Universität Leipzig
Institut für Informatik

Bachelorarbeit

Evaluierung von Generatoren für aussagenlogische Formeln mit Anwendungen für Abstract Dialectical Frameworks

Stefan Wetzig

17. Oktober 2015

Betreuer: Prof. Dr. Brewka
Dr. Hannes Straß

Inhaltsverzeichnis

Kurzzusammenfassung

Diese Arbeit untersucht mit ausgewählten technischen, syntaktischen und semantischen Eigenschaften verschiedene vollständige Generatoren für aussagenlogische Formeln und zeigt mittels Anwendung von Abstract Dialectical Frameworks, dass die Ergebnisse dieser Evaluierung stark von den untersuchten Generatoren abhängen. Außerdem wird eine breite Übersicht geboten, welche die Wahl eines Formelgenerators für Anwender erleichtern soll. Es zeigt sich, dass sehr ähnliche Eigenschaften durch verschiedene Formelgeneratoren erreicht werden können und der neu vorgestellte STF-Generator zusätzlich ein enormes Einsparen der Generierungsdauer hervorbringen kann.

1 Einleitung

Die Generierung von aussagenlogischen Formeln ist ein fundamentales Zufallsexperiment, welches für ein breites Spektrum verschiedenster Simulationen im Bereich der Theoretischen Informatik Anwendung findet, da Aussagenlogische Formeln eine elementare Wissensrepräsentation darstellen. Für massive Formelgrößen sind die bisherigen elementaren Algorithmen leider nicht realistisch anwendbar. Da es für n abhängige Variablen 2^{2^n} semantisch verschiedene Formeln gibt, wird die Mächtigkeit der Menge von möglichen Formeln schon bei geringen n extrem unüberschaubar. Da ich in dieser Arbeit nur vollständige Generatoren untersuchen werde und die Formeln, die der Einfachheit halber ausschließlich in Disjunktiver Normalform (DNF) repräsentiert werden, auch von mehr als 20 Variablen abhängen sollen, hat es zur Folge, dass mit unseren Mitteln in keiner real vorstellbaren Zeit jemals jede dieser Formeln generiert werden kann ($2^{2^{20}} = 6.7 * 10^{315652}$). Der klassische Ansatz, der dessen Formeln für die Generierung die gleiche Wahrscheinlichkeit besitzen, birgt erstens ein immenses Komplexitätsproblem und zweitens schwinden die Wahrscheinlichkeiten mancher Formeln, als Beispiel die trivialen Formeln $wahr$ und $falsch$, so rapide, dass sie für zunehmendes n schwindend gering werden, obwohl sie für mache Probleme einen intuitiv viel höheren Stellenwert genießen müssten. Für manch ein Experiment ist vielleicht solch ein Generator, der sukzessiv die komplette Wahrheitstafel mit zufälligen Wahrheitswerten füllt, kein gutes Abbild der Realität, da manche Formeln in der Anwendung häufiger auftreten als in manch einer Simulation von Gleichwahrscheinlichkeiten angenommen wird. Somit ist es wichtig für eine Simulation dieses Fachbereiches zu prüfen, welche Formelarten man als Modell für die Realität generieren möchte, damit vom Experiment nicht durch die Generierung falsche statistische Schlüsse gezogen werden. Da solch ein Prüfen sehr stark von der Anwendung abhängt, wird dies in der Arbeit nicht weiter untersucht. In erster Linie werden anwendungsunabhängig die Formeln der verschiedenen Generatoren untersucht und verglichen. Da nach der Generierung einer Formel eine Minimierung erfolgt, stellt sich die Frage, ob mehr Generierungszeit gespart werden kann, wenn aussagenlogische Formeln in einem Ansatz schon minimiert generiert werden. Somit könnte man vielleicht, intuitiv gesprochen, Formeln nicht auf doppeltem, gegensätzlichem Wege generieren, sondern beide Wege sparen. Mit dem in dieser Arbeit neu entwickelten Generator soll dieser Gedanke aufgefasst werden und damit das Zeitproblem lösen. Außerdem soll mit diesem Generator es ermöglicht werden, durch eine individuelle Parameterwahl die Art der Formel zu variieren. Dadurch kann nach eigenem Ermessen ein spezieller Formelgenerator erstellt werden, welcher eine bestimmte, nahezu beliebige Formelmenge mit größerer Wahrscheinlichkeit generiert. Um festzustellen, ob die Ergebnisse einer Anwendung wirklich abhängig vom gewählten Generator ist, werden ADFs (Abstract Dialectical Frameworks) mittels der Software Diamond ausgewertet und das Ausmaß der Abhängigkeit untersucht. Somit ist das Ziel dieser Arbeit in erster Linie einen vergleichenden Überblick über verschiedene, ausgewählte vollständige Formel-Generatoren darzustellen und darauffolgend festzustellen welchen unterschiedlichen Einfluss diese Generatoren auf eine Simulation wirklich haben können.

2 Theoretische Grundlagen

In diesem Kapitel wird vom Basiswissen bis hin zu interessanten Sätzen die Theorie für diese Arbeit vermittelt. Vieles davon sollte jedem Informatiker bekannt sein, jedoch ist es für das allgemeine Verständnis, für die Wiederholung und aus Gründen der Vollständigkeit wichtig folgend im Kleinen anzufangen.

2.1 Allgemein

Ein Modell für die Formalisierung von Aussagen ist die Boolesche Aussagenlogik. Sie beschäftigt sich mit Aussagen, welche genau *wahr* oder *falsch* sein können. Hierfür gibt es elementare Bausteine: die sogenannten Atome. Induktiv kann man daraus die Gestalt einer aussagenlogischen Formel, wie es U. Schöning in seinem Standardwerk [1] tat, definieren.

- Jede atomare Formel ist selbst eine Formel.
- Falls F ein Formel ist, so ist auch $\neg F$ eine Formel
- Falls F und G Formeln sind, so ist $F \vee G$ auch eine Formel.
- Falls F und G Formeln sind, so ist $F \wedge G$ auch eine Formel

Eine Wahrheitstafel zeigt, für welche Belegung der atomaren Formeln eine aussagenlogische Formel als Ergebnis *wahr* oder *falsch* ergibt. Hierbei gilt, dass zwei Formeln F_1 und F_2 semantisch äquivalent sind ($F_1 \equiv F_2$), genau dann wenn alle Belegungen einen gleichen Wahrheitswert liefern. Die Definition dieser Äquivalenzrelation ist notwendig, da eine unterschiedliche syntaktische Darstellung auch semantisch gleich sein kann. Aus einer Wahrheitstafel kann man nun eine Formel ableiten, welche aus Klauseln der Belegungen für Modelle, für jene Belegungen die Formel wahr ist, besteht. Mit einem Minimierungsalgorithmus ist es oft möglich die Formel F_1 zu vereinfachen, indem unter Umständen eine syntaktisch kürzere Formel F_2 entsteht, welche aber semantisch äquivalent zur ursprünglichen Formel ist. Für die Darstellung einer Formel wähle ich die Disjunktive Normalform, da sie aus der Wahrheitstafel gut und nachvollziehbar abzulesen ist. Außerdem lässt sich jede aussagenlogische Formel in einer DNF darstellen, dessen Beweis in [1] nachzulesen ist. Betrachte man nun die Äquivalenzklasse $\{F|F \equiv F_1\}$ mit F_1 die DNF einer Wahrheitstafel. Es ist offensichtlich, dass es 2^{2^n} solcher verschiedener Äquivalenzklassen gibt, wobei diese Mengen disjunkt sind.

2.2 Minimierung

Um neben dem Sparen an Speicherplatz und Zeit auch die Einfachheit einer Formel zu ermöglichen, versucht man die Anzahl und die Länge der Klauseln so zu minimieren, dass die dadurch erstellte Formel trotzdem noch semantisch äquivalent bezüglich der ursprünglich aus der Wahrheitstafel erstellten Formel ist. Im Folgenden wird ein allgemein bekannter Minimierungsalgorithmus dargestellt, welcher von W.V. Quine vorgestellt und später von E.J. McCluskey weiterentwickelt wurde. Von Edmund Burke und Eric Foxley wurde in der Standardliteratur [2] dieses Verfahren ausführlich erläutert. Auf die Implementierung wird

später eingegangen, jedoch werden in den folgenden Schritten schon erste Funktionen der Software erwähnt.

1. **Sortieren nach Klauselgrößen**
   In der Methode *divideClauses()* werden die einzelnen Klauseln der eingegebenen DNF nach der Größe sortiert. Dabei wird in dieser Repräsentation die Anzahl der Zweien, und damit die Anzahl der *don't cares*, gezählt. Hierfür werden n neue Listen erstellt, dessen Einträge die Klauseln nach der Größe geordnet hinzugefügt werden.

2. **Sortieren nach Anzahl der wahren Literale**
   In den einzelnen Listen von Klauseln gleicher Größe werden die Einsen mittels *sortNumOfOnes()* gezählt und die Einträge jeweils nach diesem Ergebnis geordnet. Das hat den Vorteil, dass das Prüfen, ob eine Minimierung möglich ist, nur auf wenige Klauseln beschränkt ist.

3. **Klauseln zusammenführen**
   Nun findet die erste eigentliche Minimierung statt, da unter Umständen in diesem Abschnitt Klauseln durch *join(k1,k2)* gelöscht werden. Falls ohne Beschränkung der Allgemeinheit $(A_1, \ldots, A_m, B)$ und $(A_1, \ldots, A_m, \neg B)$ in der zu minimierenden Formel zwei Klauseln sind, so kann man beide Klauseln entfernen und $(A_1, \ldots, A_m)$ der Liste von Klauseln der Größe m hinzufügen.

4. **Implikationen löschen**
   Um überflüssige Klauseln zu löschen überprüft man in *deleteImpl()*, ob eine Klausel eine andere impliziert. Das tritt auf, wenn $(A_1, \ldots, A_m)$ und $(A_1, \ldots, A_m, B_1, \ldots, B_k)$ in der Klauselmenge der Formel enthalten ist. In dem Fall werden die implizierten Klauseln gelöscht.

5. **weitere Redundanzen löschen**
   Falls (B) und $(A_1, \ldots, A_m, \neg B)$ in der Formel vorhanden sind, kann die zweite Klausel zu $(A_1, \ldots, A_m)$ verkleinert werden. Diese hier verallgemeinerte Klauseln werden durch *deleteRedundant()* minimiert.

Für die Minimierung gibt es noch weitere Möglichkeiten, die aber nicht Gegenstand dieser Arbeit sind. Außerdem ist diese Minimierung für die Allgemeinheit leider nicht nachgewiesen eindeutig. Für das Untersuchen der Formeln ist dieses Verfahren jedoch ausreichend um fundamentale Aussagen über eine Formel zu treffen.

2.3 Eigenschaften aussagenlogischer Formel

Wichtig für die Evaluierung der Generatoren ist es, welche Eigenschaften die generierten Formeln aufweisen. Die einfachste Klassifikation der Eigenschaften ist die eindeutige Formelzuordnung. Für wenige abhängige Variablen ist das noch einfach zu analysieren, jedoch ist das für höhere Zahlen viel zu aufwendig. Daher ist es für die Allgemeinheit von Aussagen besser, größere Teilmengen von aussagenlogischen Formeln zu definieren. Im besten Fall sollten diese Teilmengen eindeutig definiert, vollständig und disjunkt sein. Außerdem muss die Mächtigkeit dieser Teilmengen einfach zu berechnen sein, damit gut überprüft werden kann, welche Klassen der Generator mit höherer Wahrscheinlichkeit erzeugt und wie stark er vom

Laplace-Generator abweicht. In dieser Bachelorarbeit wähle ich fünf Charakteristiken für aussagenlogische Formeln mit maximal n Variablen.

- eigentliche Anzahl von abhängigen Variablen
- benötigte Zeit zum Generieren
- Anzahl der Klauseln
- Anteile $[0, 1]$ der Klauseln von Größe m
- Anteile $[0, 1]$ der Variablen in den Klauseln

Da die Minimierung nicht bis ins Detail eindeutig ist, folgt leider, dass die syntaktischen Eigenschaften einer Formel nicht von allgemeiner, eindeutiger Aussagekraft zeugen, sondern nur eine Andeutung für die Art der Formel bieten. Deshalb wird der Fokus auf die Semantik, in diesem Fall die eigentliche Anzahl von abhängigen Variablen, gelegt. Auf der anderen Seite werden die technischen und syntaktischen Eigenschaften nur im Ansatz analysiert.

