Friedrich-Alexander-Universität
Erlangen-Nürnberg

T.cs
Chair for Computer Science 8
Theoretical Computer Science

SMT-basierte Analyse von Konsistenzregeln für digitale Formulare

Bachelorarbeit in der Informatik

Julian Pollinger

Betreuer:

Prof. Dr. Lutz Schröder Merlin Humml

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Erlangen, 23. Februar 2023

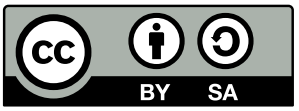
Disclaimer

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 23. Februar 2023 _____

Julian Pollinger

Lizensierung



Abstract

Verwendet man eine große Menge von Regeln zum Evaluieren der Gültigkeit von Daten, kann es sehr nützlich sein, eine Möglichkeit zu haben, diese Regelmengen zu analysieren. Dazu gehören z. B. die Prüfung auf Erfüllbarkeit solcher Mengen und ob einzelne Regeln durch andere impliziert werden. Diese Bachelorarbeit definiert zunächst die RITA-Sprache, eine Sprache, die wir bei Educorvi verwenden, um Regelmengen zur Evaluation von Formularen zu spezifizieren. Weiterhin wird erklärt, wie mit Hilfe von SMT und dem SMT-Solver CVC5 ein Prototyp eines Kommandozeilenwerkzeugs entwickelt wurde, das unter anderem die oben genannten Prüfungen durchführen kann. Zudem wird untersucht, ob dieses für einen produktiven Einsatz geeignet ist.

Inhaltsverzeichnis

1 Einleitung

Digitale Formulare sind allgegenwärtig. Ob es sich dabei um Anträge im Berufsleben oder das Registrieren bei einer Online Plattform handelt, wir kommen ständig mit ihnen in Berührung. Daraus ergibt sich natürlich auch eine große Menge Formulare, die ausgewertet werden muss. Wir bei Educorvi kamen deshalb auf die Idee, unseren Kunden eine Möglichkeit an die Hand zu geben, Formulare automatisiert anhand vorher definierter Regeln auswerten zu können.

Die erste Hürde hierbei war, eine Notation für diese Regeln zu finden. Im Vordergrund stehen sollte dabei eine möglichst simple und auch für einen Laien intuitive Notation sowie natürlich die Abdeckung aller von uns benötigten Funktionen. Hierbei erwiesen sich insbesondere Arithmetik über reelle Zahlen und Berechnungen mit Zeitstempel, wie zum Beispiel der Abstand zwischen zwei Zeitpunkten als problematisch. Am geeignetsten erschien uns hierbei das Projekt RuleML, das die Notation von Regeln in XML ermöglichte und mit MathML sogar um Arithmetik erweitert werden konnte. Allerdings war das Projekt unzureichend dokumentiert (zum Zeitpunkt dieser Arbeit ist außerdem die gesamte Webseite[1] vom Netz genommen), was, gemeinsam mit der schieren Größe des Projekts, der Voraussetzung der einfachen Erlernbarkeit widersprach. Andere Projekte scheiterten entweder daran, dass sie keine Unterstützung für die von uns benötigten mathematischen Funktionen hatten, zu komplex zu erlernen waren oder dass schlicht keine (zeitgemäße) Implementierung zum Auswerten der Regeln existierte.

Deshalb entschieden wir uns, unsere eigene simple und auf unsere Bedürfnisse zugeschnittene Regelsprache zu entwickeln: Rita[2] (kurz für „**R**ule **it** **a**ll“). Die Sprache setzt dabei auf JSON auf und hat als Kernkonzept sogenannte Rulesets, bei denen es sich um eine Menge von Regeln handelt. Sind alle Regeln erfüllt, ist auch das entsprechende Ruleset erfüllt. Mehr zur Definition der Sprache findet sich in Kapitel 2.

Bedingt durch den Verwendungszweck des Projekts muss die Sprache alle Funktionen unterstützen, die bei der Bearbeitung von Formularen notwendig werden könnten. Im Folgenden ein grober Überblick:

- **Konstanten**: In den Regeln müssen Konstanten definiert werden können.
- **Atome**: Eine der grundlegendsten Voraussetzungen ist, Daten aus den Formularen auslesen zu können. Dies wird durch sogenannte Atome realisiert. Die Daten können entweder vom Typ String, Number oder Boolean sein. Zudem können Datumsangaben eingelesen werden. Diese müssen dafür als String im ISO 8601 Format [1] angegeben werden (z. B. `2022-12-04T09:38:02.375Z`).
- **Logische Verknüpfungen**: Um Daten vom Typ Boolean, die entweder aus Atomen oder aus Funktionen der Sprache stammen, kombinieren zu können, gibt es die logischen Verknüpfungen $\wedge$, $\vee$ und $\neg$.
- **Vergleiche**: Weiterhin können Daten aller Typen verglichen werden. Vergleiche zwischen unterschiedlichen Datentypen führen dabei zu einem Fehler.
- **Berechnungen**: Es ist möglich, simple Berechnungen durchzuführen. Unterstützte Berechnungen sind Addition, Subtraktion, Multiplikation, Division und Modulo.

[1] `www.ruleml.org`
[2] `https://github.com/educorvi/rita`

- **Berechnungen mit Zeitstempeln**: Dies umfasst im wesentlichen zwei Funktionen:
  1. Subtraktion zweier Zeitstempel, um das Zeitintervall zwischen diesen zu berechnen
  2. Addition oder Subtraktion eines Zeitintervalls auf/von einen/einem Zeitstempel
- **Quantoren**: Enthalten die übergebenen Daten ein Array von Werten bzw. Objekten, kann überprüft werden, ob eine Regel für alle oder mindestens ein Element dieses Arrays erfüllt ist.
- **Macros**: Aktuell gibt es zwei Macros:
  1. `now`: Kann in einer Regel angegeben werden, um zum Zeitpunkt der Evaluation durch die aktuelle Kombination aus Datum und Uhrzeit ersetzt zu werden
  2. `length`: Wertet zur Länge des übergebenen Strings aus
- **Plugins**: Plugins dienen dazu, dem Nutzer die Möglichkeit zu geben, Regeln mit eigenen Funktionen zu erweitern. Das Konzept dahinter ist, dass das Plugin die Daten zu Beginn der Auswertung übergeben bekommt, um sie dann mit eigenen Ergebnissen anreichern zu können, welche dann wiederum im weiteren Verlauf der Regel zur Verfügung stehen.

Zusätzlich zur Sprache der Regeln braucht es auch noch eine Implementierung, die in der Lage ist, Daten gegen diese Regeln auszuwerten. Diese wurde in TypeScript umgesetzt und als Bibliothek auf npm[3] veröffentlicht. Die Wahl der Sprache ermöglicht es uns, das Projekt sowohl serverseitig mit NodeJS[4] verwenden zu können, als auch nativ in eine Webanwendung einzubinden. So kann zum Beispiel einem Nutzer schon clientseitig Auskunft über die Gültigkeit der Daten gegeben werden.

Aufbauend auf dieser Bibliothek existiert zusätzlich noch eine weitere Bibliothek zum einfachen Persistieren von Rulesets in einer Datenbank sowie ein Server, der das Speichern und Bearbeiten von Rulesets und das Evaluieren von Daten gegen selbige über eine REST-API ermöglicht.

Im Rahmen dieser Arbeit soll das Rita Projekt zusätzlich um ein Tool erweitert werden, das in der Lage ist, die in Rita formulierten Rulesets auf SMT (Satisfiability Modulo Theories) Probleme abzubilden, um sie so mit Hilfe existierender SMT Solver analysieren zu können.

Hinweis

Da RITA stetig weiterentwickelt wird, wird für diese Arbeit die Version der Sprache auf Branch `alpha`[5] unter Ausschluss von Quantoren, Macros und Plugins festgesetzt.

[3] `https://www.npmjs.com/package/@educorvi/rita`
[4] `https://nodejs.org`
[5] Commithash zum Zeitpunkt der Abgabe: 85bd96ce893967a898f1e7cff3edf168bafa744f

2 Ritas Sprache

In diesem Kapitel soll nun die Sprache deiniert werden, in der Regeln für Rita formuliert werden können.

2.1 Syntax

Die Sprache baut syntaktisch auf JSON auf. Das ist eine Syntax, die der Notation von Objekten in JavaScript ähnelt. JSONs Zielsetzung ist es, die Repräsentation von strukturierten Daten zu ermöglichen [2]. Der genaue Aufbau dieser Sprache ist im Standard ECMA-404 [2] definiert, doch hier eine kurze Zusammenfassung:
JSON kennt grundsätzlich zwei Konstrukte: Objekte und Arrays. Ein Objekt ist eine Menge aus Schlüssel-Wert-Paaren. Der Schlüssel ist hierbei ein String, der Wert kann vom Typ String, Number, Boolean, Array, Object oder null sein. Das Objekt wird durch geschweifte Klammern begrenzt, die Schlüssel-Wert-Paare werden durch Kommata separiert und die Zuweisung eines Werts zu einem Schlüssel geschieht mit einem Doppelpunkt. Strings, insbesondere auch die Schlüssel, werden in Anführungszeichen gesetzt. Ein Array ist eine Menge aus Werten, die einen der eben erwähnten Werte-Typen haben und wird durch eckige Klammern begrenzt, die Werte werden durch Kommata separiert. Die Wurzel eines JSON-Dokuments muss ein Objekt oder ein Array sein. Ein Beispiel für ein einfaches JSON-Dokument ist in Listing 1 zu sehen.

Die Rita Sprache ist durch ein JSON-Schema definiert[1]. Ein JSON-Schema ist ein in JSON geschriebenes Metadokument, das die Struktur anderer JSON-Dokumente beschreibt [3]. Die Struktur der Sprache ist dabei wie folgt: Gesendete Daten werden immer gegen ein sogenanntes Ruleset ausgewertet. Das ist eine Menge von Regeln, die diese Daten erfüllen müssen. Eine solche

[1] Das Schema ist unter `https://github.com/educorvi/rita/tree/alpha/rita-core/src/schema` zu finden. Der Einstieg ist die Datei `schema.json`

```json
{
    "schlüssel1": "Wert",
    "schlüssel2": 42,
    "schlüsselFürEinArray": [1, 2, 3.14],
    "schlüsselFürEinObjekt": {
        "schlüssel3": "Ein anderer Wert"
    }
}
```

Listing 1: Beispiel für ein JSON-Objekt

Regel kann in Backus Naur Form wie folgt beschrieben werden:

$$
\begin{aligned}
Rule &::= Formula \\
Formula &::= a \mid Connectives \mid Comparison && (a \in \mathcal{A}) \\
Connectives &::= and(Formula, Formula) \mid or(Formula, Formula) \mid not(Formula) \\
Comparison &::= comp_{=}(CA, CA) \mid comp_{<}(CA, CA) \mid comp_{>}(CA, CA) \mid \\
&\quad comp_{\leq}(CA, CA) \mid comp_{\geq}(CA, CA) \\
CA &::= a \mid c \mid Calculation \mid DateCalculation && (a \in \mathcal{A}, c \in \mathcal{C}) \\
Calculation &::= calc_{+}(CA, CA) \mid calc_{-}(CA, CA) \mid \\
&\quad calc_{*}(CA, CA) \mid calc_{\div}(CA, CA) \mid calc_{\%}(CA, CA) \\
DateUnit &::= seconds \mid minutes \mid hours \mid days \mid months \mid years \\
DateCalculation &::= dateCalc_{+}(CA, CA, DateUnit) \mid dateCalc_{-}(CA, CA, DateUnit)
\end{aligned}
$$

