

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



Erlangen, 20. Februar 2023

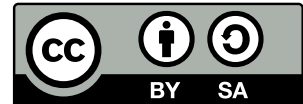
Disclaimer

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, 20. Februar 2023 _____
Julian Pollinger

Lizensierung

Dieses Werk ist lizenziert unter einer Creative Commons “Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International” Lizenz.



Inhaltsverzeichnis

1	Einleitung	9
2	Ritas Sprache	11
2.1	Syntax	11
2.2	Semantik	13
2.3	Beispiel	16
3	SMT basierte Analyse von Rulesets	21
3.1	Satisfiability Modulo Theories	21
3.2	SMT-LIB	21
3.3	SMT Solver	22
3.4	Übersetzung	22
3.5	Überprüfung der Erfüllbarkeit	25
3.6	Suche nach implizierten Regeln	26
3.7	Vereinfachung von Rulesets	27
4	Evaluation	29
4.1	Verein	29
4.2	Skalierung	31
4.2.1	Geradengleichungen	31
4.2.2	Polynome	33
5	Fazit	37
	Literaturverzeichnis	39
A	Allgemeiner Appendix	41
A.1	Rita Setup	41
A.1.1	Voraussetzungen	41
A.1.2	Installation	41
A.1.3	Verwendung	41
A.2	Umrechnungsmatrix	41
A.2.1	Matrix	42
A.2.2	Lizenz und Copyright	43
B	Evaluations-Rulesets	45
B.1	Verein	45
B.2	Skalierung	49
B.2.1	Ausführung	49
B.2.2	Geradengleichungen	49
B.2.3	Polynom	50

1 Einleitung



Hinweis

Der Inhalt dieser Arbeit ist lediglich ein Entwurf. Gerade Kapitel 3 bekommt je nach Zeit bestimmt noch die ein oder andere Ergänzung/Verbesserung.

Konkret geplante Erweiterungen sind:

- *Entscheidbarkeit der SMT Übersetzung anhand der verwendeten Theorien beweisen*
- *Stellenweise müssen Aussagen noch belegt werden.*
- *Ausbauen von Abschnitt 3.1*

Digitale Formulare sind allgegenwärtig. Seien es Anträge im Berufsleben oder das Registrieren bei einer Online Plattform, wir kommen ständig in Berührung mit ihnen. Daraus ergibt sich natürlich auch eine große Menge Formulare, die ausgewertet werden müssen. 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, wobei sich insbesondere Mathematik und Berechnungen mit Zeitstempel, wie zum Beispiel der Abstand zwischen zwei Zeitpunkten als problematisch erweisen sollten. Am geeignetsten erschien uns hierbei das Projekt RuleML, das die Notation von Regeln in XML ermöglichte und mit MathML sogar um Mathematik erweitert werden konnte. Allerdings war die Dokumentation dieses Projekts nicht wirklich hilfreich (zum Zeitpunkt dieser Arbeit ist außerdem die gesamte Webseite¹ vom Netz genommen), was, gemeinsam mit der schieren Größe des Projekt, der Voraussetzung der einfachen Erlernbarkeit widersprach. Andere Projekte scheiterten entweder daran, dass sie keine Unterstützung für Mathematik hatten, zu Komplex zu erlernen waren, oder dass schlicht keine (zeitgemäße) Implementierung zum Auswerten dieser Regeln existierte.

Deshalb entschieden wir uns, unsere eigene simple und auf unsere Bedürfnisse zugeschnittene Regelsprache zu entwickeln: Rita² (kurz für "**R**ule **i**t **a**ll"). Die Sprache setzt dabei auf JSON auf und hat als Kernkonzept sogenannte Rulesets, die eine Menge von Regeln sind. Sind alle Regeln erfüllt, ist auch ein 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. Hier 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.

¹www.ruleml.org

²<https://github.com/educorvi/rita>

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.
- **Berechnungen mit Zeitstempeln:** Dies umfasst im wesentlichen zwei Funktionen:
 1. Subtraktion zweier Zeitstempeln, 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 Bedingung für alle oder mindestens ein Element dieses Arrays gilt.
- **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/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 natürlich auch noch eine Implementierung, die in der Lage ist, Daten gegen diese Regeln auszuwerten. Diese wurde in TypeScript umgesetzt und als Bibliothek auf npm³ veröffentlicht. Die Wahl der Sprache ermöglicht es uns sowohl das Projekt serverseitig mit NodeJS⁴ verwenden zu können, als auch es 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 ein Server, der das Speichern und Bearbeiten von Rulesets, sowie 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>

⁵Commithash zum Zeitpunkt der Abgabe: eced7f18b26716333ccf9aeda02f9e5c71134a07

2 Ritas Sprache

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

2.1 Syntax

Die Sprache baut syntaktisch auf JSON auf. JSON ist eine Syntax, die der Notation von Objekten in JavaScript ähnelt und dessen Zielsetzung es ist, 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.

Die Rita Sprache ist durch ein JSON Schema definiert¹. 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 Regel kann in Backus Naur Form wie folgt beschrieben werden:

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

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

Die Signatur der Sprache ist dabei wie folgt:

Sei R ein Ruleset, und r eine Regel. Sei weiterhin

$$S_0 = \emptyset$$

$$S = \{Bool, String, Date, Number, DurationType\}$$

$$F = \{$$

$$comp_{Bool,=} : Bool \times Bool \rightarrow Bool,$$

$$comp_{Bool,<} : Bool \times Bool \rightarrow Bool,$$

$$comp_{Bool,>} : Bool \times Bool \rightarrow Bool,$$

$$comp_{Bool,\leq} : Bool \times Bool \rightarrow Bool,$$

$$comp_{Bool,\geq} : Bool \times Bool \rightarrow Bool,$$

$$comp_{String,=} : String \times String \rightarrow Bool,$$

$$comp_{String,<} : String \times String \rightarrow Bool,$$

$$comp_{String,>} : String \times String \rightarrow Bool,$$

$$comp_{String,\leq} : String \times String \rightarrow Bool,$$

$$comp_{String,\geq} : String \times String \rightarrow Bool,$$

$$comp_{Date,=} : Date \times Date \rightarrow Bool,$$

$$comp_{Date,<} : Date \times Date \rightarrow Bool,$$

$$comp_{Date,>} : Date \times Date \rightarrow Bool,$$

$$comp_{Date,\leq} : Date \times Date \rightarrow Bool,$$

$$comp_{Date,\geq} : Date \times Date \rightarrow Bool,$$

$$comp_{Number,=} : Number \times Number \rightarrow Bool,$$

$$comp_{Number,<} : Number \times Number \rightarrow Bool,$$

$$comp_{Number,>} : Number \times Number \rightarrow Bool,$$

$$comp_{Number,\leq} : Number \times Number \rightarrow Bool,$$

$$comp_{Number,\geq} : Number \times Number \rightarrow Bool,$$

$$comp_{Mixed,=} : M \times N \rightarrow Bool,$$

$$M, N \in S; M \neq N$$

$$comp_{Mixed,<} : M \times N \rightarrow Bool,$$

$$M, N \in S; M \neq N$$

$$comp_{Mixed,>} : M \times N \rightarrow Bool,$$

$$M, N \in S; M \neq N$$

$$comp_{Mixed,\leq} : M \times N \rightarrow Bool,$$

$$M, N \in S; M \neq N$$

$$comp_{Mixed,\geq} : M \times N \rightarrow Bool,$$

$$M, N \in S; M \neq N$$

$$and : Bool \times Bool \rightarrow Bool,$$

$$or : Bool \times Bool \rightarrow Bool,$$

$$not : Bool \rightarrow Bool$$

$$calc_+ : Number \times Number \rightarrow Number,$$

$$calc_- : Number \times Number \rightarrow Number,$$

$$calc_{\div} : Number \times Number \rightarrow Number + Error,$$

$$calc_{\%} : Number \times Number \rightarrow Number + Error,$$

$$dateCalc_{(Date \times Date),-} : Date \times Date \times DurationType \rightarrow Number,$$

$$dateCalc_{(Date \times Number),+} : Date \times Number \times DurationType \rightarrow Date,$$

$$dateCalc_{(Date \times Number),-} : Date \times Number \times DurationType \rightarrow Date,$$

$$dateCalc_{(Number \times Date),+} : Number \times Date \times DurationType \rightarrow Date,$$

$$getTime : Date \rightarrow Number,$$

$getTime^{-1} : Number \rightarrow Date,$
 $convertInterval : Number \times DurationType \times DurationType \rightarrow Number$
 $\}$

Dann hat Rita die Signatur $\Sigma = (S_0, S, F)$

2.2 Semantik

Die Interpretation eines Rulesets geschieht rekursiv nach folgendem Ablauf:

η = Umgebung aus den gesendeten Daten in Form von Key-Value Paaren

$\eta(x)$ = Der Wert für das Datum mit dem Key x in den Daten

ε = Error

C = Menge der Konstanten

$\mathfrak{M}[R]\eta \in \{\top, \perp, \varepsilon\}$

$\mathfrak{M}[r]\eta \in \{\top, \perp, \varepsilon\}$

$$\mathfrak{M}[R]\eta = \begin{cases} \varepsilon & \text{wenn } \exists r \in R. \mathfrak{M}[r]\eta = \varepsilon \\ \top & \text{sonst wenn } \forall r \in R. \mathfrak{M}[r]\eta = \top \\ \perp & \text{sonst} \end{cases}$$

$$\mathfrak{M}[f(arg_1, \dots, arg_n)]\eta = \begin{cases} \varepsilon & \text{wenn } \mathfrak{M}[arg_i]\eta = \varepsilon \\ \mathfrak{M}[f](\mathfrak{M}[arg_1]\eta, \dots, \mathfrak{M}[arg_n]\eta) & \text{sonst} \end{cases}$$

$$\mathfrak{M}[x]\eta = \begin{cases} \eta(x) & \text{wenn } x \text{ in } \eta \text{ definiert} \\ \varepsilon & \text{sonst} \end{cases}$$

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:

$$\text{Domain } M = \{\top, \perp, S, D, \mathbb{R}\}$$

$$\mathfrak{M}[Bool] = \{\top, \perp\}$$

$$\mathfrak{M}[String] = S$$

$$\mathfrak{M}[Date] = D$$

$$\mathfrak{M}[Number] = \mathbb{R}$$

$$\mathfrak{M}[DurationType] = \{milliseconds, seconds, minutes, hours, days, months, years\}$$

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

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

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

$$\mathfrak{M}[\![not]\!](a) = \neg a$$

$$\mathfrak{M}[\![comp_{Bool,=}\!]\!](a, b) = (a \wedge b) \vee (\neg a \wedge \neg b)$$

$$\mathfrak{M}[\![comp_{Bool,<}\!]\!](a, b) = \neg a \wedge b$$

$$\mathfrak{M}[\![comp_{Bool,>}\!]\!](a, b) = a \wedge \neg b$$

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

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

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

$$\mathfrak{M}[\![comp_{Mixed,=}\!]\!](a, b) = \varepsilon$$

$$\mathfrak{M}[\![comp_{Mixed,<}\!]\!](a, b) = \varepsilon$$

$$\mathfrak{M}[\![comp_{Mixed,>}\!]\!](a, b) = \varepsilon$$

$$\mathfrak{M}[\![comp_{Mixed,\leq}\!]\!](a, b) = \varepsilon$$

$$\mathfrak{M}[\![comp_{Mixed,\geq}\!]\!](a, b) = \varepsilon$$

Bei Vergleichen zwischen Strings wird die lexikalische Ordnung verwendet.

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

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

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

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

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

Berechnungen und Vergleiche mit Zahlen folgen den für reelle Zahlen üblichen Interpretationen,

wobei Divisionen oder Moduloberechnungen mit Divisor 0 in einem Fehler resultieren.

$$\begin{aligned}
\mathfrak{M}[\![comp_{Number,=}\!](a, b) &= \begin{cases} \top & \text{wenn } (a = b) \\ \perp & \text{sonst} \end{cases} \\
\mathfrak{M}[\![comp_{Number,<}\!](a, b) &= \begin{cases} \top & \text{wenn } (a < b) \\ \perp & \text{sonst} \end{cases} \\
\mathfrak{M}[\![comp_{Number,>}\!](a, b) &= \begin{cases} \top & \text{wenn } (a > b) \\ \perp & \text{sonst} \end{cases} \\
\mathfrak{M}[\![comp_{Number,\leq}\!](a, b) &= \mathfrak{M}[\![comp_{Number,<}\!](a, b) \vee \mathfrak{M}[\![comp_{Number,=}\!](a, b) \\
\mathfrak{M}[\![comp_{Number,\geq}\!](a, b) &= \mathfrak{M}[\![comp_{Number,>}\!](a, b) \vee \mathfrak{M}[\![comp_{Number,=}\!](a, b) \\
\mathfrak{M}[\![calc_+\!](a, b) &= a + b \\
\mathfrak{M}[\![calc_-\!](a, b) &= a - b \\
\mathfrak{M}[\![calc_*\!](a, b) &= a * b \\
\mathfrak{M}[\![calc_{\div}\!](a, b) &= \begin{cases} a \div b & \text{wenn } b \text{ nicht } 0 \text{ ist} \\ \varepsilon & \text{sonst} \end{cases} \\
\mathfrak{M}[\![calc_{\%}\!](a, b) &= \begin{cases} a - (\lfloor \frac{a}{b} \rfloor * b) & \text{wenn } b \text{ nicht } 0 \text{ ist} \\ \varepsilon & \text{sonst} \end{cases}
\end{aligned}$$

