BOSCH
Technik fürs Leben

DHBW
Duale Hochschule
Baden-Württemberg

Implementierung von Set-Of-Support-Strategien für zur Theorem-Beweisführung in Python

Studienarbeit

Bachelor of Science

des Studiengangs Informatik

an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Nico Makowe

10.06.2022

Bearbeitungszeitraum	29.09.2021 - 10.06.2022
Matrikelnummer, Kurs	9275184, TINF19ITA
Ausbildungsfirma	Robert Bosch GmbH, Stuttgart
Betreuer	Professor Dr. Stephan Schulz

Selbstständigkeitserklärung

Ich versichere hiermit, dass ich meine Studienarbeit mit dem Thema: *Implementierung von Set-Of-Support-Strategien für zur Theorem-Beweisführung in Python* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Stuttgart, 10.06.2022

Nico Makowe

Abstract

Abstract Inhalt

Abstract

Abstract Content

Inhaltsverzeichnis

Abkürzungsverzeichnis

SOS	Set of Support

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

1.1 Problemstellung

Beim Testen von Software können Theorembeweiser eingesetzt werden, um Anforderungen zu verifizieren. Diese Beweiser können mithilfe gegebener Aussagen neue Zusammenhänge und auch Widersprüche herleiten.

Ein typisches Beispiel für den Einsatz von Theorembeweisern ist die Verifikation von Gleitkommarechenwerken [2]. Sie kommen in fast allen modernen Prozessoren vor. Die Schwierigkeit bei Gleitkommazahlen liegt darin, dass reelle Zahlen mit beliebiger Größe von einer Bitsequenz repräsentiert werden soll, die eine begrenzte Größe besitzt. Die Zahlen sind deshalb als Potenz kodiert, was sowohl für sehr große als auch für sehr kleine Zahlen eine präzise Abstufung ermöglicht. Für aufwändige Berechnungen wie Division, werden die Algorithmen entsprechend komplex. Nachdem bei Intel-Prozessoren im Jahr 1994 ein Fehler bei der Division mancher Zahlen entdeckt worden war, wurden bei den Nachfolgermodellen Theorembeweiser eingesetzt, um die Korrektheit aller Berechnungen mathematisch zu beweisen.

Es gibt Beweisverfahren, die vollständig fürs Widerlegen sind. Das heißt, dass es aus einer widersprüchlichen Menge von Aussagen, immer möglich ist, einen Widerspruch nachzuweisen. In der Praxis wird dieses Ziel nicht immer erreicht, da häufig unendlich viele Formeln abgeleitet werden können und es bei wachsender Formelmenge immer unwahrscheinlicher wird, einen Widerspruch zu finden. Es gibt verschiedene Ansätzen, um dem schnellen Anwachsen der Formelmenge entgegenzuwirken. Zum Beispiel kann die Anzahl der gegebenen Aussagen verringert werden, wenn bestimmtes Wissen nicht zum Beweis beiträgt, oder es können Verfahren zur Reduzierung von Formeln eingesetzt werden. Bei der Set-of-Support-Strategie (SOS) werden die gegebenen Aussagen in zwei Formelmengen eingeteilt und es werden Regeln definiert, die die Ableitung neuer Formeln einschränken. Auf diese Weise können weniger neue Formeln ableiten werden können, wodurch die Wahrscheinlichkeit, einen Widerspruch zu finden, erhöht wird. Der gesamte Beweisprozess soll mit der SOS-Strategie sowohl effektiver werden, als auch effizienter.

Das Ziel dieser Arbeit ist die Konzeption und Implementierung einer Set-Of-Support-Strategie im bestehenden Theorembeweiser "PyRes". Außerdem soll das Laufzeitverhalten der Beweisführung mit und ohne SOS-Strategie verglichen werden. Ein weiter Ziel ist, die Implementierung mit anderen Optimierungsstrategien, wie Subsumption, Literal-Selektion oder geordneter Resolution kompatibel zu machen.

