Leibniz Universität Hannover
Institut für Theoretische Informatik

Bachelorarbeit
Tools für modale Logiken

Sergey Kartamyshev

09.08.2012

Prüfer: Prof. Dr. Heribert Vollmer
Zweitprüfer: Dr. Arne Meier
Betreuer: M.Sc. Julian-Steffen Müller

Zusammenfassung

In dieser Arbeit wird die praktische Umsetzung mehrerer theoretischer Konzepte aus dem Bereich der modalen Logiken erörtert. Als erstes wird eine geeignete Implementierung eines Modells der modalen Logik gezeigt. Anschließend werden die Algorithmen für **Model Checking** und der **Erfüllbarkeit auf S5 Kripke Strukturen** beschrieben und die wesentlichen Implementierungsdetails der Algorithmen begründet. Des Weiteren wird gezeigt, wie die Prüfung auf Erfüllung der Axiome **K4**, **B**, **T**, **S4**, **S5**, **D** und **E** erfolgen kann.

Die Umsetzung der Algorithmen in der Programmiersprache Java befindet sich samt dem dazugehörigen Source-Code auf der dieser Bachelorarbeit beigefügten CD.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit und die zugehörige Implementierung selbstständig verfasst und dabei nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Hannover, 09. August 2012

Sergey Kartamyshev

Inhaltsverzeichnis

1. Einleitung

In der Logik geht es um korrekte Schlussfolgerungen aus bereits gegebenen Prämissen. Aus diesem Grund ist Logik in den unterschiedlichsten Wissenschaften, wie zum Beispiel Philosophie, Mathematik und Informatik von Relevanz. *Die* allumfassende Logik existiert jedoch nicht. Vielmehr gibt es eine Vielzahl an Logiken, die für unterschiedliche Aufgaben und Anforderungen entwickelt wurden.

1.1. Geschichtlicher Hintergrund

Zu Beginn dieser Arbeit erfolgt ein kurzer Überblick über die Geschichte der Logik. Dieser Abschnitt basiert auf [Sch02a].

Die Ursprünge der Logik, wie wir sie heute kennen, liegen in der griechischen Antike. Aristoteles (384-322 v. u. Z.) beschrieb als einer der Ersten, wie man korrekte Schlüsse zieht. Er erkannte, dass für die Logik nur die äußere Form eines Schlusses relevant ist und verwendete Variablen anstelle konkreter Beispiele.

Nach Aristoteles entwickelten vor allem die Stoiker (ab etwa 300 v. u.Z.) die Logik weiter. Danach geriet die Disziplin in den Hintergrund und wurde erst ab dem Mittelalter wieder stärker beachtet.

In der Neuzeit erkannte vor allem Gottfried Wilhelm Leibniz (1646-1716) die Notwendigkeit der Weiterentwicklung der Logik. Sein Ziel war es, die Umgangssprache durch eine formalisierte Wissenschaftssprache zu ersetzten. Die erste umfassende Darstellung der modernen Logik präsentierte der Mathematiker und Philosoph Gottlob Frege (1848-1925). Die von ihm entwickelte „Begriffsschrift“ stellt eine streng formalisierte Notation dar. Sie lässt keine unterschiedlichen Interpretationen zu, ist jedoch auch schwer lesbar. Bertrand Russell (1872-1970) entwickelte die heute übliche logische Notation.

Wesentliche Beiträge zur Weiterentwicklung der Logik stammen von Mathematikern wie Leopold Löwenheim(1878-1957), Alfred Tarski(1902-1983) und Kurt Gödel(1906-1978). Die Entwicklung der modalen Logik hat insbesondere Saul Kripke(*1940) vorangetrieben.

Ihre wesentliche Anwendung aber fand die Logik erst während des Computerzeitalters. Es übertrug die Bedeutung von „wahr“ und „falsch“ auf technische Realisierungen, wies den überwiegend theoretisch motivierten Überlegungen praktische Bedeutungen zu und

bewirkte aufgrund von weitverbreiteter Nutzung der Heimcomputer, digitalen Kameras, Mobiltelefone und anderen technischen Produkten, den Einzug der Logik in fast jeden privaten Haushalt.

1.2. Ziel und Aufbau dieser Arbeit

Die in dieser Arbeit beschriebene Form der Logik ist bekannt als „modale Logik“. Dieser Zweig der Logik erweitert die klassische Aussagenlogik um die Begriffe des Möglichen und des Notwendigen. Modale Logik wird unter anderem im Bereich der künstlichen Intelligenz, sowie zur Erkennung von Deadlocks (z.B. beim Chipentwurf) verwendet.

Ziel dieser Arbeit ist es, die gegebenen Algorithmen zum Model Checking, Erfüllbarkeit und Axiomenprüfung praktisch umzusetzen und geeignete Benutzerschnittstellen anzubieten. Die Umsetzung der Algorithmen soll die in der Theorie geforderte Laufzeit einhalten. Ein wesentliches Kriterium ist auch die Erweiterbarkeit des Codes, um zusätzliche Algorithmen zu einem späteren Zeitpunkt unkompliziert ergänzen zu können.

Um die Implementierung zu erläutern, wird zunächst in Kapitel 2 die zugrunde liegende Theorie betrachtet, die im Wesentlichen auf [Mül09] und [Sch02b] basiert. Die verwendete Notation und die Struktur der Erläuterungen orientiert sich an [Bey11].

Nachdem die theoretischen Grundlagen genannt wurden, wird im Kapitel 3 die praktische Umsetzung beschrieben. Dabei werden die Entwurfsentscheidungen begründet. Abschließend folgt ein Ausblick auf die Möglichkeiten einer Weiterentwicklung der Tools.

