist innerhalb der „index.js“-Datei überwiegend in HTML programmiert, die Funktionalitäten der Schaltflächen wurden jedoch mithilfe von JavaScript-Funktionen realisiert und das Übermitteln von Übergabeparametern wurde mithilfe der JSX-Syntax umgesetzt.

4.2.3 Backend

Im Hintergrund der Applikation findet die Datenverarbeitung und Aufbereitung statt. Da das temporäre Speichern der Eingabedaten auf dem Webserver zu unnötigen Verzögerungen im Programmablauf führen könnte, wurde zur Speicherung der Daten innerhalb des Browsers auf Cookies zurückgegriffen. Dadurch konnte nicht nur die Zuordnung der Daten zur jeweiligen Sitzung eingespart werden, sondern auch eine einfache Umsetzung mehrerer Visualisierungen realisiert werden, die sich parallel zueinander anzeigen lassen. Innerhalb des Cookies werden die Daten im JSON-Format gespeichert, da so auf Parser zurückgegriffen werden kann, um die Daten beim Lesen des Cookies zurück in JavaScript-Objekte zu konvertieren.

Für das Anzeigen der eingegebenen Parameter muss so lediglich der Cookie eingelesen und dann elementweise in der Konsole ausgegeben werden. Dies wird über die Datei „cookiePrinter.js“ realisiert, innerhalb derer die Funktion „CookiePrinter()“ den Namen des auszugebenden Cookies entgegennimmt und den Inhalt des Selbigen in der Konsole ausgibt.

Die Datenverwaltung wird innerhalb der „DataHandler.js“-Datei geregelt. Sie verfügt über Funktionen zum Löschen und Hinzufügen von Daten zu besagtem Cookie. Die „dataHandler()“-Funktion nimmt für das Verwalten der Cookie-Daten den Operator und Wert, der dem Cookie hinzugefügt werden soll, entgegen. Zusätzlich wird der Name des Cookies, der manipuliert werden soll, übergeben. Wird als Operator „delete“ übergeben, wird der Inhalt des Cookies gelöscht. Andernfalls wird der Inhalt des Cookies ausgelesen und in ein JavaScript-Objekt umgewandelt, an

dessen Ende die neuen Daten angefügt werden. Im Anschluss daran wird das JavaScript-Objekt zurück in einen String umgewandelt und in den Cookie geschrieben.

Sind alle Parameter eingegeben, wird die „Zahlenstrahl()“-Funktion innerhalb der „Zahlenstrahl.tsx“-Datei aufgerufen. Die Funktion erwartet ebenfalls den Namen des Cookies als Übergabeparameter und bedient sich der React-Funktion zum Erstellen eines SVG-Objektes, um ein solches zu erstellen und an die Funktion zum Erstellen der Visualisierung zu übergeben.
Die „drawSvg()“-Funktion, welche ein SVG-Element und den Namen des zu visualisierenden Cookies erwartet, greift zu Beginn wiederum eine Funktion zurück, die die Vorverarbeitung der Daten durchführt. Die Funktion „processDataTable()“ (Zeile 5) erwartet als Übergabeparameter lediglich den Namen des Cookies, dessen Daten verarbeitet werden sollen und gibt ein Datenstruct zurück, welches die obere und untere Grenze des in der Visualisierung abgebildeten Zahlenstrahls, ein Array mit den Werten, die vom Definitionsbereich ausgeschlossen werden und mehrere Flags zur Verwaltung der Darstellung der oberen und unteren Grenze beinhaltet.
Über eine Fallunterscheidung bezüglich der Operatoren (Zeile 60) werden die Grenzen des zu visualisierenden Wertebereiches ermittelt und die Parameter für die Visualisierung im Datenstruct gespeichert, welches anschließend zurückgegeben wird.