Die Signatur der Sprache ist dabei $\Sigma = (S_0, S, F)$, wobei

$$
\begin{aligned}
S_0 &= \emptyset \\
S &= \{Bool, String, Date, Number, DurationType, Error\} \\
F &= \{ \\
&\quad comp_{Bool,=} : Bool \times Bool \to Bool, \\
&\quad comp_{Bool,<} : Bool \times Bool \to Bool, \\
&\quad comp_{Bool,>} : Bool \times Bool \to Bool, \\
&\quad comp_{Bool,\leq} : Bool \times Bool \to Bool, \\
&\quad comp_{Bool,\geq} : Bool \times Bool \to Bool, \\
&\quad comp_{String,=} : String \times String \to Bool, \\
&\quad comp_{String,<} : String \times String \to Bool, \\
&\quad comp_{String,>} : String \times String \to Bool, \\
&\quad comp_{String,\leq} : String \times String \to Bool, \\
&\quad comp_{String,\geq} : String \times String \to Bool, \\
&\quad comp_{Date,=} : Date \times Date \to Bool, \\
&\quad comp_{Date,<} : Date \times Date \to Bool, \\
&\quad comp_{Date,>} : Date \times Date \to Bool, \\
&\quad comp_{Date,\leq} : Date \times Date \to Bool, \\
&\quad comp_{Date,\geq} : Date \times Date \to Bool, \\
&\quad comp_{Number,=} : Number \times Number \to Bool, \\
&\quad comp_{Number,<} : Number \times Number \to Bool, \\
&\quad comp_{Number,>} : Number \times Number \to Bool, \\
&\quad comp_{Number,\leq} : Number \times Number \to Bool, \\
&\quad comp_{Number,\geq} : Number \times Number \to Bool, \\
&\quad comp_{Mixed,=} : M \times N \to Error, && M, N \in S; M \neq N \\
&\quad comp_{Mixed,<} : M \times N \to Error, && M, N \in S; M \neq N \\
&\quad comp_{Mixed,>} : M \times N \to Error, && M, N \in S; M \neq N \\
&\quad comp_{Mixed,\leq} : M \times N \to Error, && M, N \in S; M \neq N \\
&\quad comp_{Mixed,\geq} : M \times N \to Error, && M, N \in S; M \neq N
\end{aligned}
$$

$$
\begin{aligned}
&and : Bool \times Bool \to Bool,\\
&or : Bool \times Bool \to Bool,\\
&not : Bool \to Bool\\
&calc_{+} : Number \times Number \to Number,\\
&calc_{-} : Number \times Number \to Number,\\
&calc_{\div} : Number \times Number \to Number + Error,\\
&calc_{\%} : Number \times Number \to Number + Error,\\
&dateCalc_{(Date \times Date),-} : Date \times Date \times DurationType \to Number,\\
&dateCalc_{(Date \times Number),+} : Date \times Number \times DurationType \to Date,\\
&dateCalc_{(Date \times Number),-} : Date \times Number \times DurationType \to Date,\\
&dateCalc_{(Number \times Date),+} : Number \times Date \times DurationType \to Date,\\
&getTime : Date \to Number,\\
&getTime^{-1} : Number \to Date,\\
&convertInterval : Number \times DurationType \times DurationType \to Number
\end{aligned}
$$

}

2.2 Semantik

Sei R ein Ruleset, und r eine Regel. Die Interpretation eines Rulesets geschieht rekursiv nach folgendem Ablauf:

$$
\begin{aligned}
\eta &= \text{ Umgebung aus den gesendeten Daten in Form von Schlüssel-Wert Paaren}\\
\eta(x) &= \text{ Der Wert für das Datum mit dem Schlüssel } x \text{ in den Daten}\\
\varepsilon &= \text{ Error}\\
C &= \text{ Menge der Konstanten}\\
\mathfrak{M}[\![R]\!]\eta &\in \{\top, \bot, \varepsilon\}\\
\mathfrak{M}[\![r]\!]\eta &\in \{\top, \bot, \varepsilon\}\\
\mathfrak{M}[\![R]\!]\eta &= \begin{cases} \varepsilon & \text{wenn } \exists r \in R.\mathfrak{M}[\![r]\!]\eta = \varepsilon \\ \top & \text{sonst wenn } \forall r \in R.\mathfrak{M}[\![r]\!]\eta = \top \\ \bot & \text{sonst} \end{cases}\\
\mathfrak{M}[\![f(arg_1, ..., arg_n)]\!]\eta &= \begin{cases} \varepsilon & \text{wenn } \mathfrak{M}[\![arg_i]\!]\eta = \varepsilon \\ \mathfrak{M}[\![f]\!](\mathfrak{M}[\![arg_1]\!]\eta, ..., \mathfrak{M}[\![arg_n]\!]\eta) & \text{sonst} \end{cases}\\
\mathfrak{M}[\![x]\!]\eta &= \begin{cases} \eta(x) & \text{wenn } x \text{ in } \eta \text{ definiert} \\ \varepsilon & \text{sonst} \end{cases}
\end{aligned}
$$

Für $c \in C$:

$$\mathfrak{M}[\![c]\!]\eta = c$$

Wobei für
S als die Menge aller Strings,
D als die Menge aller Zeitstempel und
um als eine Umrechnungsmatrix für Zeitintervalle (siehe Appendix A.2)
die Symbole der Signatur durch $\mathfrak{M}$ wie folgt interpretiert werden:

$$\begin{aligned}
\text{Träger } M =& \{\top, \bot, \varepsilon\} \cup S \cup D \cup \mathbb{R} \\
\mathfrak{M}[\![Bool]\!] =& \{\top, \bot\} \\
\mathfrak{M}[\![String]\!] =& S \\
\mathfrak{M}[\![Date]\!] =& D \\
\mathfrak{M}[\![Number]\!] =& \mathbb{R} \\
\mathfrak{M}[\![DurationType]\!] =& \{milliseconds, seconds, minutes, hours, days, months, years\} \\
\mathfrak{M}[\![Error]\!] =& \{\varepsilon\}
\end{aligned}$$

Die Funktionen *and* und *or* werden faul ausgewertet, bei Vergleichen von Booleans gilt $\top > \bot$.

$$\begin{aligned}
\mathfrak{M}[\![and]\!](a,b) =& \begin{cases} b & \text{wenn } a = \top \\ \bot & \text{sonst} \end{cases} \\
\mathfrak{M}[\![or]\!](a,b) =& \begin{cases} \top & \text{wenn } a = \top \\ b & \text{sonst} \end{cases} \\
\mathfrak{M}[\![not]\!](a) =& \neg a \\
\mathfrak{M}[\![comp_{Bool,=}]\!](a,b) =& (a \wedge b) \vee (\neg a \wedge \neg b) \\
\mathfrak{M}[\![comp_{Bool,<}]\!](a,b) =& \neg a \wedge b \\
\mathfrak{M}[\![comp_{Bool,>}]\!](a,b) =& a \wedge \neg b \\
\mathfrak{M}[\![comp_{Bool,\leq}]\!](a,b) =& \mathfrak{M}[\![comp_{Bool,<}(a,b)]\!] \vee \mathfrak{M}[\![comp_{Bool,=}(a,b)]\!] \\
\mathfrak{M}[\![comp_{Bool,\geq}]\!](a,b) =& \mathfrak{M}[\![comp_{Bool,>}(a,b)]\!] \vee \mathfrak{M}[\![comp_{Bool,=}(a,b)]\!]
\end{aligned}$$

Vergleiche zwischen Daten mit unterschiedlichen Datentypen werten stets zu einem Fehler aus.

$$\begin{aligned}
\mathfrak{M}[\![comp_{Mixed,=}]\!](a,b) =& \varepsilon \\
\mathfrak{M}[\![comp_{Mixed,<}]\!](a,b) =& \varepsilon \\
\mathfrak{M}[\![comp_{Mixed,>}]\!](a,b) =& \varepsilon \\
\mathfrak{M}[\![comp_{Mixed,\leq}]\!](a,b) =& \varepsilon \\
\mathfrak{M}[\![comp_{Mixed,\geq}]\!](a,b) =& \varepsilon
\end{aligned}$$

Bei Vergleichen zwischen Strings wird die lexikographische Ordnung verwendet.

$$\mathfrak{M}[\![comp_{String,=}]\!](a,b) = \begin{cases} \top & \text{wenn } a \text{ zeichenweise identisch zu } b \text{ ist} \\ \bot & \text{sonst} \end{cases}$$

$$\mathfrak{M}[\![comp_{String,<}]\!](a,b) = \begin{cases} \top & \text{wenn } a \text{ lexikographisch kleiner als } b \text{ ist} \\ \bot & \text{sonst} \end{cases}$$

$$\mathfrak{M}[\![comp_{String,>}]\!](a,b) = \begin{cases} \top & \text{wenn } a \text{ lexikographisch größer als } b \text{ ist} \\ \bot & \text{sonst} \end{cases}$$

$$\mathfrak{M}[\![comp_{String,\leq}]\!](a,b) = \mathfrak{M}[\![comp_{String,<}(a,b)]\!] \vee \mathfrak{M}[\![comp_{String,=}(a,b)]\!]$$
$$\mathfrak{M}[\![comp_{String,\geq}]\!](a,b) = \mathfrak{M}[\![comp_{String,>}(a,b)]\!] \vee \mathfrak{M}[\![comp_{String,=}(a,b)]\!]$$

Berechnungen und Vergleiche mit Zahlen folgen den für reelle Zahlen üblichen Interpretationen, wobei Divisionen oder Moduloberechnungen mit Divisor 0 in einem Fehler resultieren.

$$\mathfrak{M}[\![comp_{Number,=}]\!](a,b) = \begin{cases} \top & \text{wenn } (a = b) \\ \bot & \text{sonst} \end{cases}$$

$$\mathfrak{M}[\![comp_{Number,<}]\!](a,b) = \begin{cases} \top & \text{wenn } (a < b) \\ \bot & \text{sonst} \end{cases}$$

$$\mathfrak{M}[\![comp_{Number,>}]\!](a,b) = \begin{cases} \top & \text{wenn } (a > b) \\ \bot & \text{sonst} \end{cases}$$

$$\mathfrak{M}[\![comp_{Number,\leq}]\!](a,b) = \mathfrak{M}[\![comp_{Number,<}(a,b)]\!] \vee \mathfrak{M}[\![comp_{Number,=}(a,b)]\!]$$
$$\mathfrak{M}[\![comp_{Number,\geq}]\!](a,b) = \mathfrak{M}[\![comp_{Number,>}(a,b)]\!] \vee \mathfrak{M}[\![comp_{Number,=}(a,b)]\!]$$
$$\mathfrak{M}[\![calc_{+}]\!](a,b) = a + b$$
$$\mathfrak{M}[\![calc_{-}]\!](a,b) = a - b$$
$$\mathfrak{M}[\![calc_{*}]\!](a,b) = a \cdot b$$

$$\mathfrak{M}[\![calc_{\div}]\!](a,b) = \begin{cases} a \div b & \text{wenn } b \text{ nicht 0 ist} \\ \varepsilon & \text{sonst} \end{cases}$$

$$\mathfrak{M}[\![calc_{\%}]\!](a,b) = \begin{cases} a - (\lfloor \frac{a}{b} \rfloor \cdot b) & \text{wenn } b \text{ nicht 0 ist} \\ \varepsilon & \text{sonst} \end{cases}$$

Vergleiche zwischen Zeitstempeln vergleichen den tatsächlichen Zeitpunkt, wobei ein Zeitpunkt kleiner ist als ein anderer, wenn er vor ihm liegt.

$$\mathfrak{M}[\![comp_{Date,=}]\!](a,b) = \begin{cases} \top & \text{wenn der Zeitpunkt } a \text{ zeitgleich zum Zeitpunkt } b \text{ ist} \\ \bot & \text{sonst} \end{cases}$$

$$\mathfrak{M}[\![comp_{Date,<}]\!](a,b) = \begin{cases} \top & \text{wenn der Zeitpunkt } a \text{ vor dem Zeitpunkt } b \text{ ist} \\ \bot & \text{sonst} \end{cases}$$

$$\mathfrak{M}[\![comp_{Date,>}]\!](a,b) = \begin{cases} \top & \text{wenn der Zeitpunkt } a \text{ nach dem Zeitpunkt } b \text{ ist} \\ \bot & \text{sonst} \end{cases}$$

$$\mathfrak{M}[\![comp_{Date,\leq}]\!](a,b) = \mathfrak{M}[\![comp_{Date,<}(a,b)]\!] \vee \mathfrak{M}[\![comp_{Date,=}(a,b)]\!]$$
$$\mathfrak{M}[\![comp_{Date,\geq}]\!](a,b) = \mathfrak{M}[\![comp_{Date,>}(a,b)]\!] \vee \mathfrak{M}[\![comp_{Date,=}(a,b)]\!]$$