1.2 Vorgehensweise der Arbeit

Im Kapitel Grundlagen wird theoretisches Basiswissen vermittelt, das für das Verständnis dieser Arbeit notwendig ist. Der Fokus liegt auf der Prädikatenlogik erster Stufe und dem Resolutionsverfahren zur Beweisführung. Außerdem wird ein Überblick über die bestehende Version des Programms PyRes geschaffen. Der Schwerpunkt bei der Konzeption liegt darauf, den formalen Ablauf der Beweisführung herauszuarbeiten und zu planen. Das fertige Konzept wird im Teil Implementierung umgesetzt und getestet. Die Validierung bewertet die Verbesserung des Programms mit dem alten Stand ohne SOS-Strategie. Die Bewertung basiert auf die Ergebnisse von Performancetests, die das Laufzeit der Beweisführungen vergleichen. Alle Ergebnisse werden in einer kritischen Reflexion zusammengefasst und reflektiert. Außerdem wird ein Ausblick für die Zukunft gegeben.

2 Grundlagen

2.1 Aussagenlogik

Mit der Aussagenlogik können viele logische Zusammenhänge formal beschrieben werden. Eine Aussage stellt eine Behauptung auf, die entweder wahr oder falsch sein kann, aber nicht beides. Beispiele für Aussagen sind: "Die Sonne scheintünd "Die Temperatur beträgt mindestens 20 °C". Solche Aussagen werden mit einzelnen Großbuchstaben $A, B, C, ...$ dargestellt und als Atome bezeichnet. Die Menge aller Atom-Symbole Σ wird als Signatur bezeichnet.

Atome können mithilfe von Junktoren verknüpft werden, um Zusammenhänge von Aussagen darzustellen. Das Ergebnis dieser Verknüpfung ist eine Formel. Ist ein Atom C nur dann wahr, wenn sowohl A als auch B wahr sind, kann man dies mit folgender Formel ausdrücken:

$$C = A \wedge B$$

Weitere Junktoren sind $\vee$ (logisches Oder), $\rightarrow$ (wenn, dann), $\leftrightarrow$ (genau dann, wenn) und $\neg$ (Negierung).

Weitere Symbole, die in der Aussagenlogik verwendet werden, sind das Symbol $\top$ für wahr und das Symbol $\bot$ für falsch.

Folgende Tabelle zeigt, wie sich aus den Wahrheitswerten zweier Atome die Wahrheitswerte einer Formel ergeben.

A	B	Negierung $\neg A$	Konjunktion $A \wedge B$	Disjunktion $A \vee B$	Implikation $A \rightarrow B$	Äquivalenz $A \leftrightarrow B$
wahr	wahr	falsch	wahr	wahr	wahr	wahr
wahr	falsch	falsch	falsch	wahr	falsch	falsch
falsch	wahr	wahr	falsch	wahr	wahr	falsch
falsch	falsch	wahr	falsch	falsch	wahr	falsch

Tabelle 2.1: Übersicht der Wahrheitswerte von Junktoren

Formeln lassen sich rekursiv konstruieren und können dadurch beliebig weit verschachtelt werden. Die Konstruktionsregeln sind folgendermaßen definiert:

1. Alle Symbole in Σ sowie die Symbole $\top$ und $\bot$ sind Formeln.

2. Wenn F eine Formel ist, dann sind auch $\neg F$ und (F) eine Formel.

3. Wenn F und G Formeln sind, dann sind auch $F \vee G$, $F \wedge G$, $F \rightarrow G$ und $F \leftrightarrow G$ Formeln.

Wie bei mathematischen Operatoren z.B. Addition und Multiplikation, haben auch in der Logik die Operatoren eine Präzedenz. Sie beschreibt, wie stark die Operatoren binden, also in welcher Reihenfolge sie aufgelöst werden.In der Literatur wird meistens folgende Rangfolge eingesetzt:

$$\neg \quad \wedge \quad \vee \quad \rightarrow \quad \leftrightarrow$$

Die Negation bindet am stärksten und wird als erstes aufgelöst. Das Symbol für die Äquivalenz wird als letztes aufgelöst. Klammern können zusätzlich eingefügt werden, sie binden noch schwächer. Zwei Junktoren der gleichen Präzedenz werden von links nach rechts aufgelöst. Das heißt die beiden folgenden Formeln beschreiben den gleichen Zusammenhang.

$$A \rightarrow B \rightarrow C \qquad\qquad ((A \rightarrow B) \rightarrow C)$$

2.1.1 Interpretation

Eine Interpretation I ist eine Abbildung, bei der jedem Atom einer der beiden Wahrheitswerte wahr oder falsch zugeordnet wird.

$$I : \Sigma \rightarrow \{\top,\bot\}$$

Unter einer Interpretation I ist dann auch jede Formel entweder wahr oder falsch.