In der Funktion zum Erstellen der Visualisierung werden nach Erhalt der Daten zunächst die Parameter für die Visualisierung wie Höhe, Breite und Farbtöne der Visualisierungselemente festgelegt. Im Anschluss daran wird die Zeichenfläche der Visualisierung initialisiert und ermittelt, ob das zu visualisierende Intervall am oberen oder unteren Ende offen oder geschlossen dargestellt werden soll (Zeile 192).
Anschließend wird ermittelt, ob es Lücken im Definitionsbereich gibt und ein Array mit den Anfangs- und Endwerten der Gültigkeitsintervalle für a wird generiert. Das bedeutet, dass für jede Lücke im Definitionsbereich der Start und das Ende der Lücke im Datenarray vermerkt werden. Die Daten werden in aufsteigender Reihenfolge sortiert und für das Erzeugen der Visualisierung vorbereitet.
Für jeden Intervallabschnitt werden nun der Start- und Endwert auf die Maße der Visualisierung skaliert (Zeile 268) und ein Rechteck, welches den Definitionsbereich dieses Intervalls abdeckt, generiert und zum SVG hinzugefügt (Zeile 304). Dabei wird eine Unterscheidung zwischen Intervallen in der Mitte und am Rand des Geltungsbereiches gemacht, da für die Ränder eine extra Abfrage stattfinden muss, ob der Wert im Geltungsbereich enthalten ist, zum Beispiel wenn die

Operatoren „≤“ oder „≥“ verwendet wurden oder ob der Wert vom Geltungsbereich ausgeschlossen wird, wenn die Operatoren „<“ oder „>“ verwendet wurden. Ein Sonderfall ist der Fall, dass „a=x“ eingegeben wird. Um dies in der Intervall-Logik der Applikation abbilden zu können, wird diese Eingabe interpretiert, als wäre „a ≤ x“ und „a ≥x“ eingegeben worden. Die Intervallgrenzen werden somit für den Fall, dass eine gültige Eingabe gemacht wurde, an a herangeschoben.
Wurden die skalierten Start- und Endpunkte eines Intervallabschnittes errechnet, wird im Anschluss daran das zugehörige Rechteck gezeichnet und an zur Visualisierung hinzugefügt. Zusätzlich dazu werden die Begrenzungsklammern wie in der Vorlage mithilfe der Path- und D3-Linien-Funktion an den Grenzen des Rechtecks erzeugt, sofern es sich nicht um ein Endstück bei nach oben oder unten offenen Intervallen handelt.
Sobald dieser Vorgang für alle Teilintervalle des Geltungsbereiches abgeschlossen ist, wird der Zahlenstrahl mitsamt seinen Beschriftungen erzeugt. Hierbei wird je nach Anzahl der Werte innerhalb des Geltungsbereiches entschieden, wie viele Werte am Zahlenstrahl angezeigt werden sollen. Da die Übersichtlichkeit der Visualisierung ab einer Anzahl von 10 Werten abnimmt, wurde die Begrenzung auf 10 anzuzeigende Werte eingestellt. Hierbei wird sich des D3-Achsengenerators bedient (Zeile 338), mithilfe dessen eine automatische Skalierung der Achse anhand vorgegebener Werte möglich ist. Im Anschluss daran wird die Achse an das SVG-Element angefügt und mit den in der Vorlage vorgegebenen Pfeilen versehen.

Nach Abschließen des Berechnens der Visualisierung wird von der Zahlenstrahl-Funktion ein HTML-Element zurückgegeben, welches das erzeugte SVG enthält. So lässt sich die Visualisierung als Komponente in das Frontend der Applikation einbinden und dort anzeigen.

