Theory of Hybrid Systems Informatik 2

RWTH AACHEN UNIVERSITY

Diese Arbeit wurde vorgelegt am LuFG Theorie hybrider Systeme

BACHELORARBEIT

SMT-BASIERTE LÖSUNG REELL-ALGEBRAISCHER PROBLEME MITTELS LINEARISIERUNG

Denis Kuksaus

Prüfer:
Prof. Dr. Erika Ábrahám
Prof. Dr. Jürgen Giesl

Zusätzlicher Berater:
Gereon Kremer

Aachen, 30.09.2019

Zusammenfassung

Erfüllbarkeit Modulo Theorien (SMT) ist das Entscheidungsproblem, ob eine logische Formel bezüglich einer Theorie erster Ordnung erfüllbar ist. SMT-Solver werden oft bei der Verifikation von Programmen oder bei der Analyse durch symbolischen Ausführung verwendet. Wenn SMT-Probleme nicht linear sind, sind sie rechenintensiv. Deshalb wurde eine Methode [1] entwickelt, um nichtlineare Probleme zu linearisieren, die in ähnlicher Weise auch in SMT-RAT implementiert wurde [2]. In dieser Bachelorarbeit wurde die Implementierung um neue Funktionen erweitert, um die Auswirkungen auf Laufzeit und Anzahl der gelösten Probleme zu untersuchen.
Insgesamt konnte die beste Anpassung des Moduls die Anzahl der gelösten Instanzen der verwendeten Tests um 50% steigern und dabei auch die durchschnittliche Laufzeit pro Probleminstanz senken.

Inhaltsverzeichnis

Kapitel 1

Einleitung

In der heutigen Zeit wird die Erfüllbarkeitsprüfung in vielen Bereichen eingesetzt. Ein Beispiel dafür ist der Einsatz von formalen Methoden zur Modellierung oder Überprüfung von Systemen. Zur Kodierung der Probleme wird häufig mathematische Logik eingesetzt. Durch SMT-Solver soll dann die Erfüllbarkeit von diesen kodierten Problemen automatisiert überprüft werden. Reale Probleme sind dabei oft arithmetische Probleme. Wenn sie zusätzlich auch nichtlinear sind, so ist die Erfüllbarkeitsüberprüfung aufwendig. Deshalb wurde in [3] und der Publikation [1] ein Verfahren entwickelt, das nichtlineare Probleme in lineare Probleme transformiert, um Methoden zur Lösung von linearen Problemen zu verwenden. Methoden zum Lösen von linearen SMT-Problemen sind weniger rechenintensiv als Methoden zum Lösen von nichtlinearen Problemen. Aus der linearen Lösung werden dann Rückschlüsse auf die Lösung des nichtlinearen Problems geschlossen. In SMT-RAT wurde diese Methode in etwas abgewandelter Form im Rahmen der Arbeit [2] implementiert. Die vorliegende Arbeit versucht das in SMT-RAT implementierte Modul, das diesen Ansatz umsetzt, zu verbessern.
In Kapitel 2 werden die theoretischen Grundlagen für diese Arbeit dargestellt. Anschließend werden in Kapitel 3 die in dieser Arbeit implementierten Methoden vorgestellt, die die Performanz des Moduls verbessern sollen. In Kapitel 4 werden die die Versuchsergebnisse evaluiert und diskutiert. Zum Schluss wird die Arbeit in Kapitel 5 kurz zusammengefasst, um anschließend einen Ausblick auf zukünftige weitere Möglichkeiten zur Verbesserung der Performanz des Moduls zu geben.

Kapitel 2

Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen für diese Arbeit erläutert. Zuerst wird das Erfüllbarkeitskeitsproblem definiert, um es danach in den Kontext von Erfüllbarkeit Modulo Theorien zu setzen. Anschließend wird die Reelle Arithmetik als Theorie beschrieben, da die in dieser Arbeit zu betrachtenden SMT-Probleme Formeln der nichtlinearen reellen Arithmetik beinhalten. Zum Abschluss dieses Kapitels wird ein Lösungsverfahren für diese Art von SMT-Problemen vorgestellt, das auf der Linearisierung von nichtlinearen Constraints beruht, um dann Methoden zum Lösen von linearen SMT-Problemen zu verwenden.

2.1 Das Erfüllbarkeitsproblem

Das *Erfüllbarkeitsproblem* (SAT) [4] beschreibt das Problem zu entscheiden, ob eine aussagenlogische Formel erfüllbar ist. Eine *aussagenlogische Formel* besteht aus einer Menge von booleschen Aussagenvariablen, die mit einer Menge von aussagenlogischen Junktoren (z.B. Negation $\neg$, Konjunktion $\wedge$, Disjunktion $\vee$, Implikation $\rightarrow$, ...) miteinander verknüpft werden. Man nennt eine aussagenlogische Formel *erfüllbar*, wenn es eine Zuordnung der Variablen zu den booleschen Werten Null und Eins gibt, sodass die Formel zur booleschen Eins evaluiert werden kann. Falls keine solche Interpretation der Variablen existiert, so wird die Formel *unerfüllbar* genannt.
Eine bestimmte Art von Algorithmen, die das Erfüllbarkeitsproblem der Aussagenlogik lösen, wird in Form von SAT-Solvern implementiert. Die meisten SAT-Solver akzeptieren nur Formeln in konjunktiver Normalform (KNF) [5]. Formeln in KNF bestehen aus Konjunktionen von Disjunktionstermen.

Beispiel 2.1.1. *Beispiele für SAT:*

- $\neg X_1 \wedge X_2 \wedge (X_2 \rightarrow X_1)$ *ist unerfüllbar*
- $(X_1 \vee X_2 \vee X_3) \wedge (X_1 \vee X_2 \vee \neg X_3) \wedge (\neg X_1 \vee X_2 \vee X_3) \wedge (\neg X_1 \vee \neg X_2 \vee X_3)$
  ist in KNF und erfüllbar mit der Belegung:
  $X_1 \mapsto 1$, $X_2 \mapsto 0$, $X_3 \mapsto 1$

2.2 Erfüllbarkeit Modulo Theorien

Für eine Theorie T der Prädikatenlogik erster Ordnung beschreibt das Problem der *Erfüllbarkeit Modulo Theorien* (SMT) [6], das Entscheidungsproblem, ob $T \cup \Phi$ erfüllbar ist, für eine Menge von prädikatenlogischen Formeln Φ.
Eine *Theorie* besteht aus einer Signatur und einer Menge an Axiomen. Die *Axiome* sind Formeln in der Prädikatenlogik ohne freie Variablen, auch *Sätze* genannt. Eine *Signatur* besteht aus Konstanten-, Funktions- und Prädikatensymbolen. Die Formelmenge Φ wird implizit als Konjunktion der enthaltenen Formeln interpretiert. Im Gegensatz zu SAT nehmen die Variablen in SMT nicht nur boolesche Werte an, sondern sie können beliebige Werte in einem festgelegten Wertebereich annehmen. Als Wertebereich kann man beispielsweise die reellen oder die ganzen Zahlen festlegen.
Die meisten SMT-Solver verfolgen einen *lazy* Ansatz, bei dem ein *SAT-Solver* mit einem *Theorie-Solver* zusammenarbeitet [7]. Bei diesem Ansatz überprüft zunächst der SAT-Solver, ob das *boolesche Skelett* der zu untersuchenden Formel erfüllbar ist. Das boolesche Skelett ist eine Abstraktion der Constraints in der Formel auf neue boolesche Variablen. Ist dieses bereits unerfüllbar, so kann die Formel in keiner Theorie erfüllbar sein. Sollte der SAT-Solver eine erfüllende Belegung für das boolesche Skelett finden, wird ein Theorie-Solver eingesetzt. Der Theorie-Solver überprüft dann, ob das von dem SAT-Solver gefundene Modell konsistent in der Theorie ist. Falls das gefundene Modell konsistent in der betrachteten Theorie ist, so ist die Formelmenge erfüllbar in der Theorie. Im anderen Fall hat der Theorie-Solver Inkonsistenzen entdeckt. Der Theorie-Solver erweitert dann die Formelmenge um weitere Constraints, die die gefundenen Inkonsistenzen und häufig auch andere „ähnliche“ Inkonsistenzen beim nächsten Durchlauf ausschließen. Danach sucht der SAT-Solver nach einem neuen Modell.
Dieser Ablauf wiederholt sich so oft, bis entweder der SAT-Solver ein Modell findet, das konsistent mit der zugrunde liegenden Theorie ist, oder keine weiteren booleschen Modelle mehr existieren. Im ersten Fall ist das SMT-Problem erfüllbar und im letzteren Fall ist es unerfüllbar.

Beispiel 2.2.1. *Beispiel für eine quantorenfreie Formel in linearer ganzzahliger Arithmetik (QFLIA):*

- $y \geqslant 1 \wedge (x < 0 \vee y < 1) \wedge \neg(y < 1)$

Boolesches Skelett:

- $A \wedge (B \vee C) \wedge \neg C$

Modell des booleschen Skeletts:

- A, B, $\neg C$

Erfüllende Variablenbelegung in QFLIA:

- $x = -1$, $y = 2$

Als SMT-Solver wird in dieser Arbeit SMT-RAT verwendet. SMT-RAT [8] ist eine quelloffene C++ Software zum Lösen von SMT-Problemen. Es können unter anderem quantorenfreie Formeln in Reell- und Ganzzahlarithmetik gelöst werden, mit Fokus auf nichtlinearer Arithmetik. Der Name SMT-RAT steht dabei für die Abkürzung von *Satisfiability Modulo Theories Real Arithmetic Toolbox*. Die Software ist modular aufgebaut, sodass verschiedene Methoden beliebig miteinander kombiniert werden können.

2.3 Reelle Arithmetik

Reelle Arithmetik beschreibt die Theorie erster Ordnung über den reellen Zahlen mit den Konstanten Null und Eins, den Funktionen Addition und Multiplikation und dem Prädikat Kleiner-Relation ($<$). Reell-arithmetische Formeln können induktiv wie folgt definiert werden, wobei X eine reellwertige Variable ist und a ein konstanter Koeffizient aus einem Ring R ist [9]. Insbesondere werden hier die reellen Zahlen $\mathbb{R}$ als Ring R verwendet.

$$\begin{aligned}
&\text{Polynome:} && p := a \mid X \mid p+p \mid p-p \mid p \cdot p \\
&\text{Constraints:} && c := p < p \\
&\text{Formeln:} && \varphi := c \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists x \varphi
\end{aligned}$$

Weitere Relationen (z.B. $=, \neq, \leqslant, ...$) und boolesche Funktionen (z.B. $\vee, \leftrightarrow, ...$) können aus den oben genannten Relationen und Funktionen abgeleitet werden.
Ein Polynom ist in *Normalform*, wenn es in der Form

$$\sum_{i \in [1,m]} a_i X_1^{e_{i,1}} \dots X_n^{e_{i,n}}$$

vorliegt, mit Exponenten $e_{i,j} \in \mathbb{N}_0$, Koeffizienten $a_i \in R \setminus \{0\}$ und Variablen X_j, für $i \in [1,m]$ und $j \in [i,n]$.
Als *Monom* bezeichnet man einen Ausdruck der Form $X_1^{e_{i,1}} \dots X_n^{e_{i,n}}$ und als *Term* bezeichnet man ein Monom mit seinem zugehörigen Koeffizenten, also $a_i X_1^{e_{i,1}} \dots X_n^{e_{i,n}}$, für ein festes $i \in [1,m]$. Bei der Normalform müssen die Monome paarweise verschieden sein.
Der *Grad* eines Monoms $X_1^{e_1} X_2^{e_2} \dots X_n^{e_n}$ bezeichnet die Summe der Exponenten aller auftretenden Variablen, also $\sum_{i \in [1,n]} e_i$. Unter dem Grad eines Polynomes p versteht man den maximalen Grad unter allen in p auftretenden Monomen.
In dieser Arbeit werden insbesondere quantorenfreie Formeln in nichtlinearer Arithmetik (QFNRA) betrachtet, wie sie in [10] unter dem Namen QF_NRA beschrieben werden. Nichtlinearität bedeutet, dass der Grad von mindestens einem auftretenden Polynom mindestens zwei beträgt. Das heißt, dass mindestens ein Polynom das Produkt von zwei oder mehr Variablen enthalten muss.

