Konnektionskalküle für automatisches Beweisen in klassischen und nicht-klassischen Logiken*

Jens Otten

Institut für Informatik, University of Potsdam
August-Bebel-Str. 89, 14482 Potsdam-Babelsberg, Germany
E-Mail: jeotten@cs.uni-potsdam.de

Abstract: Die Automatisierung des formalen logischen Beweisens ist ein grundlegendes Forschungsgebiet innerhalb der Künstlichen Intelligenz. Der vorliegende Artikel stellt Konnektionskalküle vor, die zur automatischen Beweissuche in klassischer, intuitionistischer und modaler Logik verwendet werden können. Dabei wird der Konnektionskalkül für klassische Logik in einer eleganten und uniformen Art erweitert. Es werden sehr kompakte Implementierungen dieser Kalküle vorgestellt, die zu den leistungsfähigsten Beweissystemen in ihrer Klasse zählen und bereits mehrfach erfolgreich an internationalen Wettbewerben teilgenommen haben.

1 Einleitung

Logisches Schlussfolgern ist nicht nur eine wichtige Grundlage menschlichen Handelns im Alltag, sondern ein elementares Prinzip, das in allen wissenschaftlichen Disziplinen angewendet wird. So werden zum Beispiel in der Mathematik neue Sätze und Theoreme durch Anwendung formaler logischer Regeln aus Axiomen und bereits bekannten Sätzen hergeleitet. Der *Beweis* eines Satzes oder Theorems ist dann eine Folge von logischen Schlüssen. Das Forschungsgebiet, das sich mit der Automatisierung dieses logischen Schlussfolgerns beschäftigt, wird daher auch *Automatisches Beweisen* genannt. Ziel dabei ist es, den Beweis eines Theorems vollautomatisch durch den Computer finden zu lassen. Ein solches Computerprogramm wird auch *Beweiser* genannt.

Die Sprache der *Prädikatenlogik* wird dabei benutzt, um Aussagen zu repräsentieren und das logische Schließen über diese Aussagen zu formalisieren. In der Sprache der Prädikatenlogik werden zum Beispiel die Aussagen (a) *Plato ist ein Mensch* und (b) *für alle* X *gilt, wenn* X *ein Mensch ist, dann ist* X *sterblich*, wie folgt formalisiert:

$$\left.\begin{array}{ll}(a) & \mathit{Mensch(Plato)} \\ (b) & \forall X\ (\mathit{Mensch}(X) \Rightarrow \mathit{sterblich}(X))\end{array}\right\} \Rightarrow \quad (c) \quad \mathit{sterblich(Plato)}\,.$$

Aus diesen beiden Aussagen lässt sich logisch Schlussfolgern, dass (c) *Plato sterblich ist*. Die Aussage (c) ist also eine *logische Konsequenz* der Aussagen (a) und (b).

*Englischer Titel der Dissertation: "Connection Calculi for Automated Theorem Proving in Classical and Non-Classical Logics".

Das *Deduktionstheorem* besagt, dass sich das logische Schlussfolgern auf das Bestimmen der Gültigkeit einer Aussage reduzieren lässt. Eine Aussage ist genau dann *gültig*, wenn sie unabhängig davon wahr ist, welche Bedeutung den einzelnen Wörtern zugewiesen wird. Eine Aussage A ist genau dann eine logische Konsequenz der Aussagen $A_1, \ldots, A_n$, falls die Aussage $(A_1 \wedge \ldots \wedge A_n) \Rightarrow A$ gültig ist. So ist zum Beispiel

$$(d) \quad (\mathit{Mensch(Plato)} \wedge \forall X(\mathit{Mensch}(X) \Rightarrow \mathit{sterblich}(X))) \Rightarrow \mathit{sterblich(Plato)}$$

gültig, das heißt die Aussage (c) folgt tatsächlich logisch aus den Aussagen (a) und (b).

Die Aufgabe eines Beweisers ist es daher herauszufinden, ob eine gegebene Aussage oder *Formel* gültig ist. Da der Suchraum bei der Beweissuche bereits für relativ einfache Theoreme sehr groß ist, stellt das automatische Beweisen eine immense Herausforderung dar. Dieses Forschungsgebiet zählt daher zum Kerngebiet der *Künstlichen Intelligenz*.

Neben der häufig verwendeten *klassischen Logik*, gehören die intuitionistische Logik und die modale Logik zu den wichtigsten *nicht-klassischen Logiken*. Sie haben zahlreiche Anwendungen, zum Beispiel in der Programmsynthese, der Programmverifikation, beim Planen, der Modellierung von Kommunikation und bei der Verarbeitung natürlicher Sprache.