2.3.1 Die Anzahl der abhängigen Variablen

Für die Generierung benötigt man eine maximale Anzahl an Variablen und durch die Minimierung der Formel wird die eigentliche Anzahl der abhängigen Variablen gezählt. Offensichtlich ist eine Variable unabhängig, wenn deren Veränderung das Ergebnis der Formel nicht beeinflusst. Ein Beispiel ist die Generierung einer Formel von maximal fünf Variablen, welche aber möglicherweise die triviale Formel *wahr* ist. Dann gehört sie zu der Teilmenge der Formeln, dessen Anzahl der abhängigen Variablen gleich Null beträgt. Diese Teilmenge hat konstant eine Mächtigkeit von zwei. Nun stellt sich die Frage, wie groß die anderen Teilmengen sind. Von zwei Parametern, die maximale Anzahl der Variablen n und die eigentliche Anzahl der abhängigen Variablen k, hängt diese Größe ab. Die Mächtigkeit kann sukkzessiv berechnet werden:

Sei $n = 0$, so gibt es $2^{2^0} = 2$ verschiedene Formeln (*wahr* und *falsch*) $2^{2^0} = a_0 = 2$
Für $n = 1$ gibt es $2^{2^1} = 4$ verschiedene Formeln $(w, f, A, \neg A)$ $2^{2^1} = a_0 + a_1 = 4$
Für $n = 2$ existieren $2^{2^2} = 16$ verschiedene Formeln $(w, f, A, \neg A, B, \neg B, \ldots)$
Hier wissen wir, dass $16 = a_0 + b * a_1 + a_2$. Dieses b kann mit dem Binomialkoeffizient berechnet werden, da für die Menge der k-Formeln k Variablen aus n ausgewählt werden. Somit gibt es $\binom{n}{k}$ Formeln in dieser Menge. In dem Fall ist nun $b = \binom{2}{1}$ und somit berechnet sich $a_2 = 2^{2^2} - a0 - \binom{2}{1} * a1 = 10$
Analog gilt für $n = 3 : a_3 = 2^{2^3} - \binom{3}{0} * a_0 - \binom{3}{1} * a_1 - \binom{3}{2} * a_2 = 218$

Zu einer gegebenen n Atomen von Aussagenlogischen Formeln sei nun a_n die Anzahl der Formeln, welche von keiner dieser n Atomen unabhängig ist. Folgendermaßen ist a_n allgemein zu formulieren:

$$a_n = 2^{2^n} - \binom{n}{0}a_0 - ... - \binom{n}{n-1}a_{n-1} = 2^{2^n} - \sum_{j=0}^{n-1}\left(a_j\binom{n}{j}\right)$$

Lemma 1. *Seien* $n, k \in \mathbb{N}$

$$\sum_{i=k}^{n}(-1)^{i-k}\binom{i}{k}\binom{n}{i} = 0$$

Beweis. Dieses Lemma wird durch Induktion gezeigt. Für $n = 0$ gilt:

$$\sum_{i=k}^{0}(-1)^{i-k}\binom{i}{k}\binom{0}{i} = 0$$

Damit ist die Induktionsvoraussetzung gegeben und durch den Induktionsschritt $(n+1)$ wird das Lemma bewiesen. Mit der Gleichung $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$ lässt sich die folgende Summe aufteilen.

$$\sum_{i=k}^{n+1}(-1)^{i-k}\binom{i}{k}\binom{n+1}{i} = \sum_{i=k}^{n+1}(-1)^{i-k}\binom{i}{k}\binom{n}{i} + \sum_{i=k}^{n+1}(-1)^{i-k}\binom{i}{k}\binom{n}{i-1}$$

$$= \sum_{i=k}^{n}(-1)^{i-k}\binom{i}{k}\binom{n}{i} + (-1)^{n+1-k}\binom{n+1}{k}\binom{n}{n+1} + \sum_{i=k}^{n+1}(-1)^{i-k}\binom{i}{k}\binom{n}{i-1}$$

Wegen der Induktionsvoraussetzung und weil für $k > n$ laut Definition $\binom{n}{k} = 0$ gilt, folgt nun

$$= 0 + 0 + \sum_{i=k}^{n+1}(-1)^{i-k}\binom{i}{k}\binom{n}{i-1}$$
$$= \sum_{i=k}^{n+1}(-1)^{i-k}\binom{i-1}{k}\binom{n}{i-1} + \sum_{i=k}^{n+1}(-1)^{i-k}\binom{i-1}{k-1}\binom{n}{i-1}$$

Nun versetze man den Index der Summe mit $j := i - 1$.

$$= \sum_{i=k-1}^{n}(-1)^{j+1-k}\binom{j}{k}\binom{n}{j} + \sum_{i=k-1}^{n}(-1)^{j+1-k}\binom{j}{k-1}\binom{n}{j}$$

Nun folgt nach Aufteilung der ersten Summe mit der Induktionsvoraussetzung und der Definitionen des Binomialkoeffizienten die Behauptung des Lemmas.

$$= \sum_{j=k}^{n} (-1)^{j+1-k} \binom{j}{k} \binom{n}{j} + (-1)^{k-k} \binom{k-1}{k} \binom{n}{k-1} + \sum_{i=k-1}^{n} (-1)^{j+1-k} \binom{j}{k-1} \binom{n}{j}$$
$$= 0 + 0 + 0$$

□

Theorem 1. *Es gilt folgende Gleichheit*

$$a_n = 2^{2^n} - \sum_{j=0}^{n-1} \left(\binom{n}{j} a_j \right) = \sum_{j=0}^{n} \left((-1)^{n-j} \binom{n}{j} 2^{2^j} \right)$$

Beweis. Mittels Induktion wird diese Behauptung gezeigt. Mit $n = 1$ ist einfach nachzuprüfen, dass $2 = (-1)^1 \binom{1}{0} 2^2 + (-1)^0 \binom{1}{1} 2^2 = 2$ gilt. Somit ist der Induktionsanfang bewiesen.

Für den Induktionsschritt überprüfe man die Behauptung für $n + 1$. Für a_i lässt sich sofort die Induktionsvoraussetzung anwenden.

$$2^{2^{n+1}} - \sum_{i=0}^{n+1-1} a_i \binom{n+1}{i} = 2^{2^{n+1}} - \sum_{i=0}^{n} \left[\binom{n+1}{i} \sum_{j=0}^{i} (-1)^{j-i} \binom{i}{j} 2^{2^j} \right]$$
$$= 2^{2^{n+1}} - \binom{n+1}{0} \left((-1)^0 \binom{0}{0} 2^{2^0} \right)$$
$$- \binom{n+1}{1} \left((-1)^0 \binom{0}{0} 2^{2^0} + (-1)^0 \binom{1}{1} 2^{2^1} \right)$$
$$- \ \ldots$$
$$- \binom{n+1}{n} \left((-1)^n \binom{n}{0} 2^{2^0} + \ \ldots \ + (-1)^0 \binom{n}{n} 2^{2^n} \right)$$
$$= 2^{2^{n+1}} + \sum_{j=0}^{n} \left[2^{2^j} \sum_{i=j}^{n} \left[(-1)^{i-j+1} \binom{n+1}{i} \binom{i}{j} \right] \right]$$

Nach diesen bisherigen, sehr technischen Umformungen wendet man das Hilfslemma an, denn aus $\sum_{i=j}^{n} \left[(-1)^{i-j+1} \binom{n+1}{i} \binom{i}{j} \right] + (-1)^{n-j+1} \binom{n+1}{j} = 0$ folgt direkt:

$$= 2^{2^{n+1}} + \sum_{j=0}^{n} \left[2^{2^j} (-1)^{n-j} \binom{n+1}{j} \right]$$
$$= \sum_{j=0}^{n+1} \left[2^{2^j} (-1)^{n-j} \binom{n+1}{j} \right]$$

□

Für diese Teilmengen ist leicht zu prüfen, wie weit ein Generator von der Gleichverteilung abweicht. Ich gehe später auf die Ergebnisse dieser Klassifizierung ein.

2.3.2 Geschwindigkeit der Generierung

Die benötigte Zeit für die Generierung spielt eine sehr große Rolle. Für reine Laplace-Generatoren ist es nicht mehr effizient, Formeln ab einer Größe von z.B. 2^{30} zu generieren. Hierbei stellen andere Generatoren, wie sich später noch zeigt, eine viel schnellere Methode bereit Formeln dieser Größe zu generieren.

2.3.3 Klauselanzahl

Diese syntaktische Charakteristik gibt die Anzahl der Klauseln in einer Formel an und damit ist sie genauso wie die Anzahl der abhängigen Variablen ein intuitives Maß für die Einfachheit der Formel. Es ist hierfür wichtig zu wissen, wie viele Klauseln eine minimierte Formel maximal enthalten kann. Der Beweis besteht aus zwei Schritten: 1. wird gezeigt, welche Anzahl an Klauseln für eine minimierte Formel nicht überschritten wird und 2. wird gezeigt, dass es minimierte Formeln gibt, welche genau diese Anzahl an Klauseln enthalten.

Theorem 2. *Die größte minimierte DNF über n Variablen hat eine Klauselanzahl von* 2^{n-1}.

Beweis. 1. $|F_n| \leq 2^{n-1}$
Der Induktionsanfang mit n=1 ist klar.
Für die Formeln $t, f, A, \neg A$ gilt jeweils: $|F_1| \leq 2^0$, da es maximal eine Klausel in einer minimalen Formel für $n = 1$ geben kann.

Induktionsschritt: $n \to n + 1$
Seien F_n und F'_n zwei beliebige Klauselmengen über die selbe Atommenge von maximaler Mächtigkeit ($F_n = \{K_1, ..., K_{2^{n-1}}\}$ und $F'_n = \{K'_1, ..., K'_{2^{n-1}}\}$) und sei Y die neue Variable. Somit lässt sich über die neue Wahrheitstabelle ableiten, dass $F_{n+1} := \{\{Y\} \cup K_1, ..., \{Y\} \cup K_{2^{n-1}}, \{\neg Y\} \cup K'_1, ..., \{\neg Y\} \cup K'_{2^{n-1}}\}$ gilt. Damit besteht die Formel F_{n+1} aus maximal 2^n Klauseln.

2. Es gibt eine minimierte Formeln F_n mit $|F_n| = 2^{n-1}$
Die Formel F_n dieser Mächtigkeit ist folgendermaßen zu konstruieren: Man wähle eine erste Klausel $K_1 = \{A_1, ..., A_n\}$ der Größe n und fügt sie der bisher leeren Klauselmenge F_n hinzu. Nun fügt man $\binom{n}{2}$ verschiedene Klauseln hinzu, wobei A_i und A_j mit $i \neq j$ und $i, j \in \{1, ..., n\}$ negiert werden. Diese Aktion wiederholt man für $k \in \mathbb{N}$ mit $\binom{n}{2k}$ bis ausschließlich $n \leq 2k$ gilt. Dadurch, dass die Hammingdistanz zwischen den Klauseln niemals 1 ergibt, ist die Formel laut der Quine-McCluskey-Minimierung nicht vereinfachbar. Die Mächtigkeit der Klauselmenge ist damit für n ungerade $|F_n| = \sum_{i=0}^{\frac{n-1}{2}} \binom{n}{2i}$ und für n gerade $|F_n| = \sum_{i=0}^{\frac{n}{2}} \binom{n}{2i}$.

Zu zeigen ist nun:

$$i)\ n\ ungerade: \sum_{i=0}^{\frac{n-1}{2}} \binom{n}{2i} = 2^{n-1}$$

$$ii)\ n\ gerade: \sum_{i=0}^{\frac{n}{2}} \binom{n}{2i} = 2^{n-1}$$

Bevor der eigentliche Beweis beginnt, betrachte man zunächst die Zusammensetzung der Summe eines allgemeinen Binomialkoeffizienten.

$$n\ ungerade: \sum_{i=0}^{\frac{n-1}{2}} \binom{n}{2i} + \sum_{i=0}^{\frac{n-1}{2}} \binom{n}{2i+1} = \sum_{i=0}^{n} \binom{n}{i} = 2^n$$

$$n\ gerade: \sum_{i=0}^{\frac{n}{2}} \binom{n}{2i} + \sum_{i=0}^{\frac{n}{2}-1} \binom{n}{2i+1} = \sum_{i=0}^{n} \binom{n}{i} = 2^n$$

Somit besteht der Beweis aus 2 Umformungen:
i)

$$\sum_{i=0}^{\frac{n-1}{2}} \binom{n}{2i}$$

$$= \binom{n}{0} + \binom{n}{2} + \ ... \ + \binom{n}{n-1}$$

$$= \binom{n}{n} + \binom{n}{n-2} + \ ... \ + \binom{n}{1}$$

$$= \sum_{i=0}^{\frac{n-1}{2}} \binom{n}{2i+1}$$

$$\Rightarrow \sum_{i=0}^{\frac{n-1}{2}} \binom{n}{2i} + \sum_{i=0}^{\frac{n-1}{2}} \binom{n}{2i+1} = 2 * \sum_{i=0}^{\frac{n-1}{2}} \binom{n}{2i} = 2^n = 2 * 2^{n-1}$$

$$\Rightarrow \sum_{i=0}^{\frac{n-1}{2}} \binom{n}{2i} = 2^{n-1}$$

ii)

$$\sum_{i=0}^{\frac{n}{2}-1} \binom{n}{2i+1} = \sum_{i=0}^{\frac{n}{2}-1} [\binom{n-1}{2i} + \binom{n-1}{2i+1}]$$

$$m := n-1 \Rightarrow = \sum_{i=0}^{\frac{m-1}{2}} \binom{m}{2i} + \sum_{i=0}^{\frac{m-1}{2}} \binom{m}{2i+1} = 2^{m-1} + 2^{m-1} = 2 * 2^{n-2} = 2^{n-1}$$

□

□ □

2.3.4 Variablenanteil und Klauselanteil

Der Variablenanteil ist eine syntaktische Eigenschaft, welche angibt, wie oft im Verhältnis eine Variable in der Formel in Erscheinung tritt. Damit wird versucht eine Näherung zu finden, wie stark der Einfluss einzelner Variablen auf das Ergebnis einer Formel ist. Analog zeigt der Klauselanteil auf, welche Vielfalt an Klauselarten bezüglich der Größe in der Formel bestehen. Durch diese beiden Eigenschaften wird deutlich, welche Struktur die Formel hat.

3 Generatortypen

Für diese Arbeit wähle ich drei verschiedene Generatoren: Laplace-Generator, KDNF-Generator und STF-Generator. Die ersten beiden wurden gewählt, da sie standardmäßig in verschiedenen Anwendungen gewählt werden. Außerdem hat der Laplace-Generator die Eigenschaft, dass jede Formel mit der gleichen Wahrscheinlichkeit generiert wird. Mit dem neu entwickelten STF-Generator werden Gemeinsamkeiten und Unterschiede untersucht und es wird die Vermutung überprüft, ob dieser Generator das Zeitproblem der beiden anderen Generatoren lösen kann. Die Vollständigkeit der Generatoren, was bedeutet, dass jede Formel möglich zu generieren ist, wird jeweils bewiesen. In Abhängigkeit davon, dass der Speicher für die Implementierung genügt und die Zufallswerte hinreichend genau sind, ist die Vollständigkeit für die Anwendung gewährleistet.

3.1 Laplace-Generator

Die naive Ansatz für die Generierung einer aussagenlogischen Formel ist für jede verschiedene Belegung der Variablen anzugeben, ob sie für die zu generierende Formel ein Modell ist oder nicht. Anders gesagt wird einfach die Wertetabelle der Formel zufällig gefüllt, wobei die Modellwahrscheinlichkeit an den Positionen unabhängig gegenüber den anderen Positionen ist und natürlich die Wahrscheinlichkeit für ein Modell gleich der Wahrscheinlichkeit von keinem Modell ist. Somit ist die Wahrscheinlichkeit für das Generieren einer Formel aus einer Äquivalenzklasse bezüglich der Semantik gleich dem Generieren einer Formel aus jeder anderen Äquivalenzklasse. Dadurch handelt es sich um ein Laplaceexperiment. Dass der Laplace-Generator vollständig ist, ist einfach zu zeigen.

