Studienarbeitsthemen 2021/2022

Stephan Schulz

stephan.schulz@dhbw-stuttgart.de
http://www.dhbw-stuttgart.de/~sschulz

Inhaltsverzeichnis

1 Einführung **1**
1.1 Rahmenbedingungen 2

2 Themen **3**
2.1 Geordnete Resolution für PyRes 3
2.2 Implementierung von Set-of-Support-Strategien in PyRes 4
2.3 Prämissenauswahl für PyRes 4
2.4 *Step by Step* - Detaillierte Beweise für E 5
2.5 Fehlertolerantes Parsing für Theorembeweiser in C 6

1 Einführung

Deduktionssysteme bilden logische Denkprozesse nach, um so aus bekannten Fakten gesicherte neue Erkenntnisse abzuleiten. Automatische Deduktion ist ein Teilgebiet der künstlichen Intelligenz, aber auch der theoretischen Informatik. Wir entwickeln an der DHBW Stuttgart zur Zeit zwei Beweiser.

Auf der einen Seite ist das der Theorembeweiser E [11, 10, 9], einer der derzeit leistungsstärksten Beweiser für die Prädikatenlogik erster Stufe, und den stärksten dieser Beweiser, der unter einer Open-Source/Free-Software-Lizenz steht. E nimmt regelmäßig an der *CADE ATP System Competition* teil, und ist in seiner Klasse in den letzten Jahren in der Regel nur vom Beweiser Vampire der Universität Manchester geschlagen worden. 2020 hat E zwei Kategorien des Wettbewerbs gewonnen. E wird in C entwickelt.

Unser zweiter Beweiser ist PyRes [12], ein resolutionsbasierter Beweiser in Python 3. Im Gegensatz zu E steht hier nicht maximale Performance im Vordergrund, sondern eine möglichst einfache und leicht verständliche Implementierung. Python ermöglicht einen leichten Einstieg in die Welt der symbolischen

künstlichen Intelligenz.

Konkret sind automatische Theorembeweiser Computerprogramme, die nachweisen, dass bestimmte Aussagen zwingend aus einer gegebenen formalen Beschreibung folgen. Ein wichtiges Anwendungsbeispiel ist die Verifikation von Software, z.B. der Nachweis, dass ein Fahrassistenzsystem einen PKW niemals beschleunigt, wenn der Fahrer die Bremse tritt, oder dass eine Banktransaktion immer entweder vollständig oder gar nicht abgewickelt wird. Ein anderes Anwendungsbeispiel ist die Beantwortung von komplexen Fragen über großen formalen Wissensbasen, etwas die Frage "Welche europäischen Großstädte sind von Überflutungen bedroht?" Um diese Fragen zu beantworten, muss ein Beweiser im Raum der möglichen logischen Ableitungen nach einem (oder auch mehreren) Beweisen für die Behauptung bzw. Vermutung suchen.

E und PyRes sind *saturierender* Theorembeweiser für die Prädikatenlogik erster Stufe. Ein solcher Beweiser stellt den Suchzustand als Menge von als wahr angenommenen logischen Formeln (*Klauseln*) dar, und kombiniert nach festen Regeln Formeln, die Aussagen über die selben Objekte machen, um so neues Wissen explizit herzuleiten. Dieser Prozess wird ausgeführt, bis der Beweis offensichtlich wird, oder bis der Beweisversuch aufgegeben wird. In der Praxis sucht das System nach einem *Beweis per Widerspruch*, bei dem dann der explizite Widerspruch zwischen den Axiomen und der negierten Vermutung hergeleitet wird.

Fast immer ist die Menge der ableitbaren Formeln unendlich, und die Größe der Wissensbasis wächst sehr schnell. Das erschwert die Beweissuche stark. Um diesen Effekt zu reduzieren haben moderne Kalküle ein Redundanzkonzept, mit der Formeln entweder vereinfacht oder sogar komplett gestrichen werden können. Moderne Beweiser verbringen einen großen Teil der Beweissuche mit solchen *Simplifikationen*.