2. Theoretische Grundlagen

In diesem Kapitel werden die Syntax und die Semantik der modalen Logik beschrieben, sowie alle für die folgenden Kapitel benötigten Eigenschaften modaler Logik erörtert. Diese theoretischen Grundlagen werden am Ende des Kapitels anhand eines Beispiels verdeutlicht.

2.1. Die modale Grundsprache

Die modale Grundsprache erweitert die klassische Aussagenlogik um die zwei modalen Operatoren Box $\Box$ und Diamond $\Diamond$. Somit besteht das Alphabet Σ_{ML} der modalen Logik aus:

1. der abzählbaren Menge atomarer Formeln $\Phi = \{p, q, r, ...\}$
2. den klassischen Operatoren $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$
3. den modalen Operatoren $\Box, \Diamond$
4. den Konstanten $\top, \bot$
5. sowie den Klammersymbolen (und)

2.1.1. Syntax der modalen Grundsprache

Definition 2.1 *Sei Φ die Menge atomarer Aussagen. Dann ist die Menge der modalen Formeln* $\mathrm{Fma}(\Phi)$ *durch folgende Regeln induktiv definiert als:*

1. *Jede atomare Variable $p \in \Phi$ und Konstante $\bot$ ist in* $\mathrm{Fma}(\Phi)$
2. *Seien $\varphi, \psi \in \mathrm{Fma}(\Phi)$ dann gilt:*
   (a) $\neg\varphi \in \mathrm{Fma}(\Phi)$
   (b) $(\varphi \vee \psi) \in \mathrm{Fma}(\Phi)$
   (c) $\Diamond\varphi \in \mathrm{Fma}(\Phi)$
3. *Zusätzlich werden folgende Kurzschreibweisen vereinbart:*
   (a) $\top := \neg\bot$

(b) $(\varphi \wedge \psi) := \neg(\neg\varphi \vee \neg\psi)$

(c) $(\varphi \rightarrow \psi) := \neg\varphi \vee \psi$

(d) $(\varphi \leftrightarrow \psi) := (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$

(e) $(\Box\varphi) := \neg\Diamond\neg\varphi$

4. *Eine Teilformel von* $\varphi \in \mathrm{Fma}(\Phi)$ *ist eine Formel, die ein Teilwort von* φ *ist. Die Menge aller Formel von* φ *heißt* $\mathrm{Sub}(\varphi)$.

Zur besseren Lesbarkeit werden folgende Regeln zum Einsparen von Klammern verwendet:

1. Äußere Klammern werden weggelassen.
2. $\neg, \Diamond$ und $\Box$ binden am stärksten.
3. $\wedge$ bindet stärker als $\vee$.
4. $\vee$ wiederum bindet stärker als $\rightarrow$ und $\leftrightarrow$.
5. Für mehrfache Konnektoren desselben Typs werden die Klammern von rechts nach links gesetzt. Die Formel $p \rightarrow q \rightarrow r$ wird also wie folgt ausgewertet: $p \rightarrow (q \rightarrow r)$.

2.1.2. Semantik der modalen Grundsprache

Die Belegungsfunktion weist jeder Variable aus Φ in jeder Welt einen Wahrheitswert zu. Dabei ist die Belegung der Variablen in einer Welt unabhängig von der in anderen Welten. Die Betrachtung der Belegungsfunktion genügt nicht, um eine Formel semantisch zu interpretieren. Vielmehr wird eine Menge an Welten, sowie die dazugehörige binäre Relation zwischen den Welten betrachtet. Je nach Kontext werden die Welten auch als Punkte oder Zustände bezeichnet. Die Relation könnte dann z.B. angeben, welcher Zustand aus einem anderen folgen kann. Anders formuliert: Welche Welten aus einer Welt direkt erreichbar sind. Eine gängige Formulierung ist auch: „Welt v sieht Welt w“.

Während die Interpretation der Standardoperatoren klar und eindeutig ist, gibt es mehrere Möglichkeiten, die modalen Operatoren $\Diamond$ und $\Box$ zu interpretieren. In dieser Arbeit wird die gängige Interpretation verwendet. Sie sieht wie folgt aus:

$\Diamond\varphi$	„φ ist möglicherweise wahr“
$\Box\varphi$	„φ ist notwendigerweise wahr“

Definition 2.2 *Sei* W *eine nicht leere Menge von Welten und* R *eine binäre Relation auf* W.

1. *Die Struktur* $F = (W, R)$ *wird Kripke-Rahmen (oder Rahmen; frame) genannt.*

2. *Die Funktion $V : \Phi \to P(W)$ wird Belegungsfunktion genannt. Diese bildet die Variablen aus Φ auf die Menge der Welten ab.*

3. *Ist $F = (W, R)$ ein Rahmen und V eine Belegungsfunktion, so heißt $M = (F, V)$ ein auf F basierendes Modell. Anders formuliert: F liegt M zugrunde.*

Definition 2.3 *Seien ψ_1, ψ_2 modale Formeln. Sei $F = (W, R)$ ein Rahmen und $M = (F, V)$ ein auf F basierendes Modell. Dann ist die Formel φ*

1. *wahr in der Welt $w \in W$ $(M, w \models \varphi)$ genau dann, wenn einer der folgenden Punkte erfüllt ist:*

   (a) $\varphi = \top$

   (b) $\varphi = x$ *falls* $w \in V(x)$ *und* $x \in \Phi$

   (c) $\varphi = \neg\psi_1$ *falls* $M, w \not\models \psi_1$

   (d) $\varphi = \psi_1 \vee \psi_2$ *falls* $M, w \models \psi_1$ *oder* $M, w \models \psi_2$

   (e) $\varphi = \Diamond\psi_1$ *falls* $\exists v \in W$ *mit* wRv *und* $M, v \models \psi_1$

2. *wahr in einem Modell M genau dann, wenn sie in jeder Welt im Modell wahr ist. ($M \models \varphi$ gdw. $\forall w \in W$ gilt: $M, w \models \varphi$)*

3. *wahr in F $(F \models \varphi)$ genau dann, wenn sie in allen auf F basierenden Modellen wahr ist.*

4. *wahr $(\models \varphi)$ genau dann, wenn sie in allen Rahmen wahr ist.*

Eine Formel φ ist erfüllbar im Modell $M = ((W, R), V)$, wenn es mindestens eine Welt $w \in W$ gibt, sodass $M, w \models \varphi$ gilt.