Beispiel 2.3.1. *Beispiel für eine QFNRA-Formel:*

$$\varphi := \underbrace{(\overbrace{7X^2Y^3Z - 2X^5Y + XYZ^2}^{Polynom\ vom\ Grad\ 6} < 0)}_{Constraint} \wedge \neg(XY > 0)$$

Es gibt verschiedene Lösungsansätze für SMT-Probleme bezüglich QFNRA, die im Folgenden genannt werden.
Eine vollständige Methode für diese Probleme ist die *zylindrisch-algebraische Zerlegung* (CAD) [11]. Sie hat aber auch einen hohen Rechenaufwand und benötigt im schlimmsten Fall doppelt exponentielle Zeit. Bei der CAD wird der Zustandsraum in eine endliche Anzahl von Zellen zerlegt, sodass in einer Zelle alle Punkte die zu prüfende Formel erfüllen oder nicht. Es genügt dann einen Vertreter jeder Zelle in die Formel aus dem Problem einzusetzen und zu überprüfen ob sie erfüllt wird.
Die Methode *Subtropical Satisfiability* (STrop) [12] ist eine schnelle aber unvollständige Methode um eine Menge von real-arithmetischen Constraints auf Erfüllbarkeit

zu überprüfen. Unvollständig bedeutet, dass nicht für alle Probleminstanzen die korrekte Lösung ermittelt werden kann. Bei STrop werden die einzelnen Constraints auf dominierende Monome untersucht. Dominierende Monome werden dabei durch Untersuchung der Exponenten gefunden. Es wird entweder eine erfüllende Belegung der Variablen gefunden, oder die Erfüllbarkeit ist unbekannt. Insbesondere kann durch diese Methode nicht die Unerfüllbarkeit einer Formel nachgewiesen werden.
Die *virtuelle Substitution* [13] ist ebenfalls ein schnelles Verfahren. Dabei werden durch Lösungen von Constraints von niedrigem Grad Variablen eliminiert. Die virtuelle Substitution ist jedoch ebenfalls unvollständig.
Interval constraint propagation (ICP) [14] ist eine effiziente und numerische aber auch unvollständige Methode. Das Verfahren terminiert immer mit dem Ergebnis möglicherweise erfüllbar oder unerfüllbar. Falls das Ergebnis möglicherweise erfüllbar lautet, so liefert ICP für jede Variable ein Intervall, in dem eine Lösung liegen könnte. Da ICP jedoch ein überapproximierendes Verfahren ist, ist es möglich, dass das untersuchte SMT-Problem trotzdem unerfüllbar ist.

2.4 Inkrementelle Linearisierung

Wie in Abschnitt 2.3 gesehen, sind SMT-Probleme bezüglich QFNRA schwierig zu lösen und nur die CAD bietet eine vollständige, jedoch auch zeitintensive Lösungsmethode. In [1] wird deshalb eine Lösungsmethode vorgestellt, mit der nichtlineare Constraints linearisiert werden können. Die Linearisierung ist für QFNRA-Formeln und für QFNTA-Formeln möglich. QFNTA-Formeln sind QFNRA-Formeln, die um transzendente Funktionen erweitert wurden. Transzendente Funktionen sind nichtalgebraische Funktionen, wie zum Beispiel Kreis- und Hyperbelfunktionen, die Exponentialfunktion oder die Logarithmusfunktion. In dieser Arbeit werden jedoch nur QFNRA-Formeln ohne transzendente Funktionen betrachtet.
Das abgeleitete lineare Problem kann schneller gelöst werden als das ursprüngliche nichtlineare Problem und mit der erhaltenen Lösung können Rückschlüsse auf die Lösung des nichtlinearen Problems geschlossen werden.
Bei der Linearisierung werden zunächst die nichtlinearen Multiplikationen von Variablen inkrementell auf uninterpretierte Funktionen abstrahiert, bis man eine lineare Formel erhält. Die erzeugte lineare Formel wird dann auf Erfüllbarkeit überprüft. Ist sie unerfüllbar, so ist auch das zugehörige nichtlineare SMT-Problem unerfüllbar. Falls sie jedoch erfüllbar ist, so kann das nichtlineare Problem auch erfüllbar sein. In diesem Fall wird versucht, die Lösung des linearen Problems auf das nichtlineare Problem zu übertragen.
Dazu wird zuerst überprüft, ob die Lösung des abstrahierten linearen Problems bereits das nichtlineare Problem löst, oder ob die gefundene Lösung mit einfachen Mitteln in eine Lösung des nichtlinearen Problems überführt werden kann. Wenn das der Fall ist, hat man auch eine Lösung für das nichtlineare Problem gefunden.
Ist das nicht der Fall, so muss die Abstraktion durch das Hinzufügen weiterer Constraints verfeinert werden. Durch das Hinzufügen von neu erzeugten Constraints wird verhindert, dass bei der nächsten Iteration das gleiche Modell für das lineare Problem gefunden wird. Das gefundene Modell wird also als Lösung ausgeschlossen. Danach wird erneut nach einem Modell für das linearisierte Problem gesucht. Diese Schritte wiederholen sich solange, bis es entweder kein Modell für das verfeinerte linearisierte

Problem gibt, oder ein gefundenes Modell auch das nichtlineare Problem löst. Außerdem wird die Ausführung nach einer bestimmten Anzahl von Iterationen abgebrochen, um die Terminierung zu gewährleisten.
Die Constraints zur Verfeinerung werden durch vordefinierte Axiome erzeugt, die in Abbildung 2.1 zu sehen sind.
Durch die Zero-Axiome wird festgelegt, dass das Produkt zweier Variablen genau dann Null ist, wenn mindestens eine der Variablen den Wert Null annimmt. Außerdem muss das Produkt von zwei Variablen mit gleichem Vorzeichen positiv und das Produkt von zwei Variablen mit unterschiedlichen Vorzeichen negativ sein.
Durch das Commutativity-Axiom wird die Kommutativität der Multiplikation in der reellen Arithmetik gesichert.
Die Sign-Axiome sichern die Auswirkungen von Vorzeichenwechseln bei der Multiplikation. Vorzeichen und Betrag des Ergebnisses einer Multiplikation ändern sich im Falle einer doppelten Multiplikation mit minus Eins nicht.
Die Monotonicity-Axiome verhindern Multiplikationen, die nicht monoton sind. Wenn der Betrag von mindestens einem der Faktoren vergrößert wird, während der andere nicht verkleinert wird, so kann sich der Betrag des Ergebnisses nicht verringern.
Die letzten Axiome sind die Tangent-plane-Axiome. Dabei betrachtet man die Tangentialebene durch einen Punkt (a,b) an der Funktion $x \cdot y$. Die Constraints, die Gleichungen sind, legen fest, dass die Multiplikation an dem Punkt (a,b) das korrekte Ergebnis liefert. Die Ungleichungen legen anhand der Tangentialebene Grenzen für andere Multiplikationen fest. Man betrachte die Multiplikation $x \cdot y$. Falls gleichzeitig $x \leqslant a$ und $y \leqslant b$ gelten, so muss das Ergebnis $x \cdot y$ über der Tangentialebene durch den Punkt (a,b) liegen. Sollten jedoch gleichzeitig $x \geqslant a$ und $y \leqslant b$ gelten, so muss das Ergebnis $x \cdot y$ unter der Tangentialebene liegen.
In SMT-RAT wurde dieses Verfahren in dem Modul *NRAIL* [2] implementiert. NRAIL ist dabei die Abkürzung für *nichtlineare reelle Arithmetik inkrementelle Linearisierung*. In SMT-RAT werden anstelle von uninterpretierten Funktionen jedoch neu generierte Variablen bei der inkrementellen Linearisierung verwendet. Das heißt, dass ein Produkt $x \cdot y$ nicht auf die uninterpretierte Funktion $f.(x,y)$ abstrahiert wird, sondern auf eine neu generierte Variable z. Dadurch kann es vorkommen, dass die lineare Lösung nicht für alle Variablen des nichtlinearen Problems eine Belegung hat, da Variablen bei der Linearisierung durch die Substitutionen verschwinden können. Deshalb muss das Modell des linearen Problems durch Belegung der fehlenden Variablen erweitert werden. In SMT-RAT werden die fehlenden Variablen mit Nullen belegt.
Neben den in Abbildung 2.1 dargestellten Axiomen wurde in dem Modul NRAIL auch noch ein weiterer Typ von Axiomen implementiert, die sogenannten ICP-Axiome. Die Abkürzung ICP steht dabei für *integral constraint propagation*. Die ICP-Axiome sind wie folgt definiert:

$$((x \geqslant a' \land y \geqslant b') \lor (x \leqslant -a' \land y \leqslant -b')) \rightarrow (z \geqslant c') \tag{2.1}$$

$$((x \geqslant a' \land y \leqslant -b') \lor (x \leqslant -a' \land y \geqslant b')) \rightarrow (z \leqslant -c') \tag{2.2}$$

$$((-a' \leqslant x \leqslant a') \land (-b' \leqslant y \leqslant b')) \rightarrow (-c' \leqslant z \leqslant c') \tag{2.3}$$

Idee der ICP-Axiome ist es Bereiche auszuschließen, in denen ein Produkt nicht liegen kann. Die ICP-Axiome können erzeugt werden, falls eine Multiplikation $z = x \cdot y$ unter dem gefundenen Modell falsch ist. Seien a, b, c die Belegungen von x, y, z im Modell. Es gilt also $c \neq a \cdot b$. Dazu wird die Multiplikation der Beträge der Variablen betrachtet. Falls das Produkt $|a| \cdot |b|$ größer als $|c|$ ist, so werden die ersten zwei Axiome (2.1) und

Zero:

$$\forall x, y.(x = 0 \vee y = 0) \leftrightarrow f_*(x, y) = 0$$

$$\forall x, y.((x > 0 \wedge y > 0) \vee (x < 0 \wedge y < 0)) \leftrightarrow f_*(x, y) > 0$$

$$\forall x, y.((x < 0 \wedge y > 0) \vee (x > 0 \wedge y < 0)) \leftrightarrow f_*(x, y) < 0$$

Sign:

$$\forall x, y.f_*(x, y) = f_*(-x, -y)$$

$$\forall x, y.f_*(x, y) = -f_*(-x, y)$$

$$\forall x, y.f_*(x, y) = -f_*(x, -y)$$

Commutativity:

$$\forall x, y.f_*(x, y) = f_*(y, x)$$

Monotonicity:

$$\forall x_1, y_1, x_2, y_2.((abs(x_1) \leq abs(x_2)) \wedge (abs(y_1) \leq abs(y_2))) \rightarrow$$
$$(abs(f_*(x_1, y_1)) \leq abs(f_*(x_2, y_2)))$$

$$\forall x_1, y_1, x_2, y_2.((abs(x_1) < abs(x_2)) \wedge (abs(y_1) \leq abs(y_2)) \wedge (y_2 \neq 0)) \rightarrow$$
$$(abs(f_*(x_1, y_1)) < abs(f_*(x_2, y_2)))$$

$$\forall x_1, y_1, x_2, y_2.((abs(x_1) \leq abs(x_2)) \wedge (abs(y_1) < abs(y_2)) \wedge (x_2 \neq 0)) \rightarrow$$
$$(abs(f_*(x_1, y_1)) < abs(f_*(x_2, y_2)))$$

Tangent plane:

$$\forall x, y.(f_*(a, y) = a * y) \wedge (f_*(x, b) = x * b) \wedge$$
$$((x > a \wedge y < b) \vee (x < a \wedge y > b)) \rightarrow f_*(x, y) < b * x + a * y - a * b \wedge$$
$$((x < a \wedge y < b) \vee (x > a \wedge y > b)) \rightarrow f_*(x, y) > b * x + a * y - a * b$$

Abbildung 2.1: Axiome zur Verfeinerung der Abstraktion aus [1]

(2.2) erzeugt. Im Fall, dass das Produkt kleiner ist, so wird das Axiom (2.3) erzeugt. Im ersteren Fall, also wenn $|a| \cdot |b| > |c|$ gilt, werden a' und b' mit $c' = a' \cdot b'$ definiert. Dabei sollen a' und b' so gewählt werden, dass für c' gilt $|c| < c' < |a| \cdot |b|$. Durch die erzeugten Axiome wird dann sichergestellt, dass das Produkt $|a| \cdot |b|$ maximal den Wert c' unter allen weiteren Modellen annehmen kann und somit der Betrag der gefundenen Belegung von z, der größer als c' war, ausgeschlossen wird.
Im anderen Fall, also wenn $|a| \cdot |b| < |c|$ gilt, wird Axiom (2.3) erzeugt. Durch dieses Axiom wird dann ein c' definiert, das kleiner als $|c|$ ist, und dann als obere Grenze für das Produkt $|a| \cdot |b|$ dient.

Kapitel 3

Methoden und Implementierung

In dem folgenden Kapitel werden Methoden beschrieben, die im Laufe dieser Arbeit entwickelt und implementiert wurden. Zuerst wurde die rekursiv implementierte Linearisierung in eine iterative Funktion geändert. Ein gefundenes Modell der linearisierten Constraints ist nicht zwangsläufig auch ein Modell der nichtlinearen Constraints. Dafür wurde im nächsten Schritt eine Methode implementiert, die durch das Zurechtziehen einzelner Variablenwerte des linearisierten Modells versucht, dieses in ein Modell für das ursprüngliche nichtlineare SMT-Problem zu überführen. Anschließend wurde die Möglichkeit geschaffen die CAD und die STrop als Submodule aufzurufen. Zuletzt wurde noch eine neue Heuristik für die Verfeinerung der Linearisierung entwickelt. Von wichtigen neuen Methoden wird zusätzlich die Implementierung kurz dargelegt.

3.1 Linearisierung

In SMT-RAT war die Linearisierung der nichtlinearen Constraints zum Teil rekursiv implementiert. Die Funktion *abstractUnivariateMonomial* wurde rekursiv aufgerufen. Zur besseren Übersichtlichkeit wurde die Funktion in dieser Arbeit iterativ implementiert. Dazu wurde Algorithmus 8 aus [2] so verändert, wie es in Algorithmus 1 zu sehen ist.
Der Algorithmus abstrahiert ein univariates Monom v^d auf eine neue Variable. Dazu wird die Schleife in Zeile 2 so lange ausgeführt, bis der Exponent d gleich eins ist, damit man am Ende ein lineares Monom erhält.
Außerdem wird in dieser Funktion eine Liste *extraVList* benötigt. Die Liste enthält Abstraktionsvariablen von v^{2^i}. Jede davon ist maximal einmal enthalten. Bei der Abstraktion von Variablen mit ungeradem Exponenten wird genau ein Faktor entfernt, damit man einen geraden Exponenten erhält. Dieser Faktor wird in *extraVList* gespeichert, damit am Ende das dort zu findende Produkt dieser Faktoren zusammen mit der letzten Abstraktionsvariable weiter abstrahiert werden kann.
Falls der Exponent d ungerade ist, wird die Basis v vorne in die initial leere Liste *extraVList* eingefügt, damit der Exponent um eins verringert werden kann und somit gerade ist. Dies geschieht in den Zeilen 3 bis 6. In Zeile 7 ist der Exponent d dadurch in jedem Fall gerade, kann also ohne Rest durch zwei dividiert werden. Nun wird ausgenutzt, dass $v^d = (v^2)^{d/2}$ gilt. In den Zeilen 7 und 8 sieht man die Anwendung dieser Gleichung.

