Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Theoretische Informatik

Resolution für Modallogik

Bachelorarbeit

im Studiengang Informatik

von

Sergej Perschin

Hannover, 17. September 2017

Erstprüfer:	Prof. Dr. Heribert Vollmer
Zweitprüfer:	Dr. Arne Meier
Betreuer:	M. Sc. Martin Lück

Erklärung der Selbstständigkeit

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 17. September 2017

Sergej Perschin

Inhaltsverzeichnis

1 Einleitung

In der formalen Logik ist Resolution eine Methode die Gültigkeit von Formeln zu überprüfen. Ihr Ursprung kann zurückverfolgt werden zu Robinson (1965) [Rob65], sowie Davis und Putnam (1960) [DP60], bis hin zur Einführung durch Blake (1937) [Bla37].
Im seiner einfachsten Form wird Resolution auf Aussagenlogik angewandt. Eine solche Resolution kann zum Beispiel wie folgt aussehen:

$$\frac{C \qquad D}{(C \setminus \{p\}) \cup (D \setminus \{\neg p\})}$$

Hier wird aus den Klauseln C und D mit $p \in C$ und $\neg p \in D$ die Resolvente gebildet.
Enjalbert und Farinas Del Cerro beschäftigen sich in „Modal Resolution in Clausal Form“ [EFDC89] hingegen mit der Anwendung von Resolution auf Modallogik.
Die Modallogik ist eine Erweiterung der Aussagenlogik. Sie ergänzt die Aussagenlogik um die Operatoren „möglich“ und „notwendig“, sodass Aussagen nicht nur die Werte „wahr“ oder „falsch“ annehmen können. So kann eine falsche Aussage möglich sein und eine wahre Aussage notwendig. Beispielsweise sind neben den Aussagen „Alle Quadrate haben vier Ecken“ und „Die Sonne scheint“ auch „Notwendigerweise haben alle Quadrate vier Ecken“ und „Möglicherweise scheint die Sonne“ möglich. Eingeführt wurde die Modallogik unter anderem durch Leibniz (Die Theodizee, 1710) [Lei10], Kripke (Naming and Necessity, 1980) [Kri80] und Carnap (Meaning and Necessity, 1947) [Car47].

Das Ziel dieser Arbeit ist es, die Ansätze aus [EFDC89] zusammenzufassen und wiederzugeben, wobei nur das Modalsystem K betrachtet wird. Außerdem soll die modale Resolution in der Programmiersprache Java implementiert werden. Das Programm soll in der Lage sein, eine Resolution auf modallogischen Formeln durchzuführen und zu entscheiden, ob eine Klauselmenge erfüllbar ist.
Hier vorerst ein erster Blick darauf, wie eine modale Resolution aussehen kann:

$$\text{R2}\ \dfrac{\text{R2}\ \dfrac{\Diamond(\neg p \vee q) \qquad \Box\neg q}{\Diamond((\neg p \vee q) \wedge \neg p)} \qquad \Box p}{\bot}$$

Es wird eine Resolutionswiderlegung der Klauselmenge $\{\Diamond(\neg p \vee q), \Box\neg q, \Box p\}$ durch mehrfache Anwendung der Schlussregel R2 erreicht.
Die Arbeit ist folgendermaßen aufgebaut:
Zunächst werden die theoretischen Grundlagen zur Modallogik und klassischen Resolution aufgegriffen. Diese werden in ein modales Resolutionssystem zusammengeführt. Dessen Axiome und Regeln sollen vorgestellt und deren Vollständigkeit soll gezeigt werden, gefolgt von Anwendungsbeispielen. Zum Abschluss des theoretischen Teils wird ein Entscheidungsalgorithmus auf Basis der modalen Resolution beschrieben.
Im zweiten Teil der Arbeit soll die Implementierung des Algorithmus in einem Java-Programm weiter ausgeführt werden. Es wird der Aufbau des Programms beschrieben und die innere Funktionsweise erklärt. Eine kurze Anleitung zeigt, wie das Programm bedient wird. Abschließend wird durch beispielhafte Eingaben und Ausgaben die Funktion veranschaulicht.

2 Modallogik

In diesem Kapitel wird knapp zusammengefasst, wie das modallogische System aufgebaut ist, für das in dieser Arbeit ein Resolutionskalkül entwickelt wird.

2.1 Modallogik

Die Modallogik ist eine Erweiterung der Aussagenlogik mit den unären Konnektoren $\Box$ und $\Diamond$. Ihre Syntax und Semantik sind wie folgt definiert:

Syntax: Die Syntax setzt sich aus folgenden Elementen zusammen:

- einer Menge von aussagenlogischen Variablen: $p, q, r, \ldots$,
- Konnektoren der aussagenlogische Sprache: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\equiv$,
- dem unären Konnektor $\Box$ (notwendig)
- dem unären Konnektor $\Diamond$ (möglich), definiert als $\Diamond A \equiv \neg\Box\neg A$,
- der Konstante $\bot$ (falsch).

Semantik: Ein Kripke-Modell ist ein Tripel $M = \langle W, R, m\rangle$ mit

- einer nichtleeren Menge von Welten W,
- einer binären Relation $R \subseteq W \times W$,
- einer Funktion m, die jeder Variable p eine Menge $m(p)$ von Welten zuordnet.

Seien A, B modale Formeln, $M = \langle W, R, m\rangle$ ein Kripke-Modell und $w \in W$ eine Welt. Die Gültigkeit von A, bzw. der Ausdruck „A ist wahr in der Welt w im Model M“ (in Symbolen $M, w \vDash A$), wird definiert durch:

- $M, w \vDash p$ gdw. $w \in m(p)$ falls p eine aussagenlogische Variable ist,
- $M, w \vDash \neg A$ gdw. nicht $M, w \vDash A$ gilt,
- $M, w \vDash A \wedge B$ gdw. $M, w \vDash A$ und $M, w \vDash B$ gilt,
- $M, w \vDash \Box A$ gdw. für alle $w' \in W$ mit wRw' gilt: $M, w' \vDash A$,

Sei $\mathscr{M}$ eine Klasse von Modellen. Eine Formel A ist $\mathscr{M}$-gültig (in Symbolen: $\vDash_{\mathscr{M}} A$) genau dann, wenn für alle $M \in \mathscr{M}$ und für jede Welt w aus der Menge der Welten von M gilt: $M, w \vDash A$. Falls $\mathscr{M}$ die Klasse aller Modelle ist, dann bezeichnet man A als „gültig“ und A als modale Tautologie (in Symbolen: $\vDash A$).
Eine Formel A ist erfüllbar (bzw. $\mathscr{M}$-erfüllbar) genau dann, wenn es ein $M = \langle W, R, V\rangle$ (bzw. ein Modell M in $\mathscr{M}$) und $w \in W$ gibt mit $M, w \vDash A$. Falls dies nicht gilt, so ist A unerfüllbar (oder inkonsistent). Analog wird die Erfüllbarkeit für Mengen von Formeln definiert.

2.2 Modalsysteme

Das syntaktische Gegenstück zu den semantischen Kripke-Modellen sind die Modalsysteme. Sie setzen sich aus Axiomen aus der Aussagenlogik, Schlussregeln und weiteren Axiomen zusammen.

2.2.1 System K

Das allgemeinste Modalsystem ist das System K, welches die Klasse aller Kripke-Modelle beschreibt. Das System K besteht aus den Axiomen der Aussagenlogik und den folgenden Axiomen und Schlussregeln:

Axiome:

$$\text{(K):} \qquad (\Box A \wedge \Box(A \to B)) \to \Box B).$$

Schlussregeln:

$$\text{Nec (Nezessisierung):} \qquad \frac{A}{\Box A},$$

$$\text{MP (Modus Ponens):} \qquad \frac{A \qquad A \to B}{B}.$$

Man schreibt „$\vdash_K A$“ genau dann, wenn A mithilfe von Axiomen und Schlussregeln aus K abgeleitet werden kann.
Des Weiteren lassen sich folgende nützliche Schlussregeln aus dem System ableiten:

$$\text{Nec':} \qquad \frac{A \to B}{\Box A \to \Box B},$$

$$\text{Pos:} \qquad \frac{A \to B}{\Diamond A \to \Diamond B}.$$

Das System besitzt folgendes Korrektheits- und Vollständigkeitstheorem:

Theorem 1. ([Kri59]) A ist genau dann gültig, wenn A in K bewiesen werden kann. In Symbolen: $\vDash A \Leftrightarrow \vdash_K A$.

2.3 Disjunktive und konjunktive Normalform

Für die Resolution ist die Notation in konjunktiver und disjunktiver Normalform wichtig. Man spricht von einer modalen Formel in disjunktiver Normalform (DNF), wenn sie als eine Disjunktion in folgender Form vorliegt

$$C = L_1 \vee L_2 \vee ... \vee L_n \vee \Box D_1 \vee \Box D_2 \vee ... \vee \Box D_p \vee \Diamond A_1 \vee \Diamond A_2 \vee ... \vee \Diamond A_q$$

wobei

- jedes L_i ein Literal (negierte oder nicht negierte Variable) ist,
- jedes D_i in disjunktiver Normalform ist,
- jedes A_i in konjunktiver Normalform ist.

Eine modale Formel liegt in konjunktiver Normalform (KNF) vor, wenn sie eine Konjunktion in der Form

$$F = C_1 \wedge C_2 \wedge ... \wedge C_q$$

ist, wobei jedes C_i in disjunktiver Normalform ist. Disjunktionsterme werden fortgehend als Klauseln bezeichnet. Dies ergibt sich daraus, dass die hier beschriebenen DNF, beschränkt auf aussagenlogische Formeln, den üblichen Klauseln aus der Aussagenlogik gleichen. Unter einer *Einheitsklausel* wird eine Klausel mit nur einem Disjunkt verstanden. Für die leere Klausel wird die Konstante $\bot$ (falsch) verwendet.