Die *intuitionistische Logik* [Dal01] stellt höhere Anforderungen an die Gültigkeit einer Formel und ist daher eine Einschränkung der klassischen Logik. Die Formel

$$(e) \quad \mathit{Mensch(Sokrates)} \vee \neg \mathit{Mensch(Sokrates)}$$

formalisiert die Aussage *Sokrates ist ein Mensch oder Sokrates ist kein Mensch* und ist in klassischer, aber *nicht* in intuitionistischer Logik gültig. Die *modale Logik* [BBW06] ist eine Erweiterung der klassischen Logik um die Modaloperatoren $\Box$ und $\Diamond$, die als "notwendigerweise" und "möglicherweise" interpretiert werden. Die gültige Formel

$$(f) \quad (\Box \mathit{Mensch(Plato)}) \Rightarrow (\Diamond \mathit{Mensch(Plato)})$$

formalisiert also die Aussage *falls Plato notwendigerweise ein Mensch ist, dann ist er auch möglicherweise ein Mensch*. Abhängig davon, wie die Modaloperatoren interpretiert werden, gibt es nicht nur eine modale Logik, sondern eine ganze Reihe von modalen Logiken.

Während die Entwicklung von effizienten Beweisern für klassische Logik in den letzten Jahrzehnten große Fortschritte gemacht hat, steht die Entwicklung von leistungsfähigen Beweisern für viele nicht-klassische Logiken noch am Anfang. Die Beweissuche in nicht-klassischen Logiken ist erheblich schwieriger als in klassischer Logik. So ist die Beweissuche in klassischer *Aussagenlogik* bereits *co-$\mathcal{NP}$-vollständig*, d.h. alle bekannten Algorithmen haben eine exponentielle Zeitkomplexität. Die Beweissuche in intuitionistischer und den (meisten) modalen Aussagenlogiken ist sogar *$\mathcal{PSPACE}$-vollständig*.

Die formale Beschreibung der Beweissuche dabei häufig mit Hilfe eines *Kalküls*, der aus *Axiomen* und *Regeln* besteht. Ein Beweiser implementiert dann einen Kalkül mit einer Strategie, die beschreibt, in welcher Reihenfolge die Regeln bei der Beweissuche angewendet werden. Bekannte Kalküle zur Beweissuche sind zum Beispiel *Sequenzenkalküle*, *Tableaukalküle*, *Konnektionskalküle*, *Resolutionskalküle* und *instanzenbasierte Kalküle*.

Dieser Artikel ist die Zusammenfassung einer Dissertation [Ott13], in der sowohl neue Konnektionskalküle als auch deren Implementierung für die automatische Beweissuche in klassischer Logik und wichtigen nicht-klassischen Logiken vorgestellt werden.

2 Klausel-Konnektionskalküle

2.1 Konnektionskalkül für klassische Logik

Eine *Konnektion* ist ein Paar von (atomaren) Formeln der Form $\{A, \neg A\}$. Zum Beispiel ist $\{$*Mensch(Plato)*, $\neg$*Mensch(Plato)*$\}$ eine Konnektion; *Mensch(Plato)* und $\neg$*Mensch(Plato)* werden als *Literale* bezeichnet. Bei Konnektionskalkülen [Bib87, LS01] wird die Beweissuche durch Konnektionen gesteuert, was eine zielgerichtete Suche ermöglicht.

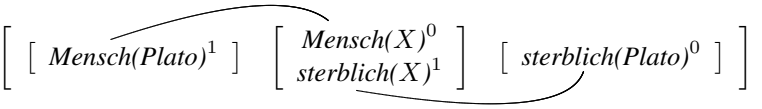
Die Formeln, deren Gültigkeit bewiesen werden sollen, müssen im Konnektionskalkül in Klauselform sein. Eine Formel in *Klauselform* hat die Form $\exists x_1 \ldots \exists x_n (C_1 \vee \ldots \vee C_n)$, wobei jede *Klausel* C_i die Form $L_1 \wedge \ldots \wedge L_m$ hat und jedes L_i ein Literal ist. In klassischer Logik kann jede Formel F in eine äquivalente Formel F' in Klauselform übersetzt werden, so dass F genau dann gültig ist, wenn F' gültig ist. Eine *Matrix* stellt ein Formel in Klauselform als eine Menge von Klauseln $\{C_1, \ldots, C_n\}$ dar, wobei jede Klausel C_i eine Menge von Literalen $\{L_1, \ldots, L_m\}$ ist, und Literale der Form A bzw. $\neg A$ durch A^0 bzw. A^1 dargestellt werden; eine Konnektion hat dann die Form $\{A^0, A^1\}$.

Die Formel (d) aus Abschnitt 1 hat die Klauselform

$$\exists X (\neg \textit{sterblich(Plato)} \vee (\textit{Mensch}(X) \wedge \neg \textit{sterblich}(X)) \vee \textit{sterblich(Plato)})$$

und $M' = \{\{\textit{Mensch(Plato)}^1\}, \{\textit{Mensch}(X)^0, \textit{sterblich}(X)^1\}, \{\textit{sterblich(Plato)}^0\}\}$ ist die zugehörige Matrix. Damit $\{\textit{Mensch}(X)^0, \textit{Mensch(Plato)}^1\}$ eine Konnektion ist, muss die Variable X durch *Plato* ersetzt werden. Dies wird durch eine *Termsubstitution* σ realisiert, d.h. $\sigma(X) = \textit{Plato}$. Die Konnektion wird dann als *σ-komplementär* bezeichnet.