Der Exponent d wird halbiert und als neue Basis wird eine neue Abstraktionsvariable als lineare Abstraktion von v^2 gewählt. Diese Schritte werden so lange wiederholt, bis der Exponent d gleich eins ist. Wenn das der Fall ist, wird die letzte Abstraktion von v vorne in die Liste *extraVList* eingefügt.
Auf die hier entstandene Liste wird zuletzt die Funktion *abstractProductRecursively* aufgerufen. Dies muss getan werden, da es auf Grund von ungeraden Exponenten weitere Variablen in der Liste geben kann. Das Produkt dieser Variablen muss ebenfalls linearisiert werden. Für ein Produkt $x \cdot y$, das durch eine Abstraktionsvariable z abstrahiert wurde, wird $z = x \cdot y$ als zugehörige Abstraktionsgleichung bezeichnet.

Algorithmus 1 abstractUnivariateMonomial(v, d) aus [2]

```
1:  extraVList := leere Liste
2:  while d > 1 do
3:    if d % 2 == 1 then
4:      d := d − 1
5:      extraVList.pushFront(v)
6:    end if
7:    d := d/2
8:    v := getAbstractVariable(v^2)
9:  end while
10: extraVList.pushFront(v)
11: return abstractProductRecursively(extraVList)
```

3.2 Zurechtziehen des Modells

Durch das Zurechtziehen der Lösung wird versucht schneller ein erfüllendes Modell für das betrachtete SMT-Problem zu finden. Dazu wird das Verfahren nach jeder Iteration angewendet, in der noch kein erfüllendes Modell für das SMT-Problem gefunden wurde. Das gefundene Modell wird dann angepasst, sodass es möglicherweise doch noch zu einer Lösung der nichtlinearen Probleminstanz wird.
Dazu werden die linearisierten Constraints und die zur Linearisierung verwendeten Gleichungen zur Definition der neu generierten Variablen betrachtet. Da das gefundene Modell nicht das SMT-Problem löst, muss in dieser Menge mindestens ein Constraint existieren, das durch das gefundene Modell nicht erfüllt wird.
Die Constraints werden einzeln nacheinander betrachtet. Zuerst wird das betrachtete Constraint auf Erfüllbarkeit unter dem aktuell generierten Modell überprüft. Wenn es erfüllt wird, so werden die darin vorkommenden Variablen als bereits betrachtet markiert und das nächste Constraint wird betrachtet. Falls es nicht erfüllt wird, so wird versucht das Constraint durch das Zurechtziehen des Wertes einer Variable im Modell zu erfüllen.
Dafür wird zuerst überprüft, ob in einem gegebenen Constraint noch eine Variable existiert, die noch nicht betrachtet wurde. Falls keine solche Variable mehr existiert, so wird das Zurechtziehen beendet und der SMT-Solver fährt mit der Verfeinerung der Abstraktion fort. Würde man Variablen mehrmals zurechtziehen, könnten bereits betrachtete Constraints unter der neuen Belegung unerfüllt werden.
Im anderen Fall wurde eine noch nicht betrachtete Variable gefunden, deren Wert nun im Modell angepasst wird, sodass das Constraint erfüllt wird. Dazu werden die Werte

der anderen Variablen aus dem Modell in das Constraint eingesetzt, sodass danach nur noch eine Variable übrig bleibt. Das Constraint wird dann nach dieser Variable aufgelöst.
Falls das Constraint eine Gleichung ist, so wird der Wert der betrachteten Variable im Modell durch das Ergebnis der Gleichung substituiert. Wenn das Constraint eine Ungleichung ist, so kann der Variable ein beliebiger Wert aus dem möglichen Wertebereich zugewiesen werden. Anschließend werden alle Variablen aus dem untersuchten Constraint als bereits betrachtet markiert und es wird mit dem nächsten Constraint fortgefahren. Dies wiederholt sich so lange, bis auch das letzte Constraint aus der Menge betrachtet wurde.
Wenn das letzte Constraint ebenfalls erfüllt ist, so werden alle linearisierten Constraints und die zur Linearisierung verwendeten Gleichungen unter dem zurechtgezogenen Modell erfülllt. Dieses Modell erfüllt dann aber auch das ursprüngliche QFNRA-Problem und der SMT-Solver kommt zu der Lösung, dass die Probleminstanz erfüllbar ist. Insbesondere können durch dieses Verfahren auch nur erfüllbare SMT-Probleme gelöst werden.
Die Reihenfolge, in der die Constraints und Variablen betrachtet werden, kann beliebig festgelegt werden. In dieser Arbeit wurden die Variablen in chronologischer Reihenfolge zum Zurechtziehen betrachtet. Chronologisch bedeutet, dass die verwendete Reihenfolge der Variablen beziehungsweise Constraints, der Reihenfolge aus der Eingabe entspricht. Danach folgen dann die Abstraktionsvariablen oder -gleichungen in der Reihenfolge ihrer Erzeugung.
Für die Reihenfolge, in der die Constraints betrachtet werden, wurden in dieser Arbeit drei verschiedene Möglichkeiten betrachtet. Die Constraints können in chronologischer und in umgekehrt chronologischer Reihenfolge betrachtet werden oder es werden zuerst die Gleichungen und danach die Ungleichungen betrachtet. Die umgekehrt chronologische Reihenfolge entspricht der chronologischen Reihenfolge in entgegengesetzter Abfolge.
Bei der dritten Reihenfolge werden zuerst die Variablen in den Gleichungen und danach die Variablen in den Ungleichungen betrachtet. Sowohl die Gleichungen als auch die Ungleichungen sind wie in der chronologischen Reihenfolge geordnet. Diese Reihenfolge wurde gewählt, da es bei Gleichungen weniger Möglichkeiten gibt den Wert einer Variable so zu verändern, dass die Gleichung nach dem Zurechtziehen erfüllt wird. In Beispiel 3.2.1 wird das Zurechtziehen für eine einzelne Gleichung beispielhaft gezeigt.

Beispiel 3.2.1. *Es wurde folgendes Modell für das lineare Problem gefunden und auf das nichtlineare Problem erweitert:*

$v_1 = -1,\ v_2 = 0,\ v_3 = 1,\ v_4 = -1,\ v_5 = -1$

Die Variablen v_1, v_3, v_4 und v_5 wurden beim Zurechtziehen als bereits betrachtet markiert und folgendes Constraint wird als nächstes betrachtet:

$$-v_1 \ + \ v_2 v_3 \ + \ v_4 v_5 \ = \ 0$$

Unter dem aktuellen Modell gilt:

$$-(-1) \ + \ 0 \cdot 1 \ + \ (-1) \cdot (-1) \ = \ 0 \ \Leftrightarrow \ 2 \ = \ 0$$

Die Variable v_2 wurde noch nicht betrachtet und wird zurechtgezogen:

$$-(-1) \ + \ 1 \cdot v_2 \ + \ (-1) \cdot (-1) \ = \ 0 \ \Leftrightarrow \ v_2 \ = \ -2$$

Die Variable v_2 wird als bereits betrachtet markiert und das nächste Constraint wird mit dem aktualisierten Modell auf die gleiche Weise untersucht. Falls dies das letzte Constraint war, wurde ein Modell gefunden und das SMT-Problem ist erfüllbar.

Das Zurechtziehen wurde wie folgt implementiert. Um die Variablen zu erkennen, welche schon betrachtet wurde, wird eine Bitmap verwendet. Initial wird jede Variable auf den Wert eins abgebildet. Nachdem ein Constraint zurechtgezogen wurde, werden alle in ihm auftretenden Variablen mit einer null markiert. Nach der Initialisierung der Bitmap wird die Funktion *orderVariables(vars, orderedVars, n)* aufgerufen. Der Parameter *vars* ist ein Container, der alle Variablen sowohl aus dem linearen als auch dem nichtlinearen Problem enthält. Diese werden dann geordnet in *orderedVars* eingefügt. Der Parameter *n* beschreibt, nach welcher Heuristik die Variablen geordnet werden sollen. Danach wird die Funktion *orderConstraints(constraints, orderedConstraints, n, size)* zum Ordnen der Constraints aufgerufen. Analog zum Ordnen der Variablen werden die Constraints aus *constraints* nach der Heuristik *n* geordnet und in *orderedConstraints* eingefügt. Zusätzlich wird mit *size* die Anzahl der Constraints übergeben.
Danach wird über die geordneten Constraints iteriert. Zuerst wird für jedes Constraint überprüft, ob es durch die aktuelle Variablenbelegung bereits erfüllt wird. Falls es erfüllt ist, wird mit dem nächsten Constraint fortgefahren. Andernfalls wird versucht eine Variable in diesem Constraint zurechtzuziehen.
Dazu wird zunächst über die Variablen in diesem Constraint iteriert, um eine Variable zu finden, die in der Bitmap noch nicht mit einer null markiert wurde. Die Iterierung erfolgt in der Reihenfolge der Variablen in *orderedVars*. Falls alle auftretenden Variablen mit einer null markiert wurden, endet das Zurechtziehen und der Wahrheitswert falsch wird zurückgegeben. Sonst endet die Iterierung über die auftretenden Variablen, sobald eine mit eins markierte Variable gefunden wurde. Die Belegung dieser Variable wird dann angepasst, sodass das Constraint erfüllt wird.
Dazu werden die übrigen Variablen in dem Constraint durch ihre Belegung im Modell substituiert. Dadurch wird ein Constraint mit einer Variablen erzeugt. Die Anpassung des Modells hängt dann von der Relation des Constraints ab.
Wenn die Relation „gleich“ lautet, so wird die Gleichung nach der Variablen aufgelöst und die Lösung der Gleichung wird die neue Belegung der Variable im Modell. Die Relation „kleiner gleich“ wird ebenfalls wie eine Gleichung behandelt. Falls die Relation „echt kleiner“ oder „ungleich“ lautet, so wird das Constraint zunächst ebenfalls nach der Variablen aufgelöst. Als neue Belegung der Variable im Modell wird die andere Seite der Ungleichung subtrahiert um eins gewählt. Weitere Relationen müssen nicht betrachtet werden, da sie in SMT-RAT auf die hier abgedeckten Relationen zurückgeführt werden.
Nachdem auch das letzte Constraint erfüllt werden konnte, gibt die Funktion den Wahrheitswert wahr zurück, da ein Modell für das nichtlineare Problem gefunden wurde.

3.3 Submodule

Wenn das Zurechtziehen nicht erfolgreich war muss weiter verfeinert werden, da keine Lösung für das nichtlineare Problem gefunden werden konnte. Vor der Verfeinerung des abstrahierten linearen Problems durch die Axiome aus Abbildung 2.1 können andere Module von SMT-RAT aufgerufen werden, um vielleicht doch ein Modell für das nichtlineare Problem zu finden. Wenn diese Module das SMT-Problem möglicherweise bereits lösen können, so sind keine weiteren Verfeinerungsschritte mehr nötig. Dazu wird zunächst das Ergebnis des SAT-Solvers betrachtet. Das Ergebnis des SAT-Solvers ist eine Menge von linearen Constraints die erfüllt werden soll, damit auch das nichtlineare Problem erfüllbar sein kann. Die linearen Constraints in dieser Menge werden nun in ihre zugehörige nichtlineare Form überführt. Danach kann diese Menge von nichtlinearen Constraints an ein anderes Modul übergeben werden, welches dann versucht eine Lösung zu finden. In dieser Arbeit wurde der Aufruf der Module *CADModule* und *STropModule* implementiert. Das *CADModule* verwendet eine zylindrische algebraische Zerlegung und das *STropModule* benutzt Subtropical Satisfiability zur Lösung von SMT-Problemen.
Die Subtropical Satisfiability ist eine unvollständige Methode zum Lösen von SMT-Problemen und gibt immer das Ergebnis UNKNOWN oder SAT zurück, kann jedoch keine Aussage zur Unerfüllbarkeit eines Problems machen. Dadurch können keine Rückschlüsse auf die Erfüllbarkeit des SMT-Problems gezogen werden, falls ein Aufruf der Subtropical Satisfiability zu dem Ergebnis UNKNOWN kommt. Nur wenn das Ergebnis des Aufrufs SAT lautet ist es für die weitere Ausführung des Programms relevant. In dem Fall ist das boolesche Modell des SAT-Solvers, das in nichtlinearer Form an das *STropModule* übergeben wurde, auch konsistent bezüglich reeller Arithmetik. Somit ist auch das ursprüngliche SMT-Problem erfüllbar.
Die CAD ist eine vollständige Methode zum Lösen von SMT-Problemen. Dadurch kann sowohl das Ergebnis SAT als auch das Ergebnis UNSAT in den darauffolgenden Schritten nach dem Aufruf der CAD verwendet werden. Analog zum Aufruf der Subtropical Satisfiability ist bei dem Ergebnis SAT auch das ursprüngliche SMT-Problem erfüllbar. Falls das Ergebnis der CAD jedoch UNSAT oder UNKNOWN lautet, dann kann kein direkter Rückschluss auf das ursprüngliche SMT-Problem gezogen werden. Bei dem Ergebnis UNSAT kann nur das aktuelle vom SAT-Solver gefundene boolesche Modell ausgeschlossen werden. Es ist jedoch möglich, dass eine Belegung der Variablen existiert, die ein anderes boolesches Modell des SMT-Problems erfüllt. Dadurch kann dieses boolesche Modell für den nächsten Aufruf des SAT-Solvers ausgeschlossen werden.
Dazu wird die kleinste unerfüllbare Teilmenge der Constraints betrachtet, die an die CAD übergeben wurden. Da diese Constraints gemeinsam nicht erfüllbar sind, muss mindestens ein Constraint aus dieser Menge beim nächsten booleschen Modell unerfüllt sein. Um das zu erreichen werden alle Constraints in dieser Menge zunächst wieder in ihre linearisierte Form überführt. Danach wird den linearen Constraints eine neue Klausel hinzugefügt. Die neue Klausel ist die Disjunktion über die Negationen dieser wieder linearisierten Constraints.
Durch das Ausschließen des aktuellen booleschen Modells, im Falle des Ergebnisses UNSAT bei Aufruf der CAD, muss keine weitere Verfeinerung durch die Axiome stattfinden. Das beschriebene Hinzufügen der unerfüllbaren Teilmenge reicht bereits aus, damit der SAT-Solver ein alternatives boolesches Modell finden muss. Wird die CAD also in jeder Iteration als Submodul aufgerufen, findet keine Verfeinerung anhand der