Beispiel 1. Beispiele für Formeln in KNF sind:

$$\Box(p \vee q \vee \Diamond(r \wedge t)),$$
$$\Diamond((p \vee q) \wedge t) \wedge \neg p,$$
$$\neg p \vee \Box(\Box p \vee \Diamond(q \wedge \Box r)).$$

Alternativ wird die Notation für Konjunktionen $C_1 \wedge C_2 \wedge ... \wedge C_n$ vereinfacht als Mengen von Klauseln $\{C_1, C_2, ..., C_n\}$ dargestellt. Für die obigen Beispiele ergibt sich somit:

$$\Box(p \vee q \vee \Diamond(r, t)),$$
$$\Diamond((p \vee q), t), \neg p,$$
$$\neg p \vee \Box(\Box p \vee \Diamond(q, \Box r)).$$

Definition 1. Die *modale Tiefe* $d(A)$ einer Formel A wird wie folgt rekursiv definiert:

- $d(A) = 0$, wenn A ein Literal ist,
- $d(A \circ B) = \max(d(A), d(B))$, falls $\circ \in \{\wedge, \vee\}$,
- $d(\neg A) = d(A)$,
- $d(\Delta A) = d(A) + 1$, falls $\Delta \in \{\Box, \Diamond\}$.

Behauptung 1. Jede modale Formel F lässt sich zu einer Formel F' in KNF umwandeln, sodass $\vdash_K F \equiv F'$ gilt.

Beweis. Unter Verwendung der Axiome und der Definition: $\neg\Box A \equiv \Diamond\neg A$ werden die Negationen nach innen bewegt, bis sie direkt vor einer Variable stehen. Dann wird die konjunktive Normalform einer Formel induktiv definiert:

- Wenn $d(F) = 0$, dann enthält F keine modalen Operatoren und wird wie eine aussagenlogische Formel in KNF umgewandelt.
- Wenn $F = \Box F_1$, dann wird F_1 zu KNF umgewandelt und das Axiom K benutzt, um $\Box$ über die Konjunktionen zu verteilen. Dadurch entsteht eine Konjunktion aus Einheitsklauseln:
$$\Box C_1 \wedge ... \wedge \Box C_n,$$
wobei $F_1 = \bigwedge_{i=1}^{n} C_i$ und jedes $\Box C_i$ eine Einheitsklausel ist, und jedes C_i eine Klausel in DNF.
- Wenn $F = \Diamond F_1$, dann sei F_2 die KNF von F_1, und $\Diamond F_2$ ist dann die Normalform für F. Es entsteht die Formel:
$$\Diamond(C_1 \wedge ... \wedge C_n),$$
welche zwar als Einheitsklausel in DNF vorliegt, aber gleichzeitig eine Formel in KNF ist, die nur ein Konjunkt besitzt.
- Ansonsten ist F eine boolsche Kombination aus Literalen und Formeln, die mit einem modalen Operator beginnen. Letztere können normalisiert werden um danach die gesamte Formel in konjunktive Normalform umzuwandeln.

□

Beispiel 2. Für die Formel
$$\neg\Box(\Diamond p \vee \neg\Diamond(q \vee (r \wedge \Diamond p)))$$
erhält man nach dem Bewegen der Negationen nach innen:
$$\Diamond(\Box\neg p \wedge \Diamond(q \vee (r \wedge \Diamond p))).$$
Nun wird $q \vee (r \wedge \Diamond p)$ normalisiert zu
$$(q \vee r) \wedge (q \vee \Diamond p),$$
sodass schließlich
$$\Diamond(\Box\neg p \wedge \Diamond((q \vee r) \wedge (q \vee \Diamond p)))$$
entsteht, welches in Klauselmengen-Notation geschrieben wird als
$$\Diamond(\Box\neg p, \Diamond(q \vee r, q \vee \Diamond p)).$$

3 Modale Resolution

In diesem Kapitel wird zunächst die Resolution für Aussagenlogik erklärt, wonach die gewünschte Funktionsweise der modalen Resolution erläutert wird. Danach wird das System RK aufgestellt, das Resolutionen auf modallogischen Formeln auf diese Weise durchführt. Zum Schluss wird ein Entscheidungsalgorithmus vorgestellt, der die Erfüllbarkeit von modalen Klauselmengen bestimmt.

3.1 Resolution

In der Aussagenlogik wird die klassische Resolution verwendet, um die Allgemeingültigkeit einer Formel zu zeigen, indem aus der Negation der Formel ein Widerspruch abgeleitet wird. Dazu wird die wie folgt definierte Resolutionsregel verwendet:
Aus zwei Klauseln, die ein komplementäres Literal enthalten, kann eine Resolvente gebildet werden, indem die beiden Klauseln ohne dieses Literal vereinigt werden.
Seien also C_1, C_2 Klauseln und p eine Variable. Angenommen die Klauseln $p \vee C_1$ und $\neg p \vee C_2$ seien bewiesen, dann ist eine Resolutionsableitung:

$$\frac{p \vee C_1 \qquad \neg p \vee C_2}{C_1 \vee C_2}.$$

Die Erfüllbarkeit der Resolvente ist eine notwendige Bedienung für die gleichzeitige Erfüllbarkeit beider Ausgangsklauseln. Daraus ergibt sich:

Theorem 2 ([BRV01])**.** Eine Menge von Klauseln S ist genau dann unerfüllbar, wenn die leere Klausel aus S abgeleitet werden kann. Eine Resolutionsableitung der leeren Klausel aus S wird als Resolutionswiderlegung von S bezeichnet.

In der Modallogik wird die Anwendung der Resolutionsregel durch die Existenz der modalen Operatoren erschwert. Folgende Resolutionen sollen möglich sein:

$$\frac{\Box(p \vee q) \qquad \Diamond\neg p}{\Diamond(\neg p, q)}$$

Bedeutung: Wenn für alle nachfolgenden Welten $(p \vee q)$ gilt, und mindestens eine Welt gibt, in der $\neg p$ gilt, so muss es auch mindestens eine Welt geben, in der $(\neg p, q)$ gilt.

$$\frac{\Box(p \vee q) \qquad \Box\neg p}{\Box q}$$

Bedeutung: In allen nachfolgenden Welten gilt $p \vee q$ und $\neg p$, somit muss auch für jede nachfolgende Welt q gelten.
Jedoch sollte eine Resolution wie

$$\frac{\Diamond(p \vee q) \qquad \Diamond\neg p}{\Diamond(\neg p, q)}$$

nicht möglich sein, da $p \vee q$ und $\neg p$ nicht zwingend in derselben Welt gelten müssen.
Unter Berücksichtigung dieser Punkte wird das Resolutionssystem RK im folgenden Abschnitt definiert.

3.1.1 System RK

Das System RK ist ein Resolutionssystem für die Bestimmung von Resolventen in K. Es besteht aus Berechnungsregeln, die die Resolventen für Formelmengen in K bestimmen, aus Vereinfachungsregeln, die diese Resolventen simplifizieren, und aus Schlussregeln, die eine Relation in eine Resolutionsableitung überführen.

Relationen: Es werden zwei Relationen auf Klauseln definiert:

$$\Sigma(A, B) \to C \qquad \text{(C ist direkte Resolvente von A und B)},$$
$$\Gamma(A) \to C \qquad \text{(C ist direkte Resolvente von A)}.$$

Die Σ-Relation ist die kleinste Relation, die die Axiome

$$(A_1): \qquad \Sigma(p, \neg p) \to \bot$$
$$(A_2): \qquad \Sigma(\bot, A) \to \bot$$

enthält und unter den folgenden Σ-Regeln abgeschlossen ist.

Σ-Regeln:

$\vee$-Regel:
$$\frac{\Sigma(A, B) \to C}{\Sigma(A \vee D_1, B \vee D_2) \to C \vee D_1 \vee D_2}$$

$\Box\Diamond$-Regel:
$$\frac{\Sigma(A, B) \to C}{\Sigma(\Box A, \Diamond(B, E)) \to \Diamond(B, C, E)}$$

$\Box\Box$-Regel:
$$\frac{\Sigma(A, B) \to C}{\Sigma(\Box A, \Box B) \to \Box C}$$

Die Γ-Relation ist die kleinste Relation, die unter den Γ-Regeln abgeschlossen ist.

Γ-Regeln:

$\Diamond$-Regel 1:
$$\frac{\Sigma(A, B) \to C}{\Gamma(\Diamond(A, B, F)) \to \Diamond(A, B, C, F)}$$

$\Diamond$-Regel 2:
$$\frac{\Gamma(A) \to B}{\Gamma(\Diamond(A, F)) \to \Diamond(B, A, F)}$$

$\vee$-Regel:
$$\frac{\Gamma(A) \to B}{\Gamma(A \vee C) \to B \vee C}$$

$\Box$-Regel:
$$\frac{\Gamma(A) \to B}{\Gamma(\Box A) \to \Box B}$$

A, B, C, D, D_1, D_2 sind hierbei Klauseln, E, F sind Mengen von Klauseln in KNF.

3.1.2 Vereinfachungsregeln

$A \approx B$ steht für die Relation „A kann zu B vereinfacht werden“. Sie enthält folgende Regeln:

$$
\begin{aligned}
(S_1): &\quad \Diamond \bot \approx \bot \\
(S_2): &\quad \bot \vee D \approx D \\
(S_3): &\quad \bot, E \approx \bot \\
(S_4): &\quad A \vee A \vee D \approx A \vee D
\end{aligned}
$$

Für jede Formel F gibt es eine Formel F', sodass $F \approx F'$ gilt und F' nicht weiter vereinfacht werden kann. F' wird als Normalform von F bezeichnet.
C wird genau dann als Resolvente von A und B bezeichnet, wenn es ein C' gibt, sodass $\Sigma(A, B) \to C'$ gilt und C die Normalform von C' ist. Geschrieben: $\Sigma(A, B) \Rightarrow C$ für „C ist Resolvente von A und B“. Analog $\Gamma(A) \Rightarrow C$ für „C ist Resolvente von A“.

3.1.3 Schlussregeln

$$
(\mathrm{R1}): \qquad \frac{C}{D} \text{ wenn } \Gamma(C) \Rightarrow D
$$

$$
(\mathrm{R2}): \qquad \frac{C1 \qquad C2}{D} \text{ wenn } \Sigma(C1, C2) \Rightarrow D
$$

wobei C, C_1, C_2, D Klauseln sind.
Die Deduktion in RK einer Klausel D aus einer Menge S ist ein Baum, dessen Wurzel D ist, und dessen Blätter Klauseln aus S sind. Jeder interne Knoten C hat Nachfolger A und B (bzw. A) genau dann, wenn die Regel R2 (bzw. R1) mit Prämissen A und B (bzw. A) und Konklusion C angewandt werden kann.
Wenn es eine Deduktion von D aus S in RK gibt, dann ist D eine RK-Konsequenz von S, geschrieben: $S \vdash_{RK} D$.
Eine Deduktion von $\bot$ aus S entspricht einer Widerlegung von S.