4.2.4 Modularität

Um für den Fall, dass mehrere Visualisierungen parallel angezeigt werden sollen, vorzusorgen, wurden alle Funktionen mit dem Namen des zugehörigen Cookies als Übergabeparameter entworfen. So lassen sich, wie in Abbildung 23 beispielhaft für zwei Instanzen dargestellt, für eine beliebige Anzahl an Instanzen der Eingabe-Oberfläche Visualisierungen generieren, sofern die Instanzen individuelle Namen für die zugehörigen Cookies erhalten. Da das Anzeigen der Visualisierung über den Rückgabewert der Zahlenstrahl-Funktion automatisiert geschieht, entstehen im Frontend der Applikation keine Zuweisungskonflikte. Allerdings mussten die Variablen in „Zahlenstrahl.tsx“ überarbeitet werden, da zuvor mit globalen Variablen gearbeitet wurde, die jedoch beim Verwenden mehrerer Komponenten im Frontend überschrieben wurden

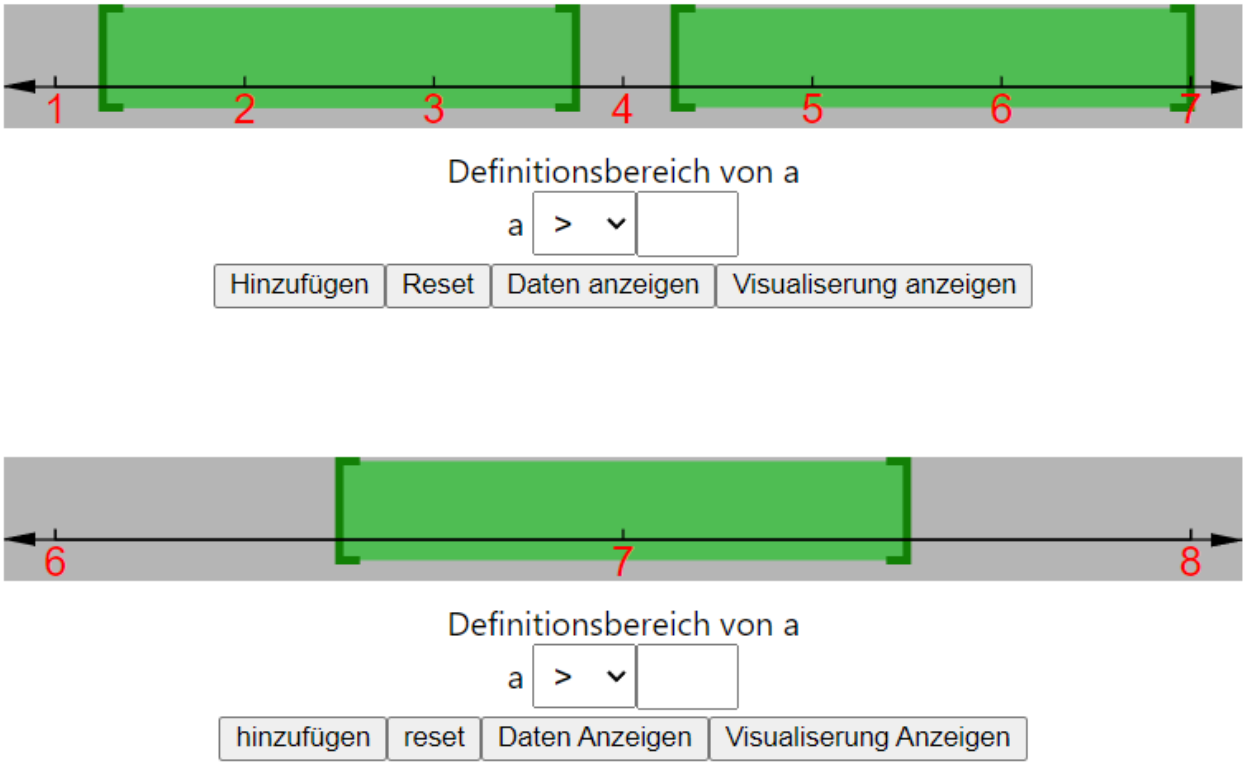

Abbildung 23: Durch modulare Bauweise ist das parallele Anzeigen mehrerer Visualisierungen möglich.

und so alle Visualisierungen das Gleiche anzeigten. Daher wurde stattdessen der Weg über das Datenstruct mit den Parametern für die Visualisierung und der direkte Aufruf der Datenverarbeitungsfunktion innerhalb der SVG-Erzeugungsfunktion gewählt. Die Instanzen der Eingabe-Oberfläche erhalten eindeutig zuzuordnende Namen, welche dann für die Zuordnung der jeweiligen Cookies verwendet werden.

