Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Theoretische Informatik

Backdoors in Default-Logik

Masterarbeit
im Studiengang Informatik

von
Daniel Benjamin Schulz
Matrikelnummer: 2788680

Prüfer: Prof. Dr. Heribert Vollmer
Zweitprüfer: Dr. Arne Meier
Betreuer: M. Sc. Anselm Haak

24. Oktober 2018

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Default Logik ist eine Erweiterung formaler Logiken um Default-Regeln, mit denen es möglich ist unter bestimmten Bedingungen Annahmen zu treffen. Dabei setzten sich diese Bedingungen jeweils aus zwei Teilen zusammen: die Vorbedingung und die Rechtfertigungen. Die Vorbedingung ist eine Formel, die erfüllt sein muss, um die Default-Regel anzuwenden, während die Rechtfertigungen eine oder mehrere Formeln sind, für die es reicht, dass nicht gezeigt werden kann, dass sie nicht gelten. Es kann also auch mit oder gerade wegen unvollständigem Wissen über den Wahrheitswert der Variablen der Rechtfertigungen diese erfüllt werden. Sind sowohl Vorbedingung als auch Rechtfertigungen erfüllt, darf angenommen werden, dass die Schlussfolgerung gilt.

Eine Default-Theorie ist ein Paar $T = \langle W, D \rangle$, wobei W, die Wissensbasis, eine Menge logischer Formeln und D eine Menge von Default-Regeln ist. Die Formeln einer Default-Theorie dürfen von einer beliebigen Logik sein, aber in dieser Arbeit betrachten wir nur Default-Theorien mit Aussagenlogischen Formeln. Durch Anwendung der Defaul-Regeln können die Schlussfolgerungen zur Wissensbasis hinzugefügt werden. Erreicht man einen Zustand, in dem keine Default-Regel mehr angewendet werden kann und die Wissensbasis keinen Widerspruch enthält, dann nennt man die so entstandene Formelmenge eine *Stabile Erweiterung* der Default-Theorie.

Wir interessieren uns für die Frage, ob eine gegebene Default-Theorie eine stabile Erweiterung hat oder nicht. Das zugehörige Entscheidungsproblem wird Extension Existence Problem (EXT) genannt. Die Komplexität von EXT ist dabei abhängig von den Formeln der Default-Theorie. Für aussagenlogische Formeln in konjunktiver Normalform (KNF) ist EXT Σ_2^P-vollständig. Nimmt man stattdessen Formeln einer eingeschränkteren Formel-

klasse, lässt sich die Komplexität reduzieren. Verwendet man beispielsweise Krom-Formeln (2-KNF) sinkt die Komplexität auf NP.

In dieser Arbeit betrachten wir EXT im Rahmen der Parametrisierten Komplexität. Dabei verwenden wir das aus der Aussagenlogik bekannte Konzept der Backdoors. Ein starkes $\mathcal{F}$-Backdoor ist eine Teilmenge der Variablen der Formeln, welche bei beliebiger Belegung dazu führt, dass die vereinfachten Formeln von der Klasse $\mathcal{F}$ sind. Fichte, Meier und Schindler [9] führten Backdoors in die Default-Logik ein. Ihrer Arbeit folgend wird gezeigt, wie sich die Definition der Backdoors für Default-Logik von der der Aussagenlogik unterscheidet. Anschließend gehen wir darauf ein, wie Backdoors für die Klassen Horn, Krom, Monotone und Positive-Unit gesucht werden können und wie sie das Lösen des Extension Existence Problems unterstützen.

Im Rahmen dieser Arbeit wurde untersucht inwieweit sich Backdoors in der Praxis eignen, um die Suche nach Erweiterungen zu unterstützen. Dabei wurde für das Lösen von EXT der von Egly et al. [6] entwickelte NML[1]-Theorem-Prover QUIP verwendet. Dieser wandelt eine eingegebene Default-Theorie in eine quantifizierte boolsche Formel (QBF) um, um diese mit den QBF-Reasoner boole zu lösen. Auf QUIP aufbauend wurde das Programm Backdoor-QUIP entwickelt, welches für eingegebene Default-Theorien nach Backdoors sucht, um dann die durch Belegung der Backdoorvariablen entstandenen Default-Theorien mit QUIP auf Erweiterungen zu testen.