Bei Berechnungen mit Zeitangaben werden Datumsangaben zunächst in die seit dem Epoch (1. Januar 1970, 00:00.000 Uhr UTC) [4] vergangene Zeit in Millisekunden umgerechnet. Mit der resultierenden Zahl kann dann der Abstand zwischen zwei Zeitpunkten durch normale Subtraktion der beiden Zahlen berechnet werden. Die Addition oder Subtraktion eines Zeitintervalls auf eine Datumsangabe erfolgt durch Addition oder Subtraktion des Intervalls in Millisekunden auf den in Millisekunden repräsentierten Zeitstempel und anschließendes Anwenden der Funktion $getTime^{-1}$, um aus der Zahl wieder eine Datumsangabe zu erhalten.

$$\mathfrak{M}[\![getTime]\!](a) = \text{Anzahl der vom Jan 1, 1970, 00:00:00.000 GMT bis a verstrichenen Millisekunden}$$

$$\mathfrak{M}[\![getTime^{-1}]\!](a) = \text{Inverse Funktion zu getTime}$$

$$\mathfrak{M}[\![convertInterval]\!](n, t_1, t_2) = \begin{cases} n \cdot um[t_1][t_2] & \text{wenn } t_1 \text{ größer als } t_2 \text{ ist, wobei gilt:} \\ & milliseconds < seconds < minutes < \\ & hours < days < months < years \\ n \div um[t_2][t_1] & \text{sonst} \end{cases}$$

$$\mathfrak{M}[\![dateCalc_{(Date \times Date),-}]\!](a, b, t) = \mathfrak{M}[\![convertInterval(calc_-(getTime(b), getTime(a)), \text{"}milliseconds\text{"}, t)]\!]$$

$$\mathfrak{M}[\![dateCalc_{(Date \times Number),+}]\!](a, b, t) = \mathfrak{M}[\![getTime^{-1}(calc_+(getTime(a), convertInterval(b, t, \text{"}milliseconds\text{"})))]\!]$$

$$\mathfrak{M}[\![dateCalc_{(Date \times Number),-}]\!](a, b, t) = \mathfrak{M}[\![getTime^{-1}(calc_-(getTime(a), convertInterval(b, t, \text{"}milliseconds\text{"})))]\!]$$

$$\mathfrak{M}[\![dateCalc_{(Number \times Date),+}]\!](a, b, t) = \mathfrak{M}[\![dateCalc_{(Date \times Number),+}]\!](b, a, t)$$

2.3 Beispiel

Nun folgt ein simples Beispiel, in welchem das Ruleset nur eine Regel enthält, die überprüfen soll, ob jemand zum Zeitpunkt eines Kaufs mindestens 18 Jahre alt und nicht angestellt ist. Bei der Evaluation werden folgende Daten übergeben:
geburtsdatum (Datum), **kaufdatum** (Datum), **istAngestellter** (Boolean)

Verwendet man für diese Regel die Notation aus Abschnitt 2.2, sähe die Formel wie folgt aus:

$$\begin{aligned} &and(\\ &\quad not(istAngestellter), \\ &\quad comp_{Number,\geq}(dateCalc_{(Date \times Date),-}(kaufdatum, geburtsdatum, years), 18) \\ &) \end{aligned}$$

Eine Visualisierung der Regel findet sich in Abbildung 2.1. Die hierarchische Struktur einer Regel wird dort gut ersichtlich. Die tatsächliche Umsetzung dieser Regel in der Rita-Sprache ist in Listing 2 zu sehen.

An diesem Beispiel wird die tatsächliche JSON Syntax gut ersichtlich: Es gibt ein Wurzelobjekt, in dem das Schema referenziert wird und das ein Array mit Regeln enthält. Jede dieser Regeln ist selbst wieder ein Objekt mit Werten für eine eindeutige ID der Regel (`id`), einem optionalen Kommentar (`comment`) und natürlich die Regel selbst (`rule`). Die Regel entspricht dem in Abschnitt 2.1

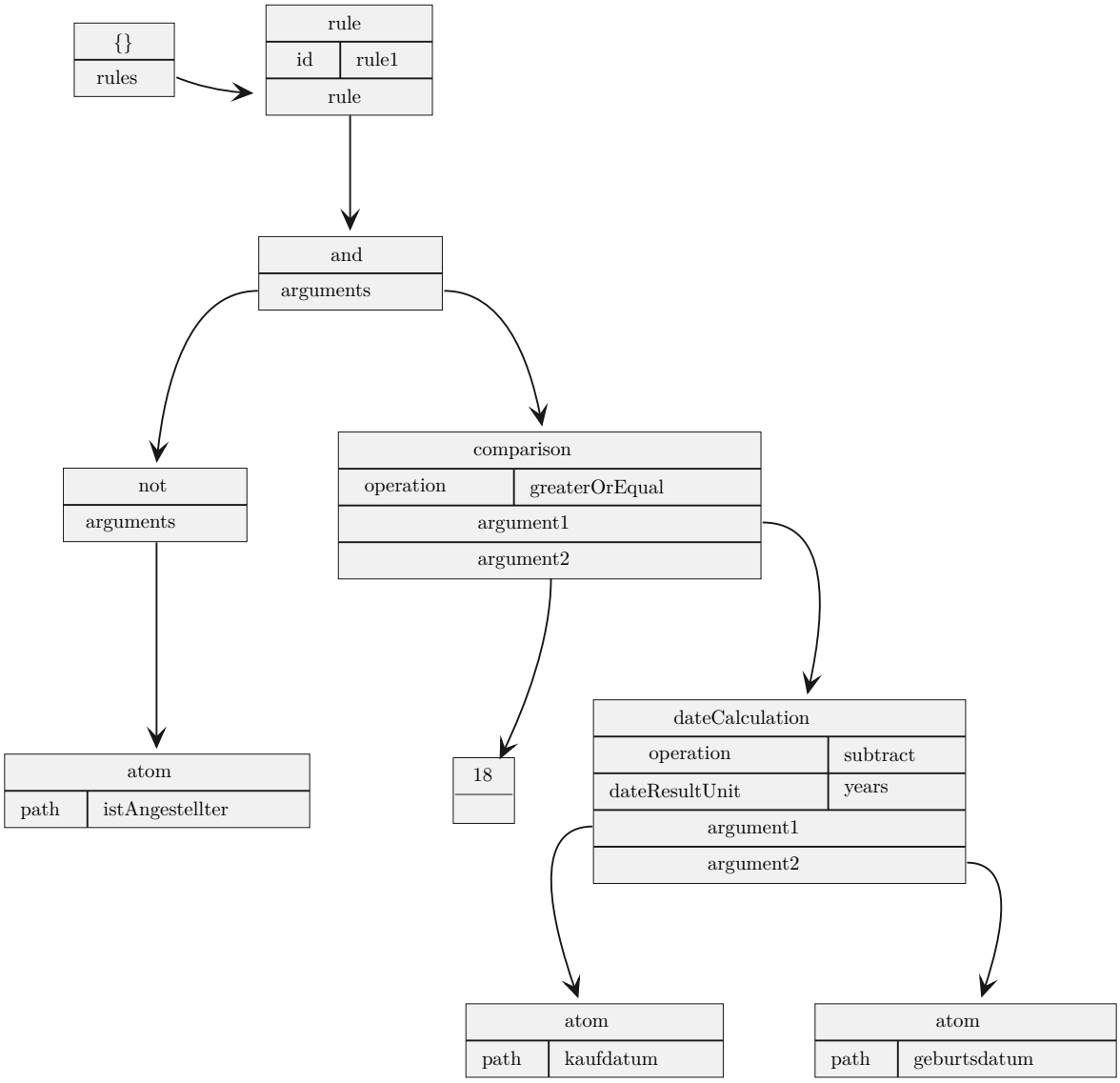


Abbildung 2.1: Visualisierung des Rulesets

```
{
    "$schema": "https://raw.githubusercontent.com/educorvi/rita/alpha/rita-core/src/schema/schema.json",
    "rules": [
        {
            "id": "rule1",
            "rule": {
                "type": "and",
                "arguments": [
                    {
                        "type": "not",
                        "arguments": [
                            {
                                "type":"atom",
                                "path": "istAngestellter"
                            }
                        ]
                    },
                    {
                        "type": "comparison",
                        "operation": "greaterOrEqual",
                        "arguments": [
                            {
                                "type": "dateCalculation",
                                "dateResultUnit": "years",
                                "operation": "subtract",
                                "arguments": [
                                    {
                                        "type": "atom",
                                        "path": "kaufdatum",
                                        "isDate": true
                                    },
                                    {
                                        "type": "atom",
                                        "path": "geburtsdatum",
                                        "isDate": true
                                    }
                                ]
                            },
                            18
                        ]
                    }
                ]
            }
        }
    ]
}
```

Listing 2: Beispielhafte Regel in der Rita-Sprache

beschriebenen Aufbau, wobei unter `type` der Typ (z. B. `and` oder `comparison`) und unter `arguments` die Argumente einer Formel angegeben werden. Gegebenenfalls werden noch weitere Attribute ergänzt, die für die jeweiligen Elemente notwendig sind, z. B. bei dem `comparison`-Element, da hier zusätzlich zu den beiden Operanden noch die Vergleichsoperation (`operation`) angegeben werden muss. Die Typen der Argumente werden dabei zur Laufzeit überprüft, weshalb es innerhalb des JSON-Dokuments nicht notwendig ist, zwischen beispielsweise $comp_{Number,<}$ und $comp_{String,<}$ zu unterscheiden.

Zunächst mag diese Notation etwas sperrig wirken, sie ist jedoch für den Laien, dem wir mit RITA das Schreiben dieser Regeln erlauben möchten, vermutlich intuitiver zu schreiben und zu verstehen, als zum Beispiel die in Kapitel 3 verwendete SMT Notation. Außerdem ist JSON für unsere Zwecke ein guter Kompromiss zwischen menschlicher und maschineller Lesbarkeit.

3 SMT basierte Analyse von Rulesets

Wenn man nun Regeln in dieser Sprache formuliert, bietet es sich an, eine Möglichkeit zur Analyse von Rulesets zu schaffen. Es könnte zum Beispiel überprüft werden, ob eine solche Menge von Regeln überhaupt erfüllbar ist, oder ob manche Regeln andere implizieren und damit obsolet machen. Um dies zu ermöglichen, sollen Rulesets nun auf SMT Probleme abgebildet werden, welche anschließend mit einem SMT Solver untersucht werden können.

3.1 Satisfiability Modulo Theories

Satisfiability Modulo Theories (SMT) bezeichnet das Problem der Erfüllbarkeit einer Formel in Logik erster Stufe vor dem Hintergrund bestimmter Theorien, die die Interpretation von Symbolen einschränken und so die Entscheidbarkeit dieses Problems gewährleisten [5]. Weiterhin können durch diese Einschränkungen beim Lösen des Problems speziell für diese Theorie entwickelte Lösungsstrategien verwendet werden [6].
Ein Beispiel für eine solche Theorie ist die „Theory of Reals“ der SMT-LIB Initiative, die unter anderem die Interpretation des Funktionensymbols $*$ auf die Multiplikation der beiden übergebenen reellen Zahlen beschränkt [7].

3.2 SMT-LIB

SMT-LIB ist, in den Worten der Projektkoordinatoren, „eine internationale Initiative, die, [...] unterstützt durch zahlreiche Forschungsgruppen weltweit, zum Ziel hat Forschung und Entwicklung im Bereich SMT zu unterstützen“ [8].

Im Fokus steht dabei die Schaffung übergreifender Standards, wie etwa einheitliche Theorien für die SMT Probleme sowie ein einheitliches Format zur Kommunikation mit SMT-Solvern [8]. In eben dieses einheitliche Format soll die Rita Sprache übersetzt werden. So wird ermöglicht, sich nicht auf einen speziellen SMT-Solver zu limitieren, da der aus den Rulesets generierte SMT-LIB Code mit minimalen Veränderungen mit den meisten SMT-Solvern kompatibel ist.