Die Auflösung einer Formel F_0 findet rekursiv statt. F_0 habe eine der beiden Zusammensetzungen, wobei F_1 und F_2 auch Teilformeln von F_0 sind und $\oplus$ ein zweistelliger Junktor ist:

$$F = \neg F_1 \qquad \text{oder} \qquad F = F_1 \oplus F_2$$

1. Wenn eine Teilformeln F_i ein Atom ist, dann ist der Wahrheitswert bekannt, da er von der Interpretation vorgegeben wird.

2. Wenn die Wahrheitswerte aller Teilformeln F_i bekannt sind, kann mit Tabelle 2.1 bestimmt werden, ob die ganze Formel F_0 wahr oder falsch ist.

3. Wenn nicht die Wahrheitswerte aller Teilformeln bekannt sind, werden die Wahrheitswerte bestimmt. Dazu werden für jedes F_i die selben Schritte durchgeführt wie für F_0.

4. Sobald die Wahrheitswerte aller Teilformeln bekannt sind, wird Schritt 2 wiederholt. Die Auflösung ist abgeschlossen.

2.1.2 Äquivalenz von Formeln

Zwei Formeln werden äquivalent genannt, wenn sie für jede Interpretation den gleichen Wahrheitswert besitzen.

Mit Äquivalenzumformungen kann aus einer Formel F eine Formel F' hergeleitet werden, die äquivalent zu F ist. Ein Beispiel für eine Äquivalenzumformung ist die doppelte Negation. Wenn F eine Formel ist, dann ist $\neg\neg F$ äquivalent zu F. Doppelte Negationen können in jeder beliebigen Position einer Formel eingefügt und entfernt werden, da die Negation am stärksten bindet.

2.1.3 Normalformen

2.2 Prädikatenlogik erster Stufe

2.2.1 Begriffe

Prädikat, Funktion, Term, Konstante, Variable Quantor

Die Prädikatenlogik erster Stufe ist eine Erweiterung zur Aussagenlogik. Um die drei Aussagen „Albert ist ein Mensch“, „Bernhard ist ein Mensch“ und „Cäsar ist ein Mensch“ in der Aussagenlogik dazustellen, müsste man drei Atome A, B und C einführen. Anstatt Aussagen, gibt es in der Prädikatenlogik Prädikate. Um das genannte Beispiel in der Prädikatenlogik zu formalisieren, könnte man ein Prädikat $P(x)$ einführen, das den Satz „x ist ein Mensch“ zum Ausdruck bringt. Für die Variable x können dann Werte eingesetzt werden, zum Beispiel a, b und c für Albert, Bernhard und Cäsar. Prädikate können eine beliebige Stelligkeit besitzen. Das zweistellige Prädikat $Q(x,y)$ könnte zum Beispiel den Zusammenhang „x ist der Bruder von y“ darstellen.

In

Zusätzlich zu Prädikaten, gibt es in der Prädikatenlogik Funktionen. Wie Prädikate können, Funktionen beliebig viele Stellen haben.

2.2.2 Ableitung

Formeln

2.2.3 Klausel

Wie in Abschnitt 2.1.3 beschrieben wurde, lässt sich jede aussagenlogische Formel in die konjunktive Normalform bringen. Eine andere Darstellungsform dieser Normalform ist die Schreibweise in Klauselform.

$$\begin{array}{ccccccc} (L_{1,1} \vee L_{1,2} \vee \cdots \vee L_{1,n_1}) & \wedge & (L_{2,1} \vee \cdots \vee L_{2,n_2}) & \wedge \cdots \wedge & (L_{m,1} \vee \cdots \vee L_{m,n_m}) \\ \{\{L_{1,1}, L_{1,2}, \cdots, L_{1,n_1}\} & , & \{L_{2,1}, \cdots, L_{2,n_2}\} & , \cdots , & \{L_{m,1}, \cdots, L_{m,n_m}\}\} \end{array}$$

Syntaktisch beschreiben die beiden Schreibweisen unterschiedliche Dinge. Bei der konjunktiven Normalform handelt es sich um eine Konjunktion einzelner Disjunktionen, wohingegen es sich bei der Klauselform um eine Menge von Mengen handelt.