2.1.3. Modale Tiefe

Die modale Tiefe gibt die maximale Schachtelungstiefe der modalen Operatoren in einer Formel wieder.

Definition 2.4 *Sei $p \in \Phi$ und seien φ, ψ modale Formeln. Die modale Tiefe wird induktiv definiert als:*

1. $md(p) := 0$
2. $md(\neg\varphi) := 0$
3. $md(\varphi \vee \psi) := max\{md(\varphi), md(\psi)\}$
4. $md(\Diamond\varphi) := 1 + md(\varphi)$

Zum Beispiel haben die Formeln $\varphi = \Diamond\Diamond\Diamond p \wedge \Box\Diamond\neg\Diamond q$ und $\psi = \Diamond(p \wedge \Box(\neg q \wedge \Diamond p))$ die gleiche modale Tiefe: $md(\varphi) = md(\psi) = 3$. Entsprechend ist die modale Tiefe bei Formeln ohne modale Operatoren gleich null.

2.1.4. Beispiel eines Modells der modalen Logik

Zur Verdeutlichung des oben genannten Sachverhaltes soll ein Modell der modalen Logik als Graph visualisiert werden. An dem hier gewählten Beispiel wird deutlich, dass die menschliche Intuition bei der Interpretation der Formeln nicht immer zutrifft.
Gegeben sei der Rahmen $F = (W, R)$ mit den Welten $W = \{w_1, w_2, w_3, w_4\}$ und der Relation $R = \{(w_1, w_2), (w_1, w_3), (w_2, w_3), (w_2, w_4)\}$. Des Weiteren sei die Menge der atomaren Variablen $\Phi = \{p, q\}$ sowie die Belegungsfunktion V mit $V(p) = \{w_1, w_2\}$ und $V(q) = \{w_2, w_3, w_4\}$ gegeben. Dann ist Abbildung 2.1 eine geeignete Darstellung für $M = (F, V)$.

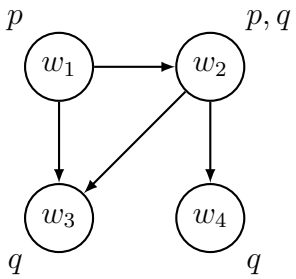

Abbildung 2.1.: Darstellung des Modells M als Graph

Unter anderem können folgende Aussagen getroffen werden:

- $M, w_1 \models p, \Diamond p, \Diamond q, \Diamond\Diamond q, \Box q, \Diamond\Box q$
- $M, w_4 \models \Box q, \Box\Box q, \Box\Box\Box q, ...$
- $M \models p \vee q$

Die Formel $\Box\varphi \rightarrow \Diamond\varphi$ könnte man als „Was notwendig ist, ist auch möglich“ auffassen. Dies scheint der Intuition nach korrekt zu sein, ist aber zum Beispiel in der Welt w_3 nicht erfüllt: Es gibt keine nachfolgende Welt, in der q gilt. Aber es gibt auch keine nachfolgende Welt in der $\neg q$ gilt. Aus der Dualität von $\Diamond$ und $\Box$ gilt in w_3 zwar $\Box q$ aber nicht $\Diamond q$.

3. Implementierung

3.1. Grundlagen

Die Implementierung erfolgt in der Programmiersprache Java. Java ist eine sehr verbreitete Programmiersprache und gehört auch an der Leibniz Universität Hannover zur Grundausbildung im Bachelorstudiengang Informatik. Die Weiterentwicklung der Tools ist somit auch zu einem späteren Zeitpunkt ohne große Einarbeitung in die Programmiersprache möglich. Ein weiterer Vorteil von Java ist die Plattformunabhängigkeit. Die erstellten Tools sind somit auf unterschiedlichen Rechnern ohne Portierung lauffähig.

Als Grundlage der Implementierung dient ein Programm für Model Checking der aussagenlogischen Formeln, das am Institut für Theoretische Informatik erstellt wurde. Eine Formel wird dabei geparst und intern als ein Formelbaum dargestellt.

3.1.1. Aufbau des Modells

In der Klasse für das Modell werden die atomaren Variablen in einem String-Array und die Welten (repräsentiert durch die Klasse World) in einem World-Array festgehalten. Wie im Abschnitt 2.1.4 erwähnt, lässt sich ein Modell der modalen Logik als Graph interpretieren. Zur Implementierung des Graphen wird das *Java Universal Network/Graph*[1] (kurz JUNG) Framework verwendet. Damit wird der Aufbau des Graphen simpel. Zusätzlich bringt das Framework zahlreiche nützliche Algorithmen mit sich. Es existieren zum Beispiel Algorithmen zur Visualisierung der Graphen.

Die Knoten des Graphen stellen die Welten dar. Eine Welt hat einen Namen, der als String gespeichert wird, sowie ein Array mit booleschen Werten. In diesem Array wird jeweils festgehalten, ob eine Variable aus dem Variablen-Array des Modells in dieser Welt gilt oder nicht.
Diese Implementierung bietet den Vorteil, dass sowohl die Frage nach Variablen (aus der Menge aller atomaren Variablen Φ) die in einer Welt gelten, als auch die Frage in welchen Welten (aus der Menge aller Welten W) eine bestimmte Variable gilt, leicht zu beantworten ist.
Für die erste Frage muss ein Mapping zwischen den Namen der Variablen und den Positionen im Array mit den booleschen Werten erfolgen. Dies erfolgt in $\mathcal{O}(|\Phi|)$. Für

[1] http://jung.sourceforge.net/

die zweite Frage genügt es, in allen Welten den jeweiligen booleschen Wert an der entsprechenden Stelle im Array zu überprüfen. Die Laufzeit liegt hierbei in $\mathcal{O}(|W|)$. Die Abbildung 3.1 verdeutlicht diesen Sachverhalt.