Laut Definition der SMT-LIB Notation [8] werden Formeln dabei in Prefix Notation geschrieben und durch runde Klammern begrenzt. Die Argumente der Funktionen sind durch Leerzeichen separiert. Außerdem gibt es Befehle zum Definieren von Funktionen und Konstanten sowie zum Assertieren von Formeln, neben anderen. Ein Beispiel in SMT-LIB, das die Erfüllbarkeit der Formel $(x \vee y) \wedge x \wedge \neg y$ überprüft, sieht wie folgt aus:

```
1    (set-logic ALL)
2    (declare-const x Bool)
3    (declare-const y Bool)
4    (assert (and (or x y) x (not y)))
5    (check-sat)
```

Zeile 1 besagt dabei, dass der Solver die gesamte SMT-LIB Logik verwenden soll. Üblicherweise empfiehlt es sich, die Logik auf eine Unterlogik einzuschränken, um so spezialisierte und meist

effizientere Algorithmen zum Überprüfen der Erfüllbarkeit verwenden zu können [9]. Zeile 2 und 3 definieren die beiden Konstanten `x` und `y`, die in Zeile 4 in der Formel verwendet werden. Zeile 4 wiederum definiert die Formel, die mit dem Befehl in Zeile 5 auf Erfüllbarkeit überprüft werden soll.

3.3 SMT Solver

Als Solver wird CVC5[1] verwendet, ein Open Source SMT Solver. Um diesen aus dem in Typescript geschriebenen Rita Projekt heraus verwenden zu können, wurde auf das Open Source Projekt node-smtlib[2] des Stanford Open Virtual Assistant Lab aufgebaut. Dieses bietet eine Schnittstelle zur Generierung von SMT-LIB Code nach Standard 2.0 und ermöglicht es, diesen anschließend durch CVC4 ausführen zu lassen. Um die Verwendung von CVC5 und SMT-LIB 2.6 zu ermöglichen, wurde die Bibliothek von uns geforked und in den Rita Stack integriert[3].

CVC5 wird dabei in einem externen Prozess ausgeführt, wobei der generierte SMT-LIB Code über dessen stdin an den Solver übergeben wird. Die Ausgabe des Solvers wird anschließend über stdout des Prozesses an das Tool zurückgeführt, dort mit Hilfe von regulären Ausdrücken geparsed und dem Nutzer der Bibliothek in Form eines Objekts übergeben. Dieses besagt, ob es ein Modell gibt, das die generierten Bedingungen erfüllt (unter dem Schlüssel `satisfiable`) und gibt gegebenenfalls ein Beispiel für ein solches Modell an (unter dem Schlüssel `model`), sofern das Produzieren von Beispielmodellen aktiviert wurde. Ein Beispiel für ein solches Objekt findet sich in Listing 5.

3.4 Übersetzung

Die Übersetzung eines Rulesets nach SMT-LIB soll nun an der folgenden, durch das Tool Rita-SMT erstellten Übersetzung des Beispiels aus Abschnitt 2.3 erläutert werden:

```
1  (set-logic QF_SNIRA)
2  (set-option :produce-assignments true)
3  (set-option :produce-models true)
4  (define-fun % ((a Real) (b Real)) Real (- a (* (to_int (/ a b)) b)))
5  (define-fun <=s ((a String) (b String)) Bool (str.<= a b))
6  (define-fun <s ((a String) (b String)) Bool (and (str.<= a b) (not (= a b))))
7  (define-fun >s ((a String) (b String)) Bool (not (and (str.<= a b) (not (= a b)))))
8  (define-fun >=s ((a String) (b String)) Bool (or (>s a b) (= a b)))
9  (declare-const istAngestellter Bool)
10 (declare-const kaufdatum Real)
11 (declare-const geburtsdatum Real)
12 (assert (and (not istAngestellter) (>= (/ (- (* kaufdatum 1000) (* geburtsdatum 1000))
   ↪  31536000000) 18.0)))
```

Logik In Zeile 1 wird zunächst die vom SMT Solver zu verwendende Logik festgesetzt. Für Rita Rulesets ist das `QF_SNIRA`. `QF` schließt die Verwendung von Quantoren aus [9] und ermöglicht so eine effizientere Lösung der SMT Problematik [6]. `S` inkludiert die „Theory of Strings“ [10] in die Logik, um Vergleiche vom Typ String durchführen zu können[4]. `NIRA` erlaubt die Verwendung

[1] `https://cvc5.github.io/`
[2] `https://github.com/stanford-oval/node-smtlib`
[3] `https://github.com/educorvi/rita/tree/alpha/rita-smt/node-smtlib`
[4] Das ist kaum dokumentiert. Lediglich in der Dokumentation zu CVC4 (`http://cvc4.cs.stanford.edu/wiki/Strings`) und einem Beispiel in der Dokumentation zu CVC5 (`https://cvc5.github.io/docs/cvc5-1.0.2/examples/strings.html`) konnte diese Information gefunden werden.

von nichtlinearer Arithmetik mit reellen und ganzen Zahlen [9].

Optionen Die Zeilen 2 und 3 werden nur benötigt, wenn eine Beispielbelegung generiert werden soll. Sie werden zum Beispiel für die in Abschnitt 3.6 erläuterte Überprüfung auf implizierte Regeln ausgelassen, da eine solche Belegung in diesem Fall nicht von Belang ist und der SMT Solver ohne diese Optionen effizienter arbeiten kann[5].

Funktionen In den Zeilen 4 bis 8 werden mehrere benötigte Funktionen definiert, beginnend mit der Funktion `%`, die die Berechnung von Modulo für reelle Zahlen implementiert. Dies ist notwendig, da SMT-LIB standardmäßig Modulo Berechnungen nur für ganzzahlige Werte spezifiziert. Umgesetzt wird die in Abschnitt 2.2 spezifizierte Modulo Funktion $mod(a, b) = a - (\lfloor \frac{a}{b} \rfloor * b)$. Der Floor Operator wird hierbei durch die `to_int` Funktion repräsentiert, da die „Theory of Reals“ in SMT-LIB keinen Floor Operator unterstützt, die Definition von `to_int` aber identisch zu Floor definiert ist [7].
In Zeilen 5 bis 8 finden sich Funktionen zum lexikographischen Vergleichen von Zeichenketten. SMT-LIB kennt hier nur die Funktion `str.<=`, die ein lexikographisches kleiner-gleich überprüft sowie `=` zur Überprüfung von Gleichheit. In Zeile 5 wird für eine konsistente Benennung zunächst ein Alias für die `str.<=` Funktion festgelegt, in den Zeilen danach werden die anderen Funktionen zum Vergleichen von Zeichenketten aus den beiden gegebenen abgeleitet.

Regeln Jede Regel wird durch eine `assert` Anweisung in SMT-LIB dargestellt. Ein Beispiel dafür ist in Zeile 12 zu sehen. Dem Solver wird mit dieser Anweisung mitgeteilt, dass die entsprechende Regel erfüllt sein muss, damit das gesamte Ruleset erfüllbar ist. Der tatsächliche Inhalt der Regel wird anschließend rekursiv übersetzt, wobei die Übersetzung der einzelnen Elemente in den folgenden Paragraphen erläutert wird.

Konstanten Konstanten können, wenn sie kein Datum sind (mehr dazu im Absatz zu Zeitstempeln), einfach als Literale übernommen werden. Bei Zahlen ist wichtig, dass sie in SMT-LIB auf mindestens eine Dezimale genau angegeben werden, da sie vom Solver sonst als ganze Zahlen interpretiert werden, die dann nicht mehr mit den Atomen vom Typ Real verrechnet werden können. Ein Beispiel für eine Konstante ist die Zahl 18.0 in Zeile 12.

Atome Im Ruleset vorkommende Atome werden in SMT-LIB mit `declare-const` als nullstellige Funktion deklariert. So kann, sollte ein Ruleset erfüllbar sein, vom SMT Solver eine Belegung für die Atome ermittelt werden, die das Ruleset erfüllt. Der Name der Funktion ist hierbei der Pfad des Atoms in den Daten, z. B. `data` oder `person.name`. Enthält der Pfad einen indexierten Zugriff auf ein Array, wie zum Beispiel `values[0]`, wird dieser zu `values.0` umgeschrieben, da eckige Klammern in SMT-LIB nicht erlaubt sind, die beiden Schreibweisen in Typescript aber äquivalent sind. Der Typ dieser Funktion wird aus dem Kontext des Atoms abgeleitet: Befindet es sich in einem logischen Konnektiv, ist es vom Typ `Bool`, ist es in einem Vergleich, ist es vom Typ `Real` oder `String`, je nachdem, ob es mit einer Zahl oder Zeichenfolge verglichen wird. Werden zwei Atome verglichen, wird bei der Übersetzung davon ausgegangen, dass es sich um zwei Reals handelt. Atome, deren Wert vom Typ Zeitstempel ist, werden als Real deklariert (mehr dazu im nächsten Abschnitt). Beispiele hierfür sind in den Zeilen 9 bis 11 zu finden. Nach der Deklaration können sie dann einfach mit ihrem Namen in den Regeln verwendet werden.

[5] siehe `https://github.com/cvc5/cvc5/issues/2386`

Zeitstempel Zeitstempel sind in SMT-LIB nicht vorgesehen. Um sie dennoch verwenden zu können, werden sie auf ganze Zahlen abgebildet. Dazu wird die Funktion $getTime : D \to \mathbb{Z}$ verwendet. Sie repräsentiert einen Zeitpunkt durch die Anzahl der seit dem Epoch (1. Januar 1970, 00:00.000 Uhr UTC) [4] bis zu diesem Zeitpunkt verstrichenen Millisekunden und folgt damit der in Abschnitt 2.2 für $getTime$ definierten Semantik.

Logische Konnektive Die Übersetzung von logischen Konnektiven ist trivial, da diese in der Core Theory von SMT-LIB enthalten sind [11].
$var_1 \wedge var_2$ beispielsweise wäre in SMT-LIB `(and var_1 var_2)`.

Vergleiche Auch Vergleiche sind in SMT-LIB für fast alle von uns benötigten Datentypen in den entsprechenden Theorien standardmäßig vorhanden. Vergleiche von Strings mussten, wie im Abschnitt „Funktionen“ erwähnt, teilweise definiert werden. Für Zeitstempel bedienen wir uns erneut der Funktion $getTime : D \to \mathbb{Z}$, da für alle $R \in \{=, <, >, \leq, \geq\}$ gilt: Für $A = (D, R), B = (\mathbb{Z}, R)$ ist $getTime$ ein Isomorphismus von A nach B. Somit können einfach die Abbilder der Zeitstempel in $\mathbb{Z}$ verglichen werden.

Berechnungen Berechnungen mit Zahlen können weitestgehend trivial in SMT-LIB umgesetzt werden, da sie Teil der „Theory of Reals“ sind [7]. $4 \cdot 3$ würde zum Beispiel zu `(* 4 3)` übersetzt werden. Die einzige Ausnahme ist die Modulo-Operation, für diese wird die im Abschnitt „Funktionen“ beschriebene Funktion `%` verwendet.

Berechnungen mit Zeitstempeln Die Berechnungen mit Zeitstempeln folgen den in Abschnitt 2.2 beschriebenen Definitionen. Zeitstempel werden durch die Funktion $getTime$ nach $\mathbb{Z}$ abgebildet. Zeitintervalle, die auf einen Zeitstempel addiert, bzw. von einem solchen subtrahiert werden sollen oder das Ergebnis einer Subtraktion zweier Zeitstempel sind, müssen in die richtige Einheit konvertiert werden. Dies geschieht durch Multiplikation oder Division mit einem entsprechenden Umrechnungsfaktor, der der Umrechnungsmatrix[6] entnommen wird.

Error Wie in Abschnitt 2.2 beschrieben, können bei Division und Modulo Fehler auftreten, wenn der Divisor gleich 0 ist. Dieser Fehler wird anschließend entsprechend der Semantik bis zum Level des Rulesets eskaliert. Um dies in SMT-LIB abzubilden, werden die Umstände, unter denen der Fehler auftreten könnte, von vornherein ausgeschlossen. Das ist für unsere Zwecke eine äquivalente Übersetzung, da ein Fehler niemals ein Ruleset erfüllt. Um also Fehler auszuschließen, wird sowohl für Moduloberechnungen als auch für Divisionen, die dem Ruleset entstammen, eine zusätzliche Bedingung eingefügt, die festlegt, dass der Divisor nicht 0 sein darf.

Zum Beispiel muss für die in Listing 3 gezeigte Regel bei der Übersetzung sichergestellt werden, dass $\neg(number - 3 = 0)$, wie in Zeile 3 des folgenden Codes zu sehen ist:

```
1 ...
2 (declare-const number Real)
3 (assert (not (= (- number 3.0) 0)))
4 (assert (< 5.0 (/ 2.0 (- number 3.0))))
```

Auch wenn in der „Theory of Reals“ die Division nur für Divisoren ungleich 0 definiert ist [7], produziert CVC5 manchmal ein Modell, das den Divisor 0 werden lässt, wenn es nicht, wie in diesem Beispiel, explizit verboten wird.