Semantisch stellt die Klauselform jedoch denselben Zusammenhang dar, wie die konjunktive Normalform. Aus jeder Klausel, also aus jeder Menge von Literalen, muss mindestens ein Literal wahr sein, damit die gesamte Klauselmenge wahr ist. Gegenüber der Konjunktiven Normalform wird die Klauselform oft bevorzugt, da sie eine kürzere und übersichtlichere Syntax hat.

2.3 Resolutionsverfahren

Theorembeweiser haben das Ziel, aus einer gegebenen Wissensbasis eine Aussage zu beweisen. Das Wissen sei formal durch eine Klauselmenge $N = \{C_1, C_2, ..., C_n\}$ gegeben, die Aussage sei eine Klausel D. Um zu beweisen, dass D aus N folgt gibt es mehrere Ansätze:

- Man weißt nach, dass aus der Klauselmenge N die Klausel D ableitbar ist.
- Man weißt nach, dass die Klauselmenge $N \cup \{\neg D\}$ unerfüllbar ist, das heißt, dass die leere Klausel $\{\}$ ableitbar ist. Das Zeichen $\neg D$ entspricht der Negierung von D.

Beim Resolutionsverfahren wird der zweite Ansatz gewählt, das heißt die negierte Form des zu beweisenden Ziels wird zur Klauselmenge hinzugefügt. Danach werden mittels Resolution jeweils zweier gegebenen Klauseln neue Klausel abgeleitet. Die neuen Klauseln werden ebenfalls zur Klauselmenge hinzugefügt. Sobald die leere Klausel abgeleitet wurde, terminiert das Programm und die Hypothese wurde bewiesen. Das Resolutionsverfahren ist vollständig fürs Widerlegen, das heißt aus jeder widersprüchlichen Klauselmenge lässt sich die leere Klausel ableiten. Das Resolutionsverfahren garantiert nicht, dass die leere Klausel in endlicher Zeit gefunden wird. [1]

2.3.1 Resolution in der Aussagenlogik

Gegeben seien zwei Klauseln C und D, die sich aus den Literalen L_i und K_i zusammensetzen. Jedes Literal ist entweder ein Atom A, oder ein negiertes Atom $\neg A$.

$$C = \{L_1, L_2, ..., L_n\} \quad , D = \{K_1, K_2, ..., K_m\}$$

Wenn das Literal L_1 aus Klausel D in negierter Form in D vorkommt, also $L_1 \equiv \neg K_1$, dann kann aus C und D die Resolvente E gebildet werden mit:

$$\frac{C \quad D}{E} L_1 = \frac{C \quad D}{(C \cup D) \backslash \{L_1, K_1\}} L_1 = \frac{\{L_1, L_2, ..., L_n\} \quad \{K_1, K_2, ..., K_m\}}{\{L_2, ..., L_n, K_2, ..., K_m\}} L_1$$

$$(C, D) \Rightarrow_{Res} E = (C \cup D) \backslash \{L_i, K_j\}$$

Ein konkretes Beispiel sieht folgendermaßen aus.

$$C = \{A,B,C\}, \quad D = \{\neg A, D, \neg E\} \Rightarrow_{Res} E = \{B,C,D,\neg E\}$$

Anschaulich kann die Resolution folgendermaßen erklärt werden: Wenn das Literal L_i unter einer Interpretation wahr ist, dann wird die gesamte Klausel C wahr. Sollte L_i falsch sein, dann muss mindestens eines der übrigen Literale in C wahr sein. Das gleiche gilt für K_j in der Klausel D. Da L_i und K_j komplementär sind, muss in jedem Fall eines der beiden Literalen L_i und K_j falsch sein. Damit beide Klauseln wahr werden, muss für eine der beiden Klauseln mindestens eines der übrigen Literale wahr sein. Die Resolvente E bringt diesen Zusammenhang zum Ausdruck, da sie die Literale aus C und D ohne L_i und K_j vereinigt.