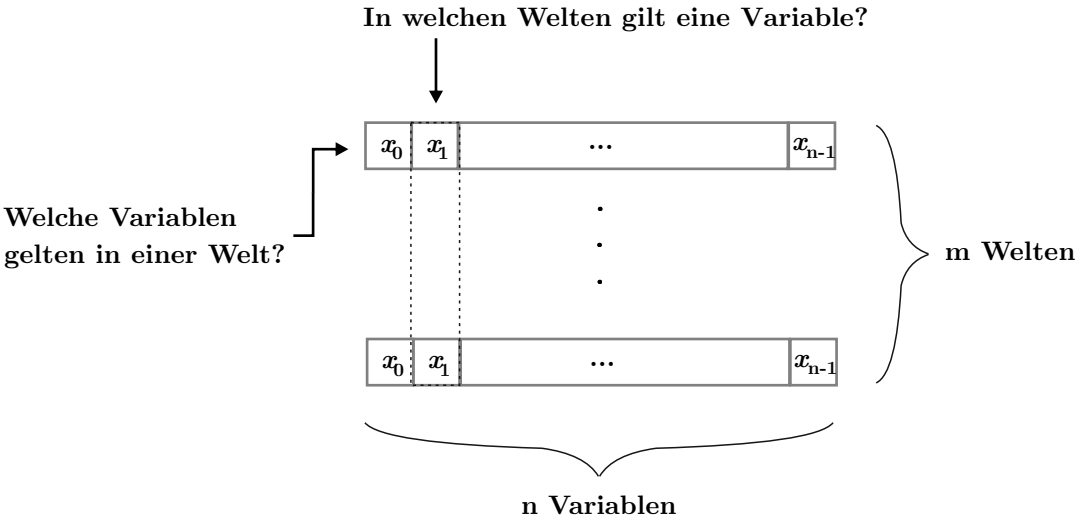

Abbildung 3.1.: Beispiel mit n Variablen und m Welten

Zur Eingabe des Modells wird eine vereinfachte Form von XML verwendet. Diese XML-Datei wird mittels *Simple API for XML* (SAX) eingelesen. Weil die Anzahl der Welten und der atomaren Variablen vor dem Einlesen nicht feststeht, wird die Datei zweimal geparst.
Beim ersten Durchlauf werden die Welten und Variablen gezählt und somit die Größen für die Arrays festgestellt. Beim zweiten Durchlauf werden die Arrays befüllt und der Graph aufgebaut. Jeder der beiden Durchläufe hat eine Laufzeit, die linear abhängig von der Anzahl der XML-Elemente ist.

Folgende XML-Datei korrespondiert mit dem Beispiel 2.1:

```
<ml>
  <worlds>
    <world name="w1"/>
    <world name="w2"/>
    <world name="w3"/>
    <world name="w4"/>
  </worlds>
  <relations>
    <relation source="w1" dest="w2"/>
    <relation source="w1" dest="w3"/>
    <relation source="w2" dest="w3"/>
    <relation source="w2" dest="w4"/>
  </relations>
  <mapping>
    <var name="p">
      <in name="w1"/>
      <in name="w2"/>
    </var>
    <var name="q">
      <in name="w2"/>
      <in name="w3"/>
      <in name="w4"/>
    </var>
  </mapping>
</ml>
```

3.2. Model Checking

Unter dem Begriff des „Model Checking“ für modale Logik versteht man Verfahren, die überprüfen, ob gegebene Aussagen in einem ebenfalls gegebenen System von Welten wahr oder unwahr sind.
Sei $M = ((W, R), V)$ ein Modell, $w_0 \in W$ eine Welt und $\varphi \in ML$ eine Formel, dann stellt sich die Frage, ob $M, w_0 \models \varphi$ gilt.

3.2.1. Algorithmus

Der Model Checking (ML-MC) Algorithmus für modale Formeln wird auf den Seiten 14 und 15 in [Mei12] beschrieben. Er beinhaltet nicht die Fälle für die Implikation und die Biimplikation. Diese müssten, sofern sie vorkommen, wie in der Definition 2.1 in Abschnitt 3c und 3d beschrieben, substituiert werden. Erst dann kann die Formel ausgewertet werden.
In dieser Arbeit wurde der Algorithmus um diese beiden zusätzlichen Fälle erweitert. Hintergrund ist, dass die Formeln intern als ein Formelbaum implementiert werden und die entsprechenden Knoten bereits existieren. So wird die interne Struktur besser ausgenutzt, da der Zwischenschritt der Substitution nicht notwendig ist.