[6] siehe Appendix A.2

```
{
    "$schema": "https://raw.githubusercontent.com/educorvi/rita/alpha/rita-core/src/s
↪   chema/schema.json",
    "rules": [
        {
            "id": "RuleWithDivision",
            "rule": {
                "type": "comparison",
                "operation": "smaller",
                "arguments": [
                    5,
                    {
                        "type": "calculation",
                        "operation": "divide",
                        "arguments": [
                            2,
                            {
                                "type": "calculation",
                                "operation": "subtract",
                                "arguments": [
                                    {
                                        "type": "atom",
                                        "path": "number"
                                    },
                                    3
                                ]
                            }
                        ]
                    }
                ]
            }
        }
    ]
}
```

Listing 3: Rita-Regel, die besagt, dass $5 < \frac{2}{number-3}$

Der Fehler, der bei in η undefinierten Werten geworfen wird, ist für die Übersetzung unerheblich, da Atome durch den Solver gerade so gesetzt werden sollen, dass das Ruleset erfüllbar ist. Dasselbe gilt für die Fehler, die beim Vergleichen von Daten unterschiedlicher Typen entstehen.

3.5 Überprüfung der Erfüllbarkeit

Nachdem die Regeln in SMT-LIB übersetzt wurden, können sie nun mit einem SMT-Solver auf Erfüllbarkeit überprüft werden. Dazu wird an das Ende der Übersetzung die Zeile `(check-sat)` angehängt. Anschließend startet das Rita-SMT Tool den SMT-Solver und übergibt ihm die Übersetzung. Dieser überprüft die Erfüllbarkeit und generiert zusätzlich ein Modell, das das Ruleset erfüllt. Diese beiden Resultate werden vom Tool eingelesen und für eine bessere Darstellung nachbearbeitet. So werden zum Beispiel Werte für Atome, die einen Zeitstempel referenzieren, zurück in ein Datum konvertiert. Das Tool gibt anschließend auf der Konsole aus, ob ein Ruleset erfüllbar ist und, wenn dem so ist, noch das gefundene Modell. Angenommen das in Abschnitt 2.3 beschriebene Beispiel läge in der Datei `example_rule.json`. Dann könnte die Erfüllbarkeit des Rulesets mit dem Befehl `rita-smt checksat example_rule.json` überprüft werden und das Ergebnis würde wie folgt aussehen:

```
{
  satisfiable: true,
  model: {
    istAngestellter: false,
    kaufdatum: 1987-12-28T00:00:00.000Z,
    geburtsdatum: 1970-01-01T00:00:00.000Z
  }
}
```

3.6 Suche nach implizierten Regeln

Die Übersetzung nach SMT-LIB ermöglicht noch weitere Anwendungen. So kann mit dem Rita-SMT Tool auch nach Regeln gesucht werden, die aus den anderen Regeln in diesem Ruleset hervorgehen und somit überflüssig sind. Dies kann gerade bei großen und über längere Zeit gewachsenen Rulesets nützlich sein.

Für die Suche nach implizierten Regeln wird nach dem folgenden Algorithmus vorgegangen: Zunächst wird eine leere Menge erstellt, die später die gefundenen überflüssigen Regeln beinhalten soll und eine Kopie des Rulesets angelegt. Anschließend werden die Regeln iteriert, die sich in dieser Kopie befinden. Für die aktuelle Regel der Iteration wird getestet, ob sie aus der Verundung der übrigen Regeln hervorgeht, indem geprüft wird, ob die Verundung der Negierung der aktuellen Regel mit der Verundung aller übrigen unerfüllbar ist. Wenn dem so ist, geht die aktuelle Regel aus den übrigen hervor und wird aus der Kopie der Regelliste gelöscht sowie in die Menge der gefundenen Regeln aufgenommen. Weiterhin wird in diesem Fall die Iteration abgebrochen und eine neue mit der reduzierten Regelmenge gestartet. Ist dem nicht so, wird die aktuelle Iteration fortgesetzt. Läuft eine Iteration vollständig durch, ohne eine implizierte Regel zu finden, wird die Suche beendet.

In Listing 4 ist der Algorithmus in Pseudocode dargestellt.

Die Iteration arbeitet die Regeln dabei nach absteigender Länge ab, wobei die Länge einer Regel in diesem Fall als die Zahl an Zeichen in ihrer JSON-Darstellung definiert ist. Damit wird versucht, zuerst Regeln zu überprüfen, die möglichst komplex in ihrer Evaluation sind und

```
1: function COMBINE(ruleset)
2:     Combine set of rules {r_1, ..., r_n} into one rule r_1 ∧ ··· ∧ r_n
3: end function
4:
5: remainingRules ← rules
6: impliedRules ← ∅
7: repeat
8:     for rule in remainingRules do
9:         prerequisites ← remainingRules \ {rule}
10:        satInput ← (combine(prerequisites) ∧ ¬rule)
11:        if satInput is unsatisfiable then          ▷ Wenn a ∧ ¬b unerfüllbar ist, gilt a ⇒ b
12:            impliedRules ← impliedRules ∪ {rule}
13:            remainingRules ← remainingRules \ {rule}
14:            break for loop
15:        end if
16:    end for
17: until No implied rules found
```

Listing 4: Pseudocode, der den Algorithmus zur Suche nach implizierten Regeln beschreibt.

sich somit am stärksten auf die spätere Evaluationslaufzeit auswirken. Zwar ist die Länge einer Regel nicht zwangsläufig proportional zu ihrem Evaluationsaufwand, sie ist jedoch ein leicht zu berechnender und oftmals guter Indikator für selbigen.

Auch diese Überprüfung ist Teil des Rita-SMT Tools und kann mit dem Befehl `rita-smt checkimp ruleset.json` gestartet werden. Das Ergebnis der Suche wird anschließend auf der Konsole ausgegeben.

Der Algorithmus liefert jedoch auf der Seite der Voraussetzungen einer Implikation meist keine minimale Lösung, sondern eine Obermenge, da stets nur getestet wird, ob die Verundung aller übrigen Regeln eine andere impliziert. Zur Verbesserung der Analysemöglichkeiten könnte man das Tool in der Zukunft um eine Option ergänzen, die diese Obermengen auf minimale Mengen reduziert.

3.7 Vereinfachung von Rulesets

Nachdem es nun möglich ist, nach überflüssigen Regeln zu suchen, sollen diese im nächsten Schritt auch automatisch entfernt werden. Dabei handelt es sich um einen recht trivialen Prozess, da einfach die mit dem Algorithmus des vorhergehenden Abschnitts gefundenen Regeln aus dem Ruleset entfernt werden müssen. Um eine korrekte Vereinfachung sicherzustellen wird anschließend überprüft, ob das originale und das vereinfachte Ruleset logisch äquivalent sind: Für das originale Ruleset R und das vereinfachte Ruleset V wird geprüft, ob $R \oplus V$ unerfüllbar ist. Ist dies der Fall, sind die beiden Rulesets logisch äquivalent und die Vereinfachung ist korrekt.

Dies ist ebenfalls Teil des Rita-SMT Tools und kann mit dem Befehl `rita-smt simplify ruleset.json` gestartet werden. Das vereinfachte Ruleset wird anschließend auf der Konsole ausgegeben.

4 Evaluation

Zur Evaluation des für diese Arbeit entwickelten Tools sollen nun fiktive Regelwerke unter Zuhilfenahme des Tools in Rita umgesetzt werden.

4.1 Verein

In diesem Szenario soll eine möglichst große Bandbreite der von Rita unterstützten Funktionen getestet werden.

Szenario

Ein Verein, der zur Förderung wissenschaftlicher Experimente gegründet wurde, möchte die Ausschüttung seiner Mittel automatisieren. Dazu soll mit Hilfe von Rita anhand von übergebenen Formulardaten überprüft werden, ob eine Auszahlung nach bestimmten Kriterien gerechtfertigt ist.

Übergebene Daten

- `projekt.genehmigt: Bool`
- `projekt.genehmigtAm: Date`
- `projekt.intialeAusschuettung: Number`
- `auszahlung.beantragungsdatum: Date`
- `auszahlung.betrag: Number`
- `auszahlung.vorangegangene: Number`
- `auszahlung.empfaenger.istMitglied: Bool`

Kriterien

1. Es soll nur an Personen ausgezahlt werden, die Mitglied im Verein sind.
2. Die Auszahlung muss zu einem der vom Verein genehmigten Projekte erfolgen. Genehmigungen, die älter als 365 Tage sind, sind ungültig.
3. Weil der Verein viele Mathematiker als Mitglieder hat, muss der auszuschüttende Betrag stets ein Vielfaches von 3.14 sein.
4. Eine Auszahlung darf nicht größer als die initiale Ausschüttung sein, muss aber größer als 0€ sein.
5. Um zu verhindern, dass Projekte häufig Geld nachfordern, wird die Anzahl der Nachzahlungen auf 4 beschränkt und die erlaubte Höhe einer automatischen Auszahlung durch die

Funktion f limitiert. Für $n \in \mathbb{N}_0$ als Nummer der bereits zuvor erfolgten Auszahlungen und $a \in \mathbb{R}^+$ als initiale Ausschüttungsmenge für dieses Projekt sei $f(n) = a * \left(\frac{1}{n-5} + \frac{6}{5}\right)$.

6. Um Manipulation bei der vorhergehenden Bedingung zu verhindern, werden die Bedingungen $n \in \mathbb{N}_0$ und $a \in \mathbb{R}^+$ explizit überprüft.

Das tatsächliche Ruleset, das sich aus diesen Kriterien ergibt, kann in Appendix B.1 nachgelesen werden.

Beobachtungen

Die Übersetzung funktioniert einwandfrei, das Tool generiert den folgenden SMT Code:

```
1  (set-logic QF_SNIRA)
2  (set-option :produce-assignments true)
3  (set-option :produce-models true)
4  (define-fun % ((a Real) (b Real)) Real (- a (* (to_int (/ a b)) b)))
5  (define-fun <=s ((a String) (b String)) Bool (str.<= a b))
6  (define-fun <s ((a String) (b String)) Bool (and (str.<= a b) (not (= a b))))
7  (define-fun >s ((a String) (b String)) Bool (not (and (str.<= a b) (not (= a b)))))
8  (define-fun >=s ((a String) (b String)) Bool (or (>s a b) (= a b)))
9  (declare-const auszahlung.empfaenger.istMitglied Bool)
10 (assert auszahlung.empfaenger.istMitglied)
11 (declare-const projekt.genehmigt Bool)
12 (declare-const auszahlung.beantragungsdatum Real)
13 (declare-const projekt.genehmigtAm Real)
14 (assert (and projekt.genehmigt (<= (/ (- auszahlung.beantragungsdatum
   ↪  projekt.genehmigtAm) 86400000) 365.0)))
15 (declare-const auszahlung.betrag Real)
16 (assert (not (= 3.14 0)))
17 (assert (= (% auszahlung.betrag 3.14) 0.0))
18 (declare-const projekt.intialeAusschuettung Real)
19 (assert (and (<= auszahlung.betrag projekt.intialeAusschuettung) (> auszahlung.betrag
   ↪  0.0)))
20 (declare-const auszahlung.vorangegangene Real)
21 (assert (not (= (- auszahlung.vorangegangene 5.0) 0)))
22 (assert (and (<= auszahlung.vorangegangene 4.0) (<= auszahlung.betrag (*
   ↪  projekt.intialeAusschuettung (+ (/ 1.0 (- auszahlung.vorangegangene 5.0)) 1.2)))))
23 (assert (not (= 1.0 0)))
24 (assert (and (> auszahlung.betrag 0.0) (and (>= auszahlung.vorangegangene 0.0) (= (%
   ↪  auszahlung.vorangegangene 1.0) 0.0))))
```

Zeilen 9 und 10 repräsentieren Kriterium 1, Zeilen 11 bis 14 Kriterium 2, Zeilen 15 bis 17 Kriterium 3, Zeilen 18 und 19 Kriterium 4, Zeilen 20 bis 22 Kriterium 5 sowie Zeilen 23 und 24 Kriterium 6. In Zeilen 16, 21 und 23 wird dabei, wie in Abschnitt 3.4 beschrieben, sichergestellt, dass nicht durch 0 dividiert wird.