Mit diesen Programmen wurde getestet, wie schnell Backdoors verschiedener Größe gefunden werden können und ob, beziehungsweise inwieweit die gefundenen Backdoors das Prüfen auf Erweiterungen beschleunigen. Dabei wurden Backdoors für die vier oben genannten Formelklassen getestet, und es wurede verglichen wie gut Backdoors dieser Klassen für EXT geeignet sind.

[1] Nicht Monotone Logik

Kapitel 2

Grundlagen

In diesem Kapitel wird zunächst die Polynomialzeithierarchie vorgestellt. Anschließend wird gezeigt, inwiefern sich nichtmonotone Logik von der klassischen Logik unterscheidet. Darauf folgt eine Einführung in die Default-Logik, in der unter anderem dargestellt wird, wie Defaults funktionieren, wie man Erweiterungen berechnet und was das Extension Existence Problem ist. Abschließend wird erläutert wie man in der parametrisierten Komplexität mithilfe von Parametern die Komplexität schwer berechenbarer Probleme senken kann. Dabei wird gezeigt was Backdoors sind und wie man das SAT-Problem mit Backdoors als Parameter effizient lösen kann.

2.1 Polynomialzeithierarchie

In der Komplexitätstheorie werden algorithmisch behandelbare Probleme anhand des zu ihrer Lösung benötigten Ressourcenverbrauchs klassifiziert. Dabei wird der Ressourcenverbrauch meistens in Rechenzeit oder Speicherplatzbedarf gemessen. Die Klasse P enthält genau die Probleme, die von einer deterministischen Turingmaschine mit von der Eingabegröße abhängigem polynomiellen Zeitverbrauch gelöst werden können. Mit einer nichtdeterministischen, polynomiell zeitbeschränkten Turingmaschine können Probleme der Klasse NP gelöst werden. Erweitert man die Turingmaschinen um Orakel, so lassen sich mit ihnen weitere Komplexitätsklassen beschreiben. Eine Orakel-Turingmaschine funktioniert zunächst wie eine normale Turingmaschine, kann aber zusätzlich Fragen einer bestimmten Sprache an das Orakel stellen, welche von diesem in einem Berechnungsschritt beantwortet werden. Als Orakel werden häufig andere Komplexitätsklassen verwendet. Eine deterministische, polynomiell zeitbeschränkte Turingmaschine mit einem NP Orakel kann also, zusätzlich zu den Berechnungen in P, auch NP Proble-

me per Frage an das Orakel in einem Schritt lösen. Die von dieser Orakel-Turingmaschine beschriebene Klasse heißt P^{NP}.

Die Polynomialzeithierarchie ist eine Hierarchie von Komplexitätsklassen zwischen P und PSPACE, die mit Orakel-Turingmaschinen definiert werden können. Sie besteht aus den drei Gruppen von Klassen Δ_i^{P}, Σ_i^{P} und Π_i^{P} die wie folgt definiert sind:

$$\begin{aligned}
\Delta_0^{\mathrm{P}} &= \Sigma_0^{\mathrm{P}} = \Pi_0^{\mathrm{P}} = \mathrm{P} \\
\Delta_{i+1}^{\mathrm{P}} &= \mathrm{P}^{\Sigma_i^{\mathrm{P}}} \\
\Sigma_{i+1}^{\mathrm{P}} &= \mathrm{NP}^{\Sigma_i^{\mathrm{P}}} \\
\Pi_{i+1}^{\mathrm{P}} &= \text{co-NP}^{\Pi_i^{\mathrm{P}}}
\end{aligned}$$

Jede Ebene der Hierarchie verwendet die Klassen der vorherigen Ebene als Orakel. Dabei sind die Σ_i^{P}- und Π_i^{P}-Orakel gleichwertig, da sie sich durch Negation ineinander überführen lassen.