Axiome statt, da die CAD diese übernimmt.
In Algorithmus 2 wird die Implementierung des Aufrufs der CAD als Submodul dargestellt. Der Aufruf der STrop funktioniert analog, mit dem Unterschied, dass das Ergebnis UNKNOWN ignoriert wird.
In jeder Iteration berechnet der SAT-Solver ein boolesches Modell des linearisierten Problems und bestimmt damit eine Menge von Constraints, die erfüllt werden soll. In den Zeilen 2 bis 4 wird jedes Constraint dieser Menge aus seiner linearisierten Form zurück in seine nichtlineare Form überführt und dem CAD-Modul übergeben. Das CAD-Modul überprüft anschließend in Zeile 5 die Erfüllbarkeit der übergebenen Constraints. Falls die Antwort des CAD-Moduls SAT lautet, so wird in Zeile 7 SAT ausgegeben, da dann auch das SMT-Problem aus der Eingabe erfüllbar ist.
Lautet die Antwort hingegen UNSAT, wird in Zeile 9 auf die gefundene unerfüllbare Teilmenge des CAD-Moduls zugegriffen. Die Formeln dieser Teilmenge werden anschließend wieder linearisiert (Zeilen 10 bis 12). In Zeile 13 wird ein neues Constraint als Disjunktion über die negierten linearisierten Constraints der unerfüllbaren Teilmenge erzeugt. Zum Schluss muss dieses neu generierte Constraint noch der Menge von linearen Constraints hinzugefügt werden, sodass der SAT-Solver ein neues boolesches Modell berechnen muss.

Algorithmus 2 Aufruf der CAD

```
1:  wahreConstraints = SATModul.wahreConstraints()
2:  for each Constraint ∈ wahreConstraints do
3:     CADModul.add(getNRAfromLRA(Constraint))
4:  end for
5:  Antwort := CADModul.check()
6:  if Antwort == SAT then
7:     return SAT
8:  else
9:     unerfüllbareMenge := CADModul.infeasibleSubsets()
10:    lineareConstraints := ∅
11:    for each Constraint ∈ unerfüllbareMenge do
12:       lineareConstraints ∪ linearizeSubformula(Constraint)
13:    end for
14:    neueKlausel := ⋁_{c∈lineareConstraints} ¬c
15:    LRAConstraints.add(neueKlausel)
16: end if
```

3.4 Axiomsequenzen

Wenn die CAD verwendet wird, ist keine Verfeinerung durch die Axiome nötig. Falls man sie jedoch nicht verwendet, so muss in jeder Iteration eine Verfeinerung der linearisierten Constraints anhand der Axiome stattfinden. Dazu muss jedoch eine Reihenfolge festgelegt werden, in der die Axiome überprüft werden.
Die Sequenz, in der die Axiome zur Verfeinerung angewendet werden sollen, kann beliebig festgelegt werden. In SMT-RAT wurden mehrere verschiedene Sequenzen definiert. Am Anfang zeigt ein Zeiger auf den ersten Axiomtyp der verwendeten Sequenz. Immer wenn ein Axiom betrachtet wurde, wird der Zeiger auf den nächsten Eintrag

gesetzt. Nach Betrachtung des letzten Axiomtyps zeigt der Zeiger dann wieder auf den ersten Axiomtyp. Bei jeder Verfeinerung wird zuerst der Axiomtyp betrachtet, auf den der Zeiger zeigt. Wenn ein Axiom des aktuell betrachteten Axiomtyps verletzt wird, so wird dieser Axiomtyp zur Verfeinerung verwendet und die verfeinerte Menge von linearen Constraints wird an den SAT-Solver übergeben. Der Zeiger zeigt danach auf den nächsten Axiomtyp der Sequenz und bei der nächsten Verfeinerung wird dieser dann zuerst betrachtet. Falls kein Axiom des betrachteten Axiomtyps von der linearen Lösung verletzt wird, so wird der Zeiger ebenfalls auf den nächsten Axiomtyp der Sequenz gesetzt. Es wird dann auf gleiche Weise überprüft, ob ein Axiom dieses Typs verletzt wird. Erst nach einer erfolgreichen Verfeinerung wird der SAT-Solver erneut aufgerufen.
In dieser Arbeit wurde die Möglichkeit implementiert die Axiomsequenzen bei jeder Iteration vom Anfang zu beginnen. Das Ziel dieser Idee ist es, dass zuerst versucht wird die Axiome zu erzeugen, die weniger rechenaufwändig sind. Die Zero-Axiome sind beispielsweise weniger rechenaufwändig als die Monotonicity-Axiome. Bei den Zero-Axiomen müssen nur Vorzeichen und Nullen untersucht werden, wohingegen bei den Monotonicity-Axiomen viele Multiplikationen miteinander verglichen werden müssen. Falls die Zero-Axiome nun also am Anfang der Axiomsequenz platziert werden, so werden diese Axiome bei jeder Iteration als erstes auf ihre Verletzung überprüft und rechenaufwändige Axiome können am Ende der Sequenz platziert werden. Dadurch kann erreicht werden, dass rechenaufwändige Axiome so selten wie möglich erzeugt werden müssen. Bei der Verwendung eines iterationsübergreifenden Zeigers ist dies nicht der Fall. Bei einer genügend großer Anzahl von Iterationen wird jedes Axiom mindestens einmal überprüft.
Es gibt also keinen Zeiger mehr, der iterationsübergreifend auf den nächsten Axiomtyp zeigt. Der Zeiger wird nach jeder erfolgreichen Verfeinerung wieder auf den ersten Axiomtyp der benutzten Sequenz gesetzt. Falls kein Axiom des ersten Axiomtyps unter dem linearisierten Modell verletzt wird, so wird der Zeiger auf den nächsten Axiomtyp gesetzt und es wird überprüft, ob Axiome dieses Typs verletzt werden. Der Zeiger wird so lange verschoben, bis das erste Axiom gefunden wurde, welches von dem Modell verletzt wird. Dann werden die verletzten Axiome dieses Typs erzeugt und der Zeiger wird für die nächste Iteration zurückgesetzt.

Kapitel 4

Ergebnisse

In diesem Kapitel wird untersucht, welche Implementierungen die Performanz des NRAIL-Modules steigern konnten. Zur Analyse der Implementierung wurden die Benchmarks QF_NRA aus aus der SMT-LIB [10] verwendet. Es wurde die Laufzeit des Lösens an sich gemessen, also ohne Linearisierung oder Vorverarbeitung der Eingabe. Ausgeführt wurden die Benchmark auf einem Intel Core i5-Prozessor bei einer Taktrate von 2,9GHz.

4.1 Linearisierung

Zur Evaluierung der iterativen Linearisierung wurden die Benchmarks mit einem Timeout von 90 Sekunden ausgeführt. Dabei wurde die Zeit vom Beginn bis zum Ende der Linearisierung gemessen. In Abbildung 4.1 sieht man das Streudiagramm für die Dauer der Linearisierung von rekursiver und iterativer Implementierung. Die rote Diagonale zeigt die Punkte, an denen beide Implementierungen die gleiche Zeit für die Linearisierung benötigen würden. Man kann sehen, dass alle Punkte des Streudiagramms nah an dieser Diagonalen liegen. Das bedeutet, dass beide Implementierungen die getesteten Probleminstanzen in ähnlicher Zeit linearisieren konnten. Es lässt sich ein leichter Trend erkennen, dass die Punkte weiter rechts im Diagramm eher über der Diagonalen liegen als darunter. Für größere Probleminstanzen ist die iterative Linearisierung also durchschnittlich ein wenig schneller als die rekursive Linearisierung. Bei den getesteten Instanzen war dies ab einer Linearisierungsdauer von circa einer Sekunde der Fall.
Tabelle 4.1 zeigt ein ähnliches Ergebnis. Die iterative Linearisierung war im Durchschnitt zwar 30ms langsamer, konnte jedoch auch fünf Probleminstanzen mehr linearisieren.

Familie	Rekursiv		Iterativ	
Sturm-MGC	9/9	0,421ms	9/9	0,440ms
Economics-Mulligan	135/135	0,518ms	135/135	0,530ms
hycomp	2.752/2.752	7,56ms	2.752/2.752	7,55ms
Heizmann	69/69	9,44ms	69/69	9,52ms
zankl	152/163	1.751ms	152/163	1.760ms
meti-tarski	7.006/7006	0,212ms	7.006/7006	0,208ms
Sturm-MBO	404/405	17.642ms	404/405	18.129ms
kissing	45/45	9,20ms	45/45	8,73ms
hong	20/20	0,155ms	20/20	0,179ms
LassoRanker	760/821	4.153ms	765/821	4.295ms
UltimateAutomizer	61/61	5.302ms	61/61	5.349ms
Gesamt	11.413/11486	961,4ms	11.418/11486	991,3ms

Tabelle 4.1: Vergleich von rekursiver und iterativer Linearisierung anhand der Anzahl von linearisierten Instanzen und der benötigten durchschnittlichen Zeit