Um einen Beweiser möglichst effizient zu machen gibt es zwei grundsätzliche Ansätze. Zum einen kann man die Geschwindigkeit steigern, mit der neue Formeln generiert und vereinfacht werden. Dadurch wird ein größerer Teil des Suchraums bearbeitet, was die Chance erhöht, einen Beweis zu finden. Zum anderen kann man versuchen, aus der Menge aller ableitbaren Formeln die für einen Beweisversuch besonders interessanten Formeln erkennen und so die Suche auf viel versprechende Teile des Suchraums konzentrieren. So kann man gezielt tiefer in den Suchraums vordringen und potentiell auch längere Beweise finden.

Neben diesen Kernfragen gibt es auch interessante Aufgabenstellung in der Vorverarbeitung der Beweisprobleme, der Axiomauswahl, der Bearbeitung von Hintergrundtheorien und der Beweisdarstellung.

1.1 Rahmenbedingungen

Für alle angebotenen Arbeiten gilt:

- Implementierungssprache in E selbst ist C, für PyRes Python 3. Für Programmieraufgaben im Umfeld kommen auch andere Sprachen mit Open-Source-Implementierungen in Frage. Viele Teilprojekte verwenden Python.

- E wird primär für UNIX-artige Betriebssysteme entwickelt, insbesondere Linux und macOS/OS-X. Das Versionskontrollsystem ist `git` (über GitHub).
  - E: `https://github.com/eprover/eprover`
  - PyRes: `https://github.com/eprover/PyRes`
- Wenn Code in einen der Beweiser integriert wird, so wird er damit implizit wie das System selbst lizenziert (im Moment GPL 2 und LGLP 2). Das heißt auch, dass nur Bibliotheken zum Einsatz kommen können, die ebenfalls unter geeigneten Open-Source-Lizenzen stehen. Wir bemühen uns im allgemeinen, wenig externe Abhängigkeiten in das Kernsystem einzubringen.
- Es wird angestrebt, Ergebnisse der Arbeiten auf Workshops, Konferenzen, oder in Journal-Artikeln zu veröffentlichen. Der Erfolg dieser Bemühungen hängt von der Qualität der Ergebnisse und von der Hartnäckigkeit der Autoren ab.

2 Themen

2.1 Geordnete Resolution für PyRes

Ein Problem bei der Beweissuche ist, dass aus einem gegebenen Beweiszustand in der Regel sehr viele Ableitungen möglich sind, von denen nur wenige zum Ziel führen. Eine Methode, diesen Effekt zumindest zu verringern, ist die Entwicklung von Kalkülen, die viele der möglichen Inferenzen verbieten, ohne dabei die Vollständigkeit des Systems zu opfern. PyRes implementiert z.B. bereits die *negative Literalselektion*, bei der in Klauseln mit mindestens einem negativen Literal eines der negativen Literale ausgewählt werden kann, und alle Inferenzen mit anderen Literalen dieser Klausel verboten werden.

Eine meist stärkere Einschränkung kann man mit der *geordneten Resolution* erreichen. Dabei wird eine sogenannte *Simplifikationsordnung* auf den Literalen definiert, und nur Resolutionsschritte mit (in dieser Ordnung) maximalen Literalen sind erlaubt. Im Rahmen dieser Arbeit soll geordnete Resolution auf Grundlage der *Knuth-Bendix-Ordnung* (sehr gut dargestellt in [5]) implementiert werden. Wenn die Arbeit an zwei Studenten vergeben wird, dann soll zusätzlich die *lexikographische Pfadordnung* [6] implementiert werden. In jedem Fall soll die Performance des Systems in den verschiedenen Ausprägungen evaluiert und verglichen werden.

Literatur

[12, 5, 6, 1]

Team

1-2 Studierende

2.2 Implementierung von Set-of-Support-Strategien in PyRes

Wie schon oben beschrieben, ist die Beweissuche schwierig, da in der Regel sehr viele Klauseln abgeleitet werden können, die zum Beweis nicht unbedingt notwendig sind. Statt einzuschränken, welche Inferenzen mit einer gegebenen Klausel notwendig sind, kann man auch die Menge der Klauseln beschränken, die überhaupt verwendet werden dürfen. Eine solche Variante ist die *Set-of-Support*-Strategie. Dabei wird die ursprüngliche Klauselmenge in eine sicher erfüllbare Teilmenge und das sogenannte *Set of Support* geteilt, durch das die Menge erst unerfüllbar wird. Beispiele für geeignete Supportmengen sind z.B. die Menge aller Klauseln, die nur aus negativen Literalen bestehen (dann enthalten alle anderen Klauseln mindestens ein positives Literal, also ist eine Interpretation, die alle Atome zu *wahr* auswertet, ein Modell dieser Klauselmenge), die Menge aller Klauseln, die nur aus positiven Literalen besteht, oder die Menge aller Klauseln, die aus dem negierten Beweisziel entstanden sind (dabei wird die Annahme gemacht, dass die eigentlichen Axiome nicht widersprüchlich sind).