Überprüft man die Erfüllbarkeit des Rulesets mit dem Rita-SMT Tool, so wird richtigerweise festgestellt, dass es erfüllbar ist und es wird eine korrekte Beispielbelegung ausgegeben. Die Ausgabe ist in Listing 5 zu sehen.

Überprüft man nun mit dem Rita-SMT Tool, ob Regeln aus den übrigen hervorgehen, stellt sich heraus, dass dies für Kriterium 4 der Fall ist. Bei genauerer Betrachtung von Kriterium 5 in Kombination mit Kriterium 6 wird auch der Grund ersichtlich: Für $f(n), n \in \{0, 1, 2, 3, 4\}, a \in \mathbb{R}^+$ gilt $0 < f \leq a$, Kriterium 4 wird also stets erfüllt sein, wenn Kriterium 5 erfüllt ist.

```
{
  satisfiable: true,
  model: {
    auszahlung: {
      empfaenger: { istMitglied: true },
      beantragungsdatum: 1970-01-01T00:00:00.000Z,
      betrag: 3.14,
      vorangegangene: 3
    },
    projekt: {
      genehmigt: true,
      genehmigtAm: 1970-01-01T00:00:00.000Z,
      intialeAusschuettung: 4.485714285714286
    }
  }
}
```

Listing 5: Beispielausgabe des Rita-SMT Tools beim Überprüfen der Erfüllbarkeit eines Rulesets

4.2 Skalierung

Wichtig ist auch, wie gut die in den vorhergehenden Kapiteln beschriebenen Tools für große Regelmengen skalieren. Um das zu untersuchen, soll die Zeit gemessen werden, die benötigt wird, um das Überprüfen auf Erfüllbarkeit und auf überflüssige Regeln eines Rulesets durchzuführen.

Dazu wurde das `rita-smt` Tool um ein Benchmarking Kommando erweitert, das für eine gegebene Obergrenze $z \in \mathbb{N}_0$ Rulesets für eines der folgenden Szenarien generiert, wobei der Paramter n der jeweiligen Szenarien iterativ alle Zahlen von 0 bis z annimmt. Für jedes n wird dabei die Zeit gemessen, die zum Überprüfen der Erfüllbarkeit und zur Überprüfung auf überflüssige Regeln benötigt wird. Eine kurze Erklärung zur Benutzung des Benchmarking Kommandos findet sich in Appendix B.2.1.

Die Messungen wurden auf einem Desktop Rechner mit einem Intel Core i7-6700K Prozessor mit 4 Kernen unter Verwendung aller 8 Threads (siehe Option `--maxWorkers` des in Appendix B.2.1 beschriebenen Tools) durchgeführt.

Die Messwerte, die den Graphen in diesem Kapitel zugrunde liegen, können unter `https://github.com/neferin12/bachelorarbeit_public` eingesehen werden.

4.2.1 Geradengleichungen

Szenario

Für n Geradengleichungen der Form $y_i = m_i x + t_i$ werden pro Geradengleichung zwei Regeln erstellt, die jeweils einen zufälligen Punkt

$p = \{(x, y) | x, y \in \mathbb{N}, 0 \leq x, y \leq 299)\}$

in die Gleichung einsetzen, um so eine Belegung für alle m_i und t_i durch den Solver bestimmen zu lassen. Da die Regelmengen für diesen Testfall erfüllbar sein sollen, wird sichergestellt, dass die zwei Punkte einer Geradengleichung nicht den selben x-Wert haben.

Für die Anzahl der Regeln des Rulesets gilt also: $a(n) = 2n$. Die Regeln werden mit zunehmendem

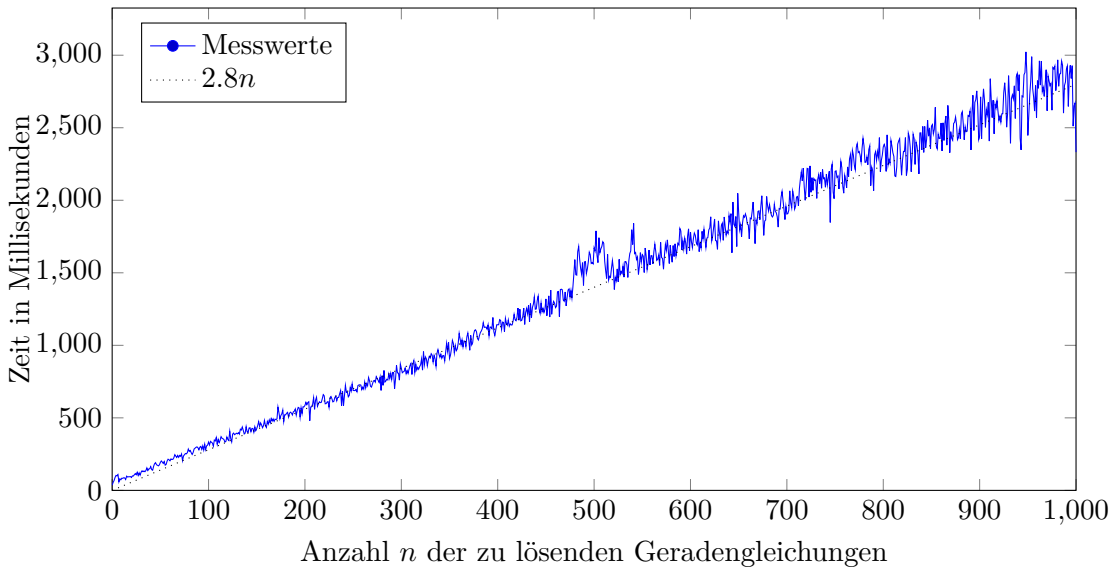


Abbildung 4.1: Zur Überprüfung der Erfüllbarkeit des generierten Rulesets benötigte Zeit in Abhängigkeit von n

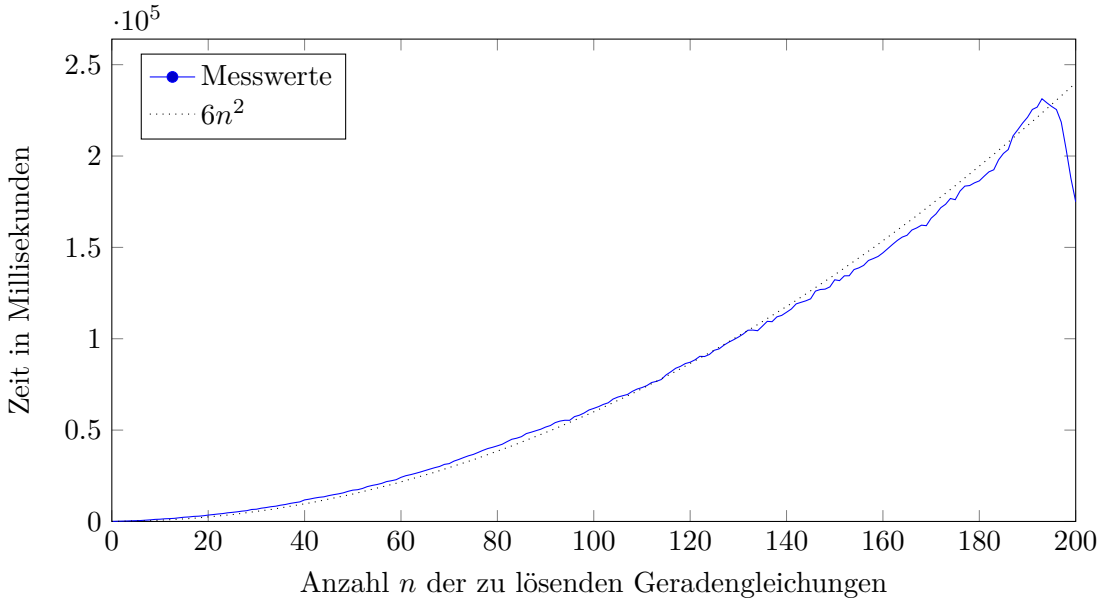


Abbildung 4.2: Zur Überprüfung auf nicht notwendige Regeln im generierten Ruleset benötigte Zeit in Abhängigkeit von n

n jedoch nicht komplexer und jede Regel teilt sich ihre Parameter m_i und t_i nur mit einer anderen Regel, nämlich der, mit der sie zusammengenommen die Geradengleichung bestimmt. Die Gleichungen sind also disjunkt.

Ein Beispiel für ein solches generiertes Ruleset ist in Appendix B.2.2 zu finden.

Beobachtungen

Wie in Abbildung 4.1 zu sehen, skaliert die Überprüfung der Erfüllbarkeit mit der Anzahl der Regeln linear.

Die Laufzeit des Algorithmus zum Finden von implizierten Regeln aus Abschnitt 3.6 skaliert quadratisch mit der Anzahl der Regeln. Die **repeat** Schleife wird, da die Gleichungen disjunkt sind und es somit keine Regel geben kann, die durch andere impliziert wird, nur einmal durchlaufen. Die innere **for** Schleife hingegen wird aus dem selben Grund n mal durchlaufen. Da die Überprüfung in Zeile 11 auch linear mit n skaliert, ergibt sich insgesamt, wie in Abbildung 4.2 zu sehen ist, ein quadratisches Laufzeitverhalten. Der Abfall am Ende des Graphen ist vermutlich auf die Parallelisierung des Benchmarking Tools zurückzuführen. Zur Realisierung der Benchmarks werden mehrere Durchläufe mit unterschiedlichen Werten für n parallel durchgeführt. Die letzten Durchläufe sind daher schneller, weil die CPU nicht mehr voll ausgelastet ist.

4.2.2 Polynome

Szenario

Für ein Polynom mit Grad n und der Form $p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$ wird pro Koeffizienten $a_i, i < n$ eine Regel erstellt, bei der der Punkt $(i, 0)$ in die Funktion eingesetzt wird. Außerdem wird eine Regel erstellt, die bestimmt, dass alle Koeffizienten außer a_0 ungleich 0 sind, sowie eine weitere Regel, dass für alle Koeffizienten gilt $-1000 < a_i < 1000$. Ziel ist es, eine solche Belegung für alle a_i durch den Solver bestimmen zu lassen, dass das Polynom $p(x)$ die Bedingungen erfüllt.

Für die Anzahl der Regeln des Rulesets gilt also: $a(n) = n + 2$. Zusätzlich dazu steigt mit dem Grad n auch die Komplexität der einzelnen Regeln.

Beobachtungen

Um die Skalierung der Laufzeit besser untersuchen zu können, wurde mit Python versucht eine Funktion der Form $f(n) = b \cdot n^a$ zu finden, die die Entwicklung der Messwerte möglichst präzise beschreibt.

Wie in Abbildung 4.3 zu sehen, entwickeln sich die Messwerte für das Überprüfen der Erfüllbarkeit des Rulesets etwa mit der Funktion $f(n) = 8 \cdot 10^{-5}n^{4.73}$ und skalieren somit wesentlich schlechter als bei den Gleichungssystemen des vorherigen Abschnitts. Das ist jedoch wenig überraschend, da nicht nur die Anzahl der Regeln, sondern auch die Komplexität der Regeln mit dem Grad des Polynoms erheblich steigt.