Vollständigkeitsbeweis:
Sei $n \in \mathbb{N}$ und sei $(b_1, ..., b_{2^n})$ eine aussagenlogische Formel F in einer Wahrheitstafel. Ausgehend davon, dass die Zufallsfunktion selbst vollständig ist, kann man die Generierung als einen vollständigen, binären Baum betrachten, dessen Zweige jeweils eine Wahrscheinlichkeit von $1/2$ zugeordnet ist. Somit gibt es jeweils 2^{2^n} verschiedene Pfade. Für den Pfad der Formel F gibt es die Wahrscheinlichkeit $P(F) = (1/2)^{2^n}$, wobei der Zufallsgenerator eine Wertabfrage des Datentyps Double aus $[0, 1)$ erhält und damit der zurückgegebene Zufallswert

diese Wahrscheinlichkeiten darstellen und unterscheiden kann.
Da nun stets $(1/2)^{2^n} > 0$ gilt, ist jede Formel möglich zu generieren.

3.2 KDNF-Generator

Der KDNF-Generator ist ein bisher häufig genutzter Generator für aussagenlogische Formeln. Von M. Franco, N. Krasnoger, J. Barcarditl wurde dieses Verfahren in [3] dargestellt. Die Besonderheit liegt darin, dass diese Methode nicht direkt die Wahrheitstafel füllt, sondern unmittelbar eine DNF generiert. Hierfür gibt es zwei Parameter, welche im Generator selbst zufällig gewählt werden. Der erste Parameter bestimmt die Anzahl k der Klauseln und der zweite deren maximale Größe g. Durch eine Gleichverteilung des Wertebereiches dieser Parameter kann der Generator jede Formel mit maximal n Variablen generieren. Der Generator könnte auch mit einer anderen Zufallsverteilung interessante Ergebnisse liefern. Obwohl dieser Umstand kein Gegenstand dieser Arbeit ist, zeigt der Beweis im Folgenden für die schwächere Forderung, $\forall m \in \{1, \ldots, n\} : P_k(m) > 0$ (Wahrscheinlichkeit der Länge), $\forall a \in \{0, \ldots, 2^{n-1}\} : P_g(a) > 0$ (Wahrscheinlichkeit der Klauselanzahl) und $\forall v \in \{1, \ldots, n\} : P_n(v) > 0$ (Wahrscheinlichkeit der Variablen), die Vollständigkeit des Generators. In beiden Fällen sei jedoch P hinreichend groß, dass die jeweilige Repräsentation in der Implementierung die Differenz zur 0 nicht so approximiert, dass die Voraussetzungen im Programm nicht mehr gelten.

Vollständigkeitsbeweis:
Sei F eine beliebige Formel mit n abhängigen Variablen. Aus 2.4.3 ist bekannt, dass mit 2^{n-1} Klauseln alle möglichen Formeln abgedeckt sind. Somit ist $P_{kdnf} := P_k(m)P_g(a)\prod_{i=1}^{m*a} P_n(v_i)$ die Wahrscheinlichkeit für das Generieren einer jeden Formel. Der Generator benötigt den Zufallsgenerator nur stückweise für ein einzelnes P, sodass für P_{kdnf} jeweils nur die Faktoren dem Toleranzbereich der Implementierung gerecht werden müssen. Da laut Voraussetzung nun für alle m, a, v_i $P_{kdnf} > 0$ gilt, ist jede Formel möglich zu generieren.

3.3 STF-Generator

Für diesen speziellen, neu entwickelten, rekursiven Generator, Special-Truth-Formula Generator, gibt es bestimmte Parameter, welche die Wahrscheinlichkeit unterschiedlicher Arten der generierten Formeln bestimmt. Intuitiv und vereinfacht gesprochen, wählt der Nutzer die Parameter in Abhängigkeit davon, ob er häufiger komplizierte oder einfache Formeln generieren möchte. Die Motivation dieses Generators liegt darin, dass die Einfachheit einer Formel und die unterschiedliche Wichtung der Abhängigkeiten der Variablen, für eine schnellere Generierung sorgen und verschiedene strukturelle Muster einer aussagenlogischen Formel ermöglichen soll. Der Generator füllt die Wahrheitstafel ähnlich dem Laplace-Generator mit den zwei Wahrheitswerten, jedoch mit rekursiver Prozedur. Der untersuchte Bitstream wird mit bestimmter Wahrscheinlichkeit (t, f) komplett auf 1 oder 0 gesetzt. Falls dieser triviale

Fall mit der Wahrscheinlichkeit $1-t-f$ nicht eintritt, wird die Liste des untersuchten Bereiches der Wahrheitstafel halbiert und dieses Vorgehen mit den Parametern (t, f) der nächsten Rekursionstiefe für beide Teile jeweils wiederholt. Genau wie in den anderen Generatoren sei n hier die Anzahl der Atome. Die Parameterfunktionen t und f mit $t, f : \{0, \ldots, n\} \longrightarrow [0, 1]$ seien mit folgenden Forderungen definiert:

1. $t(n) + f(n) = 1$
2. $\forall k \in \mathbb{N} : 0 \leq l < n : 0 \leq t(l) + f(l) < 1$

Der Generator erstellt einen Bitstream, welcher die Formel durch eine Wahrheitstafel repräsentiert. Also definiere man $F(k)$ als ein 2^k-Tupel $(b_1, \ldots, b_{2^k})$, wobei $\forall i \in \{1, \ldots, 2^k\} : b_i \in \{0, 1\}$. Weiterhin sei $o(k) := (1, ..., 1)$ ein mit Einsen gefülltes 2^k-Tupel und $z(k) := (0, \ldots, 0)$ ein Tupel selber Länge mit Nullen gefüllt. Außerdem seien $F_1(k) = (a_1, \ldots, a_{2^k})$ und $F_2(m) = (b_1, \ldots, b_{2^m})$ zwei Tupel unabhängiger Länge, dann definiert $F_1(k)F_2(m)$ ein (2^k+2^m)-Tupel mit $F_1(k)F_2(m) := (a_1, \ldots, a_{2^k}, b_1, \ldots, b_{2^m})$. Für komfortable Darstellung wird noch $s(k) := 1 - t(k) - f(k)$ zur Terminologie hinzugefügt. Im folgenden wird der fundamentale Algorithmus im Pseudocode dargestellt.

```
2^k-Tupel F(int k){
        double r= Math.random();
        if (t(n-k)<r) {
                return o(k);
        } else if (f(n-r)>r) {
                return z(k);
        } else {
                return F(k-1)F(k-1);
        }
}
```

Die intuitivere Darstellung ist mittels eines Wahrscheinlichkeitsbaum leichter getan.

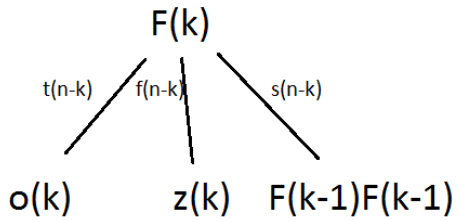

Abbildung 1: STF-Generator

Die Visualisierung dieses Wahrscheinlichkeitsbaumes mit den Wahrscheinlichkeiten t, f, s zeigt den rekursiven Aufbau dieses Generators. Der Rekursionsabbruch ist ebenfalls mit inbegriffen, da mit $k = 0$ auch $s(n-0) = 1 - t(n) - f(n) = 0$ folgt. Zusammenfassend kann man sagen, dass durch diesen Algorithmus $F(n)$ als eine Konkatenation von Tupeln generiert wird, wobei deren Größe und Anzahl die Komplexität und auch die Generierungsdauer konträr beeinflussen soll. Ein Beispiel für diesen Generator ist im Folgenden mit $n = 1$ gegeben:

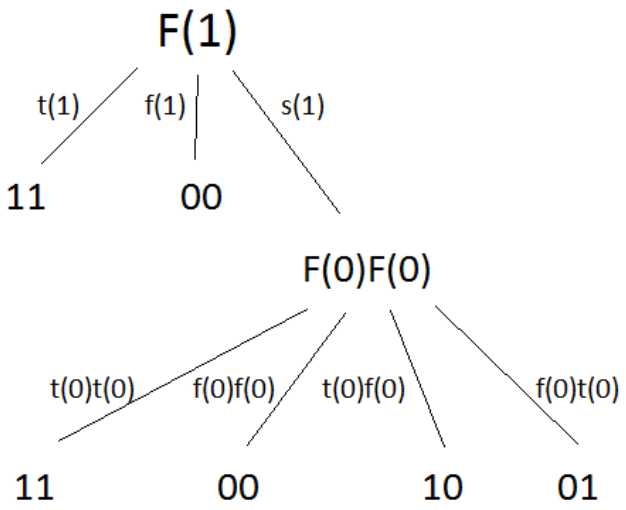

Abbildung 2: STF-Generator Baumdiagrammbeispiel für n=1

Das Beispiel zeigt, für ein besseres Verständnis, ein einfaches Baumdiagramm für maximal 2 abhängige Variablen. Für höhere n wird die Darstellung schnell sehr unübersichtlich. Im folgenden wird auch hier bewiesen, dass der Generator jede mögliche Formel generieren kann:

Vollständigkeitsbeweis:
Da $\forall k < n : s(k) > 0$ folgt direkt, dass eine beliebige Formel $F(n) = (b_1, ..., b_{2^n})$ mindestens mit einer Wahrscheinlichkeit von $P(F) > P'(F) := \prod_{j=0}^{2^n} u_i(n) * \prod_{i=0}^{n-1} [s(i)^{2^i}]$, wobei $u_i \in \{t(n), f(n)\}$ gilt, generiert wird. Weil $P'(F) > 0$ folgt, dass $P(F) > 0$. Sei nun genau wie in den vorherigen Beispielen die Approximation in der Implementierung geeignet, dann ist jede Formel möglich zu generieren.

Aus der Definition dieses Generators lassen sich leicht wichtige Wahrscheinlichkeiten ablesen. Hierfür wird eine kleine rekursive Funktion definiert: $Q(k) := t(n-k) + s(n-k)Q(k-1)^2$. Da $s(n) = 0$ gilt, hat diese Funktion genauso ein Ende in der Rekursion. Durch diese Funktion wird die Wahrscheinlichkeit für $F(n) = o(n)$ charakterisiert: $P(F(n) = o(n)) = Q(n)$. Weiterhin gilt mit $t = f$ für die trivialen Formeln $P(F(n) = o(n) \vee F(n) = z(n)) = 2Q(n)$. Außerdem gilt für eine Formel, welche nur von einer Variable abhängt, folgende Gleichung: $P(k = 1) = \sum_{i=0}^{n(n-1)} [Q(n-i)^{2^i} \prod_{j=0}^{i} (s(n-j)^{2^j}]$. Die Beweise sind rein technischer Natur

und können sukzessive im Baumdiagramm nachvollzogen werden. Daher wird hier auf diese verzichtet. Die Wahrscheinlichkeiten für mehrere Variablen sind sehr schwer zu formalisieren, weshalb in dieser Arbeit nicht weiter nach allgemeinen Lösungen dafür gesucht wird. Das Vorgehen dieses und genauso auch der anderen Algorithmen genauer zu untersuchen ist damit intensiver von statistischer Art. Die Parameter lassen leider kein offensichtliches Schließen bezüglich dem zu, welche Formeln durch Veränderung der Funktionen t, f mit höherer Wahrscheinlichkeit generiert werden. Somit wird wohl eine umfangreiche Testreihe nötig sein um Aussagen darüber zu treffen, welche Formelmengen wahrscheinlicher sind.

4 Evaluierung

Um die Unterschiede und Gemeinsamkeiten der verschiedenen Formelgeneratoren auch empirisch feststellen zu können, wurden diese in einer Software implementiert. Die Visualisierungen der erstellten Daten sind teilweise mit SciDAVis, R und ImageKlebor erstellt.

4.1 Implementierung

Es werden keine zusätzlichen Bibliotheken benötigt um dieses Programm auszuführen, da jeder Baustein dieser Software in dieser Arbeit neu implementiert und keine externen Programme genutzt wurden. Für die Anwendung wird es später jedoch eine Schnittstelle (DIAMOND) geben, was aber erst im Kapitel 5 eine Rolle spielen wird. Die Repräsentation einer aussagenlogischen Formel wurde als eine Liste von Arrays realisiert, welche die Klauseln (Konjunktion von Literalen) darstellen sollen. Die Elemente dieser Arrays sind aus der Menge $\{0, 1, 2\}$. Die Position dieses Arrays bezieht sich auf eine Variable, welche den Zustand *false*, *true* und *don't care* haben kann. In *Generator.java* werden die Formeln nach Parametereingabe ausschließlich in DNF generiert und in *QMC.java* minimiert. Die Vollständigkeit der Generatoren ist jeweils bis $n = 31$ gewährleistet. Für größere n genügen die Vollständigkeitsbeweise aus dem vorhergehen Abschnitt theoretisch auch, jedoch müssten für Listenimplementierung von Java noch bessere Datentypen verwendet werden, da die einfachen Listen nur Integer als Indizes verwenden. Dadurch kann die Klauselmenge maximal eine Mächtigkeit von 2^{31} haben. Für die Untersuchung der Generatoren genügt jedoch diese Anzahl völlig. Es obliegt dem Anwender größerer Formeln sich spezielle Listen selbst zu implementieren, falls auf die Vollständigkeit für größere Formeln Wert gelegt wird. Weitere Details der Programmierung finden sich im Programmcode selbst (https://github.com/Derstefan/ADFGenerator.git).

4.2 Laplace und KDNF

Die beiden herkömmlichen Generatoren lassen sich sehr gut vergleichen, da sie ohne weitere Parameter als das gegebene n funktionieren. Somit kann man z.B. in zwei einfachen Diagrammen die Generierungsdauer durch den Mittelwert und die Standartabweichung darstellen.