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

$$\begin{aligned}
\mathfrak{M}[\![comp_{Date,=}\!](a, b) &= \begin{cases} \top & \text{wenn der Zeitpunkt } a \text{ zeitgleich zum Zeitpunkt } b \text{ ist} \\ \perp & \text{sonst} \end{cases} \\
\mathfrak{M}[\![comp_{Date,<}\!](a, b) &= \begin{cases} \top & \text{wenn der Zeitpunkt } a \text{ vor dem Zeitpunkt } b \text{ ist} \\ \perp & \text{sonst} \end{cases} \\
\mathfrak{M}[\![comp_{Date,>}\!](a, b) &= \begin{cases} \top & \text{wenn der Zeitpunkt } a \text{ nach dem Zeitpunkt } b \text{ ist} \\ \perp & \text{sonst} \end{cases} \\
\mathfrak{M}[\![comp_{Date,\leq}\!](a, b) &= \mathfrak{M}[\![comp_{Date,<}\!](a, b) \vee \mathfrak{M}[\![comp_{Date,=}\!](a, b) \\
\mathfrak{M}[\![comp_{Date,\geq}\!](a, b) &= \mathfrak{M}[\![comp_{Date,>}\!](a, b) \vee \mathfrak{M}[\![comp_{Date,=}\!](a, b)
\end{aligned}$$

Bei Berechnungen mit Zeitangaben werden Datumsangaben zunächst in die seit dem 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.

$$\begin{aligned}
\mathfrak{M}[\![getTime]\!](a) &= \text{Von Jan 1, 1970, 00:00:00.000 GMT bis} \\
&\quad \text{a verstrichene Millisekunden} \\
\mathfrak{M}[\![getTime^{-1}]\!](a) &= \text{Inverse Funktion zu getTime} \\
\mathfrak{M}[\![convertInterval]\!](n, t_1, t_2) &= \begin{cases} n * um[t_1][t_2] & \text{wenn } t_1 \text{ größer als } t_2 \text{ ist, wobei gilt:} \\ & \text{milliseconds} < \text{seconds} < \text{minutes} < \\ & \text{hours} < \text{days} < \text{months} < \text{years} \\ n \div um[t_2][t_1] & \text{sonst} \end{cases} \\
\mathfrak{M}[\![calc_{(Date \times Date), -}]\!](a, b, t) &= \\
&\quad \mathfrak{M}[\![convertInterval(calc_{-}(getTime(b), getTime(a)), "milliseconds", t)]\!] \\
\mathfrak{M}[\![calc_{(Date \times Number), +}]\!](a, b, t) &= \\
&\quad \mathfrak{M}[\![getTime^{-1}(calc_{+}(getTime(a), convertInterval(b, t, "milliseconds")))]\!] \\
\mathfrak{M}[\![calc_{(Date \times Number), -}]\!](a, b, t) &= \\
&\quad \mathfrak{M}[\![getTime^{-1}(calc_{-}(getTime(a), convertInterval(b, t, "milliseconds")))]\!] \\
\mathfrak{M}[\![calc_{(Number \times Date), +}]\!](a, b, t) &= \mathfrak{M}[\![calc_{(Date \times Number), +}]\!](b, a, t)
\end{aligned}$$

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:

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

Eine Visualisierung der Regel findet sich in Abbildung 2.1. Die hierarchische Struktur einer Regel wird dort gut ersichtlich.