Bei der graphischen Darstellung eines Beweises im Konnektionskalkül werden die Klauseln der Matrix nebeneinander, Literale jeder Klausel übereinander angeordnet; Literale, die eine Konnektion bilden, werden durch einen Linie miteinander verbunden. Der *graphische Konnektionsbeweis* der Matrix M' mit $\sigma(X) = \textit{Plato}$ sieht dann wir folgt aus:



Ein formaler *Klausel-Konnektionskalkül* wurde von Otten und Bibel [OB03] angegeben und ist in Abbildung 1 dargestellt. Die Wörter des Kalküls haben die Form "$C, M, Path$"; M ist eine Matrix, C und $Path$ sind Mengen von Literalen oder ε und werden als *Zielklausel* bzw. *aktiver Pfad* bezeichnet. Ein *Klausel-Konnektionsbeweis* für eine Formel F ist eine Ableitung von $\varepsilon, M, \varepsilon$ im Klausel-Konnektionskalkül, wobei M die Matrix von F ist und jeder Zweig der Ableitung durch ein Axiom abgeschlossen ist. Die folgende Ableitung ist ein Beweis der Matrix M', und damit der Formel (d), im Klausel-Konnektionskalkül.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{\}, M', \{\textit{sterblich(Plato)}^0, \textit{Mensch}(X')^0\}}\,A \quad \dfrac{}{\{\}, M', \{\textit{sterblich(Plato)}^0\}}\,A}{\{\textit{Mensch}(X')^0\}, \{\{\textit{Mensch(Plato)}^1\}, \ldots\}, \{\textit{sterblich(Plato)}^0\}}\,E \quad \dfrac{}{\{\}, M', \{\}}\,A}{\{\textit{sterblich(Plato)}^0\}, \{\{\textit{Mensch(Plato)}^1\}, \{\textit{Mensch}(X)^0, \textit{sterblich}(X)^1\}, \ldots\}, \{\}}\,E}{\varepsilon, \{\{\textit{Mensch(Plato)}^1\}, \{\textit{Mensch}(X)^0, \textit{sterblich}(X)^1\}, \{\textit{sterblich(Plato)}^0\}\}, \varepsilon}\,S$$

Axiom (A)	$\dfrac{}{\{\}, M, Path}$	
Start (S)	$\dfrac{C_2, M, \{\}}{\varepsilon,\ M,\ \varepsilon}$	und C_2 ist Kopie von $C_1 \in M$
Reduktion (R)	$\dfrac{C, M, Path \cup \{L_2\}}{C \cup \{L_1\}, M, Path \cup \{L_2\}}$	und $\{L_1, L_2\}$ ist σ-komplementär
Extension (E)	$\dfrac{C_2 \backslash \{L_2\}, M, Path \cup \{L_1\} \quad C, M, Path}{C \cup \{L_1\}, M, Path}$	und C_2 ist eine Kopie von $C_1 \in M,\ L_2 \in C_2$, $\{L_1, L_2\}$ ist σ-kompl.

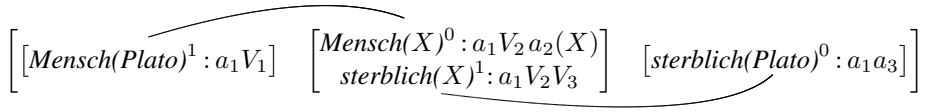
Abbildung 1: Der Klausel-Konnektionskalkül für klassische Logik

Der Beweis der *Korrektheit* und der *Vollständigkeit* des Kalküls beruht auf der *Matrixcharakterisierung* [Bib87] für klassische Logik. Die *Beweissuche* im Klausel-Konnektionskalkül startet mit $\varepsilon, M, \varepsilon$ und die Regeln werden analytisch, d.h. von unten nach oben, angewendet. Falls es bei der Anwendung von Regeln oder Regelinstanzen mehrere Alternativen gibt, muss durch *Backtracking* sichergestellt werden, dass gegebenfalls alle alternative Regeln oder Regelinstanzen berücksichtigt werden. Die Termsubstitution σ wird dabei schrittweise durch einen der bekannten Algorithmen zur *Termunifikation* berechnet.

2.2 Konnektionskalkül für intuitionistische Logik

Um den Konnektionskalkül für klassische Logik auf intuitionistische Logik zu erweitern, wird jedem Literal ein Präfix zugeordnet. Ein *Präfix* ist eine Zeichenkette, die aus *Präfixvariablen* und *Präfixkonstanten* besteht. Er wird bestimmt durch den Kontext, in dem das Literal in der Formel vorkommt. Eine Konnektion hat dann die Form $\{A(t_1)^0 : p_1,\ A(t_2)^1 : p_2\}$, in der die Literale den Präfix p_1 bzw. p_2 haben. Aufgrund der Matrixcharakterisierung für intuitionistische Logik [Wal90] ist diese Konnektion genau dann σ-komplementär, wenn es zusätzlich zu der Termsubstitution σ_T eine *Präfixsubstitution* σ_P gibt, so dass $\sigma_T(t_1) = \sigma_T(t_2)$ und $\sigma_P(p_1) = \sigma_P(p_2)$. Da es für intuitionistische Logik keine Klauselform gibt, wird die Matrixcharakterisierung angepasst. Dazu wird die für klassische Logik bekannte *Skolemisierungstechnik* auf Präfixkonstanten erweitert [Ott05].