Durch das Arbeiten mit Cookies sind die Visualisierungen dazu über die Sitzung hinaus auf dem verwendeten Rechner gespeichert und können zu einem späteren Zeitpunkt erneut visualisiert oder bearbeitet werden.

5. Auswertung

5.1 Interpretation der Implementierung

Im Rahmen der für diese Arbeit durchgeführten Implementierung einer Visualisierung wurde sich nach der Recherche und Dokumentation der wesentlichen Merkmale mehrerer unterschiedlicher Bibliotheken für Visualisierungen in JavaScript für die Bibliothek entschieden, die am besten zu den Ansprüchen, die durch die vorgegebene Visualisierung gestellt wurden, passt. Da es sich um eine zweidimensionale Darstellung eines Geltungsbereiches an einem Zahlenstrahl handelt, wurde sich für D3.js entschieden, obwohl auch eine Umsetzung mit Two.js möglich gewesen wäre. Jedoch war die Wahl der Implementierung mit D3.js von Vorteil, da Ressourcen in Form von Entwicklungszeit gespart werden konnten, da D3 genau für diesen Anwendungsfall passende Funktionen bereitstellt. Mit den anderen Bibliotheken wäre es auch möglich gewesen, das gewünschte Ziel zu erreichen, jedoch hätte mehr Entwicklungsaufwand betrieben werden müssen, um ein entsprechendes Ergebnis zu erzielen.
Die Verwendung einer weniger umfangreichen Bibliothek wie Two.js erscheint auf den ersten Blick ebenfalls sinnvoll, da die Einarbeitungszeit hier deutlich geringer ausfällt als bei D3, jedoch muss zum Umsetzen eines Projektes nicht der gesamte Funktionsumfang von D3 erlernt werden und durch die gute Dokumentation und Hilfestellung, die bei der Einarbeitung in die Bibliotheken zur Verfügung steht, ist es möglich, die Funktionalitäten, die für den jeweiligen Anwendungsfall benötigt werden, isoliert zu betrachten und sich mit deren Funktionsweise vertraut zu machen. So ist es möglich, sich die Vorteile der umfangreichen Bibliothek zunutze zu machen, während man den Nachteil der Unübersichtlichkeit durch die Vielzahl an Funktionalitäten umgeht. Der zeitliche Vorteil, den eine Implementierung mit Two.js bei der Einarbeitung geboten hätte, wäre durch eben dieses Fehlen von Funktionen in diesem Anwendungsfall zunichtegemacht worden.

5.2 Ergebnis

Anhand der Implementierung, die im Rahmen dieser Arbeit durchgeführt wurde, lässt sich die These, dass sich durch die Wahl des richtigen Frameworks oder der richtigen Bibliothek ressourcenschonend Visualisierungen passend zu Beweisen umsetzen lassen, bestätigen. Die

Komplexität der Implementierung fällt durch die Auswahl mit knapp über 250 Zeilen Code gering aus. Die Visualisierung selbst hat dabei einen noch geringeren Anteil, da ein Großteil des Codes für die Verarbeitung und Aufbereitung der Daten geschrieben wurde. Gerade die D3-Funktionen zum Erstellen, Skalieren und Beschriften des Zahlenstrahls konnten hierbei eine große Erleichterung schaffen, sodass der Fokus während der Phase der Implementierung auf das Lösen des Problems der Darstellung des Geltungsbereiches gerichtet werden konnte.