Set-of-Support-Resolution erlaubt nur Ableitungen, bei denen mindestens ein Elternteil aus der Supportmenge stammt. Dabei neu entstehende Klauseln werden auch in die Supportmenge aufgenommen.

Wenn die Arbeit an zwei Studierende vergeben wird, dann sollen mindestens die drei genannten Strategien implementiert und verglichen werden, bei einem Studierenden mindestens eine. In jedem Fall soll die Performance des Systems mit und ohne SoS in den verschiedenen Ausprägungen evaluiert und verglichen werden.

Literatur

[12, 13, 2]

Team

1-2 Studierende

2.3 Prämissenauswahl für PyRes

Ein Problem bei großen Beweisaufgaben ist oft, dass nur ein Bruchteil des spezifizieren Wissens für ein konkretes Beweisziel gebraucht wird. Da ein automatischer Beweiser Klauseln in der Regel relativ blind und systematisch kombiniert, erzeugt er mit den irrelevanten Fakten trotzdem immer weitere Ableitungen, die nicht zum Beweis beitragen, trotzdem aber den Suchraum und den Ressourcenbedarf (Zeit, Rechenleistung) erhöhen. Eine Methode, diesem Problem

zu begegnen, ist die Prämissenauswahl. Dabei wird an Hand des Beweisziels eine kleine Teilmenge von (hoffentlich) relevanten Axiomen aus der Spezifikation gezogen.

In dieser Arbeit sollen von jedem Studierenden mindestens eine Prämissenauswahlmethode entwickelt und evaluiert werden - entweder aus den aus der Literatur bekannten Verfahren (SInE, Alternating Path), oder auch selbst vorgeschlagene Verfahren.

Literatur

[12, 7, 4]

Team

1-2 Studierende

2.4 *Step by Step* - Detaillierte Beweise für E

Für viele Anwendungen von Theorembeweisern werden explizite Beweisobjekte gefordert. Es reicht also nicht, wenn der Beweiser intern ein Theorem nachweisen kann, der Nutzer erwartet auch eine nachvollziehbare Begründung, *warum* das Theorem gültig ist. Solche Beweise können mit unabhängigen Werkzeugen oder manuell verifiziert werden, und die Struktur des Beweises und interessante Zwischenergebnisse geben einen Einblick in die Anwendungsdomäne und das Suchverhalten.

E kann interne Beweisobjekte mit minimalem Aufwand erzeugen und diese als Folge von begründeten Schritten ausgeben. Allerdings sind die einzelnen Beweisschritte dieses Beweisobjektes in der Regel weder eindeutig noch atomar. Sie fassen typischerweise eine generierende Inferenz und alle darauf folgenden Vereinfachungen zusammen. Damit sind sie für manche Anwendungen nicht optimal geeignet.

Im Rahmen dieser Studienarbeit soll der vorhandene Mechanismus für die Erzeugung von Beweisobjekten verfeinert werden. Dabei stehen folgende Teilaufgaben an (die nicht alle von der selben Person gemacht werden müssen):

1. Genauere Beschreibung der Ableitungen im Beweisobjekt, insbesondere Buchführung darüber, an welchem Positionen in Formeln und Termen Inferenzregeln angewendet werden. In einem ersten Schritt sollte das mindestens die generierenden Inferenzregeln abdecken, idealerweise alle Regeln.

2. Erstellung einer Bibliothek und eines Programms, die die Beweise in Einzelschritte zerlegen und alle Zwischenergebnisse explizit macht

3. Update eines Beweisprüfers, der einzelne Beweisschritte mit anderen Theorembeweisern nachvollzieht

Die Arbeit eignet sich für zwei Studenten. Wenn Sie von einem einzelnen Studenten bearbeitet wird, sollten mindesten Punkt 1 und die Bibliothek von Punkt 2 erstellt werden.