Für die Formel (d) ist $\{\{$*Mensch(Plato)*$^1 : a_1 V_1\}, \{$*Mensch(X)*$^0 : a_1 V_2\, a_2(X)$, *sterblich(X)*$^1 : a_1 V_2 V_3\}, \{$*sterblich(Plato)*$^0 : a_1 a_3\}\}$ die *intuitionistische Matrix*, wobei a_1, $a_2(X)$, a_3 Präfixkonstanten und V_1, V_2, V_3 Präfixvariablen sind. Der graphische intuitionistische Konnektionsbeweis dieser Matrix mit $\sigma_T(X) = Plato$, $\sigma_P(V_1) = a_2(Plato)$, $\sigma_P(V_2) = \varepsilon$ (wobei ε die leere Zeichenkette ist) und $\sigma_J(V_3) = a_3$ sieht dann wie folgt aus:



Axiom (A) $\dfrac{}{\{\}, M, Path}$

Start (S) $\dfrac{C_2, M, \{\}}{\varepsilon,\ M,\ \varepsilon}$ und C_2 ist Kopie von $C_1 \in M$

Reduktion (R) $\dfrac{C, M, Path \cup \{L_2 : p_2\}}{C \cup \{L_1 : p_1\}, M, Path \cup \{L_2 : p_2\}}$ und $\{L_1 : p_1, L_2 : p_2\}$ ist σ-komplementär

Extension (E) $\dfrac{C_2 \backslash \{L_2 : p_2\}, M, Path \cup \{L_1 : p_1\} \quad C, M, Path}{C \cup \{L_1 : p_1\}, M, Path}$

und C_2 ist ein Kopie von $C_1 \in M$, $L_2{:}p_2 \in C_2$, $\{L_1{:}p_1, L_2{:}p_2\}$ ist σ-kompl.

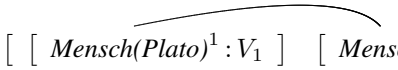
Abbildung 2: Der Klausel-Konnektionskalkül für intuitionistische und modale Logik

Für die Formel (e) ergibt sich die intuitionistische Matrix $\{\{$*Mensch(Sokrates)*$^0 : a_1\}$, $\{$*Mensch(Sokrates)*$^1 : a_2\}\}$. Da es keine Substitution σ_P gibt, die die beiden Präfixe a_1 und a_2 der Konnektion gleichmacht, ist die Formel intuitionistisch *nicht* gültig.

Der formale *Klausel-Konnektionskalkül für intuitionistische Logik* [Ott05] ist in Abbildung 2 dargestellt, wobei M eine intuitionistische Matrix ist. Der Beweis des Korrektheit und der Vollständigkeit des Kalküls basiert auf der angepassten Matrixcharakterisierung. Die Präfixsubstitution σ_P wird von einem speziellen *Präfixunifikations-Algorithmus* berechnet [Ott05]. Der Algorithmus berechnet aus einer Menge von Präfix-Gleichungen mit Hilfe von Ersetzungsregeln eine *minimale* Menge *allgemeinster* Präfixsubstitutionen.

2.3 Konnektionskalkül für modale Logiken

Der *Klausel-Konnektionskalkül für modale Logik* [Ott12, BOR12] ist im Wesentlichen identisch mit dem Kalkül für intuitionistische Logik in Abbildung 2. Die Definition der Präfixe ist nun abhängig von den Modaloperatoren, in deren Kontext ein Literal in der Formel vorkommt [Wal90]. Der Kalkül verwendet dabei die *modale Matrix*, in der wiederum eine angepasste Skolemisierungstechnik benutzt wird [Ott12]. Zum Beispiel ist die modale Matrix der Formel (f) aus Abschnitt 1 $\{\{$*Mensch(Plato)*$^1{:}V_1\}$, $\{$*Mensch(Plato)*$^0{:}V_2\}\}$, wobei V_1 und V_2 Präfixvariablen sind. Der graphische modale Konnektionsbeweis dieser Matrix mit $\sigma_P(V_1) = V_2$ sieht wie folgt aus:



Der Algorithmus zur Präfixunifikation wird angepasst und berücksichtigt nun die *Erreichbarkeitsrelation* der *Kripke-Semantik* der gewählten modalen Logik. Dabei wurden entsprechende Algorithmen für die modalen Logiken D, T, S4 und S5 entwickelt [Ott12]. Eine zusätzliche *Domänen-Bedingung* spezifiziert, ob *konstante*, *kumulative* oder *variierende Domäne* betrachtet werden.

3 Implementierung von Klausel-Konnektionskalkülen

3.1 leanCoP für klassische Logik

leanCoP ist ein Beweiser für klassische Prädikatenlogik [OB03, Ott08] und stellt eine sehr kompakte Prolog-Implementierung des Klausel-Konnektionskalküls aus Abschnitt 2.1 dar. Während leanCoP 1.0 [OB03] im Wesentlichen den Basis-Kalkül implementiert, verwendet leanCoP 2.0 zusätzliche Techniken, die die Beweissuche optimieren [Ott08, Ott10]. Der Quellcode des Kernbeweisers ist in Abbildung 3 dargestellt.