Beispiel 3. Gesucht ist die Resolvente aus $\Box p$ und $\Diamond(\neg p \vee q)$.

$$
\Box\Diamond\text{-Regel}\ \frac{\vee\text{-Regel}\ \dfrac{\Sigma(p, \neg p) \to \bot}{\Sigma(p, \neg p \vee q) \to \bot \vee q}}{\Sigma(\Box p, \Diamond(\neg p \vee q)) \to \Diamond(\neg p \vee q, \bot \vee q)}
$$

Dann wird simplifiziert:

$$
\Diamond(\neg p \vee q, \bot \vee q) \approx \Diamond(\neg p \vee q, q).
$$

Somit ergibt sich

$$
\Sigma(\Box p, \Diamond(\neg p \vee q)) \to \Diamond(\neg p \vee q, q).
$$

Zusammen mit $\Box\neg q$ erhält man eine Resolutionswiderlegung:

$$\Box\Diamond\text{-Regel}\ \frac{\Sigma(p,\neg p)\to\bot}{\Sigma(\Diamond(\neg p\vee q,q),\Box\neg q)\to\Diamond(\neg p\vee q,q,\bot)},$$

und aus $\Diamond(\neg p\vee q,q,\bot)\approx\Diamond\bot\approx\bot$ folgt $\Sigma(\Diamond(\neg p\vee q,q),\Box\neg q)\to\bot$.
Damit ergibt sich folgende Widerlegung von $\{\Box p,\Diamond(\neg p\vee q),\Box\neg q\}$:

$$\text{R2}\ \frac{\text{R2}\ \dfrac{\Box p\qquad\Diamond(\neg p\vee q)}{\Diamond(\neg p\vee q,q)}\qquad\Box\neg q}{\bot}$$

3.2 Widerspruchsfreiheit

Theorem 3. RK ist widerspruchsfrei relativ zu K.

In anderen Worten, es lässt sich kein Widerspruch aus RK ableiten, der nicht bereits aus K ableitbar ist.

Lemma 1.

(i)	Wenn $\Sigma(A,B)\to C$, dann $\vdash_K A\wedge B\to C$.
(ii)	Wenn $\Gamma(A)\to C$, dann $\vdash_K A\to C$.
(iii)	Wenn $S\vdash_{RK} C$, dann $\vdash_K S\to C$.

Beweis. Für den Beweis von (i) und (ii) wird eine Induktion über die Anzahl verwendeter Resolutionsregeln durchgeführt: Wenn $\Sigma(A,B)\to C$ ein Axiom ist, so ist es entweder $\Sigma(p,\neg p)\to\bot$ und unter Verwendung von (i) folgt $\vdash_K p\wedge\neg p\to\bot$, oder es ist $\Sigma(\bot,A)\to\bot$ und und dann $\vdash_K\bot\wedge A\to\bot$.
Ist die zuletzt verwendete Regel eine Σ-$\vee$-Regel der Form

$$\Sigma(A\vee D_1,B\vee D_2)\to C\vee D_1\vee D_2,$$

dann folgt nach der Induktionsvoraussetzung $\vdash_K A\wedge B\to C$ (IV) und aussagenlogischer Umformungen:

$$\vdash_K(A\vee D_1)\wedge(B\vee D_2)\to C\vee D_1\vee D_2$$

Im Fall einer Σ-$\Box\Diamond$-Regel folgt aus (IV) und der Schlussregel Nec:

$$\vdash_K\Box(A\wedge B\to C),$$

und aus $\vdash_K\Box X\wedge\Diamond Y\to\Diamond(X\wedge Y)$ erhält man:

$$\begin{aligned}&\vdash_K\Box A\wedge\Diamond(B\wedge E)\to\Diamond(A\wedge B\wedge E)\\&\vdash_K\Box(A\wedge B\to C)\wedge\Diamond(A\wedge B\wedge E)\to\Diamond(A\wedge B\wedge E\wedge(A\wedge B\to C))\\&\vdash_K(A\wedge B\wedge E\wedge(A\wedge B\to C))\to B\wedge C\wedge E.\end{aligned}$$

Nach Anwendung der Schlussregel Pos folgt schließlich:

$$\vdash_K \Diamond(A \wedge B \wedge E \wedge (A \wedge B \rightarrow C)) \rightarrow \Diamond(B \wedge C \wedge E).$$

Beweis für die Σ-□□-Regel erfolgt analog.
Wenn die zuletzt verwendete Regel eine Γ-Regel ist, dann kann (ii) auf ähnlichem Weg bewiesen werden. Der Beweis für (iii) folgt direkt über Verwendung von Modus Ponens unter der Betrachtung, dass A und B in K logisch äquivalent sind, wenn $A \approx B$. □

3.3 Vollständigkeit

Im folgenden Theorem wird die Widerlegungsvollständigkeit von RK gezeigt.

Theorem 4. Jede unerfüllbare Menge von Klauseln ist RK-widerlegbar.

Beweis. Für den Beweis wird für jede Menge S von Klauseln ein so genannter K-Baum definiert. Ist S erfüllbar, dann kann ein Modell aus dem K-Baum von S konstruiert werden. Wenn S unerfüllbar ist, dann kann daraus eine RK-Widerlegung konstruiert werden.

Definition 2 (Baum)**.** Ein *Baum* ist ein Tripel $\langle A, T, r\rangle$ mit einer Menge von Knoten A, der Relation T und der Wurzel r.

- y wird als *Nachfolger* von x bezeichnet, wenn xTy gilt.
- ein Baum $u' = (A', T', r')$ wird als Teilbaum von $u = (A, T, r)$ bezeichnet genau dann, wenn:
  - $A' \subseteq A$,
  - $T' \subseteq T$,
  - $r' = r$,
  - jedes Blatt in u' ein Blatt in u ist.
- die *Tiefe* $d(w)$ eines Knotens w ist wie folgt definiert:
  - Blätter haben die Tiefe 0,
  - wenn w die Nachfolger $w_1, ..., w_n$ hat, dann $d(w) = \max\{d(w_1), \ldots, d(w_n)\} + 1$.

3.3.1 K-Baum-Konstruktion

Sei S eine Menge von Klauseln. Der K-Baum u von S wird folgenderweise aufgebaut: Die Knoten von u sind Klauselmengen. Die Wurzel von u ist S selbst. u wird konstruiert, indem die Operationen OP1 und OP2 alternierend verwendet werden, bis die Endbedingung zutrifft.

OP1: Für jedes Blatt w in u finde eine Klausel $C \in w$ in der Form $C = C_1 \vee C_2$.
Füge zwei Nachfolger zu w hinzu:

$$w_1 = w - \{C\} \cup \{C_1\}$$
$$w_2 = w - \{C\} \cup \{C_2\}$$

Am Ende von OP1 ist jedes Blatt eine Menge von Einheitsklauseln.
OP2: Für jedes Blatt w in u:

- Wenn p und $\neg p$ in w vorkommen, schreite mit dem nächsten Knoten fort.
- Ansonsten: w ist eine Menge von Einheitsklauseln der Form

  $$w = L_1, ..., L_m, \Box A_1, ..., \Box A_n, \Diamond P_1, ..., \Diamond P_q.$$

  Wenn $q > 0$, dann füge alle $w_i = \{P_i, A_1, ..., A_n\}$ für $i = 1, .., q$ als Nachfolger von w hinzu. Jedes w_i wird als *K-Projektion* von w bezeichnet.

Endbedingung: OP2 ist nicht anwendbar und somit ist jedes Blatt von u eine Menge von Einheitsklauseln, die entweder p und $\neg p$ enthalten, oder keine Klausel der Form $\Diamond A$.

Das Verfahren terminiert, da zum einen die Klauselmengen von Knoten, die aus OP1 entstehen, strikt kleiner sind als die ihrer Eltern, während ihre modale Tiefe gleich bleibt. Und zum anderen haben die Knoten, die aus OP2 entstehen, eine strikt kleinere modale Tiefe, als die ihrer Eltern.

Definition 3. Für K-Bäume werden folgende Knotentypen festgelegt:

- Typ 1: jeder Knoten, auf dem OP1 angewandt wurde
- Typ 2: jeder andere Knoten, der kein Typ-1-Knoten ist
- Geschlossene Knoten:
  - jeder Knoten, der für eine beliebige Variable p sowohl p als auch $\neg p$ enthält
  - jeder Knoten vom Typ 1, dessen Nachfolger alle geschlossene Knoten sind
  - jeder Knoten vom Typ 2, der einen geschlossenen Nachfolger besitzt

Ein K-Baum ist genau dann geschlossen, wenn seine Wurzel geschlossen ist.

3.3.2 Umwandlung zum Modell

Lemma 2. Sei S eine Menge von Klauseln. Wenn der K-Baum von S nicht geschlossen ist, dann hat S ein Modell.

Beweis. Sei $u = (A, T, r)$ der nicht geschlossene K-Baum von S. u hat einen Teilbaum $u' = (A', T', r')$ mit $A' \subseteq A$, $T' \subseteq T$, $r' = r$, für den gilt:

- jedes Blatt von u ist ein Blatt von u',
- jeder Knoten von u' ist nicht geschlossen,
- jeder Typ-1-Knoten hat genau einen Nachfolger,
- wenn w ein Typ-2-Knoten ist, dann stimmen die Nachfolger von w in u' mit den Nachfolgern von w in u überein.

Man nehme alle Paare (w, w') aus u', für die wTw' gilt und w vom Typ 1 ist und bilde die transitive, reflexive und symmetrische Hülle. Die so entstandene Äquivalenzrelation wird als ρ bezeichnet. Die Äquivalenzklasse von w wird als $|w|$ bezeichnet. Das K-Model $\langle W, R, m\rangle$ wird folgendermaßen konstruiert:

- W ist die Menge der Äquivalenzklassen von u' für ρ.
- für $|w|$ und $|w'|$ in W gilt: $|w|R|w'|$ genau dann, wenn $|w| \neq |w'|$ und es gibt ein $w_1 \in |w|$ und $w_1' \in |w'|$, sodass $w_1 T w_1'$.
- $|w| \in m(p)$ genau dann, wenn $p \in w_1$ für ein $w_1 \in |w|$.