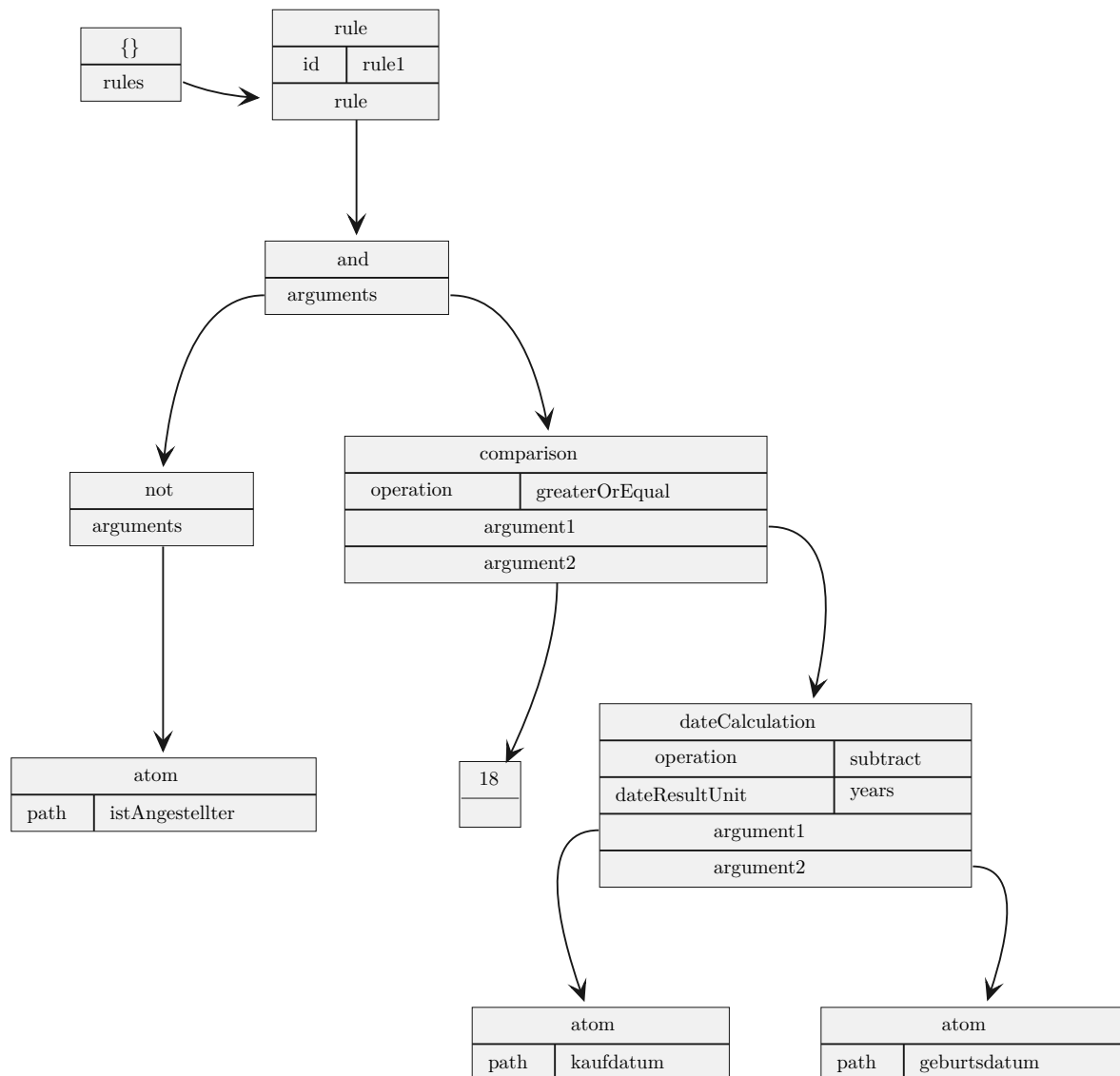


Abbildung 2.1: Visualisierung des Rulesets

Die tatsächliche Umsetzung dieser Regel sähe dann wie folgt aus:

```

1  {
2    "$schema": "https://raw.githubusercontent.com/educorvi/rita/alpha/rita-core/src/s_
↪  chema/schema.json",
3    "rules": [
4      {
5        "id": "rule1",
6        "rule": {
7          "type": "and",
8          "arguments": [
9            {
10             "type": "not",
11             "arguments": [
12               {
13                 "type": "atom",
14                 "path": "istAngestellter"
15               }
16             ]
17           },
18           {
19             "type": "comparison",
20             "operation": "greaterOrEqual",
21             "arguments": [
22               {
23                 "type": "dateCalculation",
24                 "dateResultUnit": "years",
25                 "operation": "subtract",
26                 "arguments": [
27                   {
28                     "type": "atom",
29                     "path": "kaufdatum",
30                     "isDate": true
31                   },
32                   {
33                     "type": "atom",
34                     "path": "geburtsdatum",
35                     "isDate": true
36                   }
37                 ]
38               },
39               18
40             ]
41           }
42         ]
43       }
44     ]
45   }
46 }
```

An diesem Beispiel wird gut die tatsächliche JSON Syntax 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 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.

Zunächst mag diese Schreibweise 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 Überprüfung von Rulesets zu schaffen, zum Beispiel ob eine solche Menge von Regeln überhaupt erfüllbar ist, oder ob manche Regeln andere implizieren und damit obsolet machen. Um dies ermöglichen zu können, sollen diese Rulesets nun auf SMT Probleme abgebildet werden, um diese dann mit einem SMT Solver untersuchen zu 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]. 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 [6].

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" [7].

Im Fokus steht dabei die Schaffung übergreifender Standards, wie etwa einheitliche Theorien für die SMT Probleme, sowie ein einheitliches Format zum Kommunizieren mit SMT-Solvern [7]. In eben dieses einheitliche Format soll die Rita Sprache übersetzt werden. Dies ermöglicht es, 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 [7] werden Formeln dabei in Prefix Notation geschrieben und durch runde Klammern begrenzt. Die Argumente der Funktionen sind durch Leerzeichen separiert. Außerdem gibt es Befehle unter anderem zum Definieren von Funktionen und Konstanten, sowie zum Assertieren von Formeln. Ein Beispiel in SMT-LIB, das die Erfüllbarkeit der Formel $(x \vee y) \wedge x \wedge \neg y$ überprüft, sähe 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 [8]. 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 CVC5¹ 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-smtlib² 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 integriert³.

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 mithilfe von regulären Ausdrücken geparsed und dem Nutzer der Bibliothek in Form eines Objekts übergeben. Dieses besagt sowohl, ob es ein Modell gibt, dass die generierten Bedingungen erfüllt (unter dem Schlüssel `satisfiable`), sowie ein Beispiel für ein solches Modell (unter dem Schlüssel `model`), sofern das Produzieren von Beispielen aktiviert wurde. Ein Beispiel für ein solches Objekt findet sich in Abschnitt 4.1.

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 und ermöglicht so eine effizientere Lösung der SMT Problematik. s ist eine für CVC5 spezifische Ergänzung und inkludiert die Theory of Strings in die Logik, um String Vergleiche durchführen zu können. NIRA erlaubt die Verwendung von nichtlinearer Arithmetik mit reellen und ganzen Zahlen.

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

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>

Regeln ausgelassen, da eine solche Belegung in diesem Fall nicht von Belang ist und der SMT Solver ohne diese Optionen effizienter arbeiten kann⁴.

Funktionen In den Zeilen 4 bis 8 werden mehrere benötigte Funktionen definiert. Dies beginnt 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 $\text{mod}(a, b) = a - (\lfloor \frac{a}{b} \rfloor * b)$. 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 ist.

In Zeilen 6 bis 9 finden sich Funktionen zum lexikalischen Vergleichen von Zeichenketten. SMT-LIB kennt hier nur die Funktion `str.<=`, die ein lexikalische kleiner-gleich überprüft sowie `=` zur Überprüfung von Gleichheit. In Zeile 6 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 diese 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 Konstante 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 müssen, 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`, `person.name` oder `values[0]`. 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 ein 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.

Zeitstempel Zeitstempel sind in SMT-LIB nicht vorgesehen. Um sie dennoch verwenden zu können, werden sie auf ganze Zahlen abgebildet. Dazu wird die Funktion $\text{getTime} : D \rightarrow \mathbb{Z}$ 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.

⁴siehe <https://github.com/cvc5/cvc5/issues/2386>

Logische Konnektive Die Übersetzung von logischen Konnektiven ist trivial, da diese in der Core Theory von SMT-LIB enthalten sind.

$var_1 \wedge var_2$ 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 $getTime : D \rightarrow \mathbb{Z}$, 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. Die einzige Ausnahme hierbei ist die Modulo-Operation, für diese wird die im Abschnitt "Funktionen" beschriebene Funktion `%` verwendet. $4 \cdot 3$ würde zum Beispiel zu `(* 4 3)` übersetzt werden.

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 Umrechnungsmatrix⁵ 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.

⁵siehe Appendix A.2

Man nehme zum Beispiel die folgende Regel:

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

Hier muss bei der Übersetzung sichergestellt werden, dass $\neg(\text{number} - 3 = 0)$, wie in Zeile 3 des folgenden Codes zu sehen:

```
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, produziert CVC5 manchmal ein Modell, dass den Divisor 0 werden lässt, wenn es nicht wie in diesem Beispiel explizit verboten wird.

Der Fehler, der bei in η undefinierten Werten geworfen wird, ist für die Übersetzung unerheblich, da Atome durch den Solver ja gerade so gesetzt werden sollen, dass das Ruleset erfüllbar ist. Das selbe gilt für die Fehler, die beim Vergleichen von Daten unterschiedlichen Types 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 noch 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 noch ein Modell, das das Ruleset erfüllt. Diese beiden Resultate werden vom Tool eingelesen und für 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 der 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, die sich in dieser Kopie befinden iteriert. 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 einfach die aktuelle Iteration fortgesetzt. Läuft eine Iteration vollständig durch, ohne eine implizierte Regel zu finden, wird die Suche beendet.

Hier ist der Algorithmus noch einmal in Pseudocode dargestellt:

```

1: function COMBINE(ruleset)
2:   Combine set of rules  $\{r_1, \dots, r_n\}$  into one rule  $r_1 \wedge \dots \wedge r_n$ 
3: end function
4:
5: remainingRules  $\leftarrow$  rules
6: impliedRules  $\leftarrow \emptyset$ 
7: repeat
8:   for rule in remainingRules do
9:     prerequisites  $\leftarrow$  remainingRules  $\setminus \{rule\}$ 
10:    satInput  $\leftarrow$  (combine(prerequisites)  $\wedge \neg rule$ )
11:    if satInput is unsatisfiable then  $\triangleright$  Wenn  $a \wedge \neg b$  unerfüllbar ist, gilt  $a \Rightarrow b$ 
12:      impliedRules  $\leftarrow$  impliedRules  $\cup \{rule\}$ 
13:      remainingRules  $\leftarrow$  remainingRules  $\setminus \{rule\}$ 
14:      break for loop
15:    end if
16:  end for
17: until No implied rules found

```

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 sich somit am meisten 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.

Auch diese Überprüfung ist Teils 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 dabei jedoch bei den 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 gefundenen Obermengen auf eine minimale Menge 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 nur die mit dem Algorithmus des vorhergehenden Abschnitts gefundenen Regeln aus dem Ruleset entfernt werden müssen. Anschließend wird 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 Tools sollen nun fiktive Regelwerke unter Zuhilfenahme des für diese Arbeit entwickelten Tools in Rita umgesetzt werden.

4.1 Verein

In diesem Szenario soll die gesamte 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 $f(n) = a * \left(\frac{1}{n-5} + \frac{6}{5}\right)$. Um Manipulation zu vermeiden, 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 sowie Zeilen 20 bis 24 Kriterium 5.

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:

```
{
  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,
      initialeAusschuettung: 4.485714285714286
    }
  }
}
```

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 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.

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 Zahl z Rulesets für eines der folgenden Szenarien generiert, wobei das 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 dieser 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. Da die Regelmengen für diesen Testfall erfüllbar sein sollen, wird

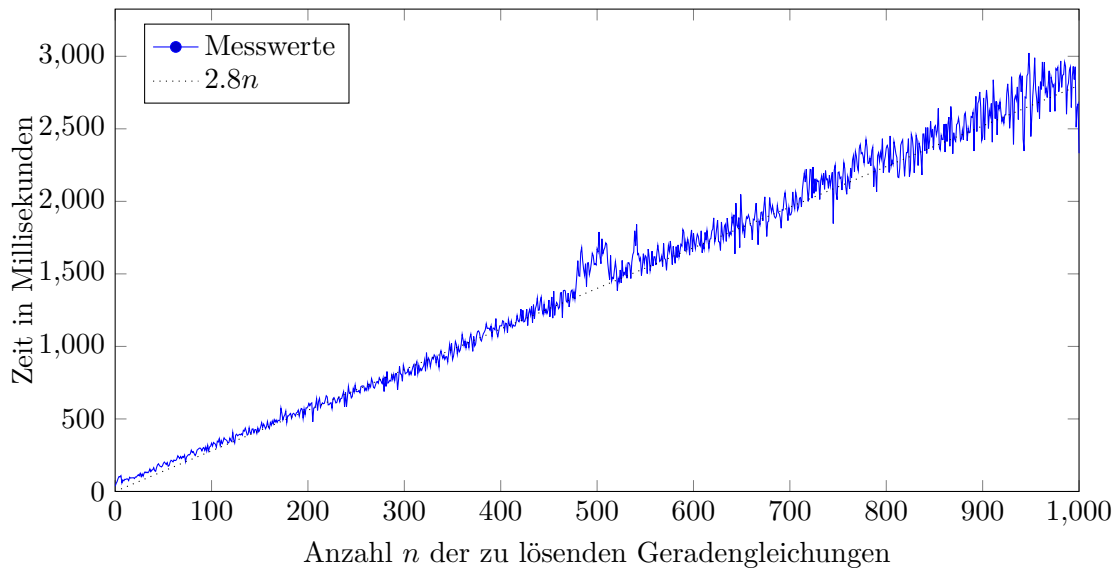


Abbildung 4.1: Zur Überprüfung der Erfüllbarkeit benötigte Zeit in Abhängigkeit von n

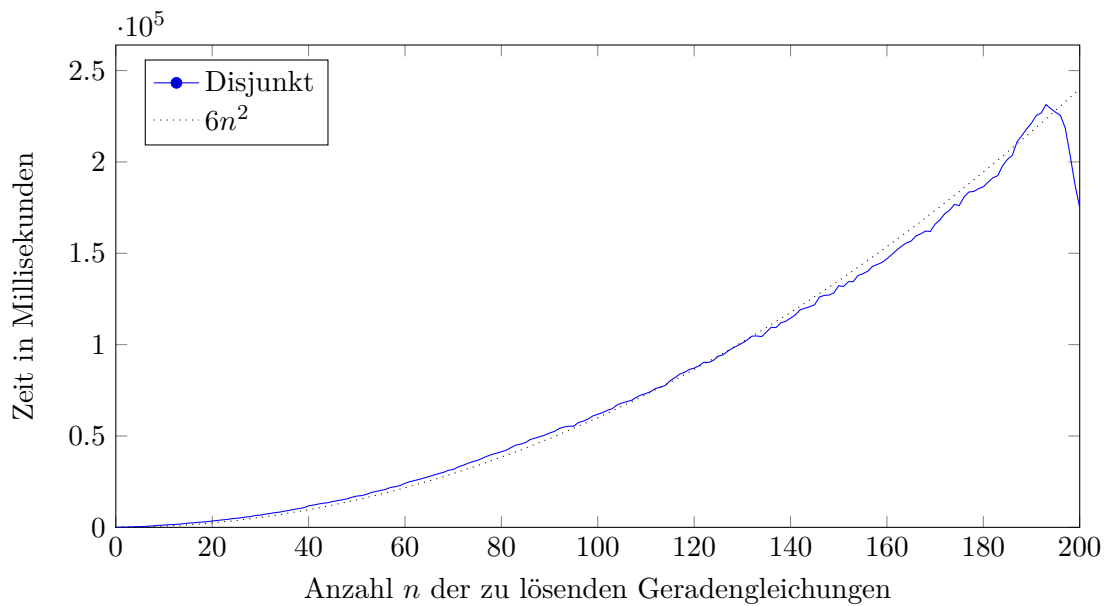


Abbildung 4.2: Zur Überprüfung auf nicht notwendige Regeln benötigte Zeit in Abhängigkeit von n

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 n jedoch nicht komplexer und jede Regel teilt sich die Parameter m_i und t_i nur mit einer anderen Regel, nämlich der mit der sie zusammengekommen 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 Line 11 auch linear mit n skaliert ergibt sich insgesamt, wie in Abbildung 4.2 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 $p(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$ werden pro Koeffizienten $a_i, i < n$ eine Regel erstellt, bei dem 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 $-1000 < a_n < 1000$.

Für die Anzahl der Regeln des Rulesets gilt also: $a(n) = n^2 + 2$. Außerdem steigt mit Grad n natürlich 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 gut 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 sei hier auch anzumerken, dass ein Ruleset aus

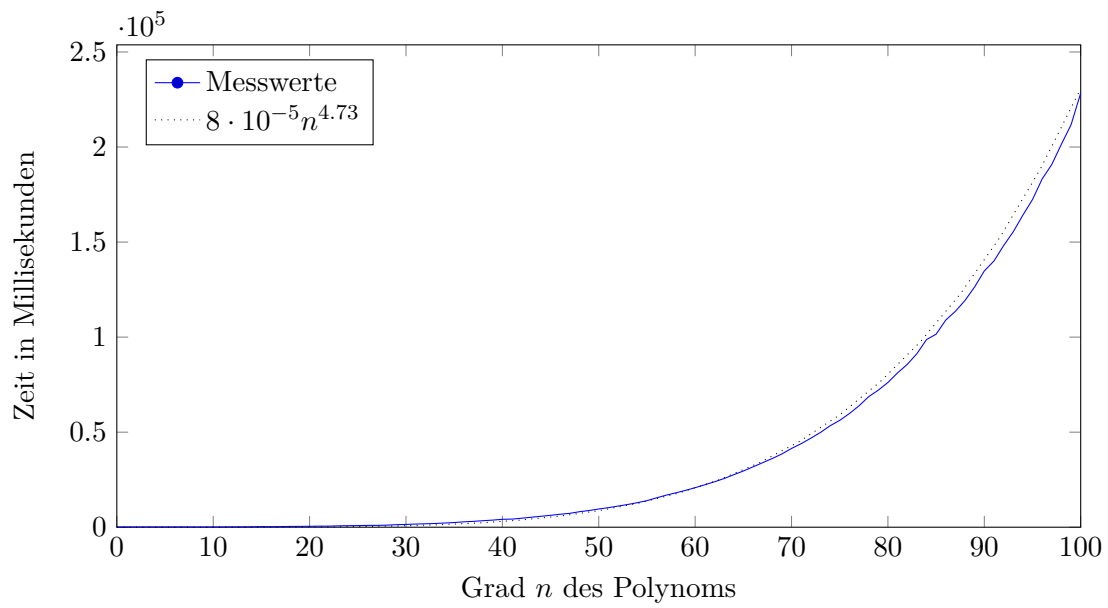


Abbildung 4.3: Zur Überprüfung der Erfüllbarkeit benötigte Zeit in Abhängigkeit von n

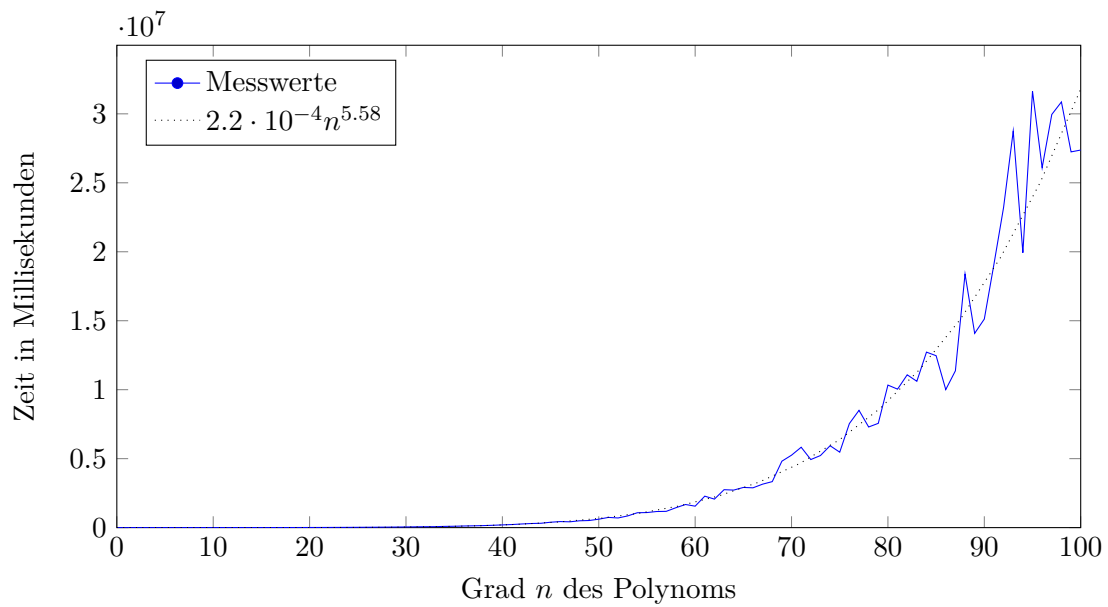


Abbildung 4.4: Zur Überprüfung auf nicht notwendige Regeln benötigte Zeit in Abhängigkeit von n

der Realität vermutlich selten dieses Maß an Komplexität erreichen wird. Interessant ist 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 noch um Quantoren und Macros erweitert werden müsste, was jeweils seine eigenen 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 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" 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.

Das Macro zur Bestimmung der Länge eines Strings wäre mit der Funktion `str.len` aus der "Theory of Strings" trivial übersetzbar, da diese genau das tut. Problematisch wird es beim Macro `now`, das zum Zeitpunkt der Evaluation des Rulesets durch den Zeitstempel eben jenes Zeitpunktes ersetzt. Hier stellt sich die Frage, wie man das sinnvoll 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 sauberste Übersetzung, würde aber die Verwendung des `forall` Quantors in SMT-LIB notwendig machen. Dies würde wiederum bedeuten, dass die Entscheidbarkeit der Probleme nicht mehr sichergestellt werden kann. 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 begrenzt 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 sehr 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 eins 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 auch transitiv weitergegeben werden (z.B. von `a1` zu `a2`).

Alle diese Probleme sind durchaus lösbar, letzten Endes wird sich aber zeigen müssen, ob die Analysemöglichkeiten, die dieses Tool bietet, relevant genug sind, um die Zeit zu rechtfertigen, die nötig wäre, um es auf das Niveau einer produktiven 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] Cesare Tinelli. Theory of reals, 2017. Available at <https://smtlib.cs.uiowa.edu/theories-Reals.shtml>.
- [7] 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>.
- [8] Smt-lib logics, 2023. <https://smtlib.cs.uiowa.edu/logics.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 18 oder 16 ¹
- `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

