Rein funktionelle Implementierung eines CDCL SAT-Solvers

Bachelorarbeit

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Das Boolean Satisfiability Problem (SAT), auf Deutsch Erfüllbarkeitsproblem der Aussagenlogik, ist das erste Problem, welches durch den „Satz von Cook“ im Jahr 1971 als NP-vollständig bewiesen wurde [3]. Das SAT-Problem ist in der Praxis sehr wichtig, weshalb trotz NP-Vollständigkeit viel Forschung in diesem Bereich betrieben wurde. Mitunter wurden mehrere SAT-Solver entwickelt, wie z.B. zChaff und MiniSAT.

Die Entwicklung von effizienten SAT-Solvern ist seit dem Jahr 2002 stetig angestiegen. Jedoch wird für die Programmierung dieser Solver hauptsächlich eine objektorientierte Programmiersprachen verwendet. Im Gegensatz dazu ist der Anteil von SAT-Solvern, welche komplett in einer rein funktionalen Programmiersprache geschrieben wurden, sehr gering.

Im Rahmen dieser Bachelorarbeit wird der Frage nachgegangen, wie eine mögliche Umsetzung eines „Conflict-driven clause learning“ (CDCL) SAT-Solvers in einer rein funktionalen Programmiersprache aussehen könnte. Das Ziel dieser Arbeit ist eine Implementierung eines CDCL SAT-Solvers in Haskell und mögliche Vorschläge wie deren Implementierung verbessert werden kann.

Anhand einer Literaturarbeit werden bestehende Algorithmen untersucht, wofür verschiedene Arbeiten über CDCL und näherstehende Arbeiten betrachtet werden. Die Literaturarbeit wurde gewählt, um bestehende Erforschungen in diesem Fachbereich zu erhalten.

In dieser Arbeit werden zuerst die verschiedenen Programmierparadigmen und die verwendete Programmiersprache Haskell in Kapitel 2 erläutert. Daraufhin werden SAT und verschiedene SAT-Solver in Kapitel 3 vorgestellt, wobei auch ein Einblick in die SAT-Competitions gewährt wird. Danach wird der Unterschied zwischen dem CDCL-Algorithmus und dem Davis-Putnam-Logemann-Loveland-Algorithmus (DPLL) in Kapitel 4 erklärt. Das darauffolgenden Kapitel 5 beinhaltet eine Beschrei-

bung für eine Umsetzung eines CDCL SAT-Solvers. Ein Vergleich der Implementierung mit anderen CDCL SAT-Solvern wird in Kapitel 6 durgeführt. Im letzten Teil der Arbeit (Kapitel 7) wird ein Fazit über die gewonnenen Erkenntnisse gezogen als auch ein Ausblick über Verbesserungsmöglichkeiten für eine bessere Effizienz des SAT-Solvers gegeben.

2 Programmierparadigmen, funktionale Programmierung und Haskell

In diesem Abschnitt werden die zwei verschiedenen Programmierparadigmen (imperative und deklarative Programmierung), funktionale Programmierung und die Programmiersprache Haskell kurz vorgestellt.

2.1 Imperative Programmierung

Folgende Definition von der Universität Passau für Imperative Programmierung wurde wörtlich übernommen: „Imperative Programme beschreiben Programmabläufe durch Operationen auf Zuständen“ [6]. Dies bedeutet dass der Ablauf des Programmes durch die Reihenfolge der Befehle maßgeblich für das erwartete Ergebnis ist. Sprachen, die zu diesem Paradigma gehören, sind z.B. Java, C++ oder C.
Die imperative Programmierung kann in weitere Paradigmen unterteilt werden, wie z.B. strukturierte Programmierung, objektoriertierte Programmierung, modulare Programmierung [1].

2.2 Deklarative Programmierung

Weiterführende Definition für Deklarative Programmierung wurde wortgetreu von der Passauer Universität übernommen: „Deklarative Programme beschreiben Berechnungen durch eine Ein-/Ausgaberelation. Der Kontrollfluß ist dem Programmierer nicht explizit zugänglich; der Ablauf der Berechnung kann aber trotzdem durch den Programmaufbau beeinflußt werden.“ [6] Sinngemäß bedeutet dies, dass nicht der Weg zum Ergebnis das Entscheidende ist, sondern das Problem und Ergebnis selbst.
Die deklarative Programmierung wird wie die imperative Programmierung auch in verschiedene Unterkategorien eingeteilt. Diese sind z.B. logische Programmierung,

Constraint Programmierung und funktionale Programmierung. Prolog, Lisp und SQL sind beispielhafte Programmiersprachen, die zu diesem Paradigma gehören. [1].