Fakt 1. Für jeden Knoten w aus u und für alle Klauseln A von w gilt: $M, |w| \vDash A$.

Beweis. Durch Induktion über die Länge von A. Wenn A ein Literal ist, geht aus der Definition von m hervor, dass eine Variable, die einem Knoten vom Typ 1 angehört, auch dessen Nachfolgern angehört, da alle Knoten von u' nicht geschlossen sind.
Wenn $A = A_1 \vee A_2$, dann ist w vom Typ 1 und A_1 oder A_2 ist in einem Nachfolger w' von w mit $w \ \rho \ w'$. Sei dies A_1. Es gilt nach Induktionsvoraussetzung $M, |w| \vDash A_1$. Somit folgt aus $|w| = |w'|$ die Aussage.
Wenn $A = \Diamond A_1$, dann gibt es einen Knoten w' vom Typ 2 in $|w|$, sodass $A \in w'$ und einen Nachfolger w'' von w' mit $A_1 \in w''$. Nach (IV) $M, |w''| \vDash A_1$ und deshalb auch $M, |w'| \vDash \Diamond A_1$.
Wenn schließlich $A = \Box A_1$, dann gibt es einen Knoten w' vom Typ 2 in $|w|$ mit $\Box A_1 \in w'$. Für jeden Nachfolger w_i' von w' gilt nach Definition $A_1 \in w_i'$. Da die Welten $|w_i|$ allen erreichbaren Welten von $|w'|$ entsprechen, folgt nach (IV) $M, |w'| \vDash \Box A_1$. □

Beweis von Lemma 2 (Abschluss). Daraus folgt, dass, wenn r die Wurzel eines K-Baums ist, dann gilt $M, |r| \vDash S$ und M ist ein Modell für S. □

Beispiel 4. Sei $S = \{p \vee t, \Box(q \vee \Diamond((r \vee s), t), \Diamond\neg q\}$. Abbildung 1 zeigt einen K-Baum von S. Ein Modell, welches aus diesem Baum konstruiert werden kann, ist das Tripel $\langle W, R, m\rangle$, sodass $W = \{w_0, w_1, w_2\}$ mit

$$w_0 = \{\{p \vee t, \Box(q \vee \Diamond((r \vee s), t), \Diamond\neg q)\}, \{p, \Box(q \vee \Diamond(r \vee s), t), \Diamond\neg q\}\},$$
$$w_1 = \{\{q \vee \Diamond((r \vee s), t), \neg q\}, \{\Diamond((r \vee s), t), \neg q\}\},$$
$$w_2 = \{\{r \vee s, t\}, \{r, t\}\}$$

und:

$$
\begin{aligned}
&R = \{(w_0, w_1), (w_1, w_2)\},\\
&m(p) = \{w_0\},\\
&m(q) = \emptyset,\\
&m(r) = \{w_2\},\\
&m(t) = \{w_2\}.
\end{aligned}
$$

Damit erhält man ein Modell für S.

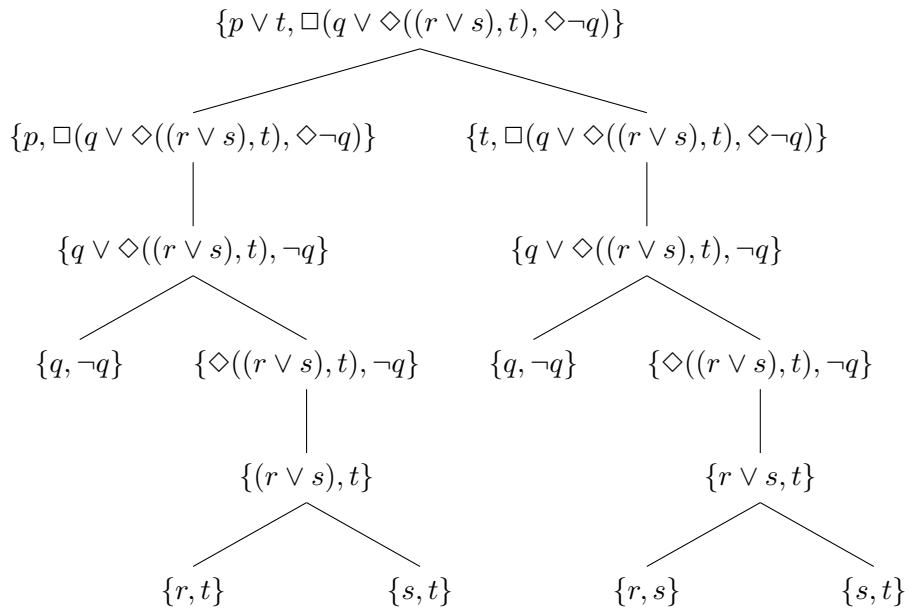

Abbildung 1: K-Baum von S

Beweis von Theorem 4 (Fortsetzung). Es wird gezeigt, dass für einen K-Baum mit geschlossener Wurzel eine RK-Widerlegung von S existiert.

Lemma 3.

(*i*) Wenn $A_1, ..., A_n \vdash_{RK} B$ dann $\Box A_1, ..., \Box A_n \vdash_{RK} \Box B$.

(*ii*) Wenn $A_1, ..., A_n, Q_1, ..., Q_r \vDash_{RK} B$ $(r \geq 1)$ und der Beweis mindestens eins der Q_j benutzt, dann $\Box A_1, ..., A_n, \Diamond(Q_1, ..., Q_r) \vdash \Diamond(B, Q_1, ..., Q_r, E)$ für eine Menge E von Klauseln wenn $B \neq \bot$, sonst $\Box A_1, ..., \Box A_n, \Diamond(Q_1, ..., Q_r) \vdash \bot$.

Beweis. (*i*): Angenommen es gibt einen Beweis für B aus $A_1, ..., A_n$. Der Beweis wird auf folgende Weise modifiziert:

- jedes Axiom A_i wird durch $\Box A_i$ ersetzt
- jedes Vorkommen einer $R2$-Regel

$$\frac{C1 \qquad C2}{D} \text{ mit } \Sigma(C_1, C_2) \Rightarrow D$$

wird durch

$$\frac{\Box C1 \qquad \Box C2}{\Box D}$$

ersetzt, wegen $\Sigma(\Box C_1, \Box C_2) \Rightarrow \Box D$ unter Benutzung der $\Box\Box$-Regel.

- jedes Vorkommen einer $R1$-Regel

$$\frac{C}{D}$$

wird durch

$$\frac{\Box C}{\Box D}$$

ersetzt, unter Benutzung der Γ-$\Box$-Regel.

Im Folgenden werden diese Abkürzungen verwendet:

$\bar{A}$ für $A_1, ..., A_n$,	$\Box\bar{A}$ für $\Box A_1, ..., A_n$,
$\bar{Q}$ für $Q_1, ..., Q_r$,	$\Diamond\bar{Q}$ für $\Diamond Q_1, ..., Q_r$.

Sei S eine Klauselmenge, C und D Klauseln und $C \in S$. D hängt von C ab, wenn C für den Beweis von D aus S benutzt wird.
(ii): Angenommen es sei $B \neq \bot$. Der Beweis erfolgt durch Induktion über die Ableitung von B.

- Ist B ein Axiom, dann gilt $B = Q_j$ für ein j. Daraus folgt:

$$\Diamond(B, Q_1, ..., Q_r) = \Diamond(Q_1, ..., Q_r).$$

- Ansonsten wird die zuletzt angewandte Schlussregel betrachtet.
  - *Fall 1:* Es ist eine $R2$-Regel. Der Beweis ergibt sich in folgender Form:

$$\frac{\begin{array}{ccccccc} \bar{A} & \ldots & \bar{Q} & \bar{A} & \ldots & \bar{Q} \\ & \vdots & & & \vdots & \\ & C_1 & & & C_2 & \end{array}}{B}$$

$$\text{mit } \Sigma(C_1, C_2) \Rightarrow B. \tag{a}$$

Mindestens eins der C_i hängt von einem Q_j ab. Angenommen es sei C_1. Dann ergeben sich zwei Möglichkeiten:

(1) $\bar{A} \vdash_{RK} C_2$. Aus (i) folgt $\Box\bar{A} \vdash_{RK} \Box C_2$.
Außerdem, nach (IV), $\Box\bar{A}, \Diamond\bar{Q} \vdash \Diamond(C_1, \bar{Q}, E)$.
Daraus ergibt sich der Beweis:

$$\text{R2}\,\frac{\begin{array}{cccc}\Box\bar{A} & \ldots & \Diamond\bar{Q} & \Box\bar{A} \\ & \vdots & & \vdots \\ & \Diamond(C_1, \bar{Q}, E) & & \Box C_2\end{array}}{}\ \text{(nach } (a) \text{ und } \Box\Diamond\text{-Regel).}$$

Dann ist $\Diamond(C_1, B, \bar{Q}, E) = \Diamond(B, \bar{Q}, F)$ für $F = (C_1, E)$.
(2) C_2 hängt von einem Q_j aus $\bar{Q}$ ab. Nach (IV) gilt:

$$\Box\bar{A}, \Diamond\bar{Q} \vdash \Diamond(C_1, \bar{Q}, E). \tag{b}$$

Aus $\bar{A}, \bar{Q} \vdash C_2$ erhält man $\bar{A}, \bar{Q}, C_1, E \vdash C_2$ und mit (IV) ergibt sich:

$$\Box\bar{A}, \Diamond(C_1, \bar{Q}, E) \vdash \Diamond(C_2, C_1, \bar{Q}, F) \text{ für ein } F. \tag{c}$$

Kombiniert man (b), (c) und die Γ-◇-Regel, so erhält man den Beweis:

$$\text{R1}\,\frac{\begin{array}{cccc}\Box\bar{A} & \ldots & \Diamond\bar{Q} & \\ & \vdots & & \\ \Diamond(C_1, \bar{Q}, E) & & \ldots & \Box\bar{A} \\ & & & \vdots \\ & & & \Diamond(C_1,C_2, \bar{Q}, F)\end{array}}{\Diamond(B, C_1,C_2, \bar{Q}, F)}$$

in der Form $\Box(B, \bar{Q}, G)$.