```

1  {
2    "years": {
3      "quarters": 4,
4      "months": 12,
5      "weeks": 52.1775,
6      "days": 365.2425,
7      "hours": 8765.82,
8      "minutes": 525949.2,
9      "seconds": 31556951.999999996,
10     "milliseconds": 31556951999.999996
11   },
12   "quarters": {
13     "months": 3,
14     "weeks": 13.044375,
15     "days": 91.310625,
16     "hours": 2191.455,
17     "minutes": 131487.3,
18     "seconds": 7889237.999999999,
19     "milliseconds": 7889237999.999999
20   },
21   "months": {
22     "weeks": 4.3481250000000005,
23     "days": 30.436875,
24     "hours": 730.485,
25     "minutes": 43829.1,
26     "seconds": 2629746,
27     "milliseconds": 2629746000
28   },
29   "weeks": {
30     "days": 7,
31     "hours": 168,
32     "minutes": 10080,
33     "seconds": 604800,
34     "milliseconds": 604800000
35   },
36   "days": {
37     "hours": 24,
38     "minutes": 1440,
39     "seconds": 86400,
40     "milliseconds": 86400000
41   },
42   "hours": {
43     "minutes": 60,
44     "seconds": 3600,
45     "milliseconds": 3600000
46   },
47   "minutes": {
48     "seconds": 60,
49     "milliseconds": 60000
50   },
51   "seconds": {
52     "milliseconds": 1000
53   }
54 }

```

A.2.2 Lizenz und Copyright

Copyright 2019 JS Foundation and other contributors

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

B Evaluations-Rulesets

Für bessere Lesbarkeit können die folgenden Dateien auch unter https://github.com/neferin12/bachelorarbeit_public/ betrachtet und heruntergeladen werden. Die Dateinamen sind in der ersten Zeile der jeweiligen Beispiele angegeben.

B.1 Verein

```
1 // verein.json
2 {
3   "$schema": "https://raw.githubusercontent.com/educorvi/rita/alpha/rita-core/src/schema/schema.json",
4   ↪ "rules": [
5     {
6       "id": "mitglied",
7       ↪ "comment": "Es soll nur an Personen ausgezahlt werden, die Mitglied im Verein sind.",
8       "rule": {
9         "type": "atom",
10        "path": "auszahlung.empfaenger.istMitglied"
11      }
12    },
13    {
14      "id": "genehmigt",
15      ↪ "comment": "Die Auszahlung muss zu einem der vom Verein genehmigten Projekte erfolgen. Genehmigungen, die älter als 365 Tage sind, sind ungültig.",
16      "rule": {
17        "type": "and",
18        "arguments": [
19          {
20            "type": "atom",
21            "path": "projekt.genehmigt"
22          },
23          {
24            "type": "comparison",
25            "operation": "smallerOrEqual",
26            "arguments": [
27              {
28                "type": "dateCalculation",
29                "dateResultUnit": "days",
30                "operation": "subtract",
31                "arguments": [
32                  {
33                    "type": "atom",
34                    "isDate": true,
35                    "path": "auszahlung.beantragungsdatum"
36                  },
37                  {
38                    "type": "atom",
39                    "isDate": true,
```

```

40                                     "path": "projekt.genehmigtAm"
41                                 }
42                             ]
43                         },
44                         365
45                     ]
46                 }
47             ]
48         }
49     },
50     {
51         "id": "pi",
52         "comment": "Der auszuschüttende Betrag muss stets ein Vielfaches von 3.14
↪ sein",
53         "rule": {
54             "type": "comparison",
55             "operation": "equal",
56             "arguments": [
57                 {
58                     "type": "calculation",
59                     "operation": "modulo",
60                     "arguments": [
61                         {
62                             "type": "atom",
63                             "path": "auszahlung.betrag"
64                         },
65                         3.14
66                     ]
67                 },
68                 0
69             ]
70         }
71     },
72     {
73         "id": "auszahlungsrahmen",
74         "comment": "Eine Auszahlung muss geringer sein als die maximal erlaubte
↪ Höhe für das Projekt, aber größer als 0€",
75         "rule": {
76             "type": "and",
77             "arguments": [
78                 {
79                     "type": "comparison",
80                     "operation": "smallerOrEqual",
81                     "arguments": [
82                         {
83                             "type": "atom",
84                             "path": "auszahlung.betrag"
85                         },
86                         {
87                             "type": "atom",
88                             "path": "projekt.intialeAusschuettung"
89                         }
90                     ]
91                 },
92                 {
93                     "type": "comparison",
94                     "operation": "greater",
95                     "arguments": [

```

```

96         {
97             "type": "atom",
98             "path": "auszahlung.betrag"
99         },
100         0
101     ]
102 }
103 ]
104 }
105 },
106 {
107     "id": "limiterung",
108     "comment": "Funktion zur Limitierung der Auszahlungen",
109     "rule": {
110         "type": "and",
111         "arguments": [
112             {
113                 "type": "comparison",
114                 "operation": "smallerOrEqual",
115                 "arguments": [
116                     {
117                         "type": "atom",
118                         "path": "auszahlung.vorangegangene"
119                     },
120                     4
121                 ]
122             },
123             {
124                 "type": "comparison",
125                 "operation": "smallerOrEqual",
126                 "arguments": [
127                     {
128                         "type": "atom",
129                         "path": "auszahlung.betrag"
130                     },
131                     {
132                         "type": "calculation",
133                         "operation": "multiply",
134                         "arguments": [
135                             {
136                                 "type": "atom",
137                                 "path": "projekt.intialeAusschuettung"
138                             },
139                             {
140                                 "type": "calculation",
141                                 "operation": "add",
142                                 "arguments": [
143                                     {
144                                         "type": "calculation",
145                                         "operation": "divide",
146                                         "arguments": [
147                                             1,
148                                             {
149                                                 "type": "calculation",
150                                                 "operation": "subtract",
151                                                 "arguments": [
152                                                     {
153                                                         "type": "atom",

```

```

154                                     "path":
↪  "auszahlung.vorangegangene"
155                                     },
156                                     5
157                                 ]
158                             }
159                         ]
160                     },
161                     1.2
162                 ]
163             }
164         ]
165     }
166 ]
167 }
168 ]
169 }
170 },
171 {
172     "id": "integritaet",
173     "comment": "n ∈ Nu{0} und a ∈ R+",
174     "rule": {
175         "type": "and",
176         "arguments": [
177             {
178                 "type": "comparison",
179                 "operation": "greater",
180                 "arguments": [
181                     {
182                         "type": "atom",
183                         "path": "auszahlung.betrag"
184                     },
185                     0
186                 ]
187             },
188             {
189                 "type": "and",
190                 "arguments": [
191                     {
192                         "type": "comparison",
193                         "operation": "greaterOrEqual",
194                         "arguments": [
195                             {
196                                 "type": "atom",
197                                 "path": "auszahlung.vorangegangene"
198                             },
199                             0
200                         ]
199                     },
201                     {
202                         "type": "comparison",
203                         "operation": "equal",
204                         "arguments": [
205                             {
206                                 "type": "calculation",
207                                 "operation": "modulo",
208                                 "arguments": [
209                                     {
210

```



```

211                                     "type": "atom",
212                                     "path": "auszahlung.vorangegangene"
213                                     },
214                                     1
215                                 ]
216                             },
217                             0
218                         ]
219                     }
220                 ]
221             }
222         ]
223     }
224 }
225 ]
226 }

```

B.2 Skalierung

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.