2.3.2 Substitution

Nicht immer kommen in zwei Klauseln zwei Literale vor, die komplementär sind. Die beiden Klauseln $C = \{P(x), Q(x)\}, D = \{\neg P(f(a)), R(x)\}$ lassen sich beispielsweise nicht resolvieren, da $P(x)$ negiert nicht $\neg P(f(a))$ ergibt. Mit einer Substitution können die Klauseln so angepasst werden, dass sich die Resolution anwenden lässt.

$$C\sigma = \{P(f(a)), Q(f(a))\} \text{ mit } \sigma = \{x \mapsto f(a)\}$$

$$D\lambda = \{\neg P(f(a)), R(x)\} \text{ mit } \lambda = \{\}$$

Anschließend lässt sich die Resolvente bilden.

$$E = \{Q(f(a)), R(x)\}$$

2.3.3 Vollständigkeit

2.4 Set-Of-Support-Strategien

Ein Problem beim Resolutionsverfahren ist, dass aus einer endlichen Menge von Axiomen häufig unendlich viele neue Klauseln abgeleitet werden können. Dadurch gibt es keine Garantie, dass in endlicher Zeit ein Beweis gefunden wird. Selbst wenn ein Beweiser bereits einen Widerspruch herleiten konnten, bedeutet das nicht, dass jeder andere Beweiser auch einen Widerspruch finden wird.

Um die Chance, dass ein Beweis gefunden wird, zu vergrößern, wurde eine Vielzahl von Strategien entwickelt, die bei der Durchführung der Resolution angewendet werden kann. Beim gewöhnlichen Resolutionsverfahren gibt es keine vorgeschriebene Reihenfolge, in der die Klauseln abgearbeitet werden. Meist iteriert der Algorithmus der Reihe nach oder willkürlich über die Klauseln. Die Grundidee diverser Strategien ist folgende: Die Klauseln sollen vom Beweiser nicht willkürlich ausgewählt werden. Stattdessen sollen Algorithmen beurteilen, welche möglichen Schritte vielversprechend sind, um unnötige Schritte zu vermeiden. Zwei sehr intuitive Strategien sind zum Beispiel:

- Heuristiken: Aus Klauseln mit wenig Literalen ist die Wahrscheinlichkeit größer, einen Widerspruch zu finden, als aus Klauseln mit sehr vielen Literalen. Kleinere Klauseln werden deshalb priorisiert verarbeitet.

- Tautologien löschen: Klauseln, die unter jeder Interpretation wahr sind, also Tautologien, tragen nichts zur Beweisführung bei. Sie können deshalb für das gesamte Beweisverfahren ignoriert werden.

Eine Reihe komplexerer Strategien sind die Set-of-Support-Strategien. Sie wurden erstmals 1965 von Lawrence Wos et al. vorgestellt. [3]

2.4.1 Prinzip

Bei den Set-Of-Support-Strategien (SOS-Strategien) wird die Menge aller gegebenen Klauseln in zwei disjunkte Teilmengen N und S unterteilt. Die Menge N ist erfüllbar. Ein Widerspruch kann erst bei Hinzunehmen vom sogennenannten Set-Of-Support S entstehen. Nach der Einteilung wird wie gewöhnlich das Resolutionsverfahren durchgeführt. Es wird jedoch verboten, die Resolvente zweier Klauseln zu bilden, wenn beide Klauseln in N liegen. Dies hat den Vorteil, dass weniger Klauseln miteinander kombiniert werden können und es weniger mögliche Resolutionsschritte gibt. Somit soll die Chance vergrößert werden, einen Widerspruch zu finden.