Algorithm 1: Model Checking Algorithmus für modale Formeln[1]

Input: Model $M = ((W, R), V)$, $w_0 \in W$ $\varphi \in ML$

1 initially set the labeling funktion $l : W \to \text{Sub}(\varphi)$ to $l(w) = \emptyset$ for all $w \in W$;
2 **forall the** $\psi \in \text{Sub}(\varphi)$ *ordered w.r.t* $|\psi|$ **do**
3 **case** $\psi = x \in PROP$
4 **forall the** $w \in W$ *s.t.* $x \in V(w)$ **do**
5 $l(w) \leftarrow l(w) \cup \{\psi\}$
6 **case** $\psi = \alpha \vee \beta$
7 **forall the** $w \in W$ *s.t.* $\alpha \in l(w)$ *or* $\beta \in l(w)$ **do**
8 $l(w) \leftarrow l(w) \cup \{\psi\}$
9 **case** $\psi = \alpha \wedge \beta$
10 **forall the** $w \in W$ *s.t.* $\alpha \in l(w)$ *and* $\beta \in l(w)$ **do**
11 $l(w) \leftarrow l(w) \cup \{\psi\}$
12 **case** $\psi = \neg\alpha$
13 **forall the** $w \in W$ *s.t.* $\alpha \notin l(w)$ **do**
14 $l(w) \leftarrow l(w) \cup \{\psi\}$
15 **case** $\psi = \alpha \to \beta$
16 **forall the** $w \in W$ *s.t.* $\alpha \notin l(w)$ *or* $\beta \in l(w)$ **do**
17 $l(w) \leftarrow l(w) \cup \{\psi\}$
18 **case** $\psi = \alpha \leftrightarrow \beta$
19 **forall the** $w \in W$ *s.t.* $(\alpha \notin l(w)$ *or* $\beta \in l(w))$ *and* $(\alpha \in l(w)$ *or* $\beta \notin l(w))$ **do**
20 $l(w) \leftarrow l(w) \cup \{\psi\}$
21 **case** $\psi = \Diamond\alpha$
22 **forall the** $w \in W$ *s.t.* $\alpha \in l(w)$ **do**
23 **forall the** $w' \in W$ *with* $w'Rw$ **do**
24 $l(w') \leftarrow l(w') \cup \{\psi\}$
25 **case** $\psi = \Box\alpha$
26 **forall the** $w \in W$ **do**
27 **if** $\forall w' \in W$ *with* $wRw' : \alpha \in l(w')$ **then**
28 $l(w) \leftarrow l(w) \cup \{\psi\}$
29 **return** *true iff* $\varphi \in l(w_0)$

Wie in [Mei12] gezeigt, hat der Algorithmus 1 die Laufzeit von $\mathcal{O}(|M|^2 \cdot |\varphi|)$. Das kommt dadurch zustande, dass die Schleife aus Zeile 2 $|\varphi|$-mal durchlaufen wird und die Laufzeit jedes **case**-Falls durch einen Polynom in $|M|^2$ beschränkt ist.
Dieser Algorithmus terminiert immer, weil $\text{Sub}(\varphi)$ endlich viele Elemente hat.

[1] Vergleiche: [Mei12, 15]

Die Korrektheit des Algorithmus folgt aus der Sortierung nach der Länge der Formeln und dem zugehörigen Labeling. Zuerst wird Sub(φ) der Länge nach sortiert. Das Labeling erfolgt anschließend gemäß dieser Sortierung. Als erstes werden die atomaren Variablen gelabelt. Beim Labeln zusammengesetzter Formeln wird dann darauf zurückgegriffen, dass die einzelnen Bestandteile der Formel bereits gelabelt wurden.

3.2.2. Implementierung des ML-MC Algorithmus

Im Folgenden wird auf die Umsetzung von Algorithmus 1 eingegangen und die wesentlichen Entscheidungen bei der Umsetzung begründet.

Die Labels werden im assoziativen Speicher abgelegt. Die Keys sind Integer-Werte, die mit den Positionen der Welten im Array korrelieren. Die Values sind HashSets, in denen die jeweiligen Teilformeln (Labels) gespeichert werden.

In Zeile 2 wird gefordert, dass die Formeln in Sub(φ) der Länge nach sortiert vorliegen. Zu diesem Zweck muss beim Aufbau des Formelbaums auch eine sortierbare Datenstruktur zur Speicherung der Teilformeln verwendet werden. In Java bietet sich hierfür zum Beispiel ein TreeSet an. Der Comparator zur Feststellung der Länge der Teilformeln - und somit der Reihenfolge der Elemente im TreeSet- muss alle Variablen als gleichlang betrachten. Es ist also unerheblich, wie lang der Name der Variable ist.

Die Zeilen 26 bis 28 beinhalten auch den Fall, dass man aus einer Welt keine weiteren Welten erreichen kann. Im Algorithmus 2 werden diese Zeilen genauer dargestellt.

Algorithm 2: Zeilen 26 bis 28 im Detail

```
1 forall the w ∈ W do
2     insert ← true;
3     forall the w' ∈ W with wRw' do
4         if α ∉ l(w') then
5             insert ← false;
6     if insert = true then
7         l(w) ← l(w) ∪ {ψ}
```

Es bietet sich an, den Algorithmus 1 in zwei Methoden aufzuteilen. So muss die Labeling-Methode, die sich von der Zeile 1 bis zur Zeile 28 erstreckt, für eine Formel φ nur einmal durchlaufen werden. Die in der Zeile 29 stattfindende Überprüfung, ob φ in der Welt $w \in W$ gelabelt wurde, kann dann für jede einzelne Welt in einer separaten Methode erfolgen. Die Überprüfung, ob eine Formel in einem HashSet enthalten ist, erfolgt im Durchschnitt in konstanter Zeit. Um zu prüfen, ob die Formel φ im ganzen Modell M gilt, genügt es, durch alle Welten zu iterieren. Wenn die Formel in jeder Welt gelabelt wurde, dann gilt sie auch in M.