Lean-Prolog-Technologie ist eine Technik, die die Klauseln der gegebenen Matrix mit Hilfe des Prädikats `lit/3` repräsentiert und vor der Beweissuche in der Prolog-Datenbank speichert. Die Beweissuche wird dann mit `prove(I,S)` gestartet; `I` ist die maximale Länge des aktiven Pfades und wird für die iterative Tiefensuche benutzt, `S` ist eine Menge von Optionen, um die Beweissuche zu steuern [Ott08, Ott10]. Die *Start*-Regel wird in den ersten beiden Zeilen implementiert, die nächsten beiden Zeilen realisieren die iterative Tiefensuche. Die *Reduktions-* und *Extensions*-Regel werden durch das Prädikat `prove(C,P,I,Q,S)` implementiert; `C` ist die Zielklausel, `P` ist der aktive Pfad und `Q` ist die Menge der bereits bewiesenen Literale und wird für eine zusätzliche *Lemma*-Regel verwendet. Das *Axiom* wird in Zeile fünf implementiert. Die Termsubstitution wird implizit durch Prolog gespeichert. Weitere Techniken sind *Regularität* (Zeile 6/7), die *kontrollierte Tiefensuche* (Zeile 9/10), sowie ein *eingeschränktes Backtracking* (Zeile 11). Das eingeschränkte Backtracking ist die zur Zeit effektivste Technik, um die Beweissuche im Konnektionskalkül zu optimieren [Ott10]. Weiterhin wird eine *definitorische Klauselform-Transformation* benutzt und ein *Strategie-Scheduling*, das nacheinander verschiedene Strategien bei der Beweissuche anwendet [Ott08, Ott10].

Diese zusätzlichen Techniken in leanCoP 2.0 führen zu einer Verbesserung der Leistung um etwa den Faktor 10^5 bezüglich der international etablierten *TPTP-Problemsammlung*. leanCoP 2.1 gibt zusätzlich einen lesbaren Konnektionsbeweis aus.

Der komplette Beweiser mit den zusätzlichen Komponenten ist auf der leanCoP-Webseite unter der Adresse `http://www.leancop.de` verfügbar.

```
prove(I,S) :- \+member(scut,S) -> prove([-(#)],[],I,[],S) ;
    lit(#,C,_) -> prove(C,[-(#)],I,[],S).
prove(I,S) :- member(comp(L),S), I=L -> prove(1,[]) ;
    (member(comp(_),S);retract(p)) -> J is I+1, prove(J,S).
prove([],_,_,_,_).
prove([L|C],P,I,Q,S) :- \+ (member(A,[L|C]), member(B,P),
    A==B), (-N=L;-L=N) -> ( member(D,Q), L==D ;
    member(E,P), unify_with_occurs_check(E,N) ; lit(N,F,H),
    (H=g -> true ; length(P,K), K<I -> true ;
    \+p -> assert(p), fail), prove(F,[L|P],I,Q,S) ),
    (member(cut,S) -> ! ; true), prove(C,P,I,[L|Q],S).
```

Abbildung 3: Der Quellcode des Kernbeweisers von leanCoP 2.0 für klassische Logik

```
prove(I,S) :- ( \+member(scut,S) ->
    prove([(-(#)):(-[])],[],I,[],[Z,T],S) ;
    lit(#):_,G:C,_) -> prove(C,[(-(#)):(-[])],I,[],[Z,R],S),
    append(R,G,T) ), check_addco(T), prefix_unify(Z).
prove(I,S) :- member(comp(L),S), I=L -> prove(1,[]) ;
    (member(comp(_),S);retract(p)) -> J is I+1, prove(J,S).
prove([],_,_,_,[[],[]],_).
prove([L:U|C],P,I,Q,[Z,T],S) :- \+(member(A,[L:U|C]), member(B,P),
    A==B), (-N=L;-L=N) -> ( member(D,Q), L:U==D, X=[], O=[] ;
    member(E:V,P), unify_with_occurs_check(E,N),
    \+ \+ prefix_unify([U=V]), X=[U=V], O=[] ;
    lit(N:V,M:F,H), \+ \+ prefix_unify([U=V]),
    (H=g -> true ; length(P,K), K<I -> true ;
    \+p -> assert(p), fail), prove(F,[L:U|P],I,Q,[W,R],S),
    X=[U=V|W], append(R,M,O) ), (member(cut,S) -> ! ; true),
    prove(C,P,I,[L:U|Q],[Y,J],S), append(X,Y,Z), append(J,O,T).
```