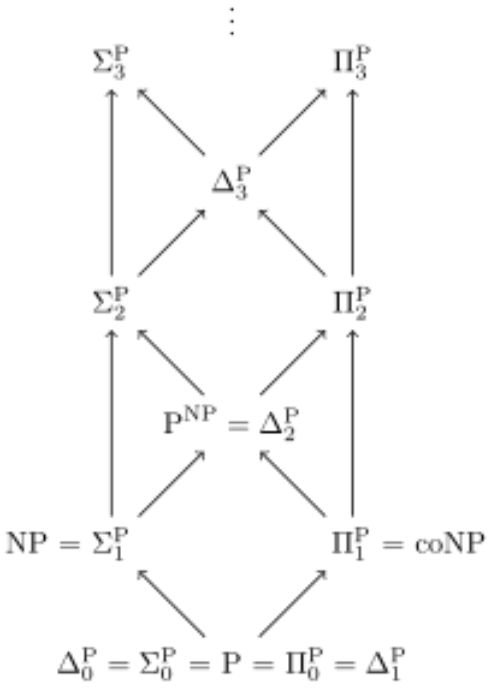

Abbildung 2.1: Die Polynomialzeit Hierarchie

In dieser Arbeit werden Probleme der ersten beiden Ebenen der Polyno-

mialzeithierarchie behandelt.

2.2 Nichtmonotone Logik

Für die meisten logischen Systemen gilt, dass ein einmal gültiges Argument auch nach dem Hinzugewinnen weiterer Informationen immer gültig bleibt; diese Eigenschaft nennt man Monotonie. Die nichtmonotonen Logiken geben diese Eigenschaft auf, um Schlussfolgerungen ziehen zu können, die mit klassischen Logiken nicht möglich gewesen wären. Beispiele für nichtmonotone Logiken sind Abduktion [7], autoepistemische Logik [11] und Default-Logik [13].

In der Abduktion sucht man für die gegebenen Fakten nach der wahrscheinlichsten Erklärung, welche aber nicht zwangsweise korrekt sein muss. Im Fall, dass eine andere (korrekte) Erklärung gefunden wird, muss die alte Annahme zurückgezogen werden.

Die autoepistemische Logik ermöglicht das Argumentieren über das Vorhandensein, beziehungsweise Fehlen von Wissen. Kommen neue Fakten hinzu, werden Aussagen über das Fehlen von Wissen über diese Fakten ungültig.

Wir betrachten in dieser Arbeit nur die Default-Logik.

2.3 Default-Logik

In der von Reiter [13] eingeführten Default-Logik ist es möglich, Annahmen über Aussagen zu treffen, deren Wahrheitswert unbekannt ist. Dazu werden zunächst Regeln definiert, die besagen, unter welchen Bedingungen, welche Annahmen getroffen werden dürfen. Diese Regeln nennt man *Default-Regeln* oder kurz *Defaults*.

Defaults bestehen aus drei Teilen: einer *Vorbedingung*, einer Menge von *Rechtfertigungen* und einer *Schlussfolgerung*. Die Vorbedingung gibt Fakten an, die erfüllt sein müssen, damit man den Default anwenden darf. Bei den Rechtfertigungen handelt es sich um Formeln, die erfüllt werden müssen, um den Default anzuwenden. Anders als bei den Vorbedingungen reicht es allerdings, wenn es konsistent ist anzunehmen, dass die Fakten gelten. Das heißt, die Annahme, dass die Rechtfertigungen gelten, darf nicht zu einem Widerspruch mit dem bereits bekannten oder später hinzugewonnenen Wissen füh-

ren. Sind Vorbedingung und Rechtfertigungen erfüllt, dürfen wir annehmen, dass die in der Schlussfolgerung definierten Fakten gelten. Die Schreibweise für einen Default ist:

$$\frac{\text{Vorbedingung} : \text{Rechtfertigung}_1, \ldots, \text{Rechtfertigung}_n}{\text{Schlussfolgerung}}$$

Defaults können wir gemeinsam mit Aussagen über unsere Welt zu Default-Theorien zusammenfassen.

Definition 2.1 *Eine Default-Theorie ist ein Paar $T = \langle W, D \rangle$, wobei W eine Menge logischer Formeln und D eine Menge von Default-Regeln ist.*

Die Formeln in W, sowie alle Formeln, die in den Defaults verwendet werden, können aus einer beliebigen formalen Logik stammen. Im Rahmen dieser Arbeit betrachten wir allerdings ausschließlich Default-Theorien mit aussagenlogischen Formeln in konjunktiver Normalform(KNF).