3.3. Axiomatisierung

Die syntaktischen Eigenschaften von Rahmen in der modalen Logik lassen sich mit Axiomen festlegen. Sei $F = (W, R)$ ein Rahmen. Seien $u, v, w \in W$ Welten in F. Dann lassen sich unter anderem folgende Axiome aufstellen:

Name	charakterisierende Eigenschaft	Gültig auf allen...
K	$\Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)$	Rahmen
Dual	$\Diamond\varphi \leftrightarrow \neg\Box\neg\varphi$	Rahmen
K4	$\Diamond\Diamond\varphi \to \Diamond\varphi$ $\forall u \forall v \forall w((uRv \wedge vRw) \Rightarrow uRw)$	transitiven Rahmen
T	$\varphi \to \Diamond\varphi$ $\forall u(uRu)$	reflexiven Rahmen
B	$\varphi \to \Box\Diamond\varphi$ $\forall u \forall v(uRv \Rightarrow vRu)$	symmetrischen Rahmen
S4	T+K4	reflexiven und transitiven Rahmen
S5	S4+B	Äquivalenzrelationen
D	$\Box\varphi \to \Diamond\varphi$ $\forall u \exists v(uRv)$	totalen Rahmen
E	$\Diamond\varphi \to \Box\Diamond\varphi$ $\forall u \forall v \forall w((uRv \wedge uRw) \Rightarrow vRw)$	euklidischen Rahmen

Aus den Axiomen lassen sich neue Formeln mittels folgender Inferenzregeln gewinnen:

MP	Modus Ponens:	Aus $\varphi \to \psi$ und φ folgere ψ
NR	Notwendigkeitsregel:	Aus φ folgere $\Box\varphi$

Eine modale Logik, die alle Tautologien über Φ, mindestens die Axiome K, Dual und die Inferenzregeln MP und NR umfasst, wird **normale Logik**, beziehungsweise **modales System** genannt. Die entsprechende Definition ist auf den Seiten 6f in [Mül09] zu finden.

3.3.1. Implementierung der Axiomenüberprüfung

Die Gültigkeit der Axiome K und Dual wird als gegeben angenommen und bedarf keiner Überprüfung. Damit eines der übrigen Axiome aus der Tabelle gilt, muss es für jede beliebige Formel gelten. Da es unendlich viele Formeln gibt, ist es nicht möglich, einfach alle Formeln mit dem Algorithmus 1 zu prüfen. Stattdessen werden die Eigenschaften des zugrunde liegenden Graphen untersucht.[1] Dabei steht n im Folgenden als die Anzahl der Knoten im Graphen.

Das Axiom K4 gilt auf allen transitiven Rahmen. Wenn eine Welt eine zweite sieht, dann sieht sie auch alle Welten, die von der zweiten aus zu sehen sind. Bei der Umsetzung

[1] Vergleiche: [Sch02b, 16]

werden für jeden Knoten die Nachfolgerknoten ermittelt und für jeden der direkten Nachfolgerknoten werden wiederum die Nachfolger bestimmt. Anschließend wird geprüft, ob zwischen dem ersten Knoten und den Knoten, die man über den Weg der Länge zwei erreichen kann, auch eine direkte Verbindung besteht. Aufgrund der drei ineinander verschachtelten Schleifen liegt die Laufzeit in $\mathcal{O}(n^3)$.

Um festzustellen, ob das Axiom T erfüllt ist, muss der Graph auf Reflexivität geprüft werden, also ob die jeweilige Welt sich selbst sieht. Hierfür muss bei jedem Knoten geprüft werden, ob eine Kante zu sich selbst existiert. Diese Überprüfung erfolgt in $\mathcal{O}(n)$.

Die Überprüfung, ob ein Graph symmetrisch ist und somit das Axiom B gilt, erfolgt in $\mathcal{O}(n^2)$. Dabei muss für jeden Knoten überprüft werden, ob es von den Knoten, die man erreichen kann, auch einen direkten Weg wieder zurück gibt. Das heißt es muss gelten: wenn eine Welt eine andere sieht, dann auch umgekehrt.

Das Axiom S4 besteht aus den Axiomen T und K4. Entsprechend ist die Schreibweise T+K4. Das bedeutet, dass das Axiom S4 genau dann gilt, wenn die Axiome T und K4 gelten. Die Laufzeit ist dementsprechend $\mathcal{O}(n^3) + \mathcal{O}(n) = \mathcal{O}(n^3)$.

Eine Äquivalenzrelation ist immer reflexiv, transitiv und symmetrisch. Es müssen also die Axiome S4 und B gelten, damit auch das Axiom S5 erfüllt ist. Die Laufzeit liegt in $\mathcal{O}(n^3)$.

Bei den totalen Rahmen muss jeder Knoten mindestens eine ausgehende Kante haben. Anders formuliert: jede Welt sieht mindestens eine andere Welt. Die Überprüfung erfolgt in linearer Zeit.

Die Euklidizität ist ähnlich zu der Transitivität definiert. Zwei Welten, die aus einer Welt zu sehen sind, sehen sich auch gegenseitig. Dementsprechend erfolgt die Überprüfung auch in diesem Fall durch drei ineinander verschachtelten Schleifen, was wiederum zur kubischen Laufzeit führt.

3.4. S5-Erfüllbarkeit

Sei φ eine modallogische Formel. Das Erfüllbarkeitsproblem S5-SAT ist definiert als:

S5-SAT=$\{\varphi : \varphi$ wird von einem S5-Modell erfüllt$\}$.

Wie in [Lad77] gezeigt ist das Problem NP-vollständig. Die Entscheidung, ob für eine Formel ein K, T oder S4 Modell existiert, auf dem die Formel gilt, ist hingegen PSPACE-vollständig. In Gegensatz zu Problemen aus NP sind die PSPACE-vollständige Probleme in polynomieller Zeit nicht verifizierbar. Eine praktische Umsetzung von PSPACE-Algorithmen ist demnach nicht sinnvoll. Aus diesem Grund wird in dieser Arbeit nur der Algorithmus zur Lösung des S5-SAT-Problems implementiert.