2.3 Funktionale Programmierung

Wie im vorherigen Abschnitt erwähnt wurde, ist die funktionale Programmierung ein Teil von der deklarativen Programmierparadigmen. Ein Programm in diesem Schema besteht hauptsächlich aus einer Zusammensetzung von Funktionen [6], verwendet unveränderliche Daten und vermeidet Zustandsänderungen.
Die Programmierung mit funktionalen Sprachen bringt mehrere Vorteile. Beispiele für solche Vorteile sind „Lazy Evaluation“, frei von Seiteneffekten und Effizienz [11]. [1]

2.4 Haskell

Haskell ist eine der weitverbreitesten funktionalen Sprachen, die 1988 ihren Namen im „Yale Meeting“ erhielt. Der Name kam vom Mathematiker Haskell B. Curry, wessen Arbeit einer der Anstöße zur Entwicklung von Haskell geführt hat. Im Jahr 1987 begann der Haskell Design Prozess in der „Functional Programming and Computer Architecture Conference“ (FPCA) und am 01. April 1990 wurde der „Haskell version 1.0 report“ veröffentlicht. Über die Jahre entwickelte sich die Sprache weiter und es wurde im Februar 1999 der „Haskell 98 Report“ veröffentlicht, wobei eine Revision im Dezember 2002 veröffentlicht wurde [5]. Mit der Veröffentlichung des „Haskell 2010 Language Report“ wurde eine wichtige Änderung für neue Revisionen beschlossen. Jedes Jahr sollte mindenstens eine neue Revision veröffentlicht werden, die eine kleine Anzahl an Änderungen und Erweiterungen beinhaltet [9]. [2]

[1] Soll ich Lazy Evaluation, etc kurz erklären? Oder wieso es Seiteneffektfrei usw. ist
[2] Reicht des für Haskell Abschnitt? Oder sollte noch irgendeine Info in den Text rein?

3 SAT

Dieses Kapitel gibt eine kurze Einführung zum Thema SAT und einen Einblick in die SAT-Competitions. Des Weiteren werden die bekannten SAT-Solver zChaff und MiniSAT vorgestellt.

3.1 SAT-Problem

Die SAT Assoziation definiert das SAT-Problem als ein Problem, in dem bestimmt werden muss, ob es für ein Problem Wertzuweisungen existieren, die dieses dann zu 1 evaluiert [13]. Stephen A. Cook hat die NP-Vollständigkeit vom SAT-Problem 1971 in seiner Publikation bewiesen [3], welche 1973 nochmals von Leonid A. Levin nachgewiesen wurde [7]. Deshalb wird der Beweis oft auch „Satz von Cook-Levin“ genannt.
Das SAT-Problem findet sich in vielen industriellen Bereichen wieder, die sich mit Informatik beschäftigen. Darunter zählen z.B. „Bounded Model Checking“ (BMC), „Artificial Intelligence“ (AI) und Theorembeweise [13].

3.2 SAT-Competition

Die internationale SAT-Competition findet seit dem Jahr 2002 statt. Der Wettbewerb wurde eingeführt, um neue SAT-Solver vorzustellen und Benchmarks zu finden, die nicht einfach zu lösen sind. Dabei werden die Solver auch mit SAT-Solvern verglichen, die den Stand der Technik darstellen. Unter anderem existieren im Wettbewerb auch unterschiedliche Disziplinen, in denen sich die Solver messen können [14].