Bei einer gegebenen Klauselmenge gibt es sehr viel Möglichkeiten, die Klauseln in zwei Mengen N und S einzuteilen. Da die Laufzeit der Beweissuche sehr wichtig ist, ist es nicht sinnvoll sehr komplexe Einteilungen vorzunehmen, da sonst der gesamte Performance-Vorteil durch den Mehraufwand der Einteilung wieder verloren geht. Stattdessen liegt der Fokus auf Einteilungen, die sehr einfach vorzunehmen sind. In dieser Arbeit werden drei verschiedene Einteilungen vorgestellt:

1. Alle Axiome werden zu N hinzugefügt und die Klauseln der negierten Vermutung werden zu S hinzugefügt. Die Bedingung, dass N erfüllbar ist, ist in jedem Fall gegeben, da von Axiomen angenommen wird, dass sie immer erfüllbar sind.

2. Alle Klauseln, die nur positive Literale enthalten, werden zu N hinzugefügt, die restlichen zu S. N ist erfüllbar, da alle Klauseln in N unter der Interpretation, dass alle Literale für beliebige Variablenersetzungen wahr sind, auch wahr werden. S enthält in diesem Fall Klauseln, die mindestens ein negatives Literal enthalten. Sollte die leere Klausel vorkommen, wird sie ebenfalls zu S hinzugefügt.

3. Alle Klauseln, die nur negative Literale enthalten, werden zu N hinzugefügt. Äquivalent zu 2) ist diese Klauselmenge erfüllbar, unter der Interpretation, dass alle Literale für beliebige Variablenersetzungen falsch sind.

Der Einsatz von SOS-Strategien ist nur dann sinnvoll, wenn zwei Bedingungen erfüllt sind:

- Das Resolutionsverfahren muss mit einer SOS-Strategie weiterhin vollständig fürs Widerlegen sein. Andernfalls könnte es passieren, dass der Beweiser eine unerfüllbare Klauselmenge für erfüllbar hält, da kein Widerspruch hergeleitet werden kann.
- Die Laufzeit der Beweisführung sollte mit einer SOS-Strategie im Mittel signifikant schneller sein, als ohne.

Die Vollständigkeit wird im folgenden Abschnitt erläutert. Das Laufzeitverhalten wird in Kapitel 5 analysiert.

2.4.2 Vollständigkeit

Die Resolution ist weiterhin vollständig fürs Widerlegen [3] Dass die Vollständigkeit erhalten bleibt, kann man sich anschaulich folgendermaßen vorstellen: Dadurch, dass die Klauselmenge N erfüllbar ist, kann aus ihren Klauseln kein Widerspruch hergeleitet werden. Eine Resolvente, die aus zwei Klauseln in N gebildet wird, trägt zur Suche des Widerspruchs nichts bei und wird deshalb gar nicht erst gebildet.

2.5 PyRes

Beweiser in Python, Fokus liegt nicht auf Performance sondern auf einfachem Verständnis, kann Formelmengen oder Klauselmengen aus Datei parsen und Resolution anwenden, drei mögliche Ausgänge: Beweis gefunden, kein Beweis vorhanden, Beweis nicht in der Zeit gefunden

3 Konzeption

Aufteilungsmöglichkeiten in Grundklauselmenge und SOS,

4 Implementierung

Prozedur zur Einteilung in SOS, Durchführung der Resolution, Klassenkonzept, Performance Test,

5 Validierung

Ergebnis Performance, eventuell Unit Tests, Erweiterungsmöglichkeiten, Kompatibilität mit anderen Erweiterungen (geordnete Resolution), Verständlichkeit der Implementierung,

6 Reflexion

Ziel und Sinn der Aufgabe erreicht, Errungenschaften, Kritik, Ausblick

Literatur

[1] Chin-Liang Chang und Richard Char-Tung Lee. *Symbolic logic and mechanical theorem proving*. Academic Press, 1973.

[2] John Harrison. „Floating-point verification using theorem proving“. In: *International School on Formal Methods for the Design of Computer, Communication and Software Systems*. Springer. 2006, S. 211–242.

[3] Lawrence Wos, George A Robinson und Daniel F Carson. „Efficiency and completeness of the set of support strategy in theorem proving“. In: *Journal of the ACM (JACM)* 12.4 (1965), S. 536–541.

Anhang