Abbildung 4: Der Quellcode der Kernbeweiser von ileanCoP 1.2 und MleanCoP 1.2

3.2 ileanCoP für intuitionistische Logik

ileanCoP ist ein Beweiser für intuitionistische Prädikatenlogik [Ott05, Ott08] und eine sehr kompakte Implementierung des intuitionistischen Kalküls aus Abschnitt 2.2. Der Quellcode des Kernbeweisers ist in Abbildung 4 dargestellt. Lediglich die unterstrichenen Zeichen wurden dem Quellcode von leanCoP 2.0 in Abbildung 3 hinzugefügt. Es wurden Präfixe hinzugefügt, als auch eine Präfixunifikation (`prefix_unify/1`), die weitere 18 Zeilen an Prolog-Code benötigt. Alle Optimierungen von leanCoP 2.0 wurden angepasst und integriert. ileanCoP 1.2 beweist erheblich mehr Probleme aus der TPTP-Problemsammlung und der *ILTP-Problemsammlung* [ROK07] als alle anderen Beweiser für intuitionistische Logik [Ott08]. Der komplette Beweiser ist auf der ileanCoP-Webseite unter der Adresse `http://www.leancop.de/ileancop/` verfügbar.

3.3 MleanCoP für modale Logiken

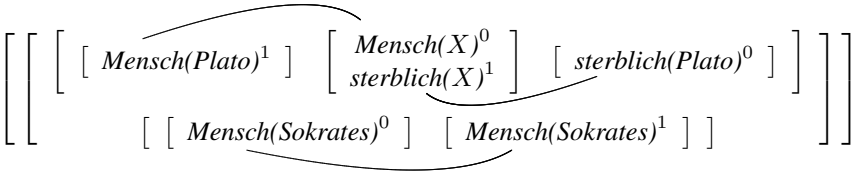
MleanCoP ist ein Beweiser für die modalen Prädikatenlogiken D, T, S4 und S5 mit konstanten, kumulativen und variierenden Domäne [Ott12] und eine sehr kompakte Implementierung des modalen Kalküls aus Abschnitt 2.3. Der Quellcode des Kernbeweisers ist im Wesentlichen identisch mit dem für ileanCoP in Abbildung 4. Dabei wird die Definition der Präfixe an die modalen Logiken angepasst. Weiterhin müssen die Präfixunifikationen für die modalen Logiken D, T, S4 und S5 hinzugefügt und die *Domänen-Bedingung* (`check_addco/1`) angepasst werden. MleanCoP 1.2 beweist erheblich mehr Probleme aus der *QMLTP-Problemsammlung* [RO12] als alle anderen Beweiser für modale Logik [BOR12]. Der komplette Beweiser ist auf der MleanCoP-Webseite unter der Adresse `http://www.leancop.de/mleancop/` verfügbar.

4 Der Nicht-Klausel-Konnektionskalkül

Die für die Klausel-Konnektionskalküle notwendige Transformation in Klauselform verändert die Struktur der Formel und kann dazu führen, dass sich die Formel deutlich vergrößert, was die Beweissuche erheblich erschweren kann. Dies gilt auch für die definitorische Klauselform-Transformation [Ott10]. Der *Nicht-Klausel-Konnektionskalkül* [Ott11] hat diese Nachteile nicht, da der Kalkül keine Klauselform benötigt und stattdessen die originale Struktur der gegebenen Formel erhält. Dafür muss die Definition der Matrix verallgemeinert werden. Eine *Nicht-Klausel-Matrix* ist eine Menge von Klauseln, wobei jede Klausel eine Menge von Literalen und (Unter-)Matrizen ist. Zum Beispiel wird die Formel

$$\begin{array}{c}(\,(\,\mathit{Mensch(Plato)} \wedge \forall X(\mathit{Mensch}(X) \Rightarrow \mathit{sterblich}(X))\,) \Rightarrow \\ \mathit{sterblich(Plato)}\,) \;\wedge\; (\,\mathit{Mensch(Sokrates)} \vee \neg \mathit{Mensch(Sokrates)}\,)\end{array}$$

durch die Nicht-Klausel-Matrix $\{\{\,\{\{\,\mathit{Mensch(Plato)}^1\}, \{\mathit{Mensch}(X)^0, \mathit{sterblich}(X)^1\}, \{\mathit{sterblich(Plato)}^0\}\}, \{\{\mathit{Mensch(Sokrates)}^0\}, \{\mathit{Mensch(Sokrates)}^1\}\}\,\}\}$ repräsentiert, die 6 Literale enthält. Die entsprechende *Klausel*form enthält mindestens 14 Literale. Der graphische *Nicht-Klausel-Konnektionsbeweis* mit $\sigma(X) = \mathit{Plato}$ hat die folgende Form:



Der *formale* Nicht-Klausel-Konnektionskalkül hat das gleiche Axiom und die gleiche Start- und Reduktions-Regel wie der Klausel-Konnektionskalkül. Die Extensions-Regel wird angepasst und eine *Dekompositions*-Regel hinzugefügt [Ott11]. Die vereinfachte Version des Kalküls, bei dem Start- und Reduktions-Regel subsumiert werden, ist in Abbildung 5 dargestellt. Für die Nicht-Klausel-Matrix M startet die Suche mit $\{M\}, M, \{\}$.