– Fall 2: Die zuletzt verwendete Regel ist eine $R1$-Regel. Der Beweis erfolgt auf ähnliche Weise.

Sei $B = \bot$. Der Beweis ist fast identisch und benutzt die Simplifizierung $\Diamond(\bot, E) \approx \bot$. □

Korollar 1. (1) Wenn $A_1, ..., A_n, Q_1, ..., Q_r \vdash_{RK} \bot$ $(r \geq 1)$ dann

$$\Box A_1, ..., \Box A_n, \Box(Q_1, ..., Q_r) \vdash_{RK} \bot .$$

(2) Sei S eine Menge von Klauseln. Wenn eine ihrer K-Projektionen S_i widerlegbar ist, ist S ebenso widerlegbar.

Beweis. (1): *Fall 1:* $\bar{A} \vdash \bot$. Dann ist $\Box\bar{A} \vdash \Box\ \bot$. Aus dem Axiom (A_2) erhält man

$$\Sigma(\Box\ \bot, \Diamond Q) \to \Diamond\ \bot$$

und nach Anwendung von Vereinfachungsregeln

$$\Sigma(\Box\ \bot, \Diamond\bar{Q}) \Rightarrow \bot\ .$$

Daher: $\Box\bar{A}, \Box\bar{Q} \vdash \bot$.
Fall 2: Der Beweis von $\bot$ hängt von $\bar{Q}$ ab. Das Lemma 3 wird angewandt.

(2) ergibt sich aus (1).

□

Lemma 4. Jeder geschlossene Knoten in einem K-Baum ist widerlegbar.

Beweis. durch Induktion über die Tiefe des geschlossenen Knotens w.

- Wenn $d(w) = 0$, dann ist w ein Blatt, und somit enthält w für eine Variable p sowohl p als auch $\neg p$ und ist damit widerlegbar durch Axiom (A_1).
- Angenommen die Aussage ist wahr für alle W' mit $d(w') < n$ und $d(w) = n$. Wenn w vom Typ 2 ist, dann sind die Nachfolger von w K-Projektionen und es kann das Korollar 1 und (IV) angewandt werden.
- Wenn w vom Typ 1 ist, dann sind die Nachfolger von w:

  $$w_1 = w - \{C\} \cup \{C_1\} \text{ und } w_2 = w - \{C\} \cup \{C_2\} \text{ für ein } C = C_1 \vee C_2.$$

  Nach (IV) gibt es Widerlegungen r_1 von w_1 und r_2 von w_2. Die Widerlegung r_1 wird modifiziert zu einer Deduktion von C_2, und mit r_2 wird die Widerlegung für w fortgeführt.

□

Beweis von Theorem 4 (Abschluss). Theorem 4 wird mithilfe von Lemma 2 und Lemma 4 wie folgt bewiesen: Angenommen S ist unerfüllbar. Aus Lemma 2 folgt, dass der dazugehörige K-Baum geschlossen sein muss. Nach Lemma 4 ist dessen Wurzel S, die ebenfalls geschlossen ist, RK-widerlegbar. □

Beispiel 5. Gegeben ist folgende Menge von Klauseln:

(1) $\Box\Diamond(p \vee \Diamond\neg q)$,

(2) $\Diamond\Box\neg p$,

(3) $\Box\Box\Box q$.

Eine Widerlegung ist:

(4) $\Diamond(\Box\neg p, \Diamond(p \vee \Diamond\neg q, \Diamond\neg q))$, aus (1) und (2),

(5) $\bot$ aus (3) und (4).

3.4 Algorithmus

In diesem Abschnitt wird der „Algorithm A“ von Enjalbert und Farinas Del Cerro vorgestellt [EFDC89].

3.4.1 Beschreibung

Aus den bisherigen Kapiteln geht hervor, dass, wenn eine Menge von Klauseln unerfüllbar ist, eine Widerlegung existiert, die die Resolutionsregeln benutzt. Es bleibt zu zeigen, wie man diese Widerlegung effektiv erhält. Dazu wird ein Entscheidungsalgorithmus konstruiert, der:

- für eine Menge unerfüllbare Klauseln die leere Klausel produziert,
- die Erfüllbarkeit einer Menge zeigt, wenn diese erfüllbar ist.

Algorithmus A

Eingabe: Klauselmenge S

1: $S_0 := \emptyset$

2: $S_1 := S$

3: $i := 1$

4: **if** $S_i \subseteq S_0 \cup ... \cup S_{i-1}$ **then return** S ist erfüllbar.

5: **if** S_i enthält die leere Klausel $\perp$ **then return** S ist unerfüllbar.

6: Sei S_{i+1} die Menge aller unären Resolventen der Klauseln von S_i und die aller binären Resolventen zwischen jeder Klausel aus S_i und jeder Klausel aus $S_0 \cup ... \cup S_i$. Diese Resolventen werden mit den Vereinfachungsregeln $(S_1) - (S_4)$ soweit wie möglich reduziert. Außerdem werden die Γ-◇-Regel 1 und 2 mit folgenden Einschränkungen angewandt:

$$\frac{\Sigma(A,B) \to C}{\Gamma(\Diamond(A,B,F)) \to \Diamond(A,B,C,F)} \quad \text{wenn } C \notin (A,B,F),$$

$$\frac{\Gamma(A) \to B}{\Gamma(\Diamond(A,F)) \to \Diamond(B,A,F)} \quad \text{wenn } B \notin (A,F).$$

7: $i := i + 1$

8: goto 4.[1]

---

[1] Vermutlich wurde in [EFDC89] an dieser Stelle ein Fehler gemacht. Es wird dort ein goto zu 6 statt zu 4 ausgeführt. Da 4 und 5 die einzigen Schritte sind, in denen der Algorithmus eine Ausgabe macht, würde der Algorithmus diese Stellen nach dem ersten Durchgang nie wieder erreichen und in eine Endlosschleife geraten.

4 Implementierung

In diesem Kapitel wird der vorherige Entscheidungsalgorithmus in einem Programm implementiert. Die interne Funktionsweise wird beschrieben und erklärt. Außerdem wird in der Bedienungsanleitung veranschaulicht, wie das Programm zu benutzen ist. Zum Schluss zeigen Beispiele die Ausführung des Programms.

4.1 Umsetzung

Die einzelnen Programmschritte werden kurz zusammengefasst:
Zuerst werden die eingegebenen Klauseln ausgelesen und auf ihre syntaktische Richtigkeit überprüft. Danach werden die Zeichen der Klauseln interpretiert und in Token umgewandelt. Aus den Token werden Klauseln gebaut, die in Syntaxbäumen abgespeichert werden. Darauf folgt die Normalisierung der Klauseln. Diese werden dann in einer KNF zusammengefasst. Die KNF wiederum wird in ihre DNF-Konjunkte aufgeteilt, welche die finale Klauselmenge darstellen.
Zum Schluss wird der eigentliche Algorithmus angewandt. Dazu werden in einem dynamischen Programmierverfahren Σ-Matrizen für die binären Resolventen und Γ-Matrizen für die unären Resolventen konstruiert, um die Menge der Resolventen in jedem Schritt zu ermitteln.

4.1.1 Syntaxanalyse

Zunächst wird die Eingabe auf mögliche Programmparameter überprüft. Dazu werden Argumente die mit „-“ beginnen der Klauselmenge entnommen und überprüft, welche der Optionen ausgeführt werden soll. Die restlichen Argumente werden untersucht, ob sie syntaktisch korrekte Klauseln enthalten. Hierfür wird die Klasse Parser verwendet. Sie implementiert einen Top-Down-Parser für modallogische Formeln, der die Grammatik $G = (N, T, P, S)$ verwendet, mit:

- Menge der Nichtterminalsymbole $N = \{S, E, P, B, U\}$,
- Menge der Terminalsymbole $T = \{var, \vee, \wedge, \Box, \Diamond, \neg, (,), \dashv\}$,
- Startvariable S,
- Menge der Produktionsregeln P, die die folgenden Produktionen enthält:

$$
\begin{array}{lllll}
S \to PE \dashv & E \to BPE & P \to (E) & B \to \wedge & U \to \Box \\
 & E \to \epsilon & P \to UP & B \to \vee & U \to \Diamond \\
 & & P \to var & & U \to \neg
\end{array}
$$

  wobei var der reguläre Ausdruck $var = [L]\ ([L] \mid [D])^*$ ist mit:

  - $[L] = A \mid B \mid ... \mid Z \mid a \mid b \mid ... \mid z$
  - $[D] = 0 \mid 1 \mid ... \mid 9$.

Da bei der Eingabe der Klausel keine Endmarke erwartet wird, wird die Endmarke $\dashv$ an das Ende jeder Klausel angehängt, bevor sie überprüft wird. Um aus der Grammatik einen Top-Down-Parser zu erstellen, wurden zunächst die folgenden Syntaxgraphen erstellt:

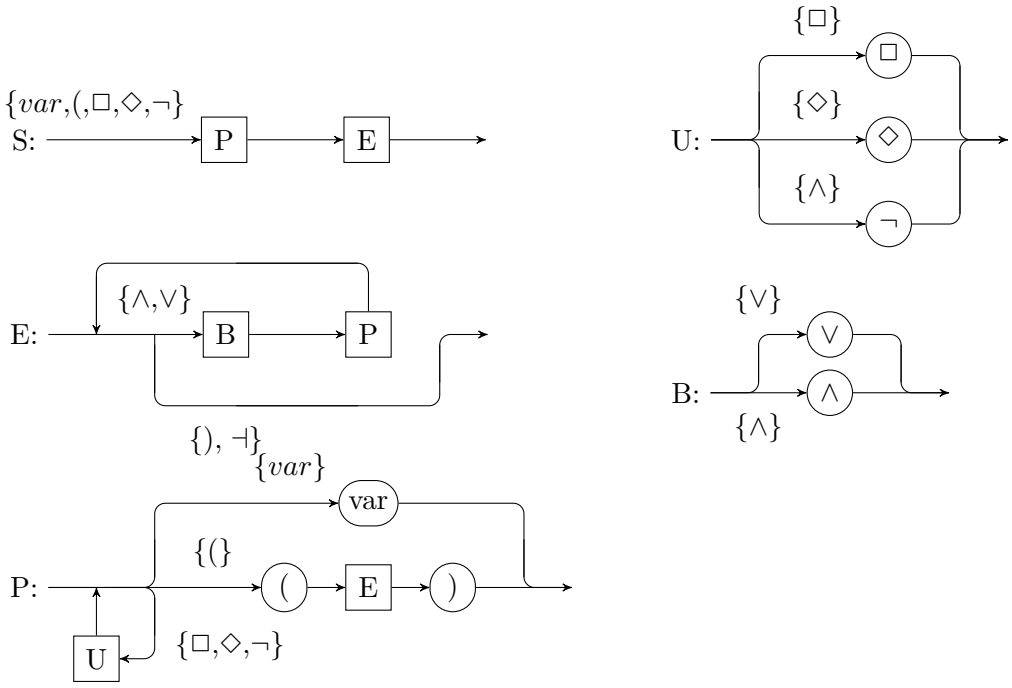

Abbildung 2: Syntaxgraphen von G

Aus den Syntaxgraphen wird schließlich der Algorithmus gebildet. Jeder Teilgraph wird in ein namensgleiches Unterprogramm umgewandelt. Hierbei wurden U und B bereits in P und E integriert.

```
public boolean check() {
  it = tokenList.iterator();
  getToken();
  S();
  if(token.equals("⊣"))
    return true;
  return false;
}

private void S() {
  P();
  E();
  return;
}

private void E() {
  if (token.equals("⊣") || token.equals(")")) {
    return;
  } else if (token.equals("∧") || token.equals("∨")) {
    getToken();
    P();
    E();
  } else {
    error();
    return;
  }
}

private void P() {
  if (token instanceof Variable) {
    getToken();
    return;
  }
  if (token.equals("(")) {
    getToken();
    S();
    if (token.equals(")")) {
      getToken();
      return;
    } else {
      error();
      return;
    }
  } else if (token.equals("□") || token.equals("◇")
        || token.equals("¬")) {
    getToken();
    P();
  } else {
    error();
    return;
  }
}
```

4.1.2 Interpretierung

Nach Abschluss der Syntaxanalyse werden die syntaktisch korrekten Klauseln interpretiert. Die Klauseln werden nach ihren Zeichen aufgetrennt, die in Token abgespeichert werden. Aufeinander folgende Buchstaben und Zahlen sollen als eine gemeinsame Variable gesehen werden und werden daher in ein einzelnes Token abgespeichert.
Jedes Token gehört einer von drei Unterklassen an:

- Operationen, welche nochmal unterteilt werden in:
  - unäre Operationen ($\Box, \Diamond, \neg$),
  - und binäre Operationen ($\wedge, \vee$),
- Variablen, die aus einer Kombination von ein oder mehreren Buchstaben und Zahlen bestehen können,
- und Klammern.

Außerdem gilt folgende Präzedenz für die Token:

$$\text{Klammern} > \text{unäre Operationen} > \wedge > \vee.$$

4.1.3 Syntaxbaum

Die Token werden zunächst in einer Liste abgespeichert. Für die nächsten Schritte werden die Klauseln in Form von Syntaxbäumen benötigt, da dort vom rekursiven Aufruf des Syntaxbaums Gebrauch gemacht wird. Aus den Token werden Knoten gebildet. Dabei haben binäre Operationen ($\wedge, \vee$) immer zwei Nachfolger, unäre Operationen ($\Box, \Diamond, \neg$) einen Nachfolger und Variablen keine Nachfolger, wodurch die Blätter des Baums immer Variablen sind. Klammern werden nicht zu Knoten umgewandelt. Stattdessen wird in jedem Knoten gespeichert, ob dieser eingeklammert ist oder nicht. Dies geht aus der Präzedenz hervor: es werden Klammern für einen Knoten gesetzt, wenn dessen Vorgänger eine höhere Präzedenz hat. Damit werden gleichzeitig überflüssige Klammern entfernt, die in den Token noch vorliegen konnten.
Für diesen Vorgang wird der Shunting-yard-Algorithmus verwendet. Dieser wandelt Formeln aus der Infixnotation zu Syntaxbäumen um. Angepasst an modallogische Formeln, sieht der Algorithmus wie folgt aus:

Algorithmus Shunting-yard[The99]

```
   Eingabe: Tokenliste Tlist
 1: Stack operandStack, operatorStack
 2: for Token T in Tlist do
 3:     if T ist eine Variable then PUSH(operandStack, T)
 4:     if T ist eine Operation then
 5:         while operatorStack ist nicht leer UND TOP(operatorStack) hat eine höhere
                Präzedenz als T do
 6:             PUT(operandStack, POP(operatorStack))
 7:         end while
 8:         PUSH(operatorStack, T)
 9:     if T ist eine Klammer then
10:         if T ist "(" then PUSH(operatorStack, T)
11:         else
12:             while TOP(operatorStack) ist nicht "(" do
13:                 PUT(operandStack, POP(operatorStack))
14:             end while
15:             POP(operatorStack)
16: while operatorStack ist nicht leer do
17:     PUT(operandStack, POP(operatorStack))
18: end while
19: return POP(operandStack)

 1: procedure PUT(Operation OP1, Stack S)
 2:     if OP1 ist eine unäre Operation then
 3:         OP2 = new Operation(OP1, POP(operandStack))
 4:         PUSH(operandStack, OP2)
 5:     else
 6:         OP2 = new Operation(OP1, POP(operandStack), POP(operandStack))
 7:         PUSH(operandStack, OP2)
```

4.1.4 Normalisierung

Für die Anwendung des Algorithmus werden die Klauseln in disjunktiver Normalform benötigt. Dazu werden sie umgewandelt, so wie es in dem Verfahren in Kapitel 2.3 beschrieben wird. Zuerst werden die Negationen direkt vor die Variablen bewegt. Dies geschieht indem der Syntaxbaum mit rekursiven Aufrufen durchlaufen wird. Bei jedem Aufruf wird ein Wahrheitswert mitgegeben, der bestimmt, ob die aktuelle Operation oder Variable zu negieren ist. Wenn dieser wahr ist, dann werden die Operationen $\Box$, $\Diamond$ und die Operationen $\wedge$, $\vee$ jeweils vertauscht. Bei Variablen wird ein neuer Knoten als Vorgänger erzeugt, der eine Negation enthält. Tritt beim Durchlaufen ein Knoten mit einer Negation auf, dann wird der Knoten entfernt und dessen Nachfolger an dessen Vorgänger angehängt.

Die Prozedur wird dann beim Nachfolger mit umgekehrten Wahrheitswert fortgesetzt.
Als nächsten wird weiterhin wie in Kapitel 2.3 vorgegangen: bei modaler Tiefe von 0 erfolgt eine Normalisierung wie in der klassischen Aussagenlogik. Das Distributivgesetz wird angewandt, so dass eine DNF im klassischen Sinne entsteht. Tritt die Operation $\Box$ auf, werden die Nachfolger normalisiert und der Operator über alle Konjunkte verteilt. Bei $\Diamond$ werden die Nachfolger ebenfalls zuerst normalisiert, danach wird der Operator vorangestellt.
Um überflüssige Elemente in Klauseln zu entfernen, werden die Klauseln bereits hier simplifiziert. Dazu wird der Baum wieder mit rekursiven Aufrufen durchlaufen und es wird verfahren wie in Kapitel 3.1.2: bei jedem Aufruf wird überprüft, ob eine der Vereinfachungsregeln anwendbar ist und, falls dies der Fall ist, auch angewandt. Außerdem wird die Vereinfachung $A \wedge A \approx A$ den Regeln hinzugefügt, damit die Klauseln noch weiter vereinfacht werden.

4.1.5 KNF und DNF

Nach der bisherigen Normalisierung besteht noch die Möglichkeit, dass eine Klausel nicht in DNF, sondern in KNF ist. Das liegt daran, dass eine Klausel in der Eingabe aus einer Konjunktion mehreren Klauseln bestehen kann, und diese bisher noch nicht aufgetrennt wurden.
Beispielsweise würde die Klausel $\Box(A \wedge B)$ zu der Klausel $\Box A \wedge \Box B$ in KNF normalisiert werden. Diese müsste auf zwei Klauseln $\Box A$ und $\Box B$ in DNF aufgeteilt werden.
Dazu werden alle Klauseln der Klauselmenge zu einer Formel in KNF zusammengefasst, indem die Klauseln mit der $\wedge$-Operation verbunden werden. Diese Formel wird vereinfacht nach demselben Verfahren wie zuvor. Somit entsteht eine Formel in KNF, und aus Kapitel 2.3 geht hervor, dass jedes Konjunkt der KNF eine DNF ist und somit auch eine Klausel. Also ergibt sich die finale Klauselmenge aus der Menge der Konjunkte.

4.1.6 Berechnung der Resolventen

Das wohl wichtigste Bestandteil des Algorithmus ist das Berechnen der binären und unären Resolventen von Klauseln. Die Resolventen werden im System RK mithilfe der Berechnungsregeln nicht-deterministisch berechnet, indem, ausgehend von einem Axiom, eine Folge von Berechnungsregeln angewandt wird, sodass die Relation zum Schluss auf der linken Seite die Ausgangsklauseln enthält und auf der rechten deren Resolvente. Bei einer Anwendung einer Σ-$\vee$-Regel oder einer Σ-$\Box\Diamond$-Regel wird eine Klausel der Relation hinzugefügt. Da es keine Einschränkungen gibt, welche Klausel das sein kann, gibt es unendlich Möglichkeiten, wie diese Regelanwendung aussehen kann. Für das Programm wäre es nicht vorteilhaft, alle Möglichkeiten zu durchlaufen, da es so potenziell nie zum Ende kommt. Deshalb wird im Folgenden ein dynamisches Programmierverfahren beschrieben, mit dem die Berechnung der Resolventen deterministisch abläuft.

Für jede Berechnungsregel gilt, dass die Resolvente nach der Anwendung nie kleiner wird und die vorherige Resolvente immer als Teilformel enthält. Dasselbe gilt für die Klauseln auf der linken Seite der Relation. Die Größe wird in diesem Fall anhand der Tiefe des Syntaxbaums gemessen.
Damit kann die Menge der möglichen Klauseln, die bei Anwendung einer Berechnungsregel hinzukommen können, auf eine endliche Menge eingeschränkt werden. Es handelt sich dabei um die Klauseln aus der Potenzmenge der Ausgangsklausel.