Beispiel 2.1 *In einem beliebten Beispiel für Default-Logik geht es um Vögel und die Annahme, dass die meisten Vögel fliegen können. Es gibt aber auch Vogelarten, die nicht fliegen können, wie Pinguine oder Straußen. Würde man versuchen, den Sachverhalt in Aussagenlogik zu formulieren, käme man auf die Formel*

$$(Vogel \wedge \neg Pinguin \wedge \neg Strauß \wedge \ldots) \Rightarrow Fliegt$$

Diese Formel hat den Nachteil, dass wir nicht von Vogel auf Fliegt schließen können, solange wir nicht sicher ausschließen können, dass der Vogel zu einer der nicht fliegenden Arten gehört. Formulieren wir das gleiche Problem in Default-Logik, kommen wir auf die Formelmenge

$$\{Vogel,\ (Pinguin \vee Strauß \vee \ldots) \Rightarrow \neg Fliegt\}$$

und die Default-Regel

$$\left\{ \frac{Vogel : Fliegt}{Fliegt} \right\}$$

In diesem Fall können wir mithilfe der Default-Regel auch ohne das Wissen über die Vogelart, beziehungsweise gerade durch das fehlende Wissen, von

Vogel auf Fliegt schließen. Würde man aber zum Beispiel die Formelmenge um Pinguin erweitern, dürfte man den Default nicht mehr verwenden, da aus Pinguin $\Rightarrow \neg$Fliegt folgt und somit die Annahme, dass Fliegt gilt, nicht konsistent wäre. Daran kann man gut erkennen, dass Default-Logik eine nichtmonotone Logik ist, da das Hinzugewinnen von Wissen (Pinguin) die Menge des folgerbaren Wissens reduziert (Fliegt).

Stimmt in einem Default die Rechtfertigung mit der Schlussfolgerung überein, wie es im Vogelbeispiel der Fall ist, so nennen wir den Default *normal*. Ist ein Default normal und hat darüber hinaus keine (oder eine tautologische) Voraussetzung, ist er *supernormal*. Ein Default dessen Schlussfolgerung komplett in allen Rechtfertigungen enthalten ist, heißt *seminormal*.

Beispiel 2.2

$$\left\{\frac{x : y}{y}\right\} \qquad \textit{normal}$$

$$\left\{\frac{\top : y}{y}\right\} \qquad \textit{supernormal}$$

$$\left\{\frac{x : y \wedge z}{y}\right\} \qquad \textit{seminormal}$$

Definition 2.2 *Sei $T = \langle W, D\rangle$ eine Default-Theorie, E eine Formelmenge, dann definiere $E_0 := W$ und*

$$E_{i+1} := Th(E_i) \cup \left\{\gamma \,\middle|\, \frac{\alpha : \beta}{\gamma} \in D, \alpha \in E_i und \neg\beta \notin E\right\}.^{1}$$

E ist eine stabile Erweiterung *von T genau dann, wenn $E = \bigcup_{i\in\mathbb{N}} E_i$ gilt. Die Menge*

$$G = \left\{\frac{\alpha : \beta}{\gamma} \in D \,\middle|\, \alpha \in E_i \wedge \neg\beta \notin E\right\}$$

nennen wir generierende Defaults. *Ist E eine Erweiterung von $T = \langle W, D\rangle$, dann ist $E = Th(W \cup \{concl(\delta) \mid \delta \in G\})$.*[2]

Mit anderen Worten bedeutet das, dass, wenn man eine Erweiterung für eine Default-Theorie $T = \langle W, D\rangle$ sucht, man zunächst mit der Wissensbasis ($W = E_0$) startet. Danach wendet man einen Default aus D an und fügt die Schlussfolgerung zur Wissensbasis hinzu (E_{i+1}). Daraufhin dürfen weitere

[1] $Th(\varphi) := \{\psi \in KNF \mid \varphi \models \psi\}$ (deduktiver Abschluss von φ)

[2] $concl(\delta) :=$ Schlussfolgerung des Defaults δ

Defaults angewendet werden, unter anderem auch solche, bei denen die Vorbedingung erst durch das neu hinzugewonnene Wissen erfüllt wurde. Wendet man nun Defaults an, bis keine weiteren mehr angewendet werden können, da die Voraussetzungen nicht erfüllt sind oder die Defaults bereits angewendet wurden, dann ist die so entstandene Theorie E eine Erweiterung von T und die dabei angewendeten Defaults bilden die Menge der generierenden Defaults G.