Der Beweis der Korrektheit und der Vollständigkeit des Nicht-Klausel-Konnektionskalküls basiert auf einer Nicht-Klausel-Matrixcharakterisierung und der Einbettung des Nicht-Klausel-Kalküls in den Klausel-Konnektionskalkül [Ott11].

Axiom (A)	$\dfrac{}{\{\}, M, Path}$	
Extension (E)	$\dfrac{C_3, M[C_1 \backslash C_2], Path \cup \{L_1\} \quad C, M, Path}{C \cup \{L_1\}, M, Path}$	und $C_3 := \beta\text{-}clause_{L_2}(C_2)$, C_2 ist Kopie von C_1, C_1 ist e-Klausel von M bzgl. $Path \cup \{L_1\}$, C_2 enthält L_2 mit $\{L_1, L_2\}$ σ-kompl.
Dekomposition (D)	$\dfrac{C \cup C_1, M, Path}{C \cup \{M_1\}, M, Path}$	und $C_1 \in M_1$

Abbildung 5: Der (vereinfachte) Nicht-Klausel-Konnektionskalkül

5 Zusammenfassung

Formales Beweisen ist eine fundamentale Tätigkeit in der Informatik und vielen verwandten Wissenschaften. Seit Frege seine *Begriffsschrift* [Fre79] im Jahr 1879 veröffentlicht hat, gilt die Entwicklung von effizienten Kalkülen zur Automatisierung des formalen Beweisens als eines der wichtigsten Forschungsziele innerhalb der Künstlichen Intelligenz. Die Entwicklung von Kalkülen und Beweisern hat in den letzten Jahrzehnten große Fortschritte gemacht, und Beweiser werden heutzutage in industriellen Anwendungen, wie etwa der Verifikation von Hard- und Software, eingesetzt.

Die vorliegende Arbeit stellt leistungsfähige Beweiser für klassische und einige wichtige nicht-klassische Logiken vor. Alle Beweiser basieren auf einem uniformen Klausel-Konnektionskalkül, der durch die Verwendung von Präfixen, einer Präfixunifikation sowie einer erweiterten Skolemisierung auf intuitionistische und modale Logiken übertragen wurde. Daneben enthalten die Beweiser weitere Techniken um die Beweissuche zu optimieren, darunter eine optimierte definitorische Klauselform-Transformation, Lean-Prolog-Technologie, Regularität und das sehr erfolgreiche "eingeschränkte Backtracking".

leanCoP ist der zur Zeit schnellste Beweiser, der auf einem Klausel-Konnektionskalkül basiert. Der Kernbeweiser besteht aus nur wenigen Zeilen Prolog-Code. leanCoP hat bereits mehrfach erfolgreich an den internationalen CASC-Wettbewerben teilgenommen. Darunter sind nicht nur Platzierungen unter den Top 3 der schnellsten Beweiser mit Beweisausgabe, sondern auch der "Best Newcomer"-Preis für leanCoP 2.0 [Sut08], der dritte Preis in der SUMO-Kategorie, die sehr große Formeln enthält, für leanCoP-SInE [Sut10], und der Gewinn der arithmetischen TFA-Kategorie durch leanCoP-Ω [Sut11].

ileanCoP und MleanCoP sind die zur Zeit leistungsfähigsten Beweiser für intuitionistische und modale Prädikatenlogik. Sie basieren auf leanCoP und erweitern diesen durch das Hinzufügen von Präfixen und einer Präfixunifikation. Die spezielle Logik wird dabei lediglich durch das Austauschen der Unifikationskomponente bestimmt. Dadurch lassen sich Optimierungstechniken, die bereits für klassische Logik benutzt werden, auf einfache Art übertragen. Die Benutzung einer zusätzlichen Präfixunifikation bei der Beweissuche in nicht-klassischen Logiken ähnelt der Benutzung der Termunifikation bei der Beweissuche in klassischer *Prädikaten*logik. Dieser Sachverhalt lässt sich wie folgt darstellen:

$$\begin{array}{lcl} \text{Prädikatenlogik} & = & \text{Aussagenlogik} \;+\; \text{Termunifikation}\,, \\ \text{intuitionistische/modale Logik} & = & \text{klassische Logik} \;+\; \text{Präfixunifikation}\,. \end{array}$$

Der Nicht-Klausel-Konnektionskalkül verallgemeinert den Klausel-Konnektionskalkül. Es wird lediglich die Extensions-Regel leicht angepasst und eine Dekompositions-Regel hinzugefügt. Dies kombiniert die Vorteile von (Nicht-Klausel-)Sequenzen- und Tableaukalkülen mit der zielgerichteten Beweisführung von Klausel-Konnektionskalkülen.

Zukünftige Arbeiten beinhalten die Erweiterung der Kalküle und Beweiser auf weitere nicht-klassische Logiken, wie etwa multimodale Logiken und Beschreibungslogiken, als auch die Implementierung des Nicht-Klausel-Konnektionskalküls und dessen Erweiterung auf nicht-klassische Logiken. Diese Arbeiten werden weitere Belege dafür geben, dass Konnektionskalküle eine solide und erfolgreiche Grundlage für die Automatisierung des formalen Beweisens in klassischen und nicht-klassischen Logiken sind.