4.2.1 Generierungsdauer

Hierbei bemerkt man, dass beide Generatoren eine exponentiell wachsende Generierungsdauer haben und der KDNF-Generator ein diesbezüglich geringeren Mittelwert aufweist. Andererseits ist die Standartabweichung für den Laplace-Generator im Vergleich zum KDNF-Generator viel geringer.

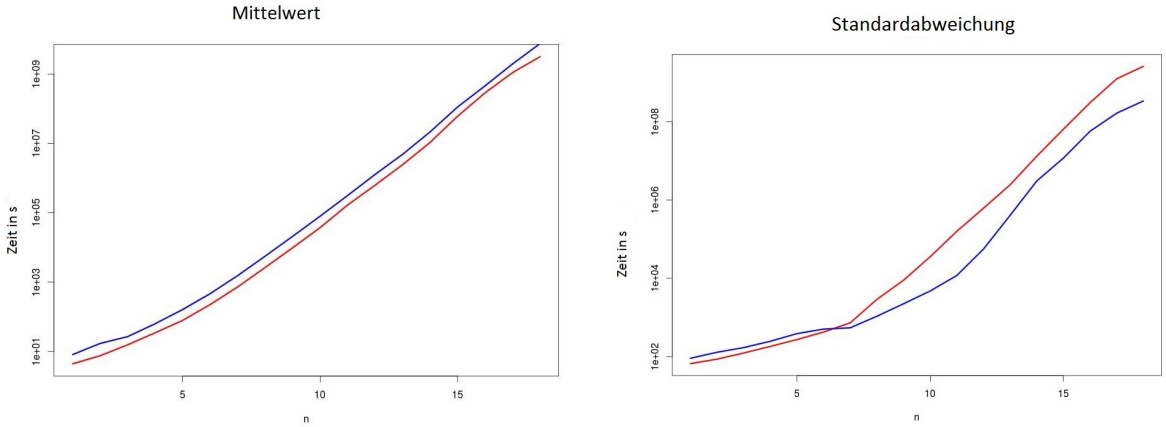

Abbildung 3: Generierungsdauer (Blau) Laplace (Rot) KDNF (links) Mittelwert (rechts) Standardabweichung, mit jeweils 10000 generierten Formeln

Für die Darstellungen dieser Daten wurden bis $n = 16$ jeweils 10000 Formeln generiert und für höhere n aus Gründen der Zeit nur 100 Stück. Da aber weder für die exakte Zeit noch für genaue Verhältnisse Aussagen getroffen werden und nur eine Schätzung der Generierungsdauer erforderlich ist, genügt die Anzahl um die Komplexitätsaussage zu treffen.

4.2.2 Variablen

Für die Anzahl der abhängigen Variablen gibt es in folgender Grafik dargestellt nur wenig zu sagen. Beim Laplace-Generator liegt so gut wie keine einzige Formel außerhalb der Menge von Formeln mit 10 abhängigen Variablen. Da dieser Generator jede Formel mit der gleichen Wahrscheinlichkeit generiert und die a_n enorm schnell ansteigen $(2, 2, 10, 218, 64594, 4294642034, 1.8 * 10^{19}, 3,4 * 10^{38}, \ldots$, diese Reihe wurde in 2.3.1 beschrieben), ist kaum Verwunderung darüber berechtigt, dass nahezu nur volle Formeln (volle Formeln := aussagenlogische Formeln mit genau n abhängigen Variablen) generiert werden. Beim KDNF-Generator ist das ähnlich, wobei er andererseits einen geringen Anteil an trivialen Formeln entstehen lässt.

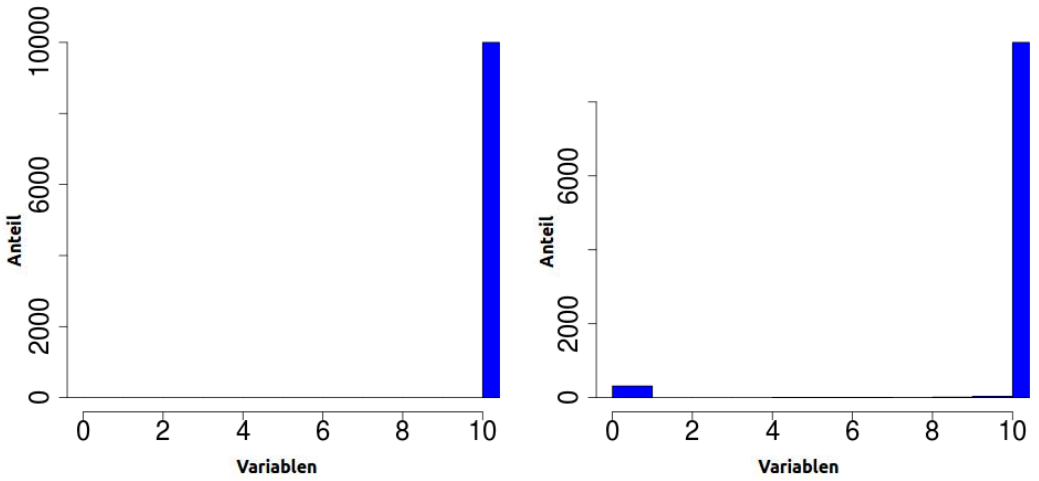

Abbildung 4: Histogramme für die Anzahl der abhängigen Variablen (links) Laplace (rechts) KDNF, mit jeweils 10000 generierten Formeln

4.2.3 Anteil der Variablen

Der Anteil der Variablen in einer Formel berechnet sich einfach als das Verhältnis der Klauseln, welche die Variable beinhalten, bezüglich aller Klauseln in der Formel. Intuitiv erwartet man, dass jede Variable den gleichen Anteil hat, da keine Variable in diesen Generatoren bevorzugt behandelt wird.

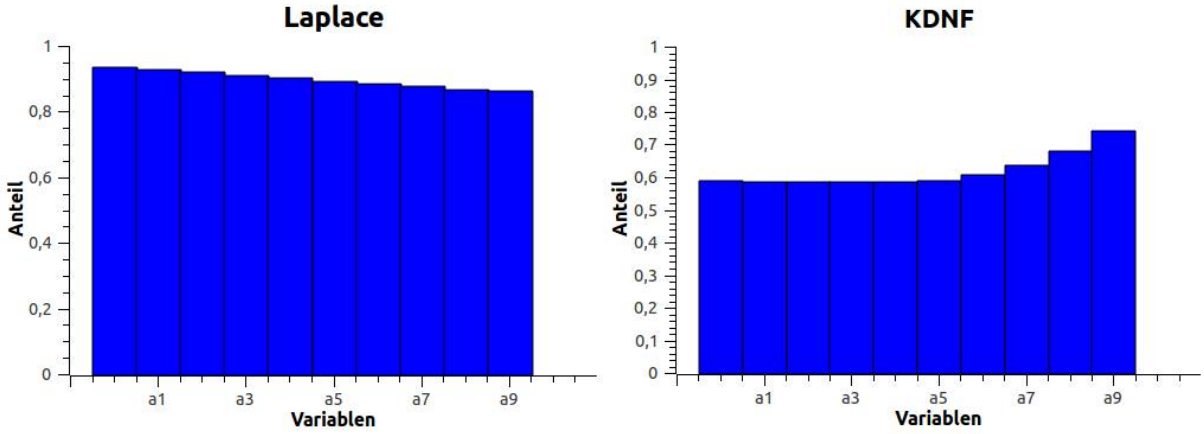

Abbildung 5: Anteil der Variablen (links) Laplace (rechts) KDNF, mit jeweils 10000 generierten Formeln

Jedoch sind hier minimale Unterschiede zu bemerken, die vermutlich durch den Minimierungsalgorithmus entstanden sind. Aus diesem Grund wird nicht weiter auf diese Differenzen eingegangen, die möglicherweise nur auf syntaktischer Ebene interessant sind.

4.2.4 Klauselanzahl

Im folgenden wird ein fundamentaler Unterschied der beiden Generatoren vorgestellt. Dafür wurde die Anzahl der Klauseln in den generierten Formeln berechnet und diese danach klassifiziert, wodurch zwei Histogramme dieser Art entstanden sind.

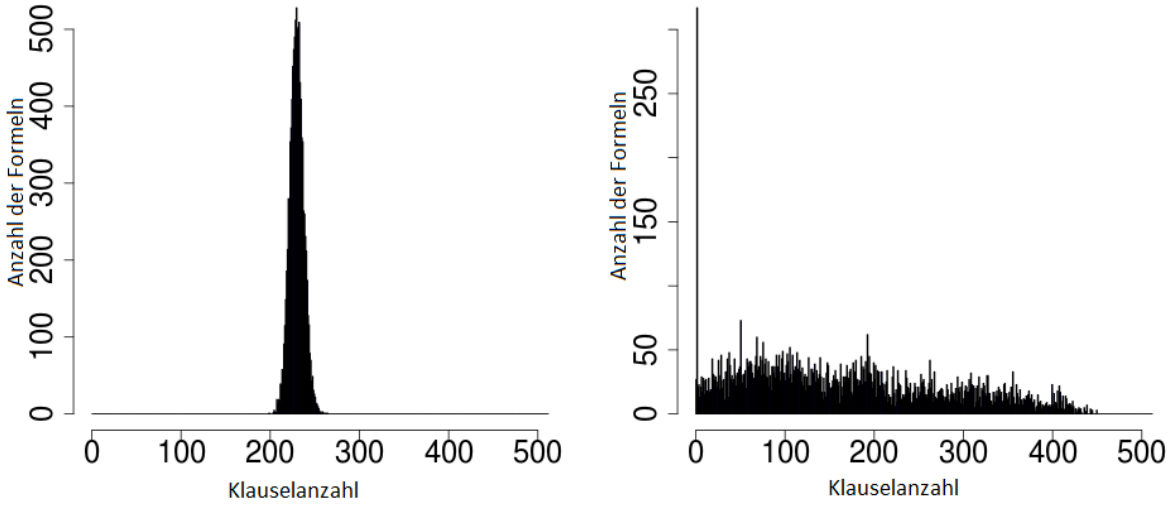

Abbildung 6: Histogramm für die Anzahl der Klauseln (links) Laplace (rechts) KDNF, mit jeweils 10000 generierten Formeln

Deutlich zu erkennen ist, dass der Laplace-Generator Formeln generiert, welche eine Klauselanzahl von ca. 200 bis 250 haben, während der KDNF-Generator eine sehr breite Masse an Formelgrößen erstellt. Dadurch ist die höhere Standardabweichung der Zeit einfach zu erklären. Außerdem ist zu sehen, dass die trivialen Formeln, wie schon in 4.2.2 gezeigt wurde, sehr präsent sind, wobei das bei einem Laplace-Generator nahezu unmöglich ist.

4.2.5 Anteil der Klauseln

Aus beiden folgenden Visualisierungen ist einfach zu sehen, dass der KDNF Generator eine breitere Vielfalt auch in Bereichen der Klauselgröße generiert, wobei der Laplace Generator zumeist nur Formeln aus Klauseln von großer Länge entstehen lässt. Diese Visualisierung zeigt jedoch nur den Mittelwert, wodurch sehr viele Informationen vernachlässigt werden und man diese Grafik nicht fehlinterpretieren darf. Es sei an dieser Stelle nicht gesagt, dass eine Formel genau so eine Zusammenstellung aus Klauseln besitzt. Jedoch lässt sich eine quantitative Aussage treffen, welchen Anteil bestimmte Klauselgrößen erhalten.

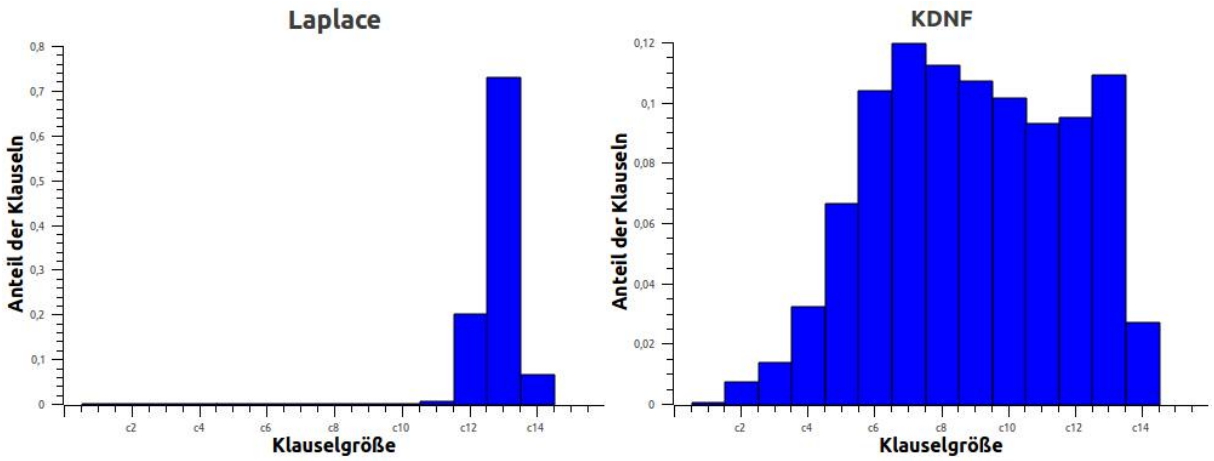

Abbildung 7: Anteil von Klauseln bestimmter Größe (links) Laplace (rechts) KDNF, mit jeweils 10000 generierten Formeln

4.3 STF-Generator mit konstanter Parameterfunktion

Für folgende Beispiele wurde aus Gründen der Einfachheit $t(k) = f(k)$ gesetzt. Somit gilt es nun zu untersuchen, welche Einflüsse diese Funktion t als Parameter für den Generator hat. Anfangs wird der triviale Fall von konstantem t untersucht.

4.3.1 Generierungsdauer

In dieser Visualisierung ist zu sehen, dass die Dauer der Generierung mit geeignetem t viel kleiner ist als in den herkömmlichen Generatoren. Der Mittelwert steigt aber sehr schnell für verschwindend kleine c und für große n.