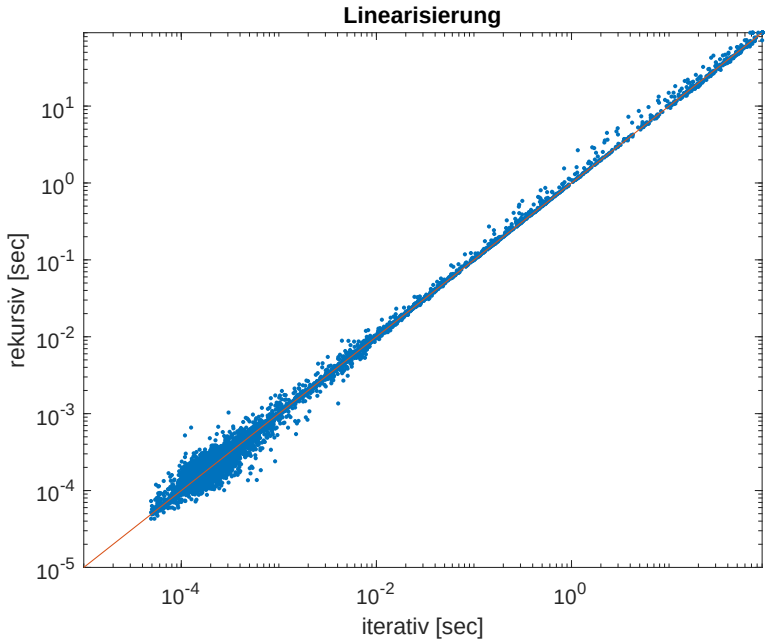

Abbildung 4.1: Streudiagramm für die iterative und rekursive Linearisierung

4.2 Zurechtziehen

Zur Evaluierung des Zurechtziehens wurden die Benchmarks mit einem Timeout von 60 Sekunden ausgeführt und als Strategie wurde *NRARefinementSolver4* verwendet. Dabei wurden die Reihenfolgen chronologisch, umgekehrt chronologisch und Gleichungen zuerst für die Constraints getestet. Die Variablen wurden in chronologischer Reihenfolge betrachtet.
Zunächst wird die Performanz der Solver auf unerfüllbaren Instanzen untersucht. In den Abbildungen 4.2, 4.3 und 4.4 sieht man Streudiagramme, die jeweils zwei Solver miteinander vergleichen. Man sieht, dass alle drei Diagramme eine hohe Symmetrie zur Diagonalen aufweisen. Auch liegen die meisten Punkte in der Nähe der Diagonalen. Die starke Streuung, insbesondere in den Bereich, in dem das Zurechtziehen schneller ist als der Verzicht darauf, entsteht durch Seiteneffekte, die Auswirkungen auf die Iterationen haben.
Auf unerfüllbaren Instanzen kann das Zurechtziehen nur negative Auswirkungen auf die Laufzeit haben, da durch das Verfahren nur erfüllbare Instanzen gelöst werden können. Beim Test einzelner Instanzen hat sich jedoch herausgestellt, dass die Zeit, die der Aufruf der Funktion benötigt, gering ist. Bei den getesteten Instanzen war ein Aufruf meist kürzer als 0,5ms.
Auf erfüllbaren Instanzen hat das Zurechtziehen einen größeren Einfluss auf die Laufzeit des Solvers als auf unerfüllbaren Instanzen. Zuerst wird das Zurechtziehen in chronologischer Reihenfolge untersucht. In Abbildung 4.5 sieht man den Vergleich mit dem Solver, der das Zurechtziehen nicht verwendet. Die Kreuze in rot stellen die Instanzen dar, die durch das Zurechtziehen gelöst wurden. Die blauen Punkte bilden die Instanzen ab, die nicht durch das Zurechtziehen gelöst wurden. Die meisten roten Kreuze liegen unterhalb Diagonalen, was bedeutet, dass das Zurechtziehen das Lösen des Problems beschleunigen konnte.
Für die umgekehrt chronologische Reihenfolge in Abbildung 4.6 ist dies noch stärker zu sehen. Insgesamt gibt es dort 270 rote Kreuze, von denen fast alle unter der Diagonalen liegen. Insbesondere gibt es auch 58 Instanzen, die durch das Zurechtziehen gelöst werden konnten, jedoch ohne Zurechtziehen in das Timeout gelaufen sind.
In Abbildung 4.7 wird die chronologische Reihenfolge mit der umgekehrt chronologischen Reihenfolge auf erfüllbaren Probleminstanzen verglichen. Die blauen Punkte stellen die Instanzen dar, bei denen das Modell von keiner Heuristik erfolgreich zurechtgezogen werden konnte. Die roten Kreuze zeigen die Instanzen, die bei Anwendung der umgekehrt chronologischen Reihenfolge erfolgreich zurechtgezogen werden konnten und die schwarzen Kreise zeigen die Instanzen bei denen das Zurechtziehen mit der chronologischen Reihenfolge erfolgreich war.
Als erstes fällt auf, dass mehr Instanzen durch die umgekehrt chronologische Reihenfolge gelöst wurden als durch die chronologische Reihenfolge. Außerdem liegen auf der rechten Achse 58 mehr Punkte als auf der oberen Achse. Durch die umgekehrt chronologische Reihenfolge konnten also 59 Instanzen gelöst und umgekehrt konnte durch die chronologische Reihenfolge eine Instanz gelöst werden, die durch die jeweils andere Heuristik nicht gelöst werden konnte.
Weiterhin fallen viele rote Kreuze unter der Diagonalen auf. Die umgekehrt chronologische Reihenfolge weist insgesamt also eine bessere Performanz auf als die chronologische Reihenfolge.
Zuletzt werden die Reihenfolgen Gleichungen zuerst und umgekehrt chronologisch auf erfüllbaren Instanzen untersucht. Abbildung 4.8 zeigt das vergleichende Streudia-

gramm für erfüllbare Probleminstanzen. Als rote Kreuze sind die Instanzen dargestellt, die erfolgreich durch die Reihenfolge Gleichungen zuerst zurechtgezogen wurden. Die schwarzen Kreise zeigen dies analog für die umgekehrt chronologische Reihenfolge. Konnte keine Heuristik das Modell erfolgreich zurechtziehen, so sind die Instanzen als blaue Punkte dargestellt. Auf der rechten Achse liegen 16 schwarze Kreise und auf der oberen Achse liegen ebenfalls 16 rote Kreuze. Das Diagramm zeigt eine ähnliche Performanz der beiden Heuristiken. Die umgekehrt chronologische Reihenfolge ist ein wenig besser, da man in der rechten Hälfte ein paar schwarze Kreise sieht, die weit unter der Diagonalen liegen. Die restlichen Punkte liegen alle nah an der Diagonalen. Dort ist also keine Heuristik wesentlich besser als die andere.
Insgesamt kann man sagen, dass es sinnvoll ist die Gleichungen zuerst zu betrachten. Auch bei der umgekehrt chronologischen Reihenfolge werden zuerst alle Abstraktionsgleichungen betrachtet, bevor mit den linearisierten Constraints fortgefahren wird. Dadurch ist die Performanz vergleichbar mit der von der Heuristik Gleichungen zuerst.
Im Gegensatz zu Ungleichungen haben Gleichungen nur eine beschränkte Anzahl an Möglichkeiten beim Zurechtziehen. Dadurch, dass beim Zurechtziehen die Belegung jeder Variable im Modell maximal einmal angepasst werden kann, sollten die Ungleichungen erst nach den Gleichungen betrachtet werden. Die betrachteten Gleichungen haben auf Grund der Linearisierung maximal den Grad zwei. Dadurch gibt es beim Zurechtziehen von Gleichungen auch maximal zwei Werte, die die betrachtete Variable annehmen kann, um die Gleichung zu erfüllen. Bei Ungleichungen ist dies anders. Dort gibt es ein Intervall von Werten, die die betrachtete Variable annehmen kann, sodass die Ungleichung trotzdem erfüllt wird. Falls nun eine Ungleichung vor einer Gleichung zurechtgezogen wird, ist unklar welcher Wert am sinnvollsten angenommen werden sollte. Insbesondere bei linearen Gleichungen ist jedoch sofort klar, welchen Wert die betrachtete Variable annehmen muss, damit die Gleichung erfüllt wird.
Eine Reihenfolge, die beide Heuristiken vereint, wäre, dass beim Zurechtziehen mit den Abstraktionsgleichungen begonnen wird, danach die Gleichungen aus den linearisierten Constraints betrachtet werden und zum Schluss die Ungleichungen. Zu Beginn des Zurechtziehens sind alle linearisierten Constraints erfüllt, da der LRA-Solver ein Modell für genau diese Constraints gefunden hat. Bei der Heuristik Gleichungen zuerst und noch stärker bei der chronologischen Reihenfolge werden also bereits viele Variablen markiert, bevor das erste unerfüllte Constraint erreicht wird.
Deshalb ist es sinnvoll mit den Abstraktionsgleichungen zu beginnen, da es zu Beginn nur dort unerfüllte Gleichungen geben kann. Danach sollten die restlichen Gleichungen untersucht werden, da es dort wie bereits beschrieben weniger Möglichkeiten zum Zurechtziehen gibt als bei den Ungleichungen.
Die umgekehrt chronologische Reihenfolge könnte auch bei der Ordnung der Variablen sinnvoll sein. Dadurch kann das Wurzelziehen häufig vermieden werden, da die letzten Abstraktionsvariablen Produkte von vorherigen Variablen sind.

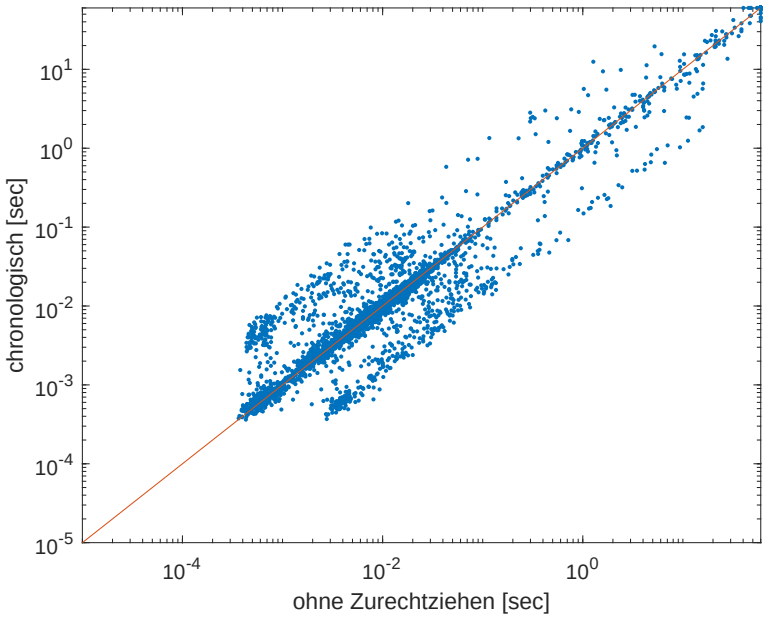

Abbildung 4.2: Streudiagramm für das Zurechtziehen in chronologischer Reihenfolge auf unerfüllbaren Instanzen

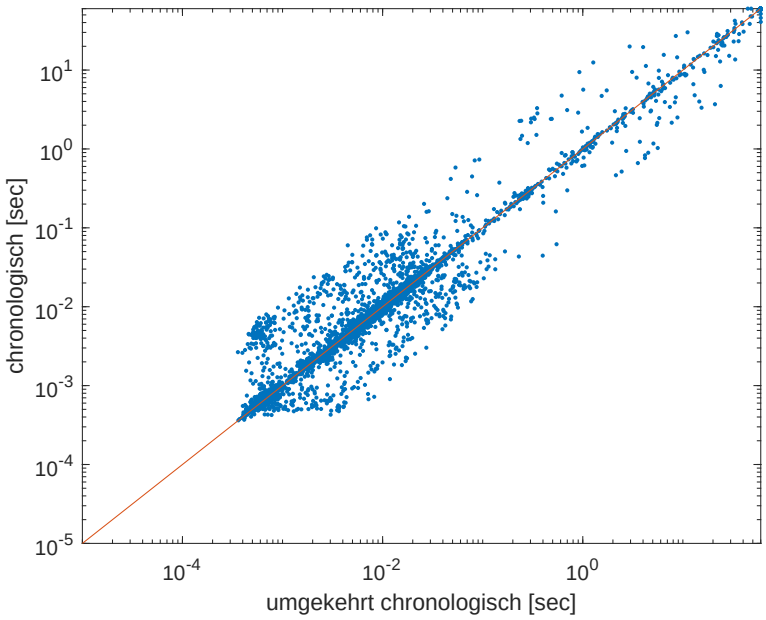

Abbildung 4.3: Streudiagramm; Vergleich des Zurechtziehens der Constraints in chronologischer und umgekehrt chronologischer Reihenfolge auf unerfüllbaren Instanzen

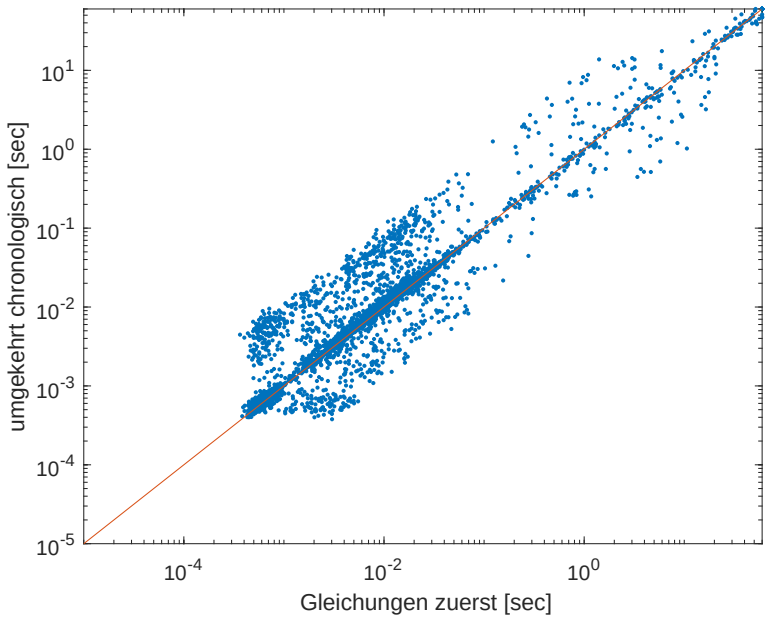

Abbildung 4.4: Streudiagramm; Vergleich des Zurechtziehens der Constraints in den Reihenfolgen Gleichungen zuerst und umgekehrt chronologische auf unerfüllbaren Instanzen

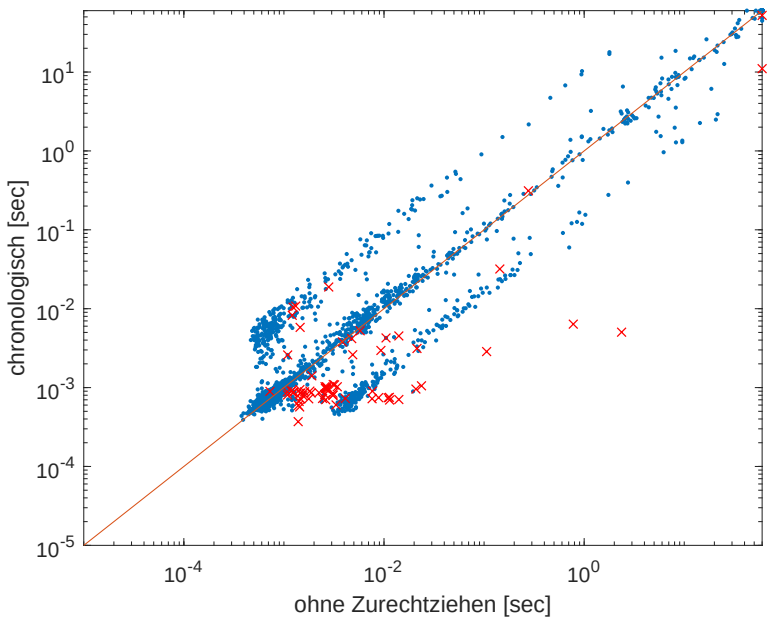

Abbildung 4.5: Streudiagramm für das Zurechtziehen in chronologischer Reihenfolge auf erfüllbaren Instanzen