3.4.1. Algorithmus des S5-SAT-Solvers

In [Lad77] wird gezeigt, dass wenn eine Formel φ mit der modalen Tiefe m S5 erfüllbar ist, dann gibt es einen Rahmen mit höchstens $m+1$ Welten, welcher φ erfüllt. Im folgenden Algorithmus wird ein S5-Modell mit genau $m+1$ Welten erstellt und die Belegung der Variablen aus Φ in den Welten nacheinander durchprobiert. Wenn φ bei einer Belegung in einer der Welten erfüllt wird, dann wird der Durchlauf beendet und das Modell zurückgegeben. In diesem Fall ist $\varphi \in$ S5-SAT.

Algorithm 3: Funktion zur Lösung des Erfüllbarkeitsproblems auf S5-Modellen

Input: $\varphi \in ML$
Output: S5-model iff φ is satisfiable

```
1 m ← modal depth of φ;
2 n ← |Φ|;
3 M ← S5-model with m+1 worlds;
4 l ← binary counter with size of (m + 1) * n;
5 for l ← 0...0 to 1...1 do
6     labeling L of the worlds in M as follows: first n bits of l for the variables in world
      w_0 etc.;
7     for i ← 0 to m do
8         if M(L), w_i ⊨ φ then
9             return M(L)
```

Der Algorithmus terminiert, da beide Schleifen endlich viele Schritte durchlaufen. Er hat eine exponentielle Laufzeit.

3.4.2. Implementierung des S5-SAT-Solvers

Die in den Zeilen 1 und 2 geforderten Werte für die modale Tiefe, sowie die Anzahl der atomaren Variablen, werden bei dem Aufbau des Formelbaums ermittelt. Die Bestimmung der modalen Tiefe erfolgt unter Ausnutzung der internen Struktur, wie in der Definition 2.4 beschrieben. Die Variablen werden in einem Set gespeichert. Dadurch wird garantiert, dass keine Variable doppelt vorkommt.

Zur Erstellung eines S5-Modells wird ein Graph mit $m+1$ Knoten aufgebaut. Dies wird in Zeile 3 gefordert. Anschließend werden alle Welten miteinander verbunden.

Der Binärzähler aus Zeile 4 wird als Boolean-Array implementiert. Die Belegung des Arrays wird mittels einer Methode, die eine Dezimalzahl in eine Dualzahl umrechnet, festgelegt. Die Anzahl der Durchläufe der Schleife aus Zeile 5 entspricht also der Anzahl der möglichen unterschiedlichen Belegungen des Arrays. Aus der Länge des Arrays von $(m+1)*n$ ergibt sich, dass die Schleife bis $2^{(m+1)*n}$ laufen muss. Bei jedem dieser

Durchläufe wird die Methode zur Belegung des Binärzählers mit dem aktuellen Wert der Laufvariable aufgerufen.

Wie in 3.1.1 beschrieben, wird in jeder Welt des Modells ein Boolean-Array angelegt. Darin wird gespeichert, ob eine Variable in der jeweiligen Welt gilt. Die Zeile 6 sieht vor, dass die ersten n Werte des Binärzählers als Belegung der ersten Welt im Modell, die nächsten n Werte für die Belegung der zweiten Welt usw. verwendet werden. Die Abbildung 3.2 verdeutlicht dieses Prinzip. Java stellt Methoden zur Verfügung, mit denen man die Feldabschnitte mit Bereichsangaben kopieren kann. Mittels entsprechender Methode für die Boolean-Arrays erfolgt auf diese Weise das Labeling der Variablen in den Welten.

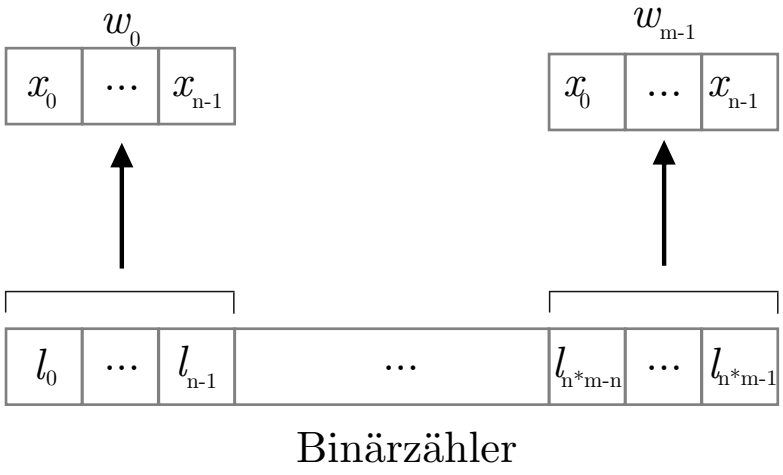

Abbildung 3.2.: Labeling unter der Verwendung des Binärzählers

Anschließend wird in jeder Welt mittels des Algorithmus 1 geprüft, ob die Formel bei der aktuellen Belegung erfüllt ist. Sollte dies bei einer Welt der Fall sein, wird das Modell mit der aktuellen Belegung zurückgegeben.

4. Ausblick

Das in dieser Arbeit entwickelte Tool ist so erstellt worden, dass eine Weiterentwicklung unproblematisch möglich ist. An dieser Stelle soll der Ausblick auf mögliche weitere Entwicklungen des Tools gegeben werden.

Die erstellten Tools decken nur einen sehr kleinen Teil des Forschungsgebietes ab. Interessant wäre zum Beispiel die Überprüfung auf Unerfüllbarkeit einer Formel mittels Resolution. Des Weiteren wäre eine Erweiterung auf multimodale Logiken, wie zum Beispiel temporale Logik, denkbar. Auf diese Weise könnten Aussagen zu unterschiedlichen Zeitpunkten verschiedene Wahrheitswerte haben.

Wie in Anhang A beschrieben, werden die Tools über die Konsole gesteuert. Damit die Benutzung intuitiver erfolgen könnte, würde sich die Entwicklung einer grafischen Benutzerschnittstelle lohnen. Diese würde auch unerfahrenen Benutzern die Arbeit erleichtern.