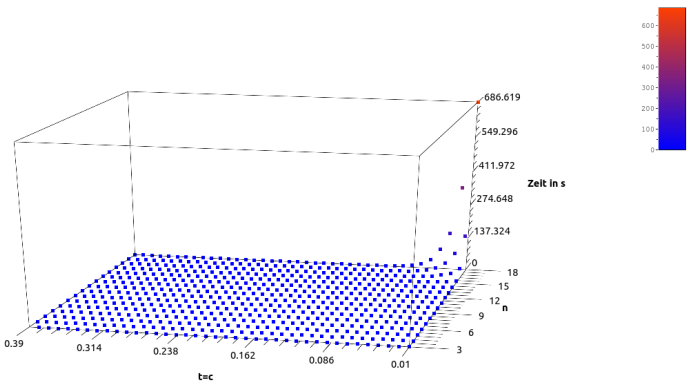

Abbildung 8: Generierungsdauer STF-Generator mit $t = const$, bis $n = 14$ wurden 10000 Formeln und für größere n auf Grund der Zeit 100 Formeln generiert

Es ist einfach nachzuprüfen, dass ein STF-Generator mit $t = 0$ äquivalent zu einem Laplace-Generator ist. Aus dem Grunde ist es nicht von großer Bedeutung, was der Generator mit t in der Nähe der Null entstehen lässt, sondern viel interessanter ist die Frage, welche Arten von Formeln der Generator für höhere t mit viel geringerer Zeit entstehen lässt.

4.3.2 Anzahl abhängiger Variablen

Die Anzahl der Formeln bestimmter Anzahl abhängiger Variablen verändert sich offensichtlich durch verschiedene n und verschiedene t. In den Histogrammen ist t mit eingetragen, wobei die gestrichelte Linie die 0.5 Marke auf der Wahrscheinlichkeitsachse darstellt. Als erstes ist zu bemerken, dass für geringe c (z.B. $t = c = 0.01$) die Gestalt der Formeln nach diesem Kriterium sehr den herkömmlichen Generatoren gleicht. Für höhere c wechselt die Formelart so sehr, dass ausschließlich die trivialen Formeln und jene von geringer Variablenabhängigkeit generiert werden. Interessant ist dabei, dass in der dargestellten Matrix man angedeutet eine Symmetrie entdecken kann. Die Formeln von mittelstarker Variablenabhängigkeit werden im Verhältnis, und das besonders mit zunehmenden n, relativ selten generiert, wobei später noch eine Möglichkeit vorgestellt wird dieses Verhältnis anzugleichen.

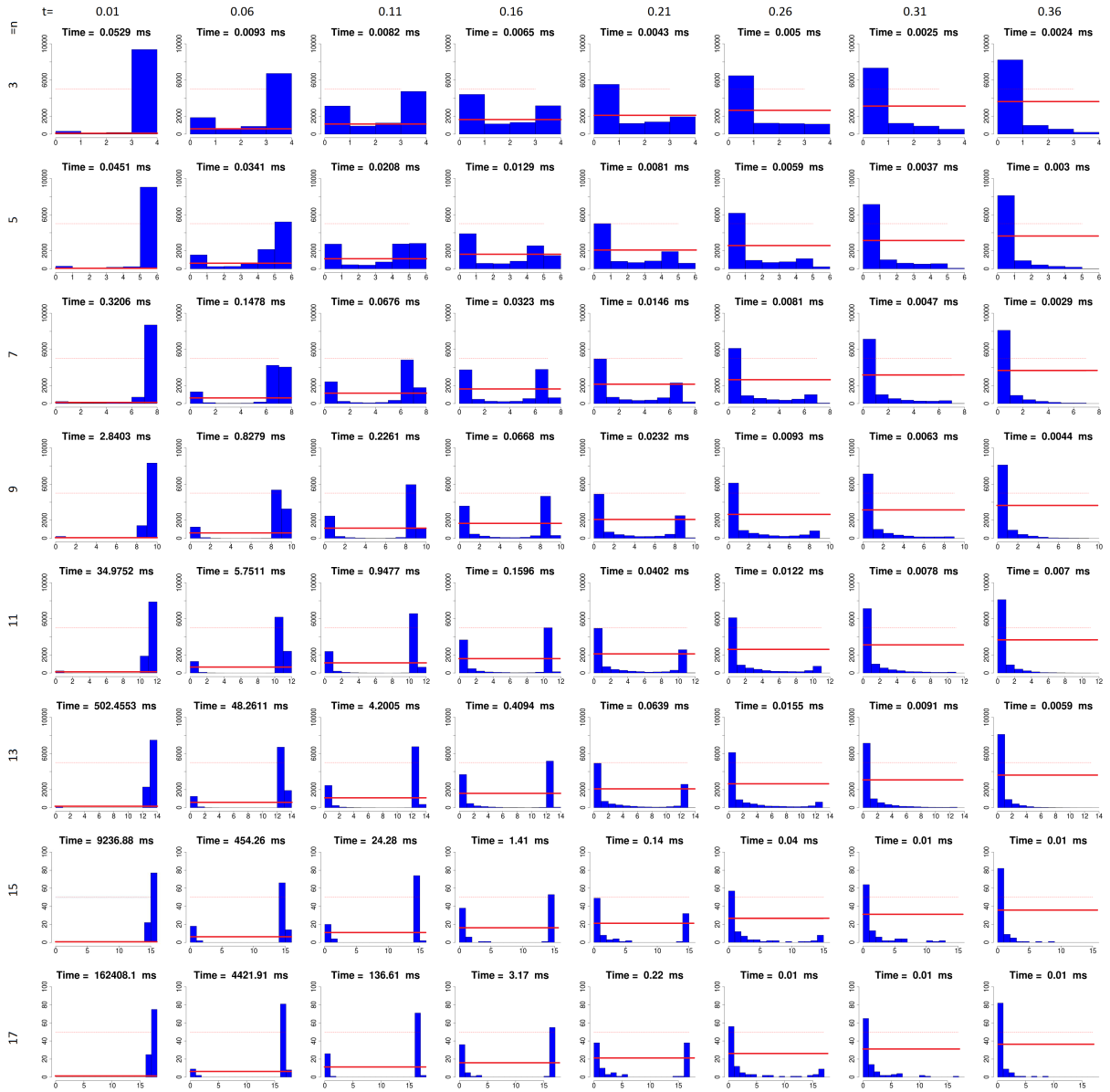

Abbildung 9: Histogramm für Anzahl abhängiger Variablen mit Generierungsdauer für STF-Generator mit t=const (roter Graph) , bis $n = 14$ wurden 10000 Formeln und für größere n auf Grund der Zeit 100 Formeln generiert

Sehr stark schwindet außerdem für leicht wachsendes c die Anzahl der Formeln von n Variablenabhängigkeiten. Somit sind die beiden konträren Säulen dieser Histogramme nicht nur die trivialen und die vollen (n abhängige Variablen) Formeln, sondern besonders jene, die von $n - 1$ Variablen abhängen. Für diese Bachelorarbeit lautet nicht die Forderung zu untersuchen, mit welchen Mitteln man die vollen und trivialen Formeln gleitend durch ein gewähltes c variieren kann. Jedoch vermute ich, dass $t(n - 1)$ viel geringer sein muss, damit die vollen Formeln größere Beachtung durch den Generator erfahren.

4.3.3 Stärke der Abhängigkeiten jeder Variable

Interessant am STF-Generator ist, dass die Variablen stark unterschiedliche Wichtungen in der generierten Formel besitzen. Das bedeutet gleichzeitig, dass die Reihenfolge der Variablen entscheidend für die Art der Formel ist. Für diese Eigenschaft genüge diese eine Visualisierung als Beispiel, da sie nur verdeutlichen soll, mit welchen Eigenschaften man für solche generierten Formeln rechnen muss. Für weitere Untersuchungen sind möglicherweise Folgearbeiten zuständig.

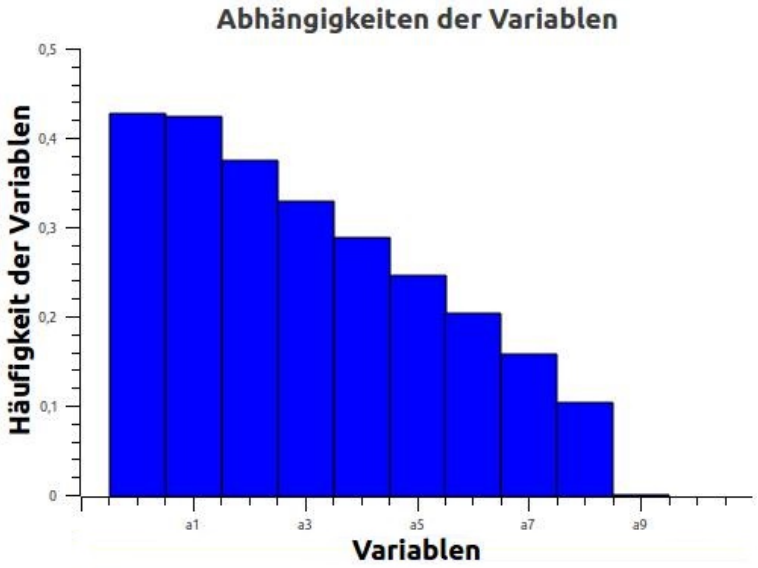

Abbildung 10: Abhängigkeiten der Variablen $n = 10$ und $t = 0.2$, mit 10000 generierten Formeln

4.3.4 Anzahl der Klauseln

Die folgenden Visualisierungen zeigen, genau wie 4.2.4, die Anzahl der Formeln einer bestimmten Klauselanzahl. Für höhere c erhält man, wie man intuitiv erwartet, Formeln von geringerer Größe.

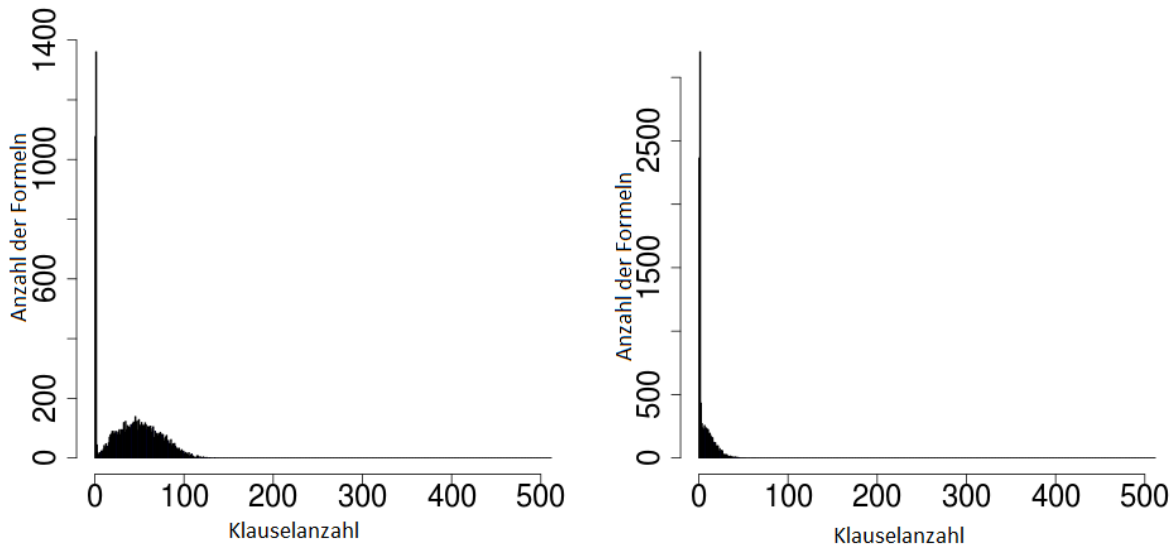

Abbildung 11: Histogramm für Formeln bestimmter Klauselanzahl $n = 10$, $t = 0.1$ (links) $t = 0.2$ (rechts), mit 10000 generierten Formeln

4.3.5 Anteil der Klauseln bestimmter Größe

Mit zunehmendem c sinken die Größen der Klauseln in den generierten Formeln. Somit ist die Aussage gerechtfertigt, dass die Formeln an Einfachheit gewinnen. In der ersten der folgenden Abbildungen sind im Gegensatz zu der Abbildung rechts daneben genau zwei Maxima zu sehen. Der Grund dafür liegt daran, dass mit den konstanten Parameterfunktionen t, f die Wahrscheinlichkeit für eine triviale Formel besonders groß ist.

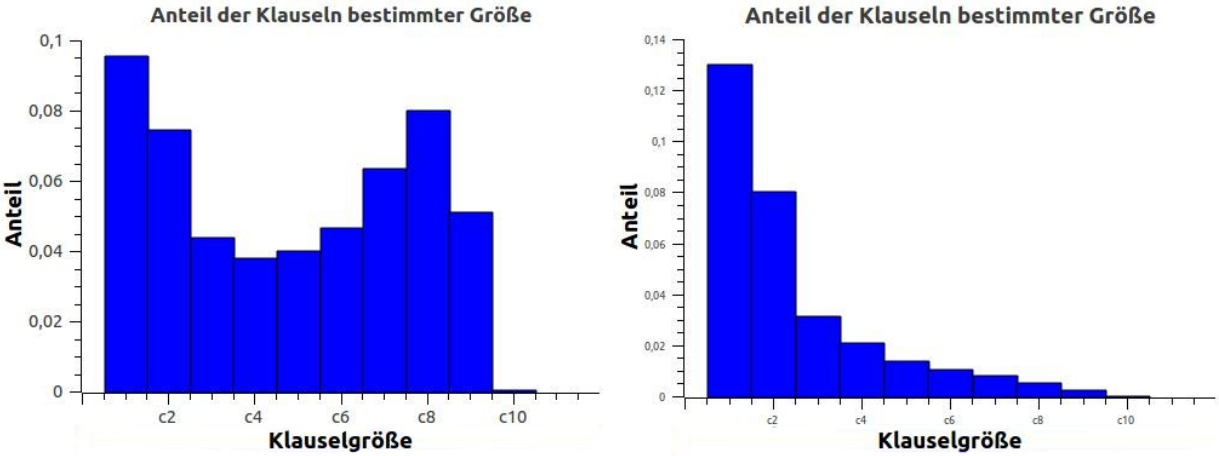

Abbildung 12: Anteil von Klauseln bestimmter Größe $n = 10$, $t = 0.2$ (links) $t = 0.3$ (rechts), mit 10000 generierten Formeln

4.4 STF-Generator mit einer Wurzelfunktion als Parameter

Nun gilt es zu untersuchen, welche anderen Funktionen für t interessante, andere Formelklassen generieren. Als Motivation dient dafür die Überlegung, dass die Formeltypen, welche in 4.3.2 dargestellt wurden, von nahezu der gleichen Verteilung generiert werden sollen. Als Parameterfunktion wähle man hierfür $t = (c) * (k/n)^m$. Für dieses neue t wird nur diese semantische und technische Eigenschaft untersucht, denn die syntaktischen unterscheiden sich nicht relevant von denen, welche für $t = constant$ untersucht wurden. In den folgenden Beispielen sind die Darstellungen abhängig der Parameter c und m gewählt und das n ist für jeden Diagrammkomplex konstant.