```

1  // generated_line_1.json
2  [
3      {
4          "id": "g0_0",
5          "rule": {
6              "type": "comparison",
7              "operation": "equal",
8              "arguments": [
9                  52,
10                 {
11                     "type": "calculation",
12                     "operation": "add",
13                     "arguments": [
14                         {
15                             "type": "calculation",
16                             "operation": "multiply",
17                             "arguments": [
18                                 {
19                                     "type": "atom",
20                                     "path": "m0"
21                                 },
22                                 126
23                             ]
24                         },
25                     {

```

```

26         "type": "atom",
27         "path": "t0"
28     }
29 ]
30 }
31 ],
32     "dates": false
33 },
34     "comment": "52=m_0*126+t"
35 },
36 {
37     "id": "g0_1",
38     "rule": {
39         "type": "comparison",
40         "operation": "equal",
41         "arguments": [
42             41,
43             {
44                 "type": "calculation",
45                 "operation": "add",
46                 "arguments": [
47                     {
48                         "type": "calculation",
49                         "operation": "multiply",
50                         "arguments": [
51                             {
52                                 "type": "atom",
53                                 "path": "m0"
54                             },
55                             216
56                         ]
57                     },
58                     {
59                         "type": "atom",
60                         "path": "t0"
61                     }
62                 ]
63             }
64         ],
65         "dates": false
66     },
67     "comment": "41=m_0*216+t"
68 }
69 ]
70

```

B.2.3 Polynom

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

```

1  // generated_pol_2.json
2  [
3      {
4          "id": "e0",
5          "rule": {
6              "type": "comparison",

```

```

7      "operation": "equal",
8      "arguments": [
9        0,
10       {
11         "type": "calculation",
12         "operation": "add",
13         "arguments": [
14           {
15             "type": "calculation",
16             "operation": "multiply",
17             "arguments": [
18               {
19                 "type": "atom",
20                 "path": "a2"
21               },
22               0
23             ]
24           },
25           {
26             "type": "calculation",
27             "operation": "add",
28             "arguments": [
29               {
30                 "type": "calculation",
31                 "operation": "multiply",
32                 "arguments": [
33                   {
34                     "type": "atom",
35                     "path": "a1"
36                   },
37                   0
38                 ]
39               },
40               {
41                 "type": "atom",
42                 "path": "a0"
43               }
44             ]
45           }
46         ]
47       },
48       "dates": false
49     },
50     "comment": "0=a_2*0^2+a_1*0^1+a0"
51   },
52   {
53     "id": "e1",
54     "rule": {
55       "type": "comparison",
56       "operation": "equal",
57       "arguments": [
58         0,
59         {
60           "type": "calculation",
61           "operation": "add",
62           "arguments": [
63             {
64

```

```

65         "type": "calculation",
66         "operation": "multiply",
67         "arguments": [
68             {
69                 "type": "atom",
70                 "path": "a2"
71             },
72             1
73         ]
74     },
75     {
76         "type": "calculation",
77         "operation": "add",
78         "arguments": [
79             {
80                 "type": "calculation",
81                 "operation": "multiply",
82                 "arguments": [
83                     {
84                         "type": "atom",
85                         "path": "a1"
86                     },
87                     1
88                 ]
89             },
90             {
91                 "type": "atom",
92                 "path": "a0"
93             }
94         ]
95     }
96 ]
97 }
98 ],
99 "dates": false
100 },
101 "comment": "0=a_2*1^2+a_1*1^1+a0"
102 },
103 {
104     "id": "zero_cond",
105     "rule": {
106         "arguments": [
107             {
108                 "arguments": [
109                     {
110                         "type": "comparison",
111                         "operation": "equal",
112                         "arguments": [
113                             0,
114                             {
115                                 "type": "atom",
116                                 "path": "a2"
117                             }
118                         ],
119                         "dates": false
120                     }
121                 ],
122                 "type": "not"

```

```

123     },
124     {
125         "arguments": [
126             {
127                 "type": "comparison",
128                 "operation": "equal",
129                 "arguments": [
130                     0,
131                     {
132                         "type": "atom",
133                         "path": "a1"
134                     }
135                 ],
136                 "dates": false
137             }
138         ],
139         "type": "not"
140     }
141 ],
142 "type": "and"
143 },
144 "comment": "for n>0: a_n != 0"
145 },
146 {
147     "id": "limits",
148     "rule": {
149         "arguments": [
150             {
151                 "arguments": [
152                     {
153                         "type": "comparison",
154                         "operation": "greater",
155                         "arguments": [
156                             1000,
157                             {
158                                 "type": "atom",
159                                 "path": "a2"
160                             }
161                         ],
162                         "dates": false
163                     },
164                     {
165                         "type": "comparison",
166                         "operation": "smaller",
167                         "arguments": [
168                             -1000,
169                             {
170                                 "type": "atom",
171                                 "path": "a2"
172                             }
173                         ],
174                         "dates": false
175                     }
176                 ],
177                 "type": "and"
178             },
179             {
180                 "arguments": [

```

```

181     {
182         "arguments": [
183             {
184                 "type": "comparison",
185                 "operation": "greater",
186                 "arguments": [
187                     1000,
188                     {
189                         "type": "atom",
190                         "path": "a1"
191                     }
192                 ],
193                 "dates": false
194             },
195             {
196                 "type": "comparison",
197                 "operation": "smaller",
198                 "arguments": [
199                     -1000,
200                     {
201                         "type": "atom",
202                         "path": "a1"
203                     }
204                 ],
205                 "dates": false
206             }
207         ],
208         "type": "and"
209     },
210     {
211         "arguments": [
212             {
213                 "type": "comparison",
214                 "operation": "greater",
215                 "arguments": [
216                     1000,
217                     {
218                         "type": "atom",
219                         "path": "a0"
220                     }
221                 ],
222                 "dates": false
223             },
224             {
225                 "type": "comparison",
226                 "operation": "smaller",
227                 "arguments": [
228                     -1000,
229                     {
230                         "type": "atom",
231                         "path": "a0"
232                     }
233                 ],
234                 "dates": false
235             }
236         ],
237         "type": "and"
238     }

```

```
239         ],
240         "type": "and"
241     }
242 ],
243     "type": "and"
244 },
245     "comment": "-1000<a_n<1000"
246 }
247 ]
248
```