Literaturverzeichnis

[BdRV02] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic.* Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2002.

[Bey11] PD Dr. Olaf Beyersdorff. Logik. *Skript zur Vorlesung SS 2011, Institut für Theoretische Informatik, Leibniz Universität Hannover*, 2011.

[Lad77] Richard E. Ladner. The Computational Complexity of Provability in Systems of Modal Propositional Logic. *SIAM Journal on Computing, vol. 6, no. 3*, pages 467–480, 1977.

[Mei12] Dr. Arne Meier. Advanced Logics. *Lecture Notes, Summer Term 2012, Institut für Theoretische Informatik, Leibniz Universität Hannover*, 2012.

[Mül09] Julian-Steffen Müller. Vollständigkeit in modalen Logiken. *Bachelorarbeit, Institut für Theoretische Informatik, Universität Hannover*, 2009.

[Sch02a] Torsten Schatz. Einführung in die Logik. *Skript zur Vorlesung SS 2002, Philosophisches Seminar der Universität Tübingen*, 2002.

[Sch02b] Thomas Schneider. Komplexität modaler Logiken. *Diplomarbeit, Fakultät für Mathemaik und Informatik, Universität Jena*, 2002.

[Weg08] Dr. Sven-Ake Wegner. Eine kurze Einführung in Gottlob Freges "Begriffsschrift", 2008. URL: http://www2.math.uni-wuppertal.de/~fa/wegner/FREGE.pdf.

A. Bedienungsanleitung der erstellten Tools

Die erstellte Anwendung setzt Java Version 1.6 voraus. Die Tools werden mittels Konsole bedient. Nachdem die Konsole gestartet wurde, sollte der Anwender in das Verzeichnis wechseln, in dem die `mltools.jar` abgelegt wurde. Die Anwendung wird mittels folgender Eingabe gestartet:

```
java -jar mltools.jar <parameter>
```

Die folgende Tabelle bietet eine Übersicht über die möglichen Parameter:

Übergabeparameter	Bedeutung
`-i <Pfad>`	Pfad zu der XML-Datei des Modells
`-o <Pfad>`	Pfad zu der Output-Datei
`-f <Formel> [<Welt>]`	Einzelne Formel. Wenn Model Checking erwünscht ist, dann muss der Name der Welt als weiterer Parameter übergeben werden.
`-f <Formel> all`	Formel, die im ganzen Modell überprüft werden soll.
`-ff <Pfad>`	Für den Fall, dass mehrere Formeln getestet werden sollen, kann der Pfad zu einer CSV-Datei übergeben werden.
`-a`	Axiomenprüfung aktivieren
`-s`	S5-Sat ausführen
`-h`	Übersicht über alle zur Verfügung stehende Befehle

Die Eingabe der Operatoren erfolgt gemäß folgender Tabelle:

Operator	Symbol
Konjunktion	& oder $*$
Disjunktion	| oder $+$
Negation	!
Implikation	->
Biimplikation	<->
Diamond	<>
Box	[]

Die Syntax der XML-Datei wurde in 3.1.1 gezeigt.

Die CSV-Datei zur Eingabe mehrerer Formeln für Model Checking kann zum Beispiel wie folgt aussehen:

```
FORMEL;WELT
<>p&(!p|<><>q);all
<>p|[]q;w1
```

Der CSV-Header wird nicht verarbeitet und dient ausschließlich als Hilfestellung für den Benutzer. In der Spalte „WELT“ kann entweder der Name einer Welt stehen oder der String „all“. Mit letzterem erfolgt die Anweisung die Formel auf dem ganzen Modell zu prüfen.

Die CSV-Datei, die nach dem Model Checking erstellt wird, kann wie folgt aussehen:

```
FORMEL;WELT;ERFUELLT
<>p&(!p|<><>q);all;nein
<>p|[]q;w1;ja
```

Die ersten beiden Spalten sind analog zu der CSV-Datei zur Eingabe der Formeln aufgebaut. In der Spalte „ERFUELLT“ steht das Ergebnis des ML-MC Algorithmus. Wenn die Prüfung positiv war, dann steht in der Spalte der String „ja“ sonst „nein“.

Manche Parameter schließen sich gegenseitig aus. Sollte z.B. das Ergebnis des Model Checking Algorithmus in die Output-Datei geschrieben werden und gleichzeitig die Axiomenprüfung erfolgen, so werden die Axiome auf der Konsole ausgegeben. In dem Fall, dass nur die Axiome geprüft werden sollen, kann die Output-Datei wie folgt aussehen:

```
AXIOM;ERFUELLT
D;ja
E;nein
S4;ja
T;ja
S5;nein
B;nein
K4;ja
```

Wenn S5-Sat ausgewählt wurde, werden die übrigen Parameter mit Ausnahme von `-f <Formel>` und `-o <Pfad>` ignoriert. Wenn keine Ausgabe in der Datei erfolgen soll, dann werden auf der Konsole die Bestandteile des Modells ausgegeben, welches die Formel erfüllt. Sollte ein solches Modell nicht existieren, wird der Benutzer darauf hingewiesen. Die Ausgabe kann wie folgt aussehen:

```
M=((W,R),V)
PHI={q,p}
W={w0,w1}
R={(w0,w0),(w0,w1),(w1,w0),(w1,w1)}
V(q)={w1}
V(p)={w1}
```

Wurde der Pfad zur Output-Datei mit übergeben, wird eine XML-Datei, wie in 3.1.1 beschrieben, erstellt.

Wenn der Parameter `-h` mit übergeben wurde, werden alle zur Verfügung stehende Befehle ausgegeben. Diese Ausgabe erfolgt auch in dem Fall, dass die übergebenen Parameter nicht korrekt waren.