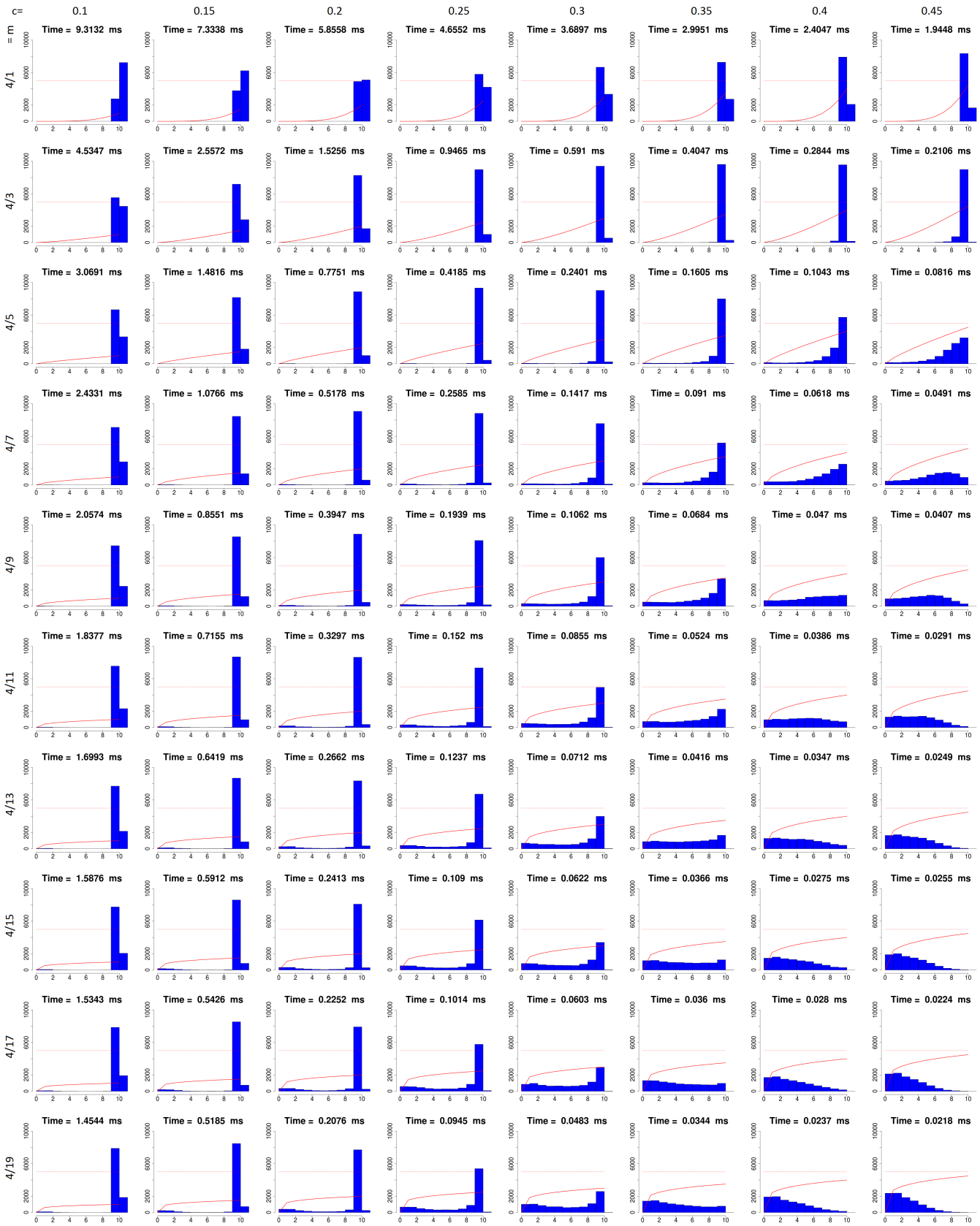

Abbildung 13: Histogramme für die Anzahl abhängiger Variablen mit Generierungsdauer für STF-Generator $n = 10$ und $t(k) = c * (k/n)^m$, mit 10000 generierten Formeln

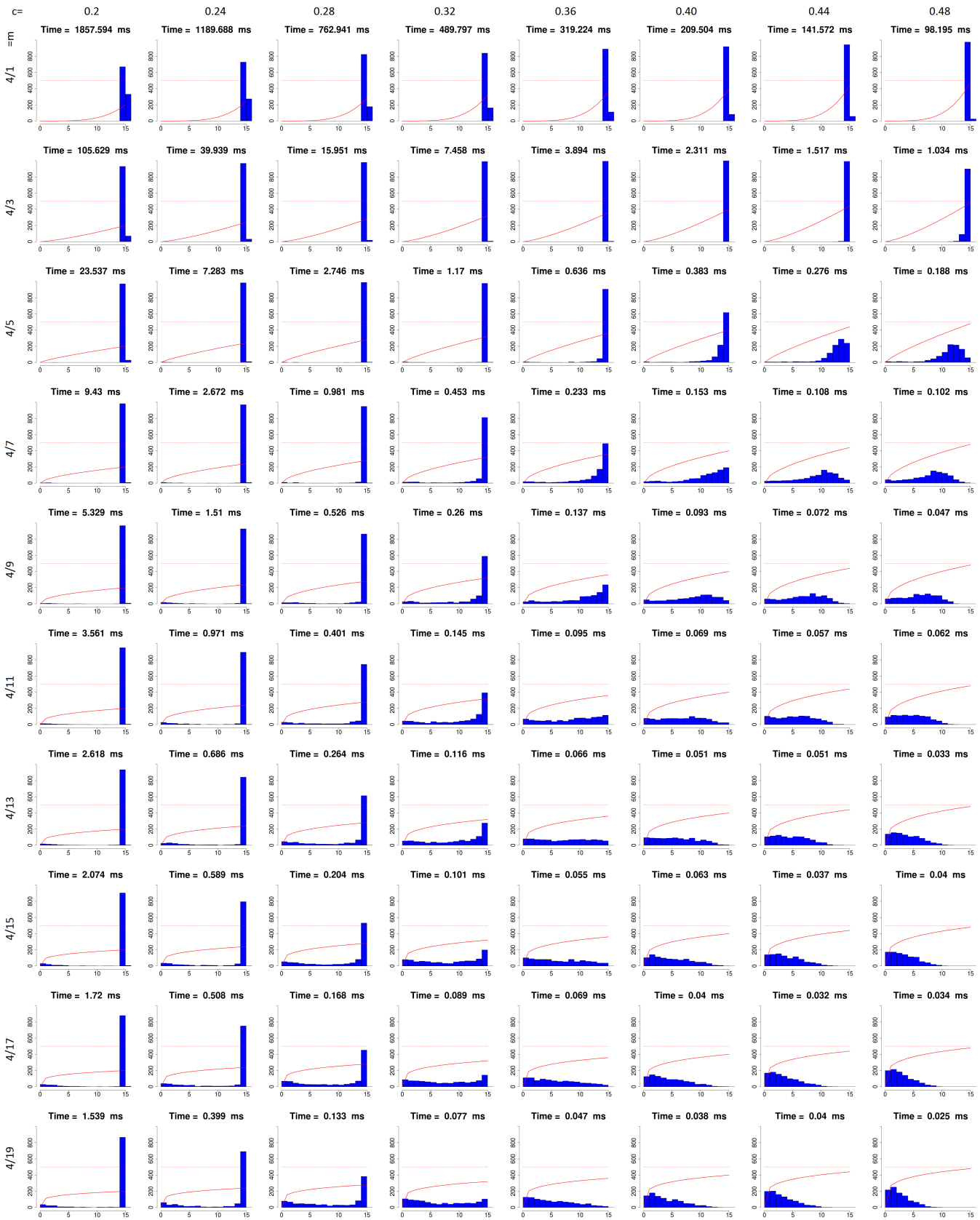

Abbildung 14: Histogramme für die Anzahl abhängiger Variablen mit Generierungsdauer für STF-Generator $n = 15$ und $t(k) = c * (k/n)^m$, mit 1000 generierten Formeln

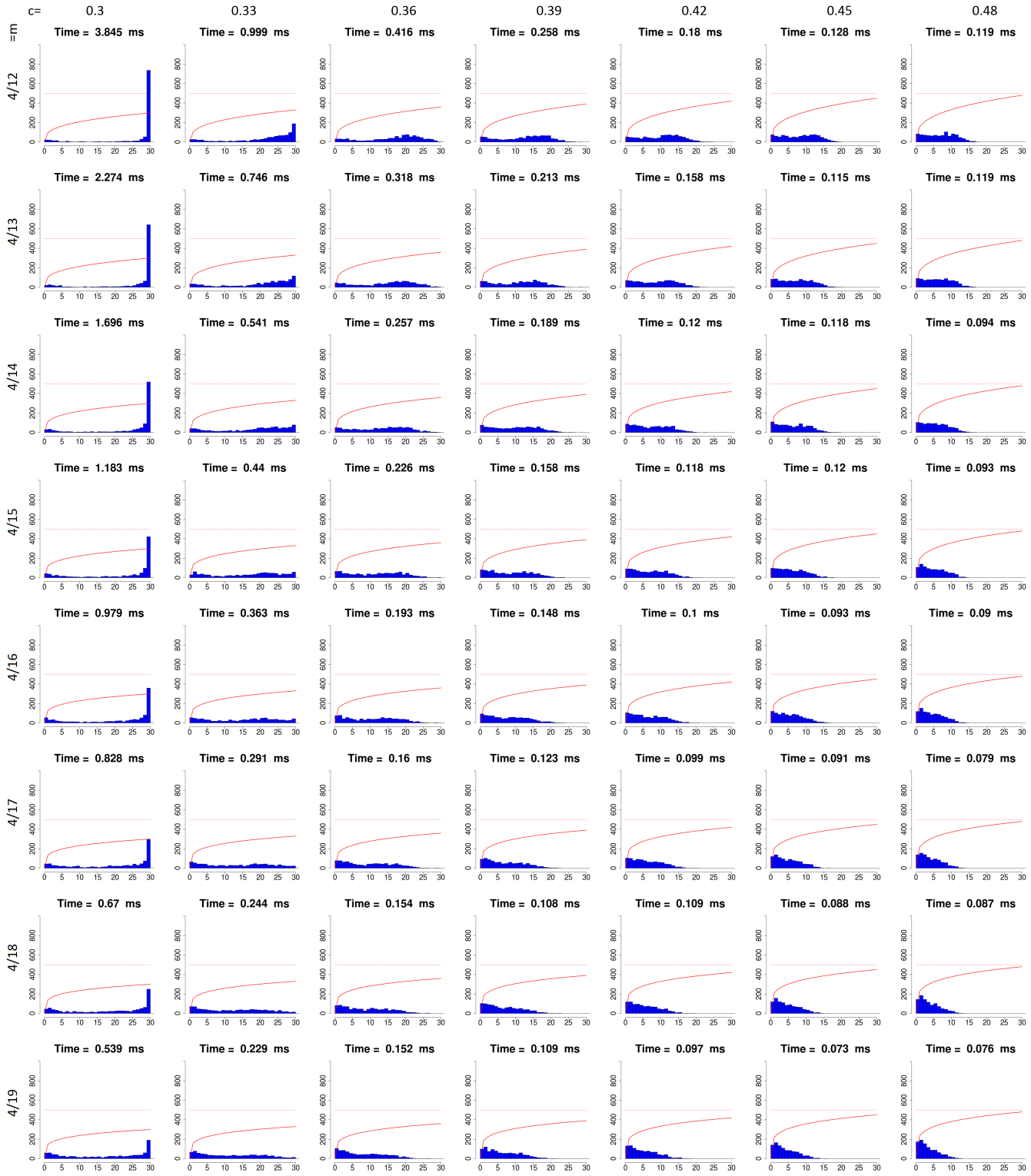

Abbildung 15: Histogramme für die Anzahl abhängiger Variablen mit Generierungsdauer für STF-Generator $n = 30$ und $t(k) = c * (k/n)^m$, mit 1000 generierten Formeln

Ähnlich der Visualisierung des Abschnittes 4.3.2 ist hier ein rapides Verschwinden der vollen Formeln zu bemerken, wobei jedoch die $n - 1$ starken Formeln wieder eine fokussierte

Menge der Generierung darstellen. Jedoch gibt es hier (bis zu einem bestimmten Wert n) eine Geichverteilung der Formeln, welche für konstantes t nicht möglich war. Um diese Gleichverteilung auch für hohe n zu ermöglichen genügt die Parametrisierung durch dieses t nicht. Außerdem sind die trivialen Formeln nicht so wahrscheinlich, wie für das konstante t, was offensichtlich der Gleichung von $t(0) = 0$ zu Schulde kommt.