6. Zusammenfassung und Ausblick

6.1 Zusammenfassung und Fazit

Im Rahmen dieser Arbeit wurde ebenfalls deutlich, dass der Aufwand, eine geeignete Visualisierung zu entwerfen, um einen Sachverhalt gut darzustellen, ebenfalls einen großen Anteil an Ressourcen in Form von Arbeitszeit in Anspruch nimmt. Je komplexer die wiederzugebende Materie ausfällt, desto mehr Aufwand muss betrieben werden, um eine geeignete Visualisierung zu entwerfen, mit der dann im Anschluss andere Menschen auch etwas anfangen können. Durch jedes Hilfsmittel, das sich auf dem Weg zu einer Visualisierung anbietet, kann der Prozess verbessert werden. Dadurch stellt auch die Auswahl des richtigen Frameworks beziehungsweise der richtigen Bibliothek und die dokumentierte Recherche, die im Zuge dessen ausgeführt werden muss, ein gutes Hilfsmittel für das Erzeugen auch von zukünftigen von Visualisierungen dar. Auch wenn die in dieser Arbeit verwendete Bibliothek D3 ein recht umfangreiches und universelles Werkzeug darstellt, mit dem sich eine Vielzahl von Visualisierungsformen realisieren lassen, ist es dennoch nützlich, über weitere mehr spezialisierte Bibliotheken zu verfügen, auf die im Falle einer Visualisierung mit speziell dafür passenden Merkmalen zurückgegriffen werden kann.
Es wurden im Rahmen der Arbeit aktuelle JavaScript-Frameworks recherchiert, dokumentiert und einander anhand einer tabellarischen Darstellung gegenübergestellt und anhand ihrer Merkmale verglichen. Dazu wurden Beispielszenarien implementiert, um ein Gefühl für die Anwendung der einzelnen Bibliotheken zu erlangen. Darüber hinaus wurde das erlangte Wissen angewandt, um eine Visualisierung für ein Projekt zu realisieren und die dieser Arbeit zugrunde liegende These zu überprüfen. Die zu Beginn dieser Arbeit gesteckten Ziele wurden somit erreicht.

6.2 Ausblick

Somit stellt das Ergebnis dieser Arbeit einen weiteren Baustein auf dem Weg hin zu guten Visualisierungen für Softwarebeweise dar und reiht sich in die Gruppe von Arbeiten auf diesem Gebiet ein, in deren Zusammenhang sie entstanden ist. Auf der Grundlage dieser Arbeit lassen sich für zukünftige Visualisierungen Entscheidungen bezüglich der verwendeten Bibliotheken und Frameworks treffen und so der Prozess von der Idee zur Visualisierung verbessern.
Im Rahmen des Projektes könnte es zukünftig interessant sein, parallel zu dieser Arbeit einen

Vergleich zu Implementierungen von Visualisierungen in Python vorzunehmen, wobei es sich gerade auch im Hinblick auf die Entwicklungen und Perspektiven auf dem Gebiet des maschinellen Lernens um eine äußerst attraktive Programmiersprache für angehende Entwicklerinnen und Entwickler handelt.

Mit Blick auf die hier getätigte Implementierung im Bezug zum Gesamtprojekt wäre es hilfreich, ein Deployment über GitHub-Pages zu realisieren, da so gewährleistet werden kann, dass auch zukünftig damit gearbeitet werden kann. Zusätzlich dazu kann der Algorithmus zur Intervallbildung ebenfalls überarbeitet werden, sodass verhindert werden kann, dass bei der Eingabe bestimmter logisch unstimmiger Parameter fehlerhafte Visualisierungen erzeugt werden.