Für die Berechnung der binären Resolvente zweier Klauseln wird das folgende Verfahren angewandt: es wird eine Matrix erstellt, die ein Klauselpaar auf eine Menge von Klauseln abbildet. Für die beiden Ausgangsklauseln, deren Resolvente gesucht wird, wird jeweils die Potenzmenge gebildet. Die Klauseln aus der Potenzmenge bilden die Einträge jeweils einer Dimension der Matrix. Die Matrix wird bottom-up aufgebaut: Für jedes Paar von Klauseln wird die Menge ihrer Resolventen bestimmt. Es wird dabei iterativ vorgegangen: Zuerst werden die kleinsten Klauseln benutzt, welche nur eine Variable enthalten. Danach wird in jeder folgenden Iteration die Größe um eins erhöht, bis zum Schluss die Ausgangsklauseln verrechnet werden, welche die maximale Größe besitzen.

Beispiel 6. Sei für $\Box p$ und $\Diamond\neg q$ die Resolvente gesucht.
Die Potenzmenge von $\Box p$ ist $\{\Box p, p, \bot\}$, die von $\Diamond\neg q$ ist $\{\Diamond\neg q, \neg q, q, \bot\}$. In der ersten Iteration werden Resolventen aus allen Paaren zwischen den Mengen $\{p, \bot\}$ und $\{q, \bot\}$ gebildet. In der zweiten Iteration werden alle Paare gebildet, von denen mindestens eine der beiden Klauseln Größe zwei hat. Dies sind: $\{(\Box p, \bot), (\Box p, q), (\Box p, \neg q), (p, \neg q), (\bot, \neg q)\}$. In der dritten und letzten Iteration werden die Klauselpaare
$\{(\bot, \Diamond\neg q), (p, \Diamond\neg q), (\Box p, \Diamond\neg q)\}$ gebildet, wodurch schließlich die gesuchte Resolvente von $(\Diamond p, \Diamond\neg q)$ erzeugt wird.

Um die Resolvente eines Paares zu bestimmen, wird zuerst überprüft, ob eines der Axiome aus System RK anwendbar ist. Das trifft zu, wenn die beiden Klauseln die linke Seite einer Relation eines Axioms bilden können. Dann wird die rechte Seite der Relation als Resolvente der Menge hinzugefügt.

Für den Fall, dass die beiden Klauseln gleich sind, wird die Klausel selbst der Menge hinzugefügt.
Danach wird überprüft, ob das Klauselpaar aus der Anwendung einer Σ-Regel aus bereits bearbeiteten Klauseln entstehen kann.
Für die Σ-□□-Regel müssen die Wurzelknoten der Syntaxbäume der beiden Klauseln die Operation □ sein. Dann werden Klauseln aus den Teilbäumen gebildet, die entstehen, wenn man die Wurzel aus dem Syntaxbaum entfernt. Das Klauselpaar wird bereits einen Eintrag für die Resolvente besitzen, da durch das Entfernen eines Operators die Länge der Klausel strikt geringer ist. Auf das neue Klauselpaar wird in Kombination mit einer ihrer Resolventen die Σ-□□-Regel angewandt, sodass die linke Seite der Relation die beiden Ausgangsklauseln enthält. Die rechte Seite der Relation enthält eine Resolvente und wird der Menge der Resolventen des Klauselpaares hinzugefügt. Dieser Vorgang wird mit jeder Resolvente des neuen Klauselpaares wiederholt.

Beispiel 7. Sei $(\Box A, \Box B)$ ein Klauselpaar, dessen Resolvente gesucht wird. Die Σ-□□-Regel ist anwendbar, da bei beiden Klauseln der □-Operator die Wurzel des Syntaxbaums ist. Es wird für jede Klausel C, welche in der Menge der Resolventen von (A, B) liegt, die binäre Relation $\Sigma(A, B) \to C$ gebildet und die Σ-□□-Regel angewandt. Auf der linken Seite entsteht dementsprechend immer $(\Box A, \Box B)$, und die rechte Seite enthält eine der gesuchten Resolventen.

Dasselbe Prinzip wird mit der Σ-□◇-Regel fortgeführt, wobei hier eine der beiden Wurzeln ◇ und die andere □ sein muss. Außerdem wird die Klausel, die aus der Klausel mit dem ◇-Operator entsteht, in ihre Konjunkte zerlegt. Jedes Konjunkt bildet ein Paar mit der Klausel, die aus der Klausel mit dem □-Operator entstanden ist. Für jedes solche Paar wird die Σ-□◇ so angewandt, dass die aktuelle Klausel auf der linken Seite entsteht. Die Resolventen werden wie zuvor entsprechend hinzugefügt.

Beispiel 8. Sei für das Klauselpaar $(\Box A, \Diamond(B \wedge E))$ die Resolvente gesucht. Die Σ-□◇-Regel kann angewandt werden. Dazu werden aus der Klausel A und jedem Konjunkt von $B \wedge E$ ein Klauselpaar gebildet. Das sind: $\{(A, B), (A, E)\}$. Für jedes Klauselpaar und für jede Resolvente C dieses Klauselpaares wird eine Relation gebildet, auf die die Σ-□◇-Regel so angewandt, dass auf der linken Seite der Relation $(\Box A, \Diamond(B \wedge E))$ entsteht. Die rechten Seiten enthalten die gesuchten Resolventen.

Für die Σ-∨-Regel wird überprüft, ob mindestens eine der beiden Klauseln die ∨-Operation als Wurzel besitzt. Ist das der Fall, werden beide Klausel in ihre Disjunkte aufgeteilt. Für jedes Paar der Disjunkte wird in Kombination mit dessen Resolventen jeweils die Σ-∨-Regel angewandt, wobei die Klausel, die bei dieser Regel hinzugefügt wird, die Disjunktion aller anderen Klauseln aus der Menge der Disjunkte ist.

Beispiel 9. Für $(A \vee D, B)$ kann die Σ-∨-Regel angewandt werden. Dazu werden aus den beiden Klauseln die Mengen der Disjunkte gebildet: $\{A, D\}$ und $\{B\}$. Jede Kombination von Klauselpaaren wird mit jeder ihrer Resolventen zu einer Relation gemacht. Auf diese wird die Σ-∨-Regel angewandt, sodass auf der linken Seite $(A \vee D, B)$ entsteht. Auf den rechten Seiten erhält man die Resolventen der Ausgangsklausel.

Bevor eine Klausel zur Menge der Resolventen eines Klauselpaares hinzugefügt wird, wird sie soweit wie möglich simplifiziert.
Falls keine der Regeln und Axiome anwendbar sind, wird die Disjunktion der beiden Klauseln als Resolvente eingetragen.
Ist die Matrix fertig konstruiert, besitzt sie Eintrage für die Resolventen der anfänglichen zwei Klauseln. Dies ist die gesuchte Menge der binären Resolventen.

Beispiel 10 (Binäre Matrix)**.** Es sei die Menge der binären Resolventen der Klauseln $\Box\neg p$ und $\Diamond(p \vee q)$ gesucht. Die binäre Matrix dieser Klauseln ist:

	$\bot$	p	$\neg p$	$\Box\neg p$
$\bot$	$\{\bot\}$	$\{\bot\}$	$\{\bot\}$	$\{\bot\}$
p	$\{\bot\}$	$\{p\}$	$\{\bot\}$	$\{p, \Box\neg p\}$
q	$\{\bot\}$	$\{p, q\}$	$\{\neg p, q\}$	$\{q, \Box\neg p\}$
$p \vee q$	$\{\bot\}$	$\{p, p \vee q\}$	$\{p \vee q, q, p \vee \neg p\}$	$\{p \vee q, q \vee \Box\neg p, p \vee \Box\neg p\}$
$\Diamond(p \vee q)$	$\{\bot\}$	$\{p, \Diamond(p \vee q)\}$	$\{\neg p, \Diamond(p \vee q)\}$	$\{\Diamond((p \vee q) \wedge (p \vee \neg p)),$ $\Diamond((p \vee q) \wedge q), \Diamond(p \vee q)\}$

Damit ergibt sich für die binären Resolventen der beiden Klauseln:

$$\{\Diamond((p \vee q) \wedge (p \vee \neg p)), \Diamond((p \vee q) \wedge q), \Diamond(p \vee q)\}.$$

Für die unären Resolventen verläuft die Prozedur größtenteils analog, nur dass die Matrix mit der Potenzmenge von nur einer Klausel befüllt wird und die angewandten Regeln die Γ-Regeln sind. Außerdem gibt es noch die Γ-$\Diamond$-1-Regel, bei der eine unäre Relation aus einer binären Relation gebildet wird. Hierbei wird für jede Klausel, auf die die Γ-$\Diamond$-2-Regel anwendbar ist, zusätzlich eine binäre Matrix gebildet, um dessen Resolvente in der Relation zu benutzen. Außerdem werden die Einschränkungen des Algorithmus aus 3.4 beachtet, sodass eine Regelanwendung von einer Γ-$\Diamond$-Regel in den entsprechenden Fällen nicht stattfindet.

4.2 Bedienung des Programms

Nach Eingabe der Klauseln gibt das Programm zurück, welche Klauseln gültig sind und benutzt werden, und ob diese Klauselmenge erfüllbar ist. Für eine unerfüllbare Klauselmenge gibt das Programm „Unerfüllbar“ aus, für eine erfüllbare Klauselmenge „Erfüllbar“.