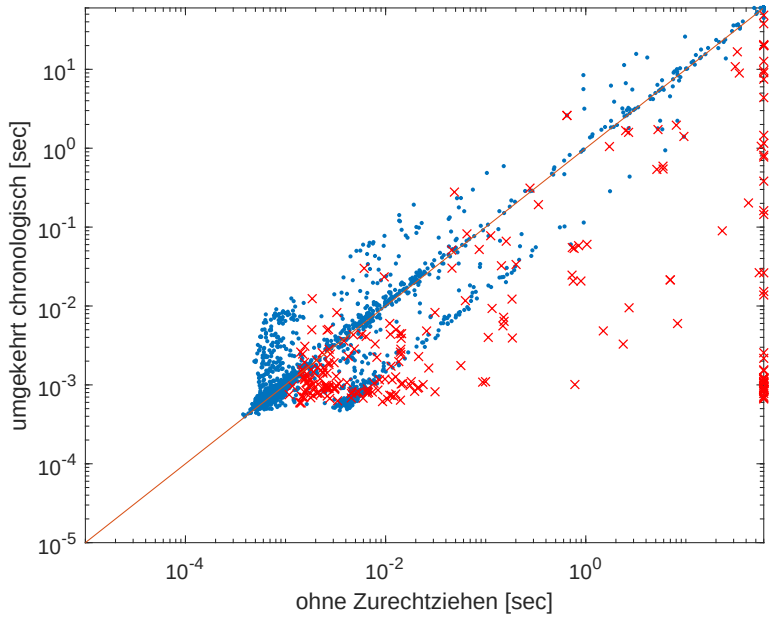

Abbildung 4.6: Streudiagramm für das Zurechtziehen in umgekehrt chronologischer Reihenfolge auf erfüllbaren Instanzen

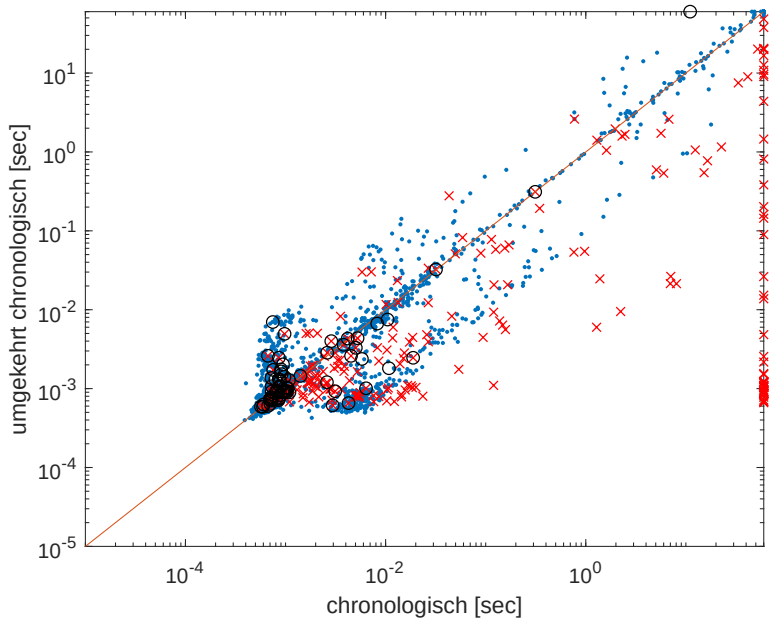

Abbildung 4.7: Streudiagramm; Vergleich des Zurechtziehens der Constraints in chronologischer und umgekehrt chronologischer Reihenfolge auf erfüllbaren Instanzen

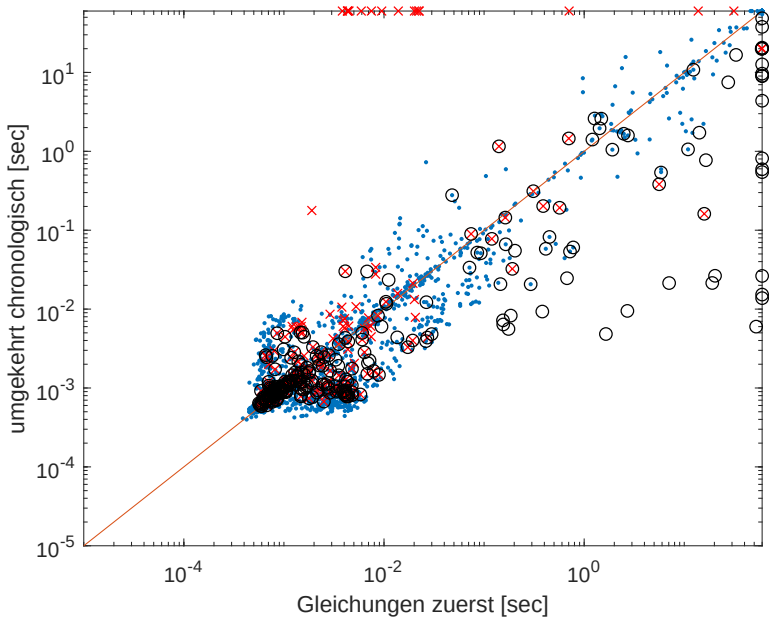

Abbildung 4.8: Streudiagramm; Vergleich des Zurechtziehens der Constraints mit den Reihenfolgen Gleichungen zuerst und umgekehrt chronologisch auf erfüllbaren Instanzen

4.3 Submodule

Zur Evaluierung des Aufrufs von Submodulen wurden die Benchmarks mit einem Timeout von 60 Sekunden ausgeführt und als Strategie wurde *NRARefinementSolver4* verwendet. Sowohl die Subtropical Satisfiability als auch die CAD konnten die Performanz des Moduls verbessern.
Zuerst wird der Einsatz der Subtropical Satisfiability betrachtet. In Tabelle 4.2 kann man sehen, dass die Performanz bei unerfüllbaren Instanzen ähnlich zur Performanz des Solvers ohne Aufruf von Submodulen ist. Sowohl die Anzahl der gelösten unerfüllbaren Instanzen als auch die durchschnittliche Dauer zum Lösen einer solchen Instanz sind fast gleich. Dies liegt daran, dass die STrop keine Aussage zu unerfüllbaren Instanzen machen kann. Dass die Performanz sich nicht wesentlich verschlechtert, zeigt auch, dass ein Aufruf der STrop nicht viel Zeit benötigt. Wären die Aufrufe sehr zeitintensiv, so würde die Performanz auf unerfüllbaren Instanzen stärker abnehmen. Da dies nicht der Fall ist, kommt die STrop also bereits nach kurzer Zeit zu dem Ergebnis UNSAT und es kann mit der Verfeinerung der linearen Constraints fortgefahren werden. Das Testen einzelner Instanzen hat gezeigt, dass die STrop auf unerfüllbaren Instanzen wenige Millisekunden pro Aufruf benötigt.
Für erfüllbare Probleminstanzen hingegen wird die Performanz durch das Aufrufen der STrop wesentlich verbessert. Es konnten mehr erfüllbare SMT-Probleme aus den Benchmarks gelöst werden als ohne den Aufruf. Außerdem wurden ungefähr 40% der gelösten erfüllbaren Instanzen durch die STrop gelöst. Auch die durchschnittliche Dauer, die zum Lösen der erfüllbaren Instanzen benötigt wurde, konnte fast halbiert

werden. Dies ist auch in Abbildung 4.9 zu sehen. Es liegen 750 Punkte auf der rechten Achse und auch sonst sind insbesondere in der rechten Hälfte der Grafik mehr Punkte unter der Diagonalen zu sehen. Diese 750 Instanzen konnten ohne den Einsatz der STrop nicht in der vorgegebenen Zeit gelöst werden. Bei den weiteren Instanzen, die weit unter der Diagonalen liegen, konnte die STrop die Laufzeit ebenfalls verbessern.

Der Einsatz der CAD liefert ebenfalls eine bessere Performanz. Von den unerfüllbaren Probleminstanzen können leicht mehr Instanzen gelöst werden. Insbesondere konnte hier jedoch auch die durchschnittliche Laufzeit zum Lösen der unerfüllbaren Instanzen um circa 50% gesenkt werden. In Abbildung 4.10 sieht man das Streudiagramm der CAD aufgetragen gegen keine Verwendung von Submodulen auf unerfüllbaren Instanzen. Auffällig sind die vielen Punkte auf den Achsen. Beide Möglichkeiten sind also auf unterschiedlichen Instanzen effizienter. So konnten 470 Instanzen mit dem Einsatz der CAD gelöst werden, die ohne die CAD nicht gelöst werden konnten und umgekehrt konnten durch den Verzicht auf die CAD 227 Probleminstanzen gelöst werden, die mit der CAD nicht bis zum Erreichen des Timeouts gelöst werden konnten.
Für erfüllbare Instanzen ist der Einsatz der CAD sogar noch effizienter. Dies liegt daran, dass die CAD diese in der ersten Iteration löst. Falls die CAD jedoch zu dem Ergebnis UNSAT kommt, muss weiter iteriert werden. Nach Tabelle 4.3 konnten mehr als doppelt so viele erfüllbare Instanzen gelöst werden als ohne den Einsatz der CAD. Auch die durchschnittliche Lösungsdauer konnte erheblich gesenkt werden. Erfüllbare Instanzen werden im Durchschnitt viermal so schnell gelöst wie durch den Solver ohne Einsatz der CAD.
Abbildung 4.11 zeigt ein ähnliches Bild. Da mit dem Einsatz der CAD mehr erfüllbare Instanzen gelöst werden konnten, sind viele Punkte auf der rechten Achse zu sehen. Auf dieser Achse liegen mehr als 2.500 Punkte. Die Probleminstanzen, die durch diese Punkte dargestellt werden, konnten ohne den Einsatz der CAD nicht gelöst werden bis das Timeout erreicht wurde. Aber auch von den übrigen Punkten liegt der größere Teil unter der Diagonalen, was bedeutet, dass der Einsatz der CAD die Performanz auf erfüllbaren Probleminstanzen steigern konnte.

Familie		SAT		UNSAT	
Sturm-MGC	mit STrop	0	-	0	-
	ohne STrop	0	-	0	-
Economics-Mulligan	mit STrop	34	1.046ms	6	104ms
	ohne STrop	34	886ms	6	98ms
hycomp	mit STrop	0	-	2.099	331ms
	ohne STrop	0	-	2.100	327ms
Heizmann	mit STrop	0	-	0	-
	ohne STrop	0	-	0	-
zankl	mit STrop	31	1.837ms	6	11.491ms
	ohne STrop	1	15,4ms	6	11.141ms
meti-tarski	mit STrop	2.486	536ms	1.712	1.761ms
	ohne STrop	1.771	1.017ms	1.713	1.701ms
Sturm-MBO	mit STrop	0	-	16	19ms
	ohne STrop	0	-	16	9ms
kissing	mit STrop	6	134ms	0	-
	ohne STrop	6	121ms	0	-
hong	mit STrop	0	-	2	33ms
	ohne STrop	0	-	3	10.424ms
LassoRanker	mit STrop	0	-	0	-
	ohne STrop	0	-	0	-
UltimateAutomizer	mit STrop	0	-	0	-
	ohne STrop	0	-	0	-
Gesamt	mit STrop	2.557	579ms	3.841	878ms
	ohne STrop	1.812	1.011ms	3.844	857ms

Tabelle 4.2: Vergleich von der Verwendung der STrop als Submodul und der Verwendung keines Submoduls