Für die Suche nach implizierten Regeln, abgebildet in Abbildung 4.4, lässt sich die generelle Entwicklung der Messwerte durch die Funktion $f(n) = 2.2 \cdot 10^{-4}n^{5.58}$ beschreiben. Auf einem Intel Core i7-6700K mit einer Taktrate von 4GHz dauerte es 7.6 Stunden ein generiertes Polynomruleset mit Grad 100 auf implizierte Regeln zu untersuchen. Das sind Zeiten, die für die praktische Anwendung völlig inakzeptabel sind. Allerdings ist hier anzumerken, dass ein Ruleset aus der Realität vermutlich selten dieses Maß an Komplexität erreichen wird. Interessant ist

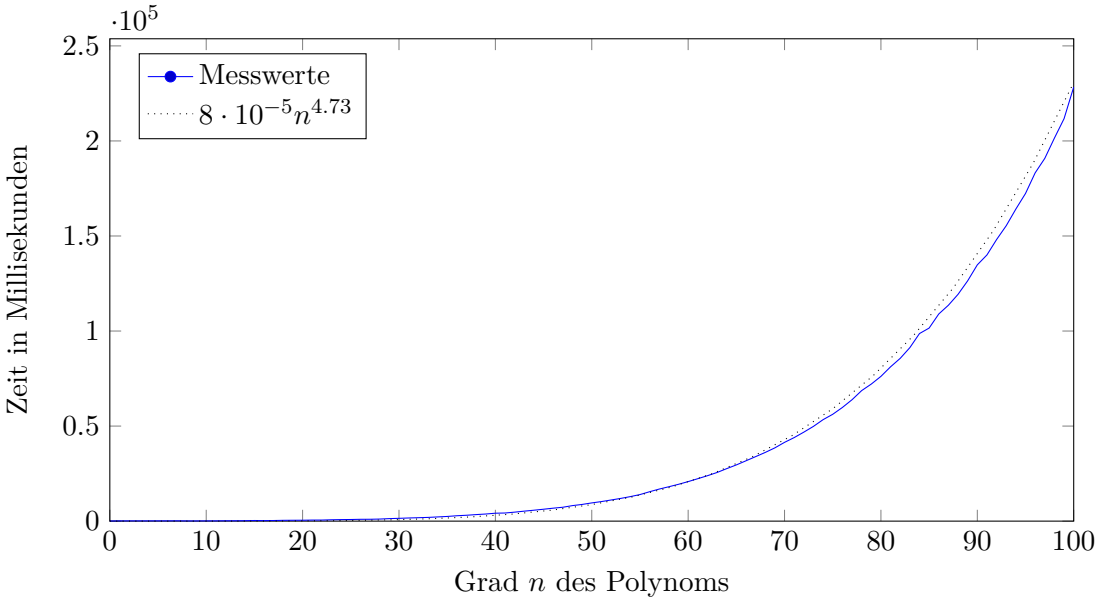


Abbildung 4.3: Zur Überprüfung der Erfüllbarkeit des generierten Rulesets benötigte Zeit in Abhängigkeit von n

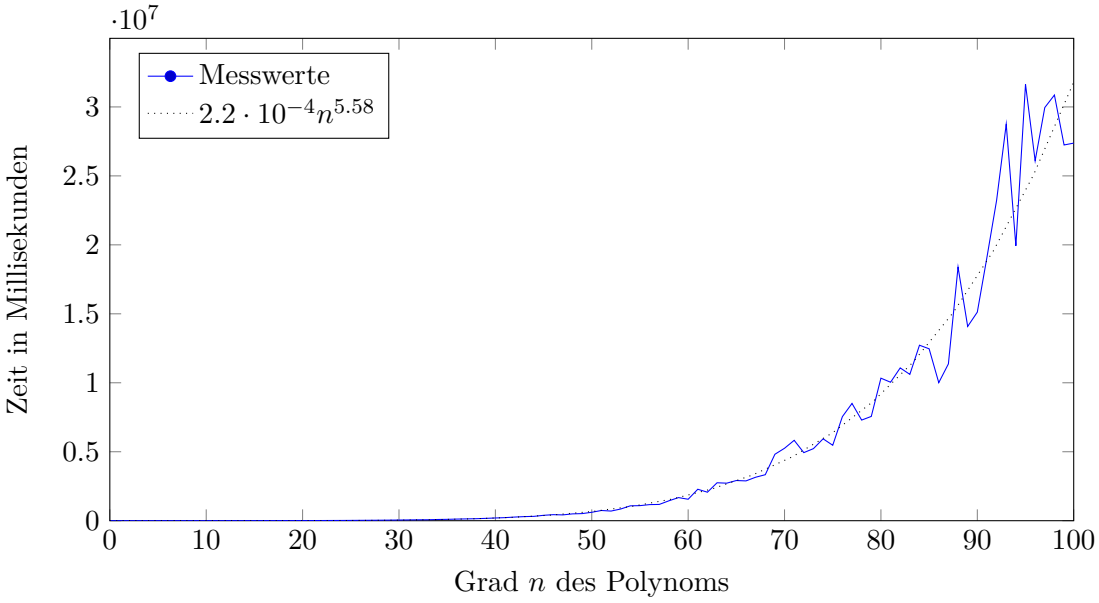


Abbildung 4.4: Zur Überprüfung auf nicht notwendige Regeln im generierten Ruleset benötigte Zeit in Abhängigkeit von n

auch die starke Unregelmäßigkeit der Messwerte, gerade wenn man sie mit den Messwerten der selben Aufgabe im Fall der Geradengleichungen in Abbildung 4.2 vergleicht. Vermutlich kommen hier besondere und schnellere Lösungsstrategien zum Tragen, die der Solver in speziellen Fällen anwenden kann. Ein Muster, wie zum Beispiel eine schnellere Analyse bei allen Polynomen mit geradem Grad, lässt sich aus den Messwerten aber nicht erkennen.

5 Fazit

Auch wenn das im Rahmen dieser Arbeit entwickelte Rita-SMT Tool, wie am Beispiel der gefundenen implizierten Regel in Abschnitt 4.1 zu sehen, gerade für unübersichtliche Rulesets durchaus nützlich sein kann, ist ein produktiver Einsatz eher fraglich.

Dies liegt vor allem daran, dass das Tool dafür um Quantoren und Macros erweitert werden müsste, was jeweils eigene Probleme mit sich bringt. Quantoren in Rita besagen, dass eine Regel auf alle oder mindestens ein Element eines Arrays zutreffen muss. Tatsächlich ließe sich dies vermutlich sogar übersetzen, ohne die Quantoren `forall` oder `exists` in SMT-LIB benutzen zu müssen. Dadurch bliebe die Entscheidbarkeit der Probleme erhalten. Allerdings wäre die Abbildung nicht trivial und Teile der Übersetzung würden deutlich an Komplexität gewinnen. So müssten zum Beispiel Atome, die auf einen Index in einem Array zugreifen, wie zum Beispiel `values[0]`, so übersetzt werden, dass tatsächlich ein Array aus der „Theory of Arrays“[1] verwendet wird, anstelle der aktuellen Übersetzung, die das Atom einfach als eigene nullstellige Funktion übersetzt. Das referenzierte Array könnte nämlich an anderer Stelle erneut im Rahmen eines Quantors Verwendung finden.

Auch problematisch wäre die Übersetzung des Macros `now`, das zum Zeitpunkt der Evaluation des Rulesets durch den Zeitstempel eben jenes Zeitpunktes ersetzt wird. Hier stellt sich die Frage, wie man das sinnvoll und korrekt übersetzen könnte: Als Zeitpunkt der Übersetzung, als schlicht völlig beliebigen Zeitpunkt, als beliebigen Zeitpunkt in der Zukunft, oder als jeden Zeitpunkt in der Zukunft? Letzteres wäre vermutlich die akkurateste Übersetzung, würde aber die Verwendung des `forall` Quantors in SMT-LIB notwendig machen. Dies könnte das Lösen des Erfüllbarkeitsproblems durch den Solver potentiell erheblich verlangsamen, oder dazu führen, dass sie unentscheidbar werden. Alle anderen Varianten sind aber auch nur begrenzt sinnvoll, weil die Aussage, dass ein Ruleset zu irgendeinem Zeitpunkt (in der Zukunft) gültig sein wird, nur eingeschränkt hilfreich ist.

Abgesehen von diesen Problemen wäre es auch notwendig, eine robustere Kommunikation mit dem Solver zu etablieren. Das Einlesen des Modells mithilfe von regulären Ausdrücken funktioniert zwar, ist aber fehleranfällig, besonders für unerwartete Ausgaben.

Eine weitere notwendige Änderung wäre, die Typinferenz bei der Übersetzung zu verbessern. Wie in Abschnitt 3.4 beschrieben, wird der Typ eines Atoms aus dem Kontext abgeleitet. Dabei wird aber nur die nähere Umgebung des Atoms verwendet. Werden zum Beispiel zwei Regeln definiert, sodass die erste Regel zwei Atome `a1` und `a2` auf Gleichheit testet, wird für die Übersetzung zu SMT davon ausgegangen, dass die Atome vom Typ `Real` sind. Wird nun in der zweiten Regel jedoch eines der beiden Atome aus Regel eins (z.B. `a1`) mit einem String verglichen, würde das bedeuten, dass beide Atome vom Typ `String` sind. Das wird von der aktuellen Implementierung jedoch nicht erkannt. Um dies umsetzen zu können, müsste man vermutlich die Typen der Atome über alle Regeln hinweg analysieren und diesen Prozess mehrmals wiederholen, damit die Typen dann auch transitiv weitergegeben werden (z. B. von `a1` zu `a2`).

Alle diese Probleme des Rita-STM Tools sind durchaus lösbar. Letzten Endes wird sich aber zeigen müssen, ob die Analysemöglichkeiten für Rita-Rulesets, die dieses Tool bietet, relevant genug sind, um die Zeit zu rechtfertigen, die nötig wäre, um es auf das Niveau einer produktiven

[1] https://smtlib.cs.uiowa.edu/theories-ArraysEx.shtml

Anwendung zu bringen.

Literaturverzeichnis

[1] ISO. Iso 8601-1:2019 standard, 2019. https://www.iso.org/obp/ui/#iso:std:iso:8601:-1:ed-1:v1:en.

[2] Douglas Crockford and Chip Morningstar. Standard ecma-404 the json data interchange syntax, 12 2017.

[3] Michael Droettboom et al. Understanding json schema, 2023. Available at https://json-schema.org/understanding-json-schema/UnderstandingJSONSchema.pdf.

[4] IEEE. Ieee standard for information technology–portable operating system interface (posix(tm)) base specifications, issue 7. *IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)*, pages 1–3951, 2018.

[5] Clark Barrett and Cesare Tinelli. *Satisfiability Modulo Theories*, pages 305–343. Springer International Publishing, Cham, 2018.

[6] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press, 2009.

[7] Cesare Tinelli. Theory of reals, 2017. Available at https://smtlib.cs.uiowa.edu/theories-Reals.shtml.

[8] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa, 2021. Available at https://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.6-r2017-07-18.pdf.

[9] Smt-lib logics, 2023. https://smtlib.cs.uiowa.edu/logics.shtml.

[10] Cesare Tinelli, Clark Barrett, and Pascal Fontaine. Theory of strings, 2017. Available at https://smtlib.cs.uiowa.edu/theories-UnicodeStrings.shtml.

[11] Cesare Tinelli. Core theory, 2015. Available at https://smtlib.cs.uiowa.edu/theories-Core.shtml.

A Allgemeiner Appendix

A.1 Rita Setup

Dieser Abschnitt soll kurz erläutern, wie Rita zu Testzwecken auf dem eigenen System aufgesetzt werden kann.

A.1.1 Voraussetzungen

- Node.js[1] in Version 18 oder 16
- CVC5[2]

A.1.2 Installation

```
# Klonen des Quellcodes
git clone --branch alpha https://github.com/educorvi/rita.git
cd rita

# Installation der Abhängigkeiten und Bauen der Anwendung
./setup.sh

# Linken des Tools rita-smt in den Pfad
cd rita-smt/main/
npm link
```

A.1.3 Verwendung

Das Tool rita-smt ist nun im Pfad verfügbar und kann einfach aufgerufen werden:

```
rita-smt --help
```

Mit dem Webserver Rita-HTTP kann außerdem das Speichern und Auswerten von Rulesets ausprobiert werden. Folgen Sie dazu einfach dem zweiten Kapitel dieser Einführung: `https://educorvi.github.io/rita_einfuehrung/`. Wichtig ist hierbei die manuelle Installation zu wählen und nicht die Docker-Installation, da die Docker Images noch nicht die für diese Arbeit verwendete Alpha Version von Rita zur Verfügung stellen.

A.2 Umrechnungsmatrix

Die für Rita verwendete Umrechnungsmatrix entstammt dem Projekt `luxon`[3].