4.2.1 Programmaufruf

Der Aufruf des Programms erfolgt über die Kommandozeile und sieht wie folgt aus:

java -jar ResolutionModal.jar [Optionen] [Klauseln]

Unter dem Feld [Klauseln] werden die Klauseln aus der zu überprüfenden Klauselmenge eingetragen. Jede Klausel ist dabei mit Anführungszeichen zu umgeben, und zwischen je zwei Klausel ist eine Leerstelle. Gültige Klauseln bestehen aus Variablen, Klammern und Operatoren und müssen syntaktisch korrekt sein. Gültige Variablen können aus mehreren auf einander folgenden Buchstaben oder Zahlen bestehen, fangen aber immer mit mindestens einem Buchstaben an. Syntaktisch inkorrekte Klause werden aus der Klauselmenge entfernt. Formeln in KNF werden in mehrere Klauseln unterteilt. Unter dem Feld [*Optionen*] können weitere Ausgaben eingeschaltet werden. Folgende sind möglich:

- -info : Die Klauselmengen, die in jeder Iteration des Algorithmus erstellt werden, werden ausgegeben.
- -klauseln : Zu jeder eingegebenen Klausel wird die Liste enthaltener Variablen angezeigt und ob die Klausel gültig ist.
- -knf : Die Klauselmenge wird in konjunktive Normalform umgewandelt. Es werden die Zwischenschritte ausgegeben.
- -comp : Alle Σ-/ und Γ-Regeln werden auf die Relation aus den ersten drei Klauseln angewandt und ausgegeben. Falls weniger als drei Klauseln angegeben werden, erfolgt keine Ausgabe.

Die Unicode Zeichen können durch folgende Zeichen substituiert werden:

- $\Box$: #
- $\Diamond$: @
- $\neg$: !
- $\vee$: |
- $\wedge$: &

Es folgen beispielhafte Eingaben und die dazugehörige Ausgaben:

Beispiel 11 (Ohne zusätzliche Optionen)**.**

java -jar ResolutionModal.jar "$\Box p$" "$\Diamond \neg p$"
Klauseln: $[\Box p, \Diamond \neg p]$
Unerfüllbar

Beispiel 12 (Alle zusätzlichen Optionen)**.** In diesem Beispiel werden alle Optionen benutzt. Sie sind in unterschiedlichen Farben gekennzeichnet, die dazugehörige Ausgabe ist in derselben Farbe.

java -jar ResolutionModal.jar -klauseln -comp -knf -info "$\Box p$" "$\Diamond(\neg p \vee q)$" "$\Box \neg q$"

Klausel 1:
Infix Notation: $\Box p$
Parsercheck: true
Variablen: $[p]$
Anzahl: 1

Klausel 2:
Infix Notation: $\Diamond(\neg p \vee q)$
Parsercheck: true
Variablen: $[p, q]$
Anzahl: 2

Klausel 3:
Infix Notation: $\Box \neg q$
Parsercheck: true
Variablen: $[q]$
Anzahl: 1

Berechnungsregeln Test:
$\Sigma[\Box p, \Diamond(\neg p \vee q)] \to \Box\neg q$
$\vee$-Regel: $\Sigma[\Box p \vee D1, \Diamond(\neg p \vee q) \vee D2] \to \Box\neg q \vee D1 \vee D2$
$\Box\Diamond$-Regel: $\Sigma[\Box\Box p, \Diamond(\Diamond(\neg p \vee q) \wedge E)] \to \Diamond(\Box\neg q \wedge \Diamond(\neg p \vee q) \wedge E)$
$\Box\Box$-Regel: $\Sigma[\Box\Box p, \Box\Diamond(\neg p \vee q)] \to \Box\Box\neg q$
$\Diamond$-Regel 1: $\Gamma[\Diamond(\Box p \wedge \Diamond(\neg p \vee q) \wedge F)] \to \Diamond(\Box\neg q \wedge \Box p \wedge \Diamond(\neg p \vee q) \wedge F)$

$\Gamma[\Box p] \to \Diamond(\neg p \vee q)$
$\Diamond$-Regel 2: $\Gamma[\Diamond(\Box p \wedge F)] \to \Diamond(\Diamond(\neg p \vee q) \wedge \Box p \wedge F)$
$\vee$-Regel: $\Gamma[\Box p \vee \Box\neg q] \to \Diamond(\neg p \vee q) \vee \Box\neg q$
$\Box$-Regel: $\Gamma[\Box\Box p] \to \Box\Diamond(\neg p \vee q)$

KNF Test:
Konjunktion: $\Box p \wedge \Diamond(\neg p \vee q) \wedge \Box\neg q$
Negationen nach innen bewegt: $\Box p \wedge \Diamond(\neg p \vee q) \wedge \Box\neg q$
Normalisiert: $\Box p \wedge \Diamond(\neg p \vee q) \wedge \Box\neg q$
Simplifiziert: $\Box\neg q \wedge \Diamond(q \vee \neg p) \wedge \Box p$
KNF: $\{\{\Box\neg q\}, \{\Box p\}, \{\Diamond(q \vee \neg p)\}\}$

Klauseln: $[\Box\neg q, \Diamond(q \vee \neg p), \Box p]$
S1: $[\Box\neg q, \Diamond(q \vee \neg p), \Box p]$ Größe: 3
S2: $[\Box\neg q, \Diamond((q \vee \neg q) \wedge (q \vee \neg p)), \Diamond(q \vee \neg p), \Diamond((q \vee \neg p) \wedge (p \vee \neg p)), \Diamond((q \vee \neg p) \wedge \neg p),$
$\Diamond(q \wedge (q \vee \neg p)), \Box p]$ Größe: 7
S3: $[\Box\neg q, \Diamond(q \vee \neg p), \bot, \Diamond((p \vee q \vee \neg p) \wedge (p \vee \neg p) \wedge (q \vee \neg p)), \Diamond(p \wedge (q \vee \neg p) \wedge (p \vee \neg p)),$
$\Diamond((q \vee \neg p) \wedge (q \vee \neg p \vee \neg q) \wedge (q \vee \neg q)), \Diamond((\neg p \vee \neg q) \wedge (q \vee \neg p) \wedge (p \vee \neg p)),$
$\Diamond((p \vee q) \wedge (q \vee \neg q) \wedge (q \vee \neg p)), \Diamond((q \vee \neg q) \wedge (q \vee \neg p) \wedge (p \vee \neg q)), \Box p, \Diamond(p \wedge q \wedge (q \vee \neg p)),$
$\Diamond((q \vee \neg q) \wedge (q \vee \neg p)), \Diamond((q \vee \neg q) \wedge (q \vee \neg p) \wedge \neg p), \Diamond((q \vee \neg q) \wedge (q \vee \neg p) \wedge \neg q),$
$\Diamond(q \wedge (q \vee \neg q) \wedge (q \vee \neg p)), \Diamond((q \vee \neg p) \wedge (p \vee \neg p)), \Diamond(q \wedge (q \vee \neg p) \wedge \neg p),$
$\Diamond((q \vee \neg p) \wedge (p \vee \neg p) \wedge (p \vee \neg q)), \Diamond((q \vee \neg p) \wedge \neg p \wedge \neg q), \Diamond((q \vee \neg p) \wedge (p \vee \neg p) \wedge \neg p),$
$\Diamond(q \wedge (q \vee \neg p) \wedge (p \vee \neg p)), \Diamond((q \vee \neg q) \wedge (q \vee \neg p) \wedge (p \vee \neg p)), \Diamond((q \vee \neg p) \wedge \neg p),$
$\Diamond(q \wedge (q \vee \neg p))]$ Größe: 24

Unerfüllbar.

5 Zusammenfassung und Ausblick

Das entwickelte Programm kann die Resolution auf modallogische Formeln durchführen und ausgeben, ob eine Klauselmenge erfüllbar ist oder nicht.
Allerdings ist die Laufzeit bei langen und bei vielen Klauseln für die praktische Anwendung oft zu hoch, weshalb die Klauselmengen, die sich zur Verwendung anbieten, eingeschränkt sind.
Es gibt daher noch einige Möglichkeiten das Programm weiter zu optimieren. Der Algorithmus erzeugt viele überflüssige Klauseln, vor allem bei der Konstruktion der binären und unären Resolventen. Hier sollten irrelevante Klauseln möglichst früh erkannt und dementsprechend entfernt werden.
Außerdem kann die Reihenfolge der Abarbeitung verbessert werden. Klauseln, die eine höhere Wahrscheinlichkeit haben, eine leere Klausel zu erzeugen, sollten mit einer höheren Priorität bearbeitet werden.
Diese Punkte sollten zu einer Verbesserung der Laufzeit führen und damit gleichzeitig die Anzahl der Klauselmengen, die sich für die praktische Anwendung anbieten, erhöhen.
Auch hinsichtlich der Theorie kann noch viel getan werden. Diese Arbeit hat sich lediglich mit der Resolution im Modalsystem K auseinandergesetzt. Enjalbert und Farinas Del Cerro haben in [EFDC89] darüber hinaus die Resolution in den Systemen Q, T, S4, K4, Q4, S5 behandelt. Jedes davon könnte untersucht und darauf überprüft werden, ob der hier beschriebene Algorithmus dort ebenfalls funktioniert.

Literaturverzeichnis

[Bla37] Blake, Archie: *Canonical expressions in boolean algebra*, University of Chicago, Ph.D. thesis, 1937

[BRV01] Blackburn, Patrick ; Rijke, Maarten d. ; Venema, Yde: *Modal Logic*. Cambridge University Press, 2001 (Cambridge Tracts in Theoretical Computer Science). http://dx.doi.org/10.1017/CBO9781107050884. http://dx.doi.org/10.1017/CBO9781107050884

[Car47] Carnap, Rudolf: *Meaning and Necessity*. University of Chicago Press, 1947

[DP60] Davis, Martin ; Putnam, Hilary: A Computing Procedure for Quantification Theory. In: *J. ACM* 7 (1960), Nr. 3, 201–215. http://dx.doi.org/10.1145/321033.321034. – DOI 10.1145/321033.321034

[EFDC89] Enjalbert, Patrice ; Farinas Del Cerro, Luis: *Modal Resolution in Clausal Form*. 1989

[Kri59] Kripke, Saul A.: A Completeness Theorem in Modal Logic. In: *Journal of Symbolic Logic* 24 (1959), Nr. 1, S. 1–14

[Kri80] Kripke, Saul A.: *Naming and Necessity*. Harvard University Press, 1980

[Lei10] Leibniz, Gottfried W.: *Die Theodizee*. 1710

[Rob65] Robinson, John A.: A Machine-Oriented Logic Based on the Resolution Principle. In: *J. ACM* 12 (1965), Nr. 1, 23–41. http://dx.doi.org/10.1145/321250.321253. – DOI 10.1145/321250.321253

[The99] Theodore Norvell: *Parsing Expressions by Recursive Descent*. http://www.engr.mun.ca/~theo/Misc/exp_parsing.htm. Version: 1999