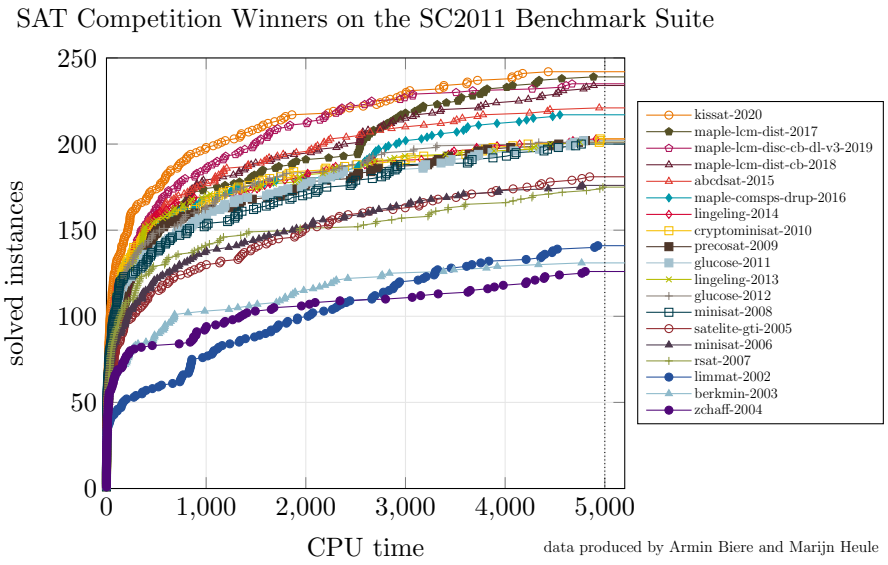

Abbildung 3.1: Benchmark 2011

Bildquelle: http://fmv.jku.at/kissat/winners-2011.pdf

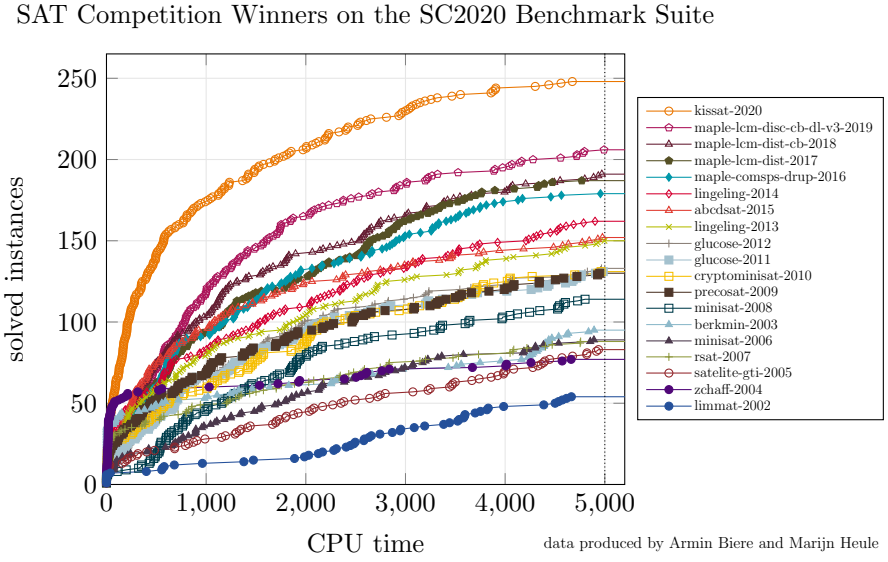

Abbildung 3.2: Benchmark 2020

Bildquelle: http://fmv.jku.at/kissat/winners-2020.pdf

Die aufgelisteten Solver in der Legende stellen die Gewinner von 2002 bis 2020 dar. Die Graphen zeigen die gelösten SAT-Probleme innerhalb einer bestimmten Zeit an. Es wird ersichtlich, dass die Performance zwischen den Solvern in der Benchmark von 2020 viel weiter auseinander liegen als in der Benchmark von 2011. Der Grund hierfür liegt darin, dass die Probleme schwieriger geworden sind und dadurch den Unterschied im Leistungsgrad ersichtlich machen [2].

3.3 SAT-Solver

Es existieren viele SAT-Solver, die gute Leistungen vollbringen beim Lösen von SAT-Problemen. In den folgenden Abschnitten werden zwei Solver vorgestellt, welche an einer SAT-Competition teilgenommen und diese auch zu ihrer Zeit gewonnen haben. Die zwei SAT-Solver, die vorgestellt werden, sind zChaff und MiniSAT. Einige Konzepte, die in den Solvern angewendet werden, werden für die Implentierung teilweise verwendet.

3.3.1 zChaff

Der SAT-Solver zChaff basiert auf den Chaff-Algorithmus und wurde an der Princeton Universität entwickelt und veröffentlicht. Der Chaff-Solver verwendet einen optimierten „Boolean Constrainst Propagation"-Algorithmus (BCP) und „Variable State Independent Decaying Sum" (VSIDS) für die Entscheidungsheuristik. Des Weiteren wendet zChaff „Restarts" und „Clause Deletion" an [10].
zChaff hat 2004 im „Industrial Track" die „ALL (SAT+UNSAT)" und „UNSAT" Kategorien gewonnen [14].

3.3.2 MiniSAT

MiniSAT wurde von Niklas Eén und Niklas Sörensson an der Chalmers University of Technology entwickelt. Hintergrund für die Entwicklung des Solvers ist die Veröffentlichung eines zugänglichen, minimalistischen CDCL-SAT-Solvers, der Watched Literals und „Dynamic Variable Ordering" (VSIDS) verwendet [4].
MiniSAT gewann den ersten Platz im SAT-Race 2006 [16].