4.5 Resultat

Der Vergleich der Generatoren ist auf unterschiedliche Weise zu betrachten. Hierbei gibt es die Generierungszeit, für welche man wohl für jeden Gebrauch ein Minimum anstrebt. Auf der anderen Seite vergleiche man die Art der generierten Formeln. Für diese Eigenschaften gibt es keine solche Qualitätsordnung wie bei der Zeit, da diese Merkmale stark von der bevorzugten Anwendung abhängen. Durch den STF-Generator werden, im Gegensatz zum Laplace-Generator und KDNF-Generator, aussagenlogische Formeln mit einer viel größeren Geschwindigkeit erzeugt. Selbst für $n = 65$ werden Formeln generiert, welche im Kriterium Variablenabhängigkeit stark den herkömmlichen Generatoren (mit dem Unterschied, dass die meisten von 64 Variablen abhängen) ähnelt, obwohl sie mit $m = 1/3$ und $c = 0.3$ eine Generierungszeit von ca. 12 Sekunden haben, wobei wohl die Standardgenerierung mit dem Laplace- oder dem KDNF-Generator für diese Größe mehr als ein Menschenleben benötigt. Die vollständige Datensammlung in Form von Visualisierungen befindet sich auf der Repository (https://github.com/Derstefan/ADFGenerator.git). Andererseits arbeiten für bestimmte triviale Parametrisierung als Beispiel $t = 0$ wohl die ursprünglichen Generatoren schneller. Die generierten Formeln unterscheiden sich dafür auf andere Weise. Es gibt für den STF-Generator als Beispiel eine andere Zusammenstellung aus verschiedensten Klauselgrößen, während im Laplacegenerator die Formeln meist ausschließlich aus großen Klauseln bestehen und der KDNF-Generator eine Vielfalt von allen Klauselgrößen erstellen lässt. Der STF-Generator erstellt Formeln, die im Allgemeinen viel weniger Klauseln beinhalten. Weiterhin gibt er im Vergleich zu den ersten Generatoren den Variablen eine stark erkennbare Wichtung. Es lässt sich allgemein formulieren, dass die STF-Generierten Formeln eine intuitiv einfachere Gestalt annehmen und dies als Zeit- und Speicherersparnis für die Generierung nutzen.

5 Anwendung für ADF-Generatoren

Die Theorie unterschiedlicher Formelgeneratoren gibt keine Auskunft darüber, welcher dieser Generatoren besser oder schlechter ist. Mit Ausnahme der Generierungsdauer werden Formeln erzeugt, deren Eigenschaften für verschiedene Anwendungen einen unterschiedlichen Fokus erhalten. Nur sehr schwer lassen sich allgemeine Aussagen darüber formulieren, welche Ergebnisse unterschiedlich generierte Formeln für bestimmte Anwendungen erzielen. Als Anwendung der bisher untersuchten Formelgeneratoren wird die Evaluierung von ADFs (Abstact Dialectical Framworks) mittels der Software DIAMOND, welche von Stefan Ellmauthaler und Hannes Strass in [4] vorgestellt wurden, überprüft. Somit wird in diesem Kapitel einerseits dargestellt, welche starken Unterschiede die Generatoren im Ergebnis hervorbringen. Andererseits gibt

es Auskunft darüber, welchen Formelgenerator man wählen sollte, damit die dargestellten Daten von den ADFs hervorgebracht werden.

Definition 1. *Ein Abstract Dialectical Framework ist ein gerichteter Graph, welcher als ein Tupel $D = (S, L, C)$ definiert ist. Hierbei ist S die Menge der Knoten dieses Graphen und L die Menge der Kanten zwischen den Knoten aus S ($L \subseteq S \times S$). $C = \{C_s\}_{s \in S}$ ist die Menge der Akzeptanzbedingungen bezüglich jeweils eines Knotens, wobei für diese $C_s : 2^{par(s)} \to \{t, f\}$ gilt.*

Auf Grund der Definition von ADFs kann man die Akzeptanzbedingungen auch als aussagenlogische Formel betrachten. Hier befindet sich die Schnittstelle für die bisher untersuchten Formelgeneratoren. Insgesamt werden acht Semantiken untersucht, welche hier nicht bis ins Detail definiert werden. Für genauere Untersuchungen sei auf die Publikationen von Brewka und Woltran [5] verwiesen. Die Generierung der ADFs ist in dieser Arbeit nur mit Hilfe der Standardbibliothek von Java neu implementiert. Die Evaluierung der ADFs geschieht mit einer Software Namens DIAMOND.

5.1 Implementierung

Der Generator wurde in der Klasse *ADFGenerator.java* implementiert. In der Funktion *generate*() wird am Anfang die Anzahl der Knoten zwischen den Parametern *min* und *max* und, durch ein Maximum begrenzt, die mögliche Anzahl der Kanten festgelegt. Anschließend wird für jeden Knoten eine Akzeptanzbedingung, in Form einer aussagenlogischen Formel, generiert. An dieser Stelle befindet sich die Verbindung zu den untersuchten Formelgeneratoren dieser Arbeit. Die ADFs werden im Pfad, welcher in der Datei *config.txt* eingegeben wird, gespeichert. Danach werden diese Graphen mit Hilfe eines ProcessBuilder im Terminal zur gewählten Semantik automatisch ausgeführt und nach Modellen untersucht. Schließlich werden die Auswertungsdaten (die Anzahl der Modelle und die Bearbeitungsdauer) mit einem Parser herausgefiltert und die Ergebnisse in einer CSV-Datei gespeichert.

5.2 Auswertung

Neben der eigentlichen Untersuchung der Formelgeneratoren ist hier ebenfalls relevant und interessant, welchen Einfluss die unterschiedlichen Semantiken auf die Bearbeitungszeit und die Anzahl der Modelle eines untersuchten ADFs haben. Die Auswertung vertieft sich jedoch nicht in die Details der speziellen Semantiken. Dieses Kapitel nutzt diese Anwendung fast ausschließlich für den Vergleich verschiedener Formelgeneratoren um festzustellen, ob es Gemeinsamkeiten und Unterschiede gibt. Für die Formelgeneratoren wurden vier Beispiele (Laplace-Generator, KDNF-Generator, STF-Generator mit konstanter Parameterfunktion $t = 0.15$ und STF-Generator mit $t(k) = 0.3(k/n)^{1/3}$) als Parameterfunktion ausgewählt, welche unterschiedliche, innerer Formelstrukturen für die Anwendung generieren sollen. Im Folgenden werden die einzelnen Semantiken untersucht und die Ergebnisse bezüglich unterschiedlicher Formelgeneratoren (mit jeweils 1000 generierten ADFs) verglichen. Als terminologische Einigung sei hier erwähnt, dass die speziellen ADF-Generatoren nach dem

Formelgenerator benannt erwähnt werden (KDNF, Laplace, STF1 für das konstante t, und STF2 für das nicht konstante t). Weiterhin findet sich in den Semantiken *stage* und *native* der Hauptgrund, weshalb die maximale Anzahl an Knoten so begrenzt wurde. Da die Dauer schon bei 10 Knoten bis zu einer Stunde am Versuchsrechner (Intel Core i3-2350M CPU @ 2.30GHz x 4 mit 3,8 GiB) anstieg, begrenzt nun der Bereich bis zu 9 Knoten die allgemeine Wertigkeit der empirischen Aussagen. Im folgenden werden die Visualisierungen der ADFs aufgelistet, deren Histogramme ihrer Modelle und deren Bearbeitungszeit kaum Unterschiede aufweisen. Anschließend werden die Semantiken mit zunehmenden Unterschieden vorgestellt.

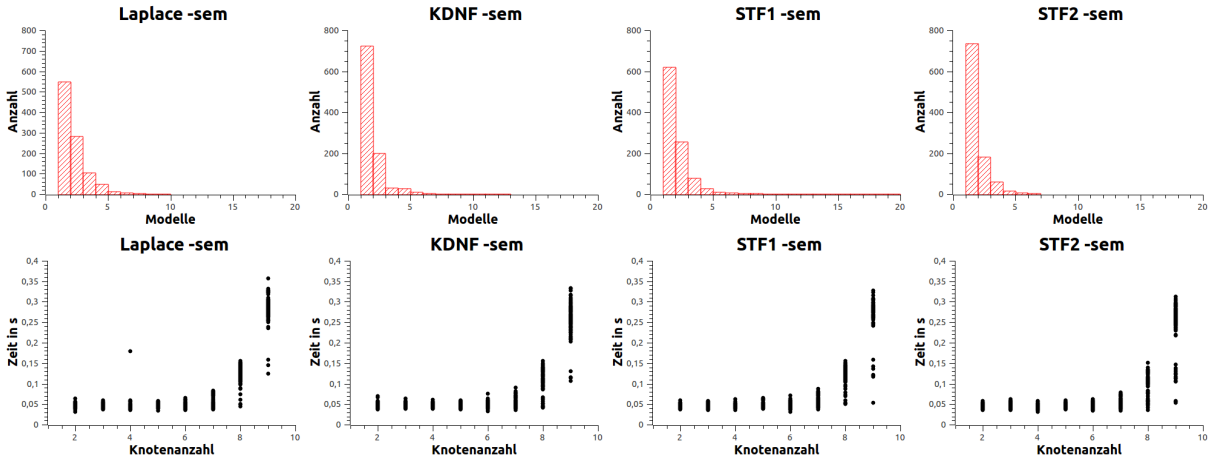

Abbildung 16: Semantik: *semi-model* mit 1000 ADFs je Formelgenerator

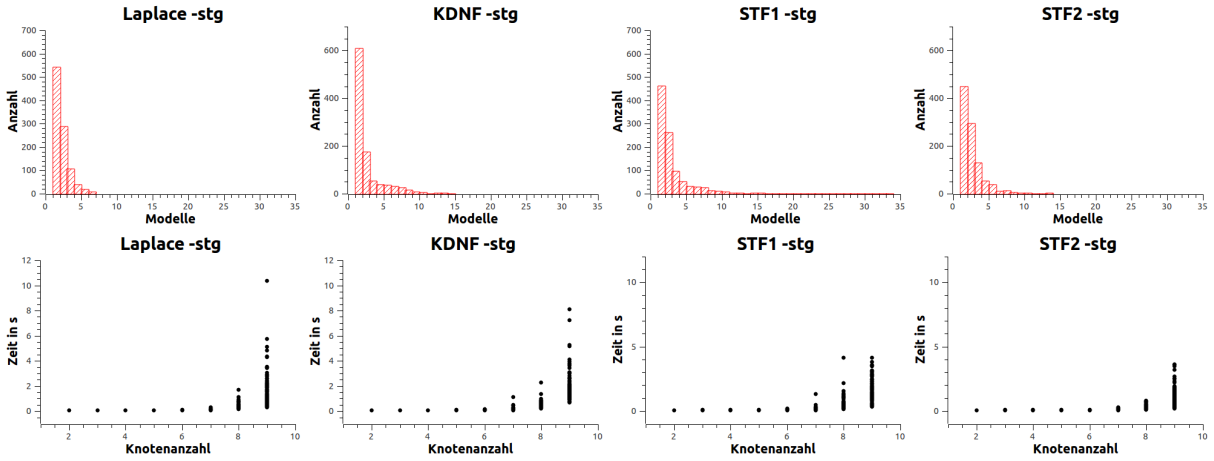

Abbildung 17: Semantik: *stage* mit 1000 ADFs je Formelgenerator

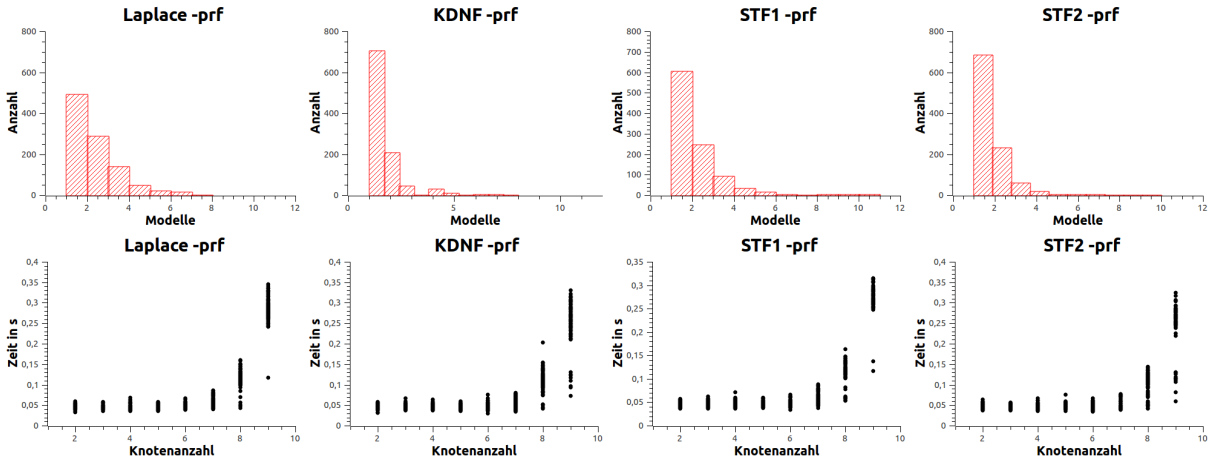

Abbildung 18: Semantik: *preferred* mit 1000 ADFs je Formelgenerator

Nur sehr kleine Unterschiede fallen auf, da z.B. STF2 und KDNF Generatoren ADFs von zumeist kleinerer Modellanzahl in diesen spezifischen Semantiken generieren. Außerdem ist z.B. von *stage* mit den STF Generatoren in den höheren Knotenzahlen eine leicht schnellere Bearbeitungsdauer zu vermerken. Jedoch werden diese Unterschiede auf Grund ihrer minimalen Gestalt vernachlässigt. Der Vergleich der folgenden Semantiken ist durch größere Unterschiede geprägt.