Literatur

[10, 9, 8, 11]

Team

1–2 Studierende

2.5 Fehlertolerantes Parsing für Theorembeweiser in C

Automatische Theorembeweiser werden traditionell als Kommandozeilenprogramme eingesetzt, und die Fehlerbehandlung beim Parsing beschränkt sich auf eine Fehlermeldung und den Programmabbruch. Das ist für Anwendungen wie z.B. einen Deduktionsserver [3] nicht geeignet. Ein solcher Server akzeptiert verschiedene Anfragen und versucht, dieses zu beantworten. Dabei muss er immer wieder neue logische Formeln (als Anfrage oder als Ergänzung der Wissensbasis) einlesen, und es ist nicht akzeptabel, wenn jeder Syntaxfehler zu einem Abbruch oder Neustart des Servers führt.

Im Rahmen dieser Studienarbeit soll der vorhandene Parser des Beweisers E so umgebaut werden, dass er auf Fehler in der Eingabe nicht mit Abbruch reagiert, sondern den Parse-Vorgang sauber beendet, den Fehler an den aufrufenden Code meldet, und so eine Fehlerbehandlung auch für interaktive Programme ermöglicht. Idealerweise ermöglicht der so verbesserte Parser auch das kontrollierte Wiederaufsetzen des Parse-Vorgangs nach einem Fehler, so dass auch spätere Fehler in der Eingabe mindestens erkannt und gemeldet werden können.

Der verbessere Parser sollte ähnlich performant sein, und die selbe Klasse von logischen Problemen lesen können, wie die aktuelle Lösung.

Literatur

[9, 11, 3]

Team

1 Studentin bzw. 1 Student

Literatur

[1] Leo Bachmair and Harald Ganzinger. Resolution Theorem Proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 2, pages 19–99. Elsevier, 2001.

[2] C. Chang and R.C. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Computer Science and Applied Mathematics. Academic Press, 1973.

[3] Mohamed Bassem Hasona and Stephan Schulz. Deduction as a service. In Pascal Fontaine, Stephan Schulz, and Josef Urban, editors, *Proc. of the 5th PAAR, Coimbra, Portugal*, CEUR Workshop Proceedings, 2016.

[4] Kryštof Hoder and Andrei Voronkov. Sine Qua Non for Large Theory Reasoning. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Proc. of the 23rd CADE, Wroclav, Poland*, volume 6803 of *LNAI*, pages 299–314. Springer, 2011.

[5] Bernd Löchner. Things to Know when Implementing KBO. *Journal of Automated Reasoning*, 36(4):289–310, 2006.

[6] Bernd Löchner. Things to Know When Implementing LPO. *International Journal on Artificial Intelligence Tools*, 15(1):53–80, 2006.

[7] David Plaisted. Properties and extensions of alternating path relevance - I. arXiv preprint 1905.08842, 2019. `https://arxiv.org/abs/1905.08842`.

[8] S. Schulz. Analyse und Transformation von Gleichheitsbeweisen. Projektarbeit in Informatik, Fachbereich Informatik, Universität Kaiserslautern, 1993. (German Language).

[9] Stephan Schulz. E – A Brainiac Theorem Prover. *Journal of AI Communications*, 15(2/3):111–126, 2002.

[10] Stephan Schulz. System Description: E 1.8. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Proc. of the 19th LPAR, Stellenbosch*, volume 8312 of *LNCS*, pages 735–743. Springer, 2013.

[11] Stephan Schulz, Simon Cruanes, and Petar Vukmirović. Faster, higher, stronger: E 2.3. In Pascal Fontaine, editor, *Proc. of the 27th CADE, Natal, Brasil*, number 11716 in LNAI, pages 495–507. Springer, 2019.

[12] Stephan Schulz and Adam Pease. Teaching automated theorem proving by example: PyRes 1.2 (system description). In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Proc. of the 10th IJCAR, Paris*, volume 12167 of *LNCS*, pages 158–166. Springer, 2020.

[13] L. Wos, G.A. Robinson, and D.F. Carson. Efficiency and Completeness of the Set of Support Strategy in Theorem Proving. *Journal of the ACM*, 12(4):536–541, 1965.