Literatur

[BBW06] P. Blackburn, J. van Bentham, F. Wolter. *Handbook of Modal Logic*. Elsevier, Amsterdam, 2006.

[Bib87] W. Bibel. *Automated Theorem Proving*. Vieweg, Wiesbaden, 2. Aufl., 1987.

[BOR12] C. Benzmüller, J. Otten, T. Raths. Implementing and Evaluating Provers for First-order Modal Logics. In L. De Raedt et al., Ed., *20th European Conference on Artificial Intelligence (ECAI 2012)*, S. 163–168. IOS Press, Amsterdam, 2012.

[Dal01] D. van Dalen. Intuitionistic Logic. In L. Goble, Ed., *The Blackwell Guide to Philosophical Logic*. Blackwell, Oxford, 2001.

[Fre79] G. Frege. *Begriffsschrift: Eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. L. Nebert, Halle, 1879.

[LS01] R. Letz, G. Stenz. Model Elimination and Connection Tableau Procedures. In A. Robinson, A. Voronkov, Ed., *Handbook of Automated Reasoning*, S. 2015–2114. Elsevier, Amsterdam, 2001.

[OB03] J. Otten, W. Bibel. leanCoP: lean connection-based theorem proving. *Journal of Symbolic Computation*, 36:139–161, 2003.

[Ott05] J. Otten. Clausal Connection-Based Theorem Proving in Intuitionistic First-Order Logic. In B. Beckert, Ed., *TABLEAUX 2005*, LNAI 3702, S. 245–261. Springer, Heidelberg, 2005.

[Ott08] J. Otten. leanCoP 2.0 and ileanCoP 1.2: High Performance Lean Theorem Proving in Classical and Intuitionistic Logic. In A. Armando, P. Baumgartner, G. Dowek, Ed., *IJCAR 2008*, LNCS 5195, S. 283–291. Springer, Heidelberg, 2008.

[Ott10] J. Otten. Restricting backtracking in connection calculi. *AI Communications*, 23:159–182, 2010.

[Ott11] J. Otten. A Non-clausal Connection Calculus. In K. Brünnler, G. Metcalfe, Ed., *TABLEAUX 2011*, LNAI 6793, S. 226–241. Springer, Heidelberg, 2011.

[Ott12] J. Otten. Implementing Connection Calculi for First-order Modal Logics. In E. Ternovska, K. Korovin, S. Schulz, Ed., *IWIL 2012*, EPiC, volume 22, S. 18–32. EasyChair, 2012.

[Ott13] J. Otten. Connection Calculi for Automated Theorem Proving in Classical and Non-Classical Logics. Dissertation, Institut für Informatik, University of Potsdam, 2013.

[RO12] T. Raths, J. Otten. The QMLTP Problem Library for First-order Modal Logics. In B. Gramlich et al., Ed., *IJCAR 2012*, LNAI 7364, S. 454–461. Springer, Heidelberg, 2012.

[ROK07] T. Raths, J. Otten, C. Kreitz. The ILTP problem library for intuitionistic logic. *Journal of Automated Reasoning*, 38:261–271, 2007.

[Sut08] G. Sutcliffe. The CADE-21 automated theorem proving system competition. *AI Communications*, 21:71–81, 2008.

[Sut10] G. Sutcliffe. The CADE-22 automated theorem proving system competition – CASC-22. *AI Communications*, 23:47–59, 2010.

[Sut11] G. Sutcliffe. The 5th IJCAR automated theorem proving system competition – CASC-J5. *AI Communications*, 24:75–89, 2011.

[Wal90] L. A. Wallen. *Automated Deduction in Nonclassical Logics*. MIT Press, Cambridge, 1990.

Jens Otten studierte Informatik an der Technischen Universität Darmstadt und arbeitete dort anschließend als Stipendiat und wissenschaftlicher Mitarbeiter. Aufenthalte als Gastwissenschaftler führten ihn an die Oxford Universität in England und an die Cornell Universität in Ithaca, New York. Nach mehrjähriger Tätigkeit als Technical Alliance Manager bei Hewlett-Packard in England, nahm er eine Stelle am Institut für Informatik der Universität Potsdam an, wo er in Theoretischer Informatik promovierte.

Seit mehr als zehn Jahren forscht er an der Automatisierung des formalen Beweisens, einem Kerngebiet der Künstlichen Intelligenz. Der Schwerpunkt liegt dabei auf der Entwicklung und Implementierung von Kalkülen für klassische und nicht-klassische Logiken. Er hat mehr als 30 Artikel in Büchern, Fachzeitschriften, Konferenz- und Workshop-Tagungsbänden veröffentlicht. Neben Gutachtertätigkeiten für internationale Fachzeitschriften und Konferenzen ist er Mitglied in Programmkomitees von mehreren Konferenzen und Workshops. Er ist Mitglied im Vorstand der "International Joint Conference on Automated Reasoning" (IJCAR) und Vorstandsvorsitzender der "International Conference on Automated Reasoning with Analytic Tableaux and Related Methods" (TABLEAUX).