4 Algorithmen

In diesem Kapitel wird der DPLL-Algorithmus und CDCL-Algorithmus vorgestellt. Anhand einer Literaturarbeit werden die Erweiterung erläutert, die das CDCL-Verfahren performanter im Gegensatz zum DPLL machen.

4.1 DPLL-Algorithmus

4.2 CDCL-Algorithmus

5 Implementierung des SAT-Solvers

6 Vergleich der Implementierung

7 Fazit

Literatur

[1] Stephan Augsten. *Was ist ein Programmierparadigma?* de. URL: https://www.dev-insider.de/was-ist-ein-programmierparadigma-a-864056/ (besucht am 22.06.2021).

[2] Armin Biere. *Kissat SAT Solver*. URL: http://fmv.jku.at/kissat/ (besucht am 03.07.2021).

[3] Stephen A. Cook. "The complexity of theorem-proving procedures". In: *In Stoc*. ACM, 1971, S. 151–158.

[4] Niklas Eén und Niklas Sörensson. *An Extensible SAT-solver*. en. 2003. URL: http://minisat.se/downloads/MiniSat.pdf (besucht am 02.07.2021).

[5] Paul Hudak u. a. "A history of Haskell: being lazy with class". en. In: *Proceedings of the third ACM SIGPLAN conference on History of programming languages*. San Diego California: ACM, Juni 2007, S. 3–5. ISBN: 978-1-59593-766-7. DOI: 10.1145/1238844.1238856. URL: https://dl.acm.org/doi/10.1145/1238844.1238856 (besucht am 30.06.2021).

[6] Christian Lengauer. *Was ist funktionale Programmierung?* URL: https://www.infosun.fim.uni-passau.de/cl/lehre/funcprog05/wasistfp.html (besucht am 22.06.2021).

[7] L A Levin. "Universal Sequential Search Problems". ru. In: *Probl. Peredachi Inf.* 9.3 (1973), S. 115–116.

[8] Jia Hui Liang u. a. "Understanding VSIDS Branching Heuristics in Conflict-Driven Clause-Learning SAT Solvers". In: *Hardware and Software: Verification and Testing*. Hrsg. von Nir Piterman. Cham: Springer International Publishing, 2015, S. 225–241. ISBN: 978-3-319-26287-1.

[9] Simon Marlow. *Haskell 2010 Language Report*. Juli 2021. URL: https://www.haskell.org/definition/haskell2010.pdf (besucht am 01.07.2021).

[10] Matthew W Moskewicz u. a. "Chaff: Engineering an Efficient SAT Solver". en. In: (), S. 6. URL: http://www.princeton.edu/~chaff/publication/DAC2001v56.pdf (besucht am 23.02.2021).

[11] Stefania Loredana Nita und Marius Mihailescu. "Functional Programming". In: *Haskell Quick Syntax Reference: A Pocket Guide to the Language, APIs, and Library*. Hrsg. von Stefania Loredana Nita und Marius Mihailescu. Berkeley, CA: Apress, 2019, S. 1–3. ISBN: 978-1-4842-4507-1. DOI: 10.1007/978-1-4842-4507-1_1. URL: https://doi.org/10.1007/978-1-4842-4507-1_1.

[12] Stefania Loredana Nita und Marius Mihailescu. *Haskell Quick Syntax Reference: A Pocket Guide to the Language, APIs, and Library*. en. Berkeley, CA: Apress, 2019. ISBN: 978-1-4842-4506-4 978-1-4842-4507-1. DOI: 10.1007/978-1-4842-4507-1. URL: http://link.springer.com/10.1007/978-1-4842-4507-1 (besucht am 24. 06. 2021).

[13] Hrsg: Satisfiability: Application and Theory (SAT) e.V. *SAT Basics*. URL: http://satassociation.org/articles/sat.pdf (besucht am 05. 07. 2021).

[14] Hrsg: Satisfiability: Application and Theory (SAT) e.V. *SAT Competitions*. URL: http://satcompetition.org/ (besucht am 03. 07. 2021).

[15] Uwe Schöning. "Das SAT-Problem". In: *Informatik-Spektrum* 33.5 (Okt. 2010), S. 479–483. ISSN: 1432-122X. DOI: 10.1007/s00287-010-0459-x. URL: https://doi.org/10.1007/s00287-010-0459-x.

[16] Carsten Sinz. *SAT-Race 2006*. URL: http://fmv.jku.at/sat-race-2006/results.html (besucht am 06. 07. 2021).