Eine weitere Verbesserung für das Projekt und die Verwendbarkeit der Implementierung könnte durch eine Verbesserung der Modularität der Komponenten erreicht werden. Die einzelnen Komponenten des Frontends für jede Instanz der Visualisierung kopiert werden müssen, treten beim Prozess der Erweiterung des Frontends leicht Fehler auf. Dies könnte vermieden werden, wenn der gesamte Block mit allen Schaltflächen und Eingabefeldern als eigene Komponente implementiert und dann nur noch importiert und eingebunden werden würde.

Darüber hinaus wäre es eine Verbesserung, wenn die eingegebenen Parameter zusätzlich zur Visualisierung außerhalb der Konsole des Browsers angezeigt werden könnten, da beim Erzeugen der Visualisierungen so leichter der Überblick behalten werden könnte, welche Parameter zu einem Schaubild geführt haben.

7. Anhang

Im Anhang dieser schriftlichen Ausarbeitung befindet sich der Quellcode der entwickelten Visualisierung, welcher ebenfalls auf GitHub unter dem folgenden Link zu finden ist:

https://github.com/bennet-leo/Zahlenstrahl/tree/Abgabe
(Stand 08.09.2022 17:00 Uhr)

Der in dieser Arbeit beschriebene Stand des Codes liegt auf dem Branch „Abgabe“.
Zusätzlich dazu ist der Quellcode auf dem Repository des zugehörigen Projektes verfügbar:

https://gitlab.rz.htw-berlin.de/proofvisualization/bennet/-/tree/main/Zahlenstrahl_vis
(Stand 08.09.2022 17:00 Uhr)

Literaturverzeichnis

Beyer, D, R Hennicker, M Hoffmann, und M Wirsing. „Software-Verifikation.“ *50 Jahre Universitäts-Informatik in München*, 2017: 75 ff.

Bostock, Mike. *D3 Gallery.* 04. 02 2020. https://observablehq.com/@d3/gallery (Zugriff am 02. 09 2022).

—. *Data-Driven Documents.* 2021. https://www.d3js.org/ (Zugriff am 28. 08 2022).

Chien, Yuin. *Two.js.* 14. 02 2022. https://two.js.org/ (Zugriff am 01. 09 2022).

DeSandro, David. *Zdog.* 23. 10 2019. https://zzz.dog/ (Zugriff am 02. 09 2022).

Google. *Angular.* 2022. https://angular.io/ (Zugriff am 03. 09 2022).

Hesse, Christine, Laura Gerken, Jutta Klaeren, und Dr. André Hein (Volontär). *INFORMATIONEN ZUR POLITISCHEN BILDUNG NR. 344/2020*. Herausgeber: Bundesszentrale für politische Bildung. 3. 11 2020. URL: https://www.bpb.de/izpb/digitalisierung-344/ (Zugriff am 15. 8 2022).

Jacobs, B, und F Piessens. *The VeriFast Program Verifier.* Technical Report CW-520, Katholieke Universiteit Leuven, Belgium: Department of Computer Science, 2008.

Jacobsen, Bennet. *Visualisierungen.* 01. 08 2022. bjacobsen.ddns.net (Zugriff am 02. 09 2022).

Kahnemann, D. *Schnelles Denken, langsames Denken.* Penguin Verlag, 2016.

MDN. *Learn Web development | MDN.* 16. 08 2022. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript (Zugriff am 28. 08 2022).

Meta-Platforms. *React.* 2022. https://reactjs.org/ (Zugriff am 03. 09 2022).

Ngan, William. *Pts Docs.* 19. 02 2021. https://ptsjs.org/docs/ (Zugriff am 02. 09 2022).

Sperling, George. „The information available in brief visual presentations.“ *Psychological Monographs: General und Applied*, 1960: 1-29.

Tran, E. „Verifikation/Validation/Certification.“ *Dependable Embedded Systems*, 1999.

You, Evan. *VUE The Progressive JavaScript Framework.* 2022. https://vuejs.org/ (Zugriff am 03. 09 2022).