[1] `https://nodejs.org/`
[2] `https://cvc5.github.io/`
[3] `https://github.com/moment/luxon`

A.2.1 Matrix


```
{
  "years": {
    "quarters": 4,
    "months": 12,
    "weeks": 52.1775,
    "days": 365.2425,
    "hours": 8765.82,
    "minutes": 525949.2,
    "seconds": 31556951.999999996,
    "milliseconds": 31556951999.999996
  },
  "quarters": {
    "months": 3,
    "weeks": 13.044375,
    "days": 91.310625,
    "hours": 2191.455,
    "minutes": 131487.3,
    "seconds": 7889237.999999999,
    "milliseconds": 7889237999.999999
  },
  "months": {
    "weeks": 4.3481250000000005,
    "days": 30.436875,
    "hours": 730.485,
    "minutes": 43829.1,
    "seconds": 2629746,
    "milliseconds": 2629746000
  },
  "weeks": {
    "days": 7,
    "hours": 168,
    "minutes": 10080,
    "seconds": 604800,
    "milliseconds": 604800000
  },
  "days": {
    "hours": 24,
    "minutes": 1440,
    "seconds": 86400,
    "milliseconds": 86400000
  },
  "hours": {
    "minutes": 60,
    "seconds": 3600,
    "milliseconds": 3600000
  },
  "minutes": {
    "seconds": 60,
    "milliseconds": 60000
  },
  "seconds": {
    "milliseconds": 1000
  }
}
```

A.2.2 Lizenz und Copyright

B Evaluations-Rulesets

Für bessere Lesbarkeit können die folgenden Dateien auch unter `https://github.com/neferin12/bachelorarbeit_public/` betrachtet und heruntergeladen werden. Die Dateinamen an den einzelnen Beispielen angegeben.

B.1 Verein

`verein.json`

```
{
    "$schema": "https://raw.githubusercontent.com/educorvi/rita/alpha/rita-core/src/schema/schema.json",
    "rules": [
        {
            "id": "mitglied",
            "comment": "Es soll nur an Personen ausgezahlt werden, die Mitglied im Verein sind.",
            "rule": {
                "type": "atom",
                "path": "auszahlung.empfaenger.istMitglied"
            }
        },
        {
            "id": "genehmigt",
            "comment": "Die Auszahlung muss zu einem der vom Verein genehmigten Projekte erfolgen. Genehmigungen, die  älter als 365 Tage sind, sind ungültig.",
            "rule": {
                "type": "and",
                "arguments": [
                    {
                        "type": "atom",
                        "path": "projekt.genehmigt"
                    },
                    {
                        "type": "comparison",
                        "operation": "smallerOrEqual",
                        "arguments": [
                            {
                                "type": "dateCalculation",
                                "dateResultUnit": "days",
                                "operation": "subtract",
                                "arguments": [
                                    {
                                        "type": "atom",
                                        "isDate": true,
                                        "path": "auszahlung.beantragungsdatum"
                                    },
                                    {
                                        "type": "atom",
```

```
                                    "isDate": true,
                                    "path": "projekt.genehmigtAm"
                                }
                            ]
                        },
                        365
                    ]
                }
            ]
        }
    },
    {
        "id": "pi",
        "comment": "Der auszuschüttende Betrag muss stets ein Vielfaches von 3.14
↪  sein",
        "rule": {
            "type": "comparison",
            "operation": "equal",
            "arguments": [
                {
                    "type": "calculation",
                    "operation": "modulo",
                    "arguments": [
                        {
                            "type": "atom",
                            "path": "auszahlung.betrag"
                        },
                        3.14
                    ]
                },
                0
            ]
        }
    },
    {
        "id": "auszahlungsrahmen",
        "comment": "Eine Auszahlung muss geringer sein als die maximal erlaubte
↪  Höhe für das Projekt,aber größer als 0€",
        "rule": {
            "type": "and",
            "arguments": [
                {
                    "type": "comparison",
                    "operation": "smallerOrEqual",
                    "arguments": [
                        {
                            "type": "atom",
                            "path": "auszahlung.betrag"
                        },
                        {
                            "type": "atom",
                            "path": "projekt.intialeAusschuettung"
                        }
                    ]
                },
                {
                    "type": "comparison",
                    "operation": "greater",
```

```
                        "arguments": [
                            {
                                "type": "atom",
                                "path": "auszahlung.betrag"
                            },
                            0
                        ]
                    }
                ]
            }
        },
        {
            "id": "limiterung",
            "comment": "Funktion zur Limitierung der Auszahlungen",
            "rule": {
                "type": "and",
                "arguments": [
                    {
                        "type": "comparison",
                        "operation": "smallerOrEqual",
                        "arguments": [
                            {
                                "type": "atom",
                                "path": "auszahlung.vorangegangene"
                            },
                            4
                        ]
                    },
                    {
                        "type": "comparison",
                        "operation": "smallerOrEqual",
                        "arguments": [
                            {
                                "type": "atom",
                                "path": "auszahlung.betrag"
                            },
                            {
                                "type": "calculation",
                                "operation": "multiply",
                                "arguments": [
                                    {
                                        "type": "atom",
                                        "path": "projekt.intialeAusschuettung"
                                    },
                                    {
                                        "type": "calculation",
                                        "operation": "add",
                                        "arguments": [
                                            {
                                                "type": "calculation",
                                                "operation": "divide",
                                                "arguments": [
                                                    1,
                                                    {
                                                        "type": "calculation",
                                                        "operation": "subtract",
                                                        "arguments": [
                                                            {
```

```
                                                    "type": "atom",
                                                    "path":
    ↪  "auszahlung.vorangegangene"
                                                },
                                                5
                                            ]
                                        }
                                    ]
                                },
                                1.2
                            ]
                        }
                    ]
                }
            ]
        }
    },
    {
        "id": "integritaet",
        "comment": "n ∈ Nu{0} und a ∈ R+",
        "rule": {
            "type": "and",
            "arguments": [
                {
                    "type": "comparison",
                    "operation": "greater",
                    "arguments": [
                        {
                            "type": "atom",
                            "path": "auszahlung.betrag"
                        },
                        0
                    ]
                },
                {
                    "type": "and",
                    "arguments": [
                        {
                            "type": "comparison",
                            "operation": "greaterOrEqual",
                            "arguments": [
                                {
                                    "type": "atom",
                                    "path": "auszahlung.vorangegangene"
                                },
                                0
                            ]
                        },
                        {
                            "type": "comparison",
                            "operation": "equal",
                            "arguments": [
                                {
                                    "type": "calculation",
                                    "operation": "modulo",
                                    "arguments": [
```

```
                                                {
                                                    "type": "atom",
                                                    "path": "auszahlung.vorangegangene"
                                                },
                                                1
                                            ]
                                        },
                                        0
                                    ]
                                }
                            ]
                        }
                    ]
                }
            }
        ]
}
```

B.2 Skalierungsbenchmarks

B.2.1 Ausführung

Die Skalierungstests können nach der Installation von Rita (siehe Appendix A.1) mit dem Befehl `rita-smt benchmark` ausgeführt werden. Mehr Details und alle Optionen für das Benchmark können mit `rita-smt benchmark --help` eingesehen werden.

B.2.2 Geradengleichungen

Der folgende Code zeigt ein beispielhaftes Ruleset, das beim Benchmarking mit einer Geradengleichung generiert wurde.

`generated_line_1.json`

```
{
  "$schema": "https://raw.githubusercontent.com/educorvi/rita/alpha/rita-core/src/sch
↪  ema/schema.json",
  "rules": [
    {
      "id": "g0_0",
      "rule": {
        "type": "comparison",
        "operation": "equal",
        "arguments": [
          196,
          {
            "type": "calculation",
            "operation": "add",
            "arguments": [
              {
                "type": "calculation",
                "operation": "multiply",
                "arguments": [
                  {
                    "type": "atom",
                    "path": "m0"
```

```
                    },
                    21
                  ]
                },
                {
                  "type": "atom",
                  "path": "t0"
                }
              ]
            }
          ],
          "dates": false
        },
        "comment": "196=m_0*21+t"
      },
      {
        "id": "g0_1",
        "rule": {
          "type": "comparison",
          "operation": "equal",
          "arguments": [
            17,
            {
              "type": "calculation",
              "operation": "add",
              "arguments": [
                {
                  "type": "calculation",
                  "operation": "multiply",
                  "arguments": [
                    {
                      "type": "atom",
                      "path": "m0"
                    },
                    127
                  ]
                },
                {
                  "type": "atom",
                  "path": "t0"
                }
              ]
            }
          ],
          "dates": false
        },
        "comment": "17=m_0*127+t"
      }
    ]
}
```

B.2.3 Polynom

Der folgende Code zeigt ein beispielhaftes Ruleset, das beim Benchmarking mit einem Polynom vom Grad 2 generiert wurde.

`generated_pol_2.json`

```
{
  "$schema": "https://raw.githubusercontent.com/educorvi/rita/alpha/rita-core/src/sch
↪  ema/schema.json",
  "rules": [
    {
      "id": "e0",
      "rule": {
        "type": "comparison",
        "operation": "equal",
        "arguments": [
          0,
          {
            "type": "calculation",
            "operation": "add",
            "arguments": [
              {
                "type": "calculation",
                "operation": "multiply",
                "arguments": [
                  {
                    "type": "atom",
                    "path": "a2"
                  },
                  0
                ]
              },
              {
                "type": "calculation",
                "operation": "add",
                "arguments": [
                  {
                    "type": "calculation",
                    "operation": "multiply",
                    "arguments": [
                      {
                        "type": "atom",
                        "path": "a1"
                      },
                      0
                    ]
                  },
                  {
                    "type": "atom",
                    "path": "a0"
                  }
                ]
              }
            ]
          }
        ],
        "dates": false
      },
      "comment": "0=a_2*0^2+a_1*0^1+a0"
    },
    {
      "id": "e1",
      "rule": {
        "type": "comparison",
```

```
        "operation": "equal",
        "arguments": [
          0,
          {
            "type": "calculation",
            "operation": "add",
            "arguments": [
              {
                "type": "calculation",
                "operation": "multiply",
                "arguments": [
                  {
                    "type": "atom",
                    "path": "a2"
                  },
                  1
                ]
              },
              {
                "type": "calculation",
                "operation": "add",
                "arguments": [
                  {
                    "type": "calculation",
                    "operation": "multiply",
                    "arguments": [
                      {
                        "type": "atom",
                        "path": "a1"
                      },
                      1
                    ]
                  },
                  {
                    "type": "atom",
                    "path": "a0"
                  }
                ]
              }
            ]
          }
        ],
        "dates": false
      },
      "comment": "0=a_2*1^2+a_1*1^1+a0"
    },
    {
      "id": "zero_cond",
      "rule": {
        "arguments": [
          {
            "arguments": [
              {
                "type": "comparison",
                "operation": "equal",
                "arguments": [
                  0,
                  {
```

```
                    "type": "atom",
                    "path": "a2"
                  }
                ],
                "dates": false
              }
            ],
            "type": "not"
          },
          {
            "arguments": [
              {
                "type": "comparison",
                "operation": "equal",
                "arguments": [
                  0,
                  {
                    "type": "atom",
                    "path": "a1"
                  }
                ],
                "dates": false
              }
            ],
            "type": "not"
          }
        ],
        "type": "and"
      },
      "comment": "for n>0: a_n != 0"
    },
    {
      "id": "limits",
      "rule": {
        "arguments": [
          {
            "arguments": [
              {
                "type": "comparison",
                "operation": "greater",
                "arguments": [
                  1000,
                  {
                    "type": "atom",
                    "path": "a2"
                  }
                ],
                "dates": false
              },
              {
                "type": "comparison",
                "operation": "smaller",
                "arguments": [
                  -1000,
                  {
                    "type": "atom",
                    "path": "a2"
                  }
```

```
                ],
                "dates": false
              }
            ],
            "type": "and"
          },
          {
            "arguments": [
              {
                "arguments": [
                  {
                    "type": "comparison",
                    "operation": "greater",
                    "arguments": [
                      1000,
                      {
                        "type": "atom",
                        "path": "a1"
                      }
                    ],
                    "dates": false
                  },
                  {
                    "type": "comparison",
                    "operation": "smaller",
                    "arguments": [
                      -1000,
                      {
                        "type": "atom",
                        "path": "a1"
                      }
                    ],
                    "dates": false
                  }
                ],
                "type": "and"
              },
              {
                "arguments": [
                  {
                    "type": "comparison",
                    "operation": "greater",
                    "arguments": [
                      1000,
                      {
                        "type": "atom",
                        "path": "a0"
                      }
                    ],
                    "dates": false
                  },
                  {
                    "type": "comparison",
                    "operation": "smaller",
                    "arguments": [
                      -1000,
                      {
                        "type": "atom",
```

```
                        "path": "a0"
                      }
                    ],
                    "dates": false
                  }
                ],
                "type": "and"
              }
            ],
            "type": "and"
          }
        ],
        "type": "and"
      },
      "comment": "-1000<a_n<1000"
    }
  ]
}
```