Da die Schlussfolgerung eines Defaults die Rechtfertigung eines anderen Defaults verhindern kann, kann die Reihenfolge in der Defaults angewendet werden eine Rolle spielen. Eine Default-Theorie kann also verschiedene Erweiterungen haben, welche durch unterschiedliche Anwendungsreihenfolgen der Defaults zustande kommen.

Beispiel 2.3 *Ein Beispiel für eine Default-Theorie mit mehreren Erweiterungen ist die Nixon-Raute. Die Nixon-Raute ist ein Szenario, in dem die folgenden vier Aussagen gelten:*

1. *Nixon ist ein Republikaner.*
2. *Nixon ist ein Quäker.*
3. *Republikaner sind normalerweise keine Pazifisten.*
4. *Quäker sind normalerweise Pazifisten.*

Daraus erhalten wir die Default-Theorie

$$T = \left\langle \{Republikaner, Quäker\}, \left\{ \frac{Republikaner : \neg Pazifist}{\neg Pazifist}, \frac{Quäker : Pazifist}{Pazifist} \right\} \right\rangle$$

Wenden wir nun den ersten Default an, können wir schließen, dass Nixon wohl kein Pazifist ist, da er Republikaner ist. Durch dieses neue Wissen ist es nicht mehr möglich, den zweiten Default anzuwenden. Obwohl Nixon auch ein Quäker ist, können wir nicht mehr annehmen, dass er Pazifist ist, da wir bereits wissen, dass er kein Pazifist ist. Da kein weiterer Default mehr angewendet werden kann, ist die Default-Theorie, in der Nixon kein Pazifist ist, eine Erweiterung unserer Ausgangstheorie.

Wenn wir aber nicht mit dem ersten Default beginnen, sondern mit dem zweiten, kommen wir auf eine Theorie in der Nixon ein Pazifist ist. Auch in diesem Fall können wir keinen weiteren Default mehr anwenden, also ist auch diese Theorie eine Erweiterung der Ausgangstheorie.

Bisher haben wir Default-Theorien gesehen, die eine (Beispiel 2.1) oder sogar mehrere (Beispiel 2.3) Erweiterungen haben. Es ist aber auch möglich, dass eine Theorie gar keine Erweiterung hat. Aber wie sieht so eine Theorie aus? Wir wissen, dass wir eine Erweiterung gefunden haben, sobald es keine Defaults mehr gibt, die noch angewendet werden können. Eine Theorie ohne Erweiterung müsste also immer in einem Zustand bleiben, in dem mindestens ein Default angewendet werden kann. Da wir aber nur eine endliche Menge an Defaults haben und jeden davon maximal einmal anwenden können, ist es also nicht möglich, in einer Dauerschleife von Default-Anwendungen zu landen. Es muss also einen Default geben, der theoretisch anwendbar wäre, aber dennoch nicht angewendet werden kann. Das ist genau dann der Fall, wenn die (theoretisch mögliche) Anwendung des Defaults dazu führen würde, dass die Default-Theorie inkonsistent wird.

Beispiel 2.4 *Das wohl einfachste Beispiel für eine Default-Theorie ohne Erweiterung ist die Theorie:*

$$T = \left\langle \{\}, \left\{ \frac{: x}{\neg x} \right\} \right\rangle$$

Die Anwendung des Defaults ist möglich, allerdings nur unter der Annahme das x gilt. Da die Schlussfolgerung aber besagt, dass $\neg x$ gilt, widerspricht die Schlussfolgerung der Annahme, die wir treffen mussten, um den Default anzuwenden. Die resultierende Default-Theorie ist also inkonsistent und damit keine Erweiterung.

Leider ist ein solcher Widerspruch nicht immer so einfach zu finden wie in dem Beispiel. Auch wenn alle Defaults für sich genommen konsistent sind, kann die Anwendung mehrerer Defaults nacheinander zu Widersprüchen führen.

Beispiel 2.5 *In der Theorie*

$$T = \left\langle \{\}, \left\{ \frac{: x}{x}, \frac{x : y}{y}, \frac{y : z}{\neg x} \right\} \right\rangle$$

ist es problemlos möglich, den ersten und danach den zweiten Default anzuwenden. Erst der dritte Default führt zur Inkonsistenz mit