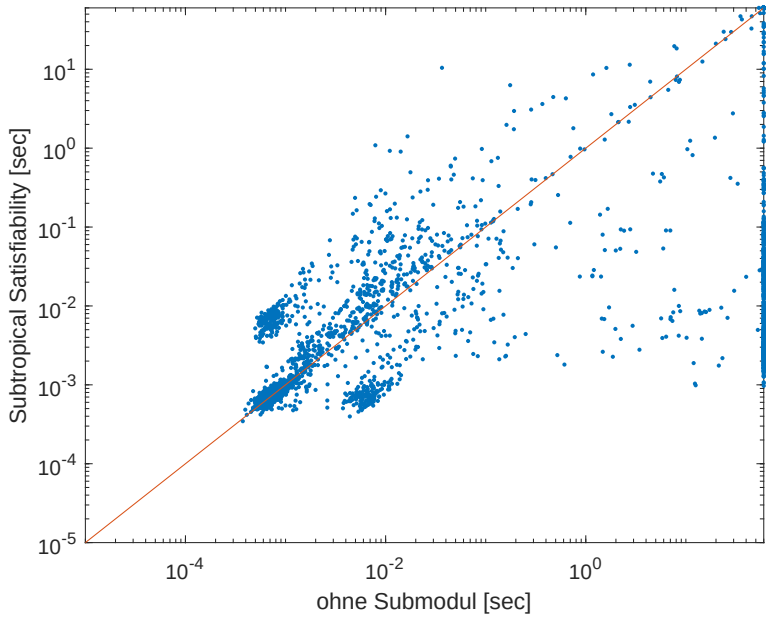

Abbildung 4.9: Streudiagramm für den Aufruf der STrop auf erfüllbaren Instanzen

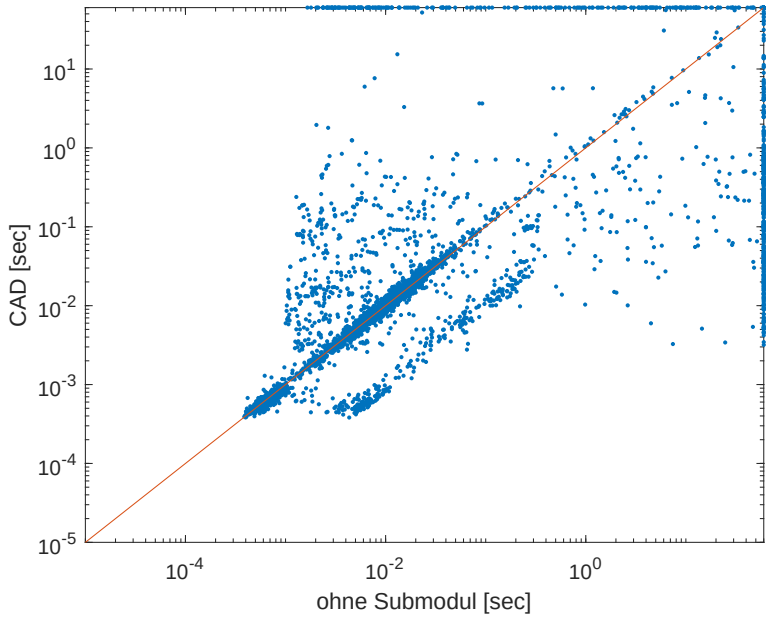

Abbildung 4.10: Streudiagramm für den Aufruf der CAD auf unerfüllbaren Instanzen

Familie		SAT		UNSAT	
Sturm-MGC	mit CAD	0	-	0	-
	ohne CAD	0	-	0	-
Economics-Mulligan	mit CAD	42	3.129ms	6	6.034ms
	ohne CAD	34	886ms	6	98ms
hycomp	mit CAD	0	-	2.051	179ms
	ohne CAD	0	-	2.100	327ms
Heizmann	mit CAD	0	-	0	-
	ohne CAD	0	-	0	-
zankl	mit CAD	14	807ms	4	37ms
	ohne CAD	1	15,4ms	6	11.141ms
meti-tarski	mit CAD	4.260	217ms	1.991	771ms
	ohne CAD	1.771	1.017ms	1.713	1.701ms
Sturm-MBO	mit CAD	0	-	31	126ms
	ohne CAD	0	-	16	9ms
kissing	mit CAD	14	1.300ms	0	-
	ohne CAD	6	121ms	0	-
hong	mit CAD	0	-	4	369ms
	ohne CAD	0	-	3	10.424ms
LassoRanker	mit CAD	0	-	0	-
	ohne CAD	0	-	0	-
UltimateAutomizer	mit CAD	0	-	0	-
	ohne CAD	0	-	0	-
Gesamt	mit CAD	4.330	251ms	4.087	439ms
	ohne CAD	1.812	1.011ms	3.844	857ms

Tabelle 4.3: Vergleich von der Verwendung der CAD als Submodul und der Verwendung keines Submoduls

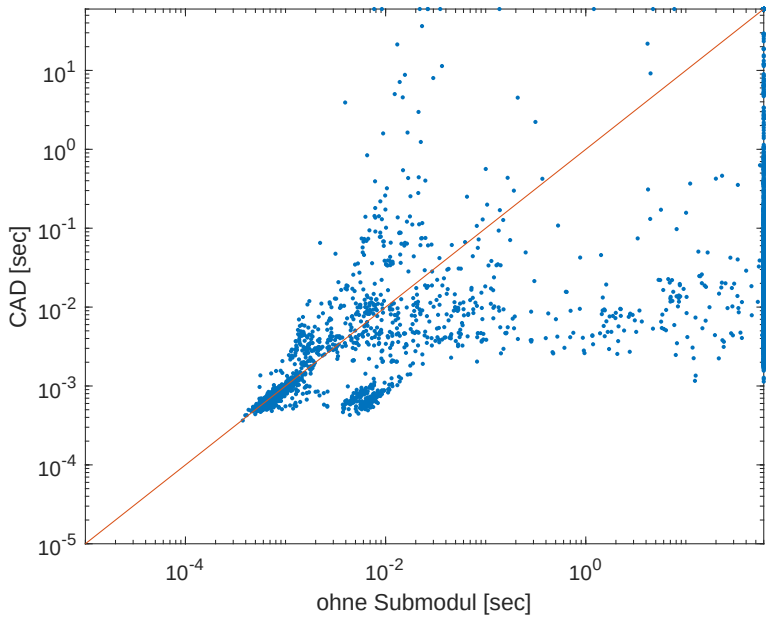

Abbildung 4.11: Streudiagramm für den Aufruf der CAD auf erfüllbaren Instanzen

4.4 Axiomsequenzen

Zur Evaluierung der Axiomsequenzen ohne iterationsübergreifenden Zeiger auf das nächste Axiom wurden die Benchmarks mit einem Timeout von 60 Sekunden ausgeführt. Dabei wurden bereits vorhandene Sequenzen aus [2] als auch eine neu definierte Sequenz untersucht. Folgende Sequenzen wurden getestet:

- NRARefinementSolver1: Zero, Tangent_Plane, ICP, Congrunece, Monotonicity (Sequenz 1)
- NRARefinementSolver2: Tangent_Plane, Zero, ICP, Congruence, Monotonicity (Sequenz 2)
- NRARefinementSolver4: ICP, Zero, Tangent_Plane, Congruence, Monotonicity (Sequenz 4)
- NRARefinementSolver26: Zero, Congruence, Monotonicity, Tangent_Plane, ICP (Sequenz 26)

Zuerst wird untersucht, ob der neue Zeiger die Performanz gegenüber dem iterationsübergreifenden Zeiger verbessern konnte. Dazu wurde Sequenz 4 mit beiden Zeigern getestet. In Abbildung 4.12 sieht man den Vergleich auf unerfüllbaren Eingaben und in Abbildung 4.13 sieht man den Vergleich für erfüllbare Probleminstanzen. In beiden Diagrammen liegt die Mehrzahl der Punkte unter der Diagonalen. Dies zeigt, dass der neu implementierte Zeiger, der nach jeder Iteration wieder auf den ersten Axiomtyp zeigt, eine bessere Performanz als der vorherige iterationsübergreifende Zeiger mit Sequenz 4 hat. Vor allem bei Instanzen deren Lösung lange dauert wird dies deutlich.

So scheiterte die Sequenz mit iterationsübergreifendem Zeiger bei 183 Instanzen am Timeout und die alternative Heuristik mit 167 Instanzen weniger häufig auf unerfüllbaren Instanzen. Auf erfüllbaren Instanzen zeigt sich mit 164 zu 126 Instanzen ein ähnliches Ergebnis.
Als nächstes wird Sequenz 4 mit Sequenz 1 verglichen. Sequenz 1 konnte von allen getesteten Sequenzen die meisten sowohl erfüllbaren als auch unerfüllbaren Probleminstanzen lösen. Für unerfüllbare Instanzen ist das Streudiagramm in Abbildung 4.14 zu sehen. Man sieht, dass vor allem in der linken Hälfte vermehrt Punkte unter der Diagonalen liegen, was eine geringere Laufzeit von Sequenz 1 auf diesen Instanzen bedeutet. Außerdem konnten mithilfe von Sequenz 1 mehr Instanzen gelöst werden, die durch Sequenz 4 nicht in einer Minute gelöst werden konnten. So liegen auf der oberen Achse 291 Punkte, wohingegen auf der rechten Achse 265 Punkte liegen.
Das Streudiagramm für erfüllbare Instanzen ist in Abbildung 4.15 zu sehen. Hier ist das Verhältnis der Timeouts noch stärker. Auf der oberen Achse liegen 257 Punkte und auf der rechten Achse liegen 116 Punkte. Durch Sequenz 1 konnten also 141 erfüllbare Probleminstanzen mehr gelöst werden. Auch sind wieder mehr Punkte unter der Diagonalen zu sehen. Insgesamt kann durch Sequenz 1 also eine leicht bessere Performanz als durch Sequenz 4 erreicht werden.
Als letztes wird Sequenz 2 mit Sequenz 26 verglichen. Sequenz 2 konnte die wenigsten unerfüllbaren Instanzen lösen und mit Sequenz 26 konnten die wenigsten erfüllbaren Instanzen von allen getesteten Sequenzen gelöst werden.
Zuerst wird die Performanz auf erfüllbaren Instanzen untersucht, die in Abbildung 4.16 zu finden ist. Man sieht, dass Sequenz 26 besser als Sequenz 2 auf unerfüllbaren Instanzen ist. Die meisten Punkte liegen unter der Diagonalen und auf der rechte Achse liegen zusätzlich 496 Punkte, wohingegen auf der oberen Achse 78 liegen. Mit Sequenz 26 konnten also mehr unerfüllbare Instanzen in durchschnittlich kürzerer Zeit gelöst werden.
Auf erfüllbaren Instanzen ist das genau umgekehrt. In Abbildung 4.17 sieht man die beiden Sequenzen in einem Streudiagramm auf erfüllbaren Instanzen. Auf erfüllbaren Instanzen schnitt Sequenz 2 besser ab als Sequenz 26. Die Punkte im Diagramm sind symmetrisch verteilt, was auf eine ähnliche durchschnittliche Laufzeit schließen lässt. Es gibt jedoch viele Instanzen, die nur mit der Verwendung von Sequenz 2 gelöst werden konnten. Von den erfüllbaren Problemen konnten 317 Instanzen nur mit Sequenz 2 und umgekehrt 86 Instanzen nur von Sequenz 26 gelöst werden.
Insgesamt wurde von den getesteten Sequenzen die beste Performanz mit Sequenz 1 erreicht.

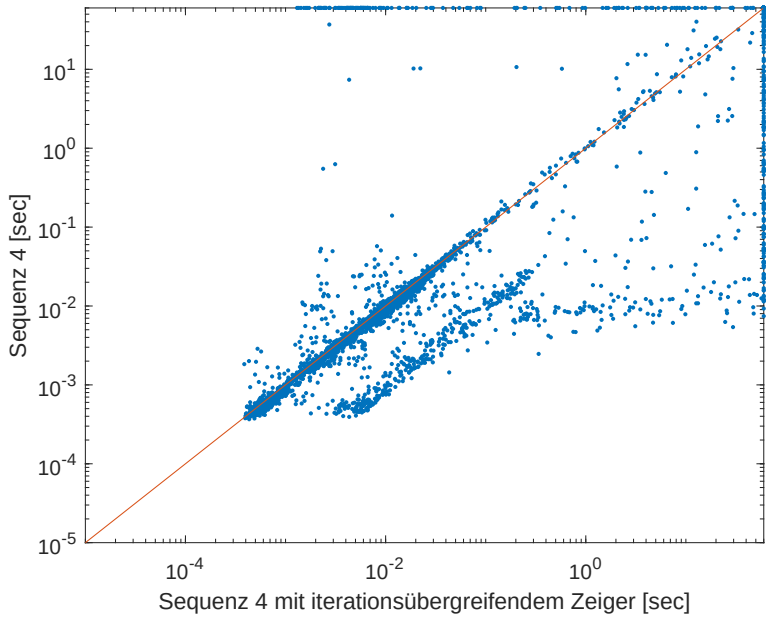

Abbildung 4.12: Streudiagramm für Sequenz 4 mit beiden Zeigern auf unerfüllbaren Instanzen

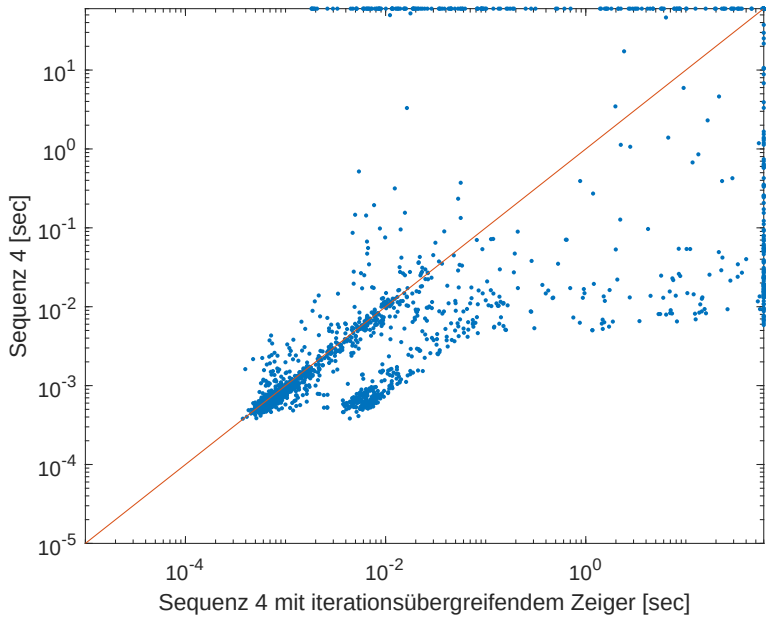

Abbildung 4.13: Streudiagramm für Sequenz 4 mit beiden Zeigern auf erfüllbaren Instanzen