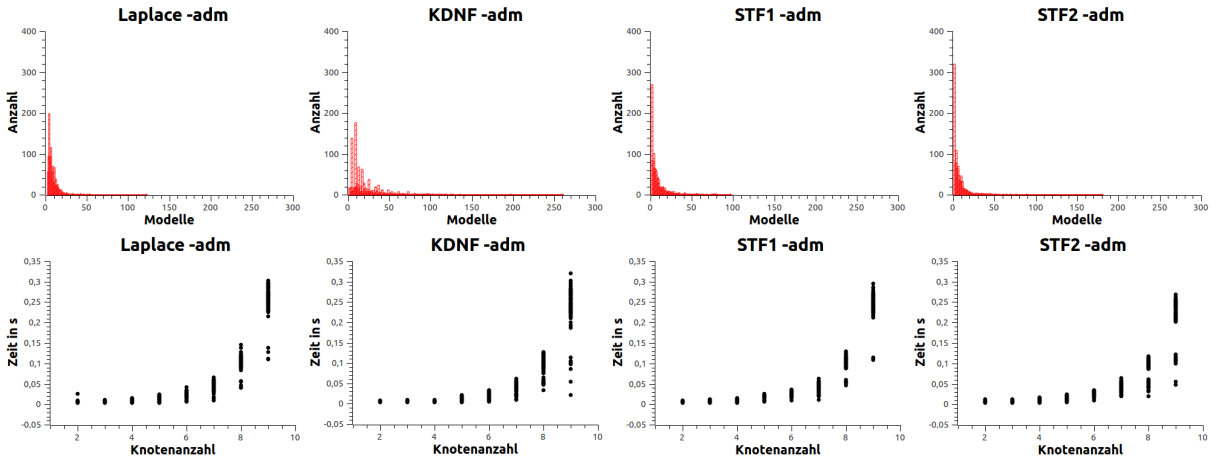

Abbildung 19: Semantik: *Admissible* mit 1000 ADFs je Formelgenerator

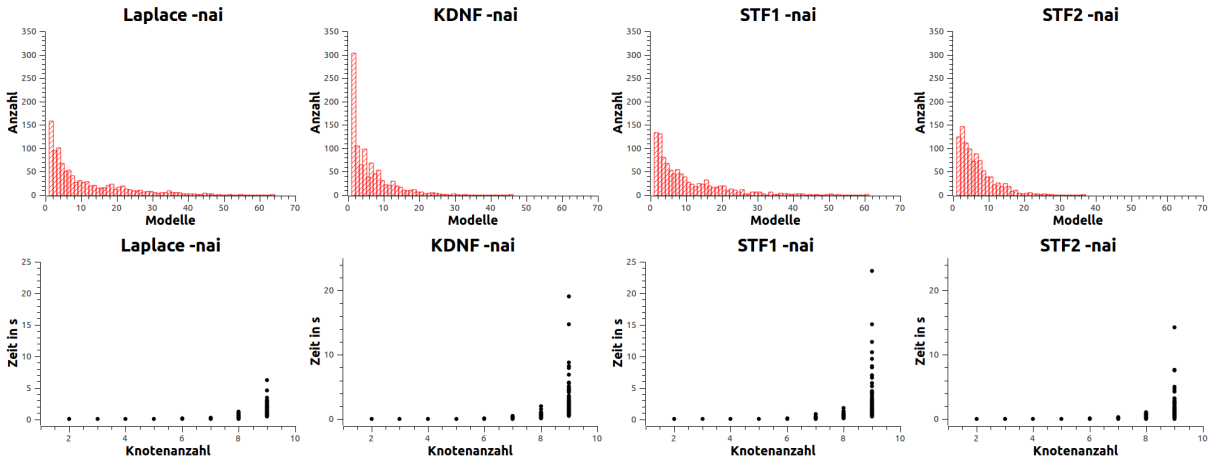

Abbildung 20: Semantik: *naive* mit 1000 ADFs je Formelgenerator

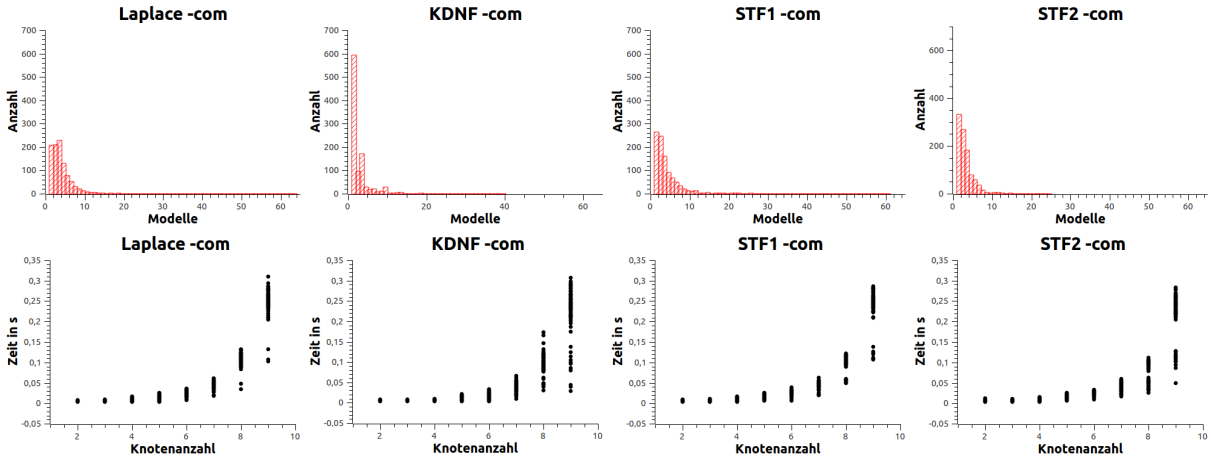

Abbildung 21: Semantik: *complete* mit 1000 ADFs je Formelgenerator

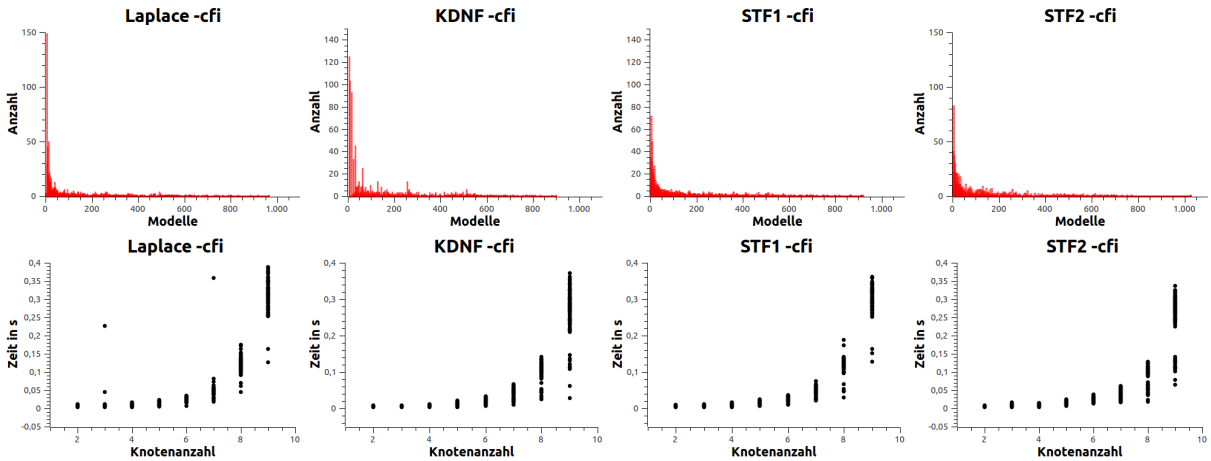

Abbildung 22: Semantik: *conflict-free* mit 1000 ADFs je Formelgenerator

Besonders fällt auf, dass der KDNF-Generator sich in den meisten Semantiken verschieden zu den anderen Generatoren verhält. Die Charakteristiken dieser Unterschiede sind jedoch stark

von der untersuchten Semantik abhängig. In den Semantiken *conflict-free* und *admissible* ist zu sehen, dass der KDNF-Generator häufiger als die übrigen Generatoren ADFs von größerer Modellanzahl entwickelt und genau wie der STF2-Generator ein Zeitersparnis (im Gegensatz zu Laplace und STF1) in der Bearbeitungsdauer für große Knotenanzahl ermöglicht. Interessant ist hier, dass die Anzahl der ADFs mit wenigen Modellen für *admissible* mit dem Laplace-Generator gering und mit dem STF2-Generator maximal ist, jedoch die für die andere Semantik *conflict-free* es nahezu umgekehr ist. Zu *Native* und *Complete* ist zu bemerken, dass der KDNF-Generator besonders viele ADFs mit genau einem Modell generiert und ausschließlich für *Native* kann der KDNF- und der STF1-Generator für große Knotenanzahl eine längere Bearbeitungsdauer benötigen. Nun wird ein interessanter Fall der Semantik *Two-valued models* präsentiert. Hier wird ein fundamentaler Unterschied festgestellt.

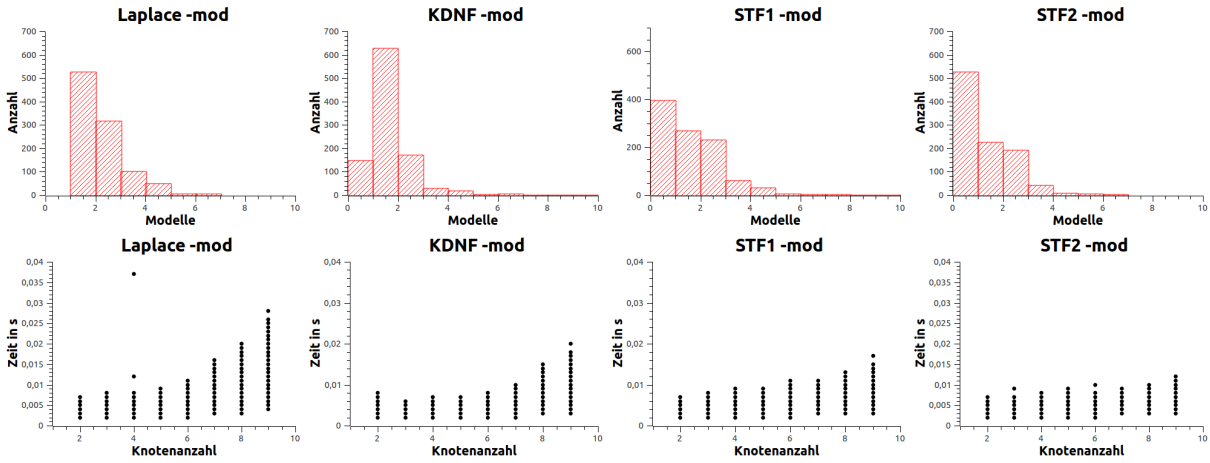

Abbildung 23: Semantik: *Two-valued models* mit 1000 ADFs je Formelgenerator

In der Visualisierung wird ersichtlich, dass die Anzahl der ADFs, welche für diese Semantik kein Modell besitzen, durch den Laplace-Generator kaum bis nicht generiert werden, jedoch für den KDNF-, STF1- und für STF2-Generator in dieser Reihenfolge bis zum Maximum zunehmen und die Bearbeitungsdauer für größere Knotenanzahl geringer ist.

5.3 Vergleichsresultate

Durch diese Anwendung wird sehr deutlich, dass unterschiedliche Formelgeneratoren allgemein entscheidende Unterschiede aufweisen können, dennoch gibt es Anwendungen deren Unterschiede verschwindend gering sind. Es lässt sich keine stärkere Aussage formulieren, als dass die Generatoren einen enormen Einfluss auf die Anwendung haben können, jedoch dies von der Anwendung selbst abhängig ist.

6 Ergebnisse dieser Arbeit

Eine der Haupterkenntnisse dieser Arbeit ist, dass die Wahl des Formelgenerators für die Ergebnisse einer Anwendung entscheidenden Einfluss haben kann. Unabhängig von Anwendungen lassen sich sehr schwer Aussagen formulieren, welche die erwarteten Ergebnisse beschreiben. Vorteilhaft ist eine Wahl, falls die semantischen Ergebnisse verschiedener Generatoren einander gleichen, jedoch die Bearbeitungsdauer verschieden ist. Dieser Fall ist in der untersuchten Anwendung eingetreten (z.B. in *stage*). Diese Arbeit bietet ein Instrumentarium für Simulationen und andere Anwendungen einen Generator auszuwählen, der individuelle Formelarten generiert. Der in dieser Arbeit entwickelte STF-Generator erstellt, mit entsprechender Eingabe, Formeln mit größerer Einfachheit, wodurch eine extrem schnellere Generierung möglich ist. Dies ist der wichtige, in dieser Arbeit nachgewiesene Vorteil des STF-Generators. Durch deren Parametrisierung wird dem Nutzer ermöglicht die generierten Formeln nach eigenem Ermessen anzupassen. Es wurde festgestellt, dass die Änderung dieser Parameter auch eine fundamentale Änderung der Formelart bewirken kann und diese auch große Unterschiede in der Anwendung für ADFs aufweist (z.B. in *two-valued models*). Die Untersuchung, welche Parameterwahl der Realität am besten entspricht, obliegt dem Anwender selbst. Der Einfluss dieser Parameterfunktionen t, f auf die Eigenschaften der generierten Formeln wurde in dieser Arbeit für zwei Beispielfunktionen untersucht. Somit gibt es für weitere Untersuchungen noch ungeklärte Fragen. Zum Beispiel wurde noch nicht überprüft, ob es eine Funktion gibt, welche die Gleichverteilung der Anzahl abhängiger Variablen bewirkt. Weiterhin wäre interessant, welchen Einfluss eine Ungleichheit der Parameterfunktionen mit sich führt.

Literatur

[1] U. Schöning, *Logik für Informatiker*. Spektrum Akademischer Verlag, 5 ed., 2000.

[2] E. Burke and E. Foxley, *Logic and its Applications*. Prentice Hall Europe, 1996.

[3] J. B. M. Franco, N. Krasnoger, *Analysing BioHEL Using Challenging Boolean Functions*. 20. Juli 2010. http://de.slideshare.net/mariauxfranco/analysing-biohel-using-challengi Zugriff 12.9.2015.

[4] S. Ellmauthaler and H. Strass, "The DIAMOND System for Computing with Abstract Dialectical Frameworks," in *Proceedings of the Fifth International Conference on Computational Models of Argument (COMMA)* (S. Parsons, N. Oren, and C. Reed, eds.), vol. 266 of *Frontiers in Artificial Intelligence and Applications*, (The Scottish Highlands, Scotland, United Kingdom), pp. 233–240, IOS Press, Sept. 2014.

[5] G. Brewka and S. Woltran, "Abstract dialectical frameworks," in *Proceedings of the Twelfth International Conference on Priciples of Knowledge Representation and Reasoning (KR)*, pp. 102–111, AAAI Press, 2010.

Abbildungsverzeichnis

Selbstständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann

Leipzig, d. _____ _____

Ort, Datum Unterschrift