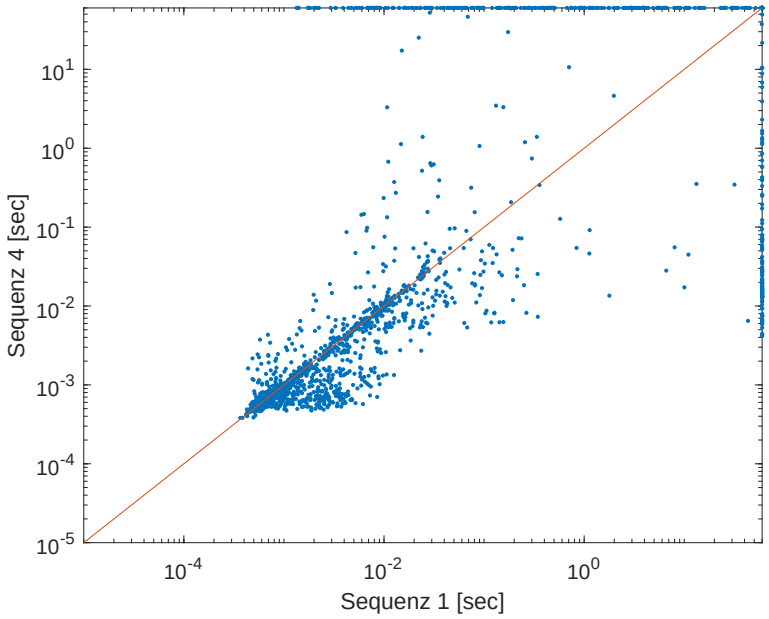

Abbildung 4.14: Streudiagramm zum Vergleich von Sequenz 1 mit Sequenz 4 auf unerfüllbaren Instanzen

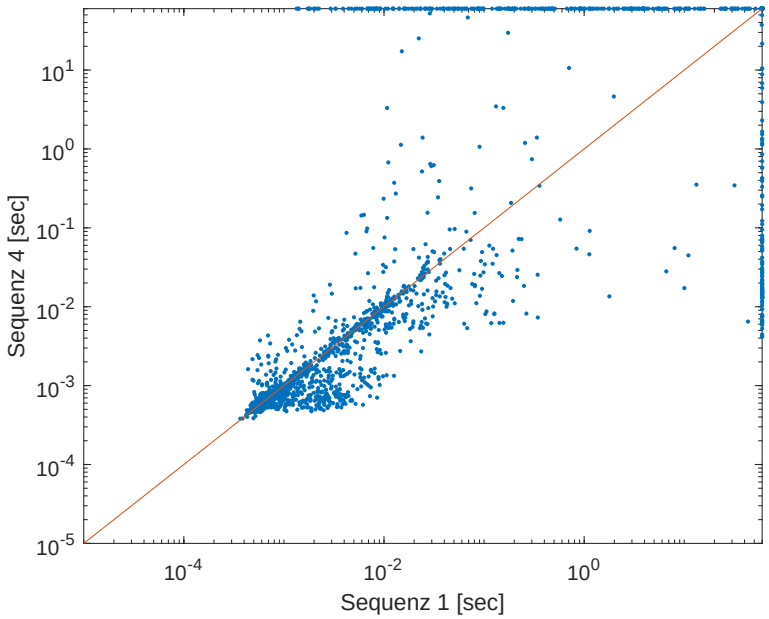

Abbildung 4.15: Streudiagramm zum Vergleich von Sequenz 1 mit Sequenz 4 auf erfüllbaren Instanzen

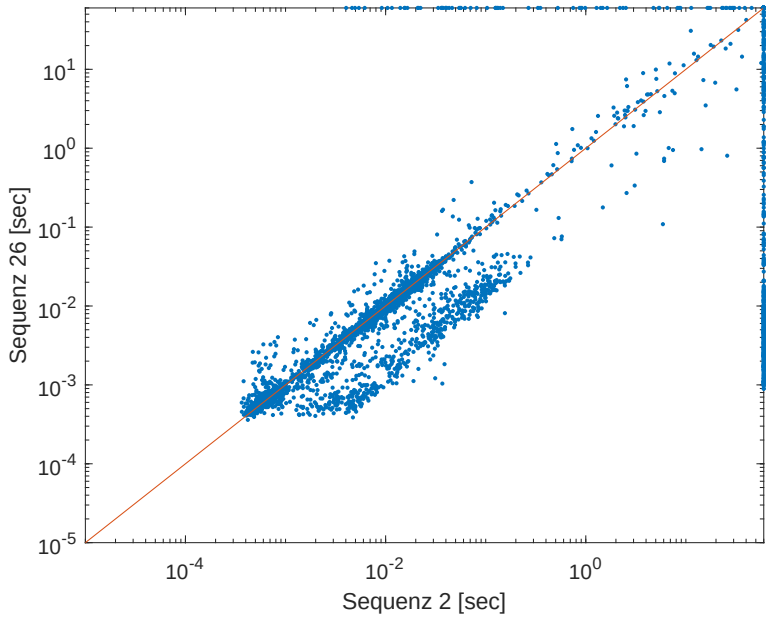

Abbildung 4.16: Streudiagramm zum Vergleich von Sequenz 2 mit Sequenz 26 auf unerfüllbaren Instanzen

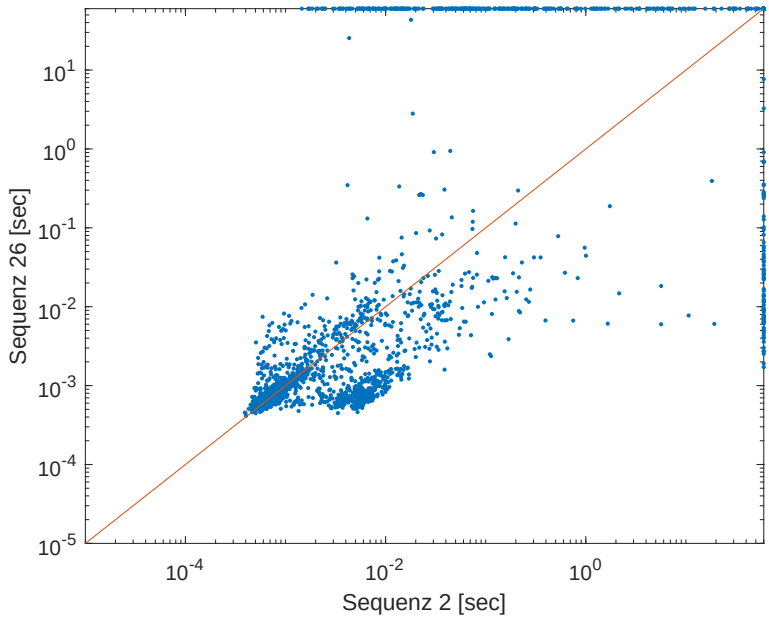

Abbildung 4.17: Streudiagramm zum Vergleich von Sequenz 2 mit Sequenz 26 auf erfüllbaren Instanzen

Kapitel 5

Fazit

5.1 Zusammenfassung

In dieser Arbeit wurde an der Lösung von SMT-Problemen durch inkrementelle Linearisierung gearbeit. Das Verfahren wurde in [1] vorgestellt und im Rahmen der Arbeit [2] als Modul in SMT-RAT integriert. Das bestehende Modul wurde um mehrere Funktionen erweitert.
Es wurde eine Funktion integriert, die versucht ein Modell der linearen Constraints auch in ein Modell der nichtlinearen Constraints umzuwandeln. Außerdem wurde die Möglichkeit zum Aufruf von anderen SMT-Solvern als Submodule implementiert.
In Kapitel 4 wurden die Auswirkungen der neuen Funktionen auf die Performanz des Moduls getestet. Insbesondere das Aufrufen von Submodulen konnte die Performanz des Moduls steigern. Die Anzahl der gelösten Instanzen konnte sowohl durch die STrop als auch die CAD signifikant erhöht werden. Auch das Zurechtziehen konnte insbesondere in der chronologischen Reihenfolge die Performanz verbessern. Diese beiden Methoden haben vor allem Auswirkungen auf das Lösen von erfüllbaren Instanzen.
Die Wahlen der Axiomsequenz und des Zeigers haben Auswirkungen sowohl auf die Lösung von erfüllbaren als auch von unerfüllbaren Instanzen.

5.2 Ausblick

Aktuell verwendet das *NRAILModule* in SMT-RAT nur Gleitkommazahlen als Wertebereich für Variablen. Dadurch können reell-algebraische SMT-Probleme, bei denen eine Variable einen irrationalen Wert annehmen muss um alle Constraints zu erfüllen, nicht gelöst werden. Dies ist zum Beispiel bei Constraints der Art $x^2 = 2$ der Fall. Deshalb könnten in der Zukunft algebraische Zahlen eingebunden werden, damit Variablen auch irrationale Werte annehmen können um mehr Probleme lösen zu können.
Eine weitere Möglichkeit für zukünftige Arbeiten ist die bessere Einbindung der zylindrisch algebraischen Zerlegung. Wenn sie aktuell verwendet wird, macht sie Verfeinerungen überflüssig, da die CAD als vollständiges Lösungsverfahren für SMT-Probleme immer ein Ergebnis liefert. Sie ist dadurch aber auch ein rechenintensives Verfahren. Man könnte deshalb versuchen die CAD wie ein Axiom zu verwenden. Anstatt sie anstelle von jeder Verfeinerung zu nutzen würde man sie dann nur manchmal aufrufen. Alternativ kann auch versucht werden die CAD parallel zur Verfeinerung aufzurufen.

Dadurch kann man das Problem umgehen, dass die CAD nach einem Aufruf viel Zeit für die Lösung benötigt und damit die weitere Ausführung des Programms verzögert. Neben der CAD und der STrop können auch noch weitere Lösungsverfahren für SMT-Probleme aufgerufen werden, wie zum Beispiel die virtuelle Substitution.
Beim Zurechtziehen des Modells ist es möglich noch den Einfluss der Ordnung der Variablen zu untersuchen. Bisher wurde nur die chronologische Reihenfolge betrachtet, jedoch könnte man auch versuchen zuerst die bei der Linearisierung generierten Abstraktionsvariablen zurechtzuziehen. Weitere Probleme beim Zurechtziehen gibt es, wenn die anzupassende Variable nach Einsetzen der Modellwerte für die übrigen Variablen durch Koeffizienten mit dem Wert null eliminiert wird.
Bei den Axiomsequenzen gibt es auch noch die Möglichkeit, beliebige andere Sequenzen zu definieren und deren Performanz zu testen.

Literaturverzeichnis

[1] Alessandro Cimatti, Alberto Griggio, Ahmed Irfan, Marco Roveri und Roberto Sebastiani: *Incremental linearization for Satisfiability and Verification Modulo Nonlinear Arithmetic and Transcendental Functions.* ACM Trans. Comput. Logic, 19(3):19:1–19:52, 2018.

[2] Aklima Zaman: *Incremental linearization for SAT modulo real arithmetic solving.* Masterarbeit, RWTH Aachen, 2019.

[3] Ahmed Irfan: *Incremental Linearization for Satisfiability and Verification Modulo Nonlinear Arithmetic and Transcendental Functions.* Dissertation, University of Trento, Italy, 2018.

[4] Holger H. Hoos und Thomas Stützle. In: *Stochastic Local Search*, The Morgan Kaufmann Series in Artificial Intelligence, Seiten 17 – 21. Morgan Kaufmann, 2005.

[5] Carla P. Gomes, Henry Kautz, Ashish Sabharwal und Bart Selman: *Satisfiability solvers.* In: *Handbook of Knowledge Representation*, Band 3 der Reihe *Foundations of Artificial Intelligence*, Seiten 89 – 134. Elsevier, 2008.

[6] Clark Barrett, Roberto Sebastiani, Sanjit Seshia und Cesare Tinelli: *Satisfiability Modulo Theories.* In: *Handbook of Satisfiability*, Band 185 der Reihe *Frontiers in Artificial Intelligence and Applications*, Seiten 825–885. IOS Press, 2009.

[7] Roberto Sebastiani: *Lazy satisfiability modulo theories.* Journal of Satisfiability, Boolean Modeling and Computation, 3:141–224, 2007.

[8] Florian Corzilius, Gereon Kremer, Sebastian Junges, Stefan Schupp und Erika Ábrahám: *SMT-RAT: An Open Source C++ Toolbox for Strategic and Parallel SMT Solving.* In: *Theory and Applications of Satisfiability Testing – SAT 2015*, Seiten 360–368. Springer International Publishing, 2015.

[9] Florian Corzilius, Ulrich Loup, Sebastian Junges und Erika Abraham: *SMT-RAT: An SMT-Compliant Nonlinear Real Arithmetic Toolbox (Tool Presentation).* In: *Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'12)*, Band 7317 der Reihe *LNCS*, Seiten 442–448. Springer Berlin Heidelberg, 2012.

[10] Clark Barrett, Pascal Fontaine und Cesare Tinelli: *The Satisfiability Modulo Theories Library (SMT-LIB).* `www.SMT-LIB.org`, 2016.

[11] George E. Collins: *Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition–preliminary Report.* SIGSAM Bull., 8(3):80–90, 1974.

[12] Pascal Fontaine, Mizuhito Ogawa, Thomas Sturm und Xuan Tung Vu: *Subtropical Satisfiability.* In: *Frontiers of Combining Systems*, Seiten 189–206. Springer International Publishing, 2017.

[13] Volker Weispfenning: *Quantifier Elimination for Real Algebra — the Quadratic Case and Beyond.* Applicable Algebra in Engineering, Communications and Computing, 8:85–101, 1997.

[14] Sicun Gao, Malay Ganai, Franjo Ivancic, Aarti Gupta, Sriram Sankaranarayanan und Edmund M. Clarke: *Integrating ICP and LRA solvers for deciding nonlinear real arithmetic problems.* In: *Proceedings of 10th International Conference on Formal Methods in Computer-Aided Design*, Seiten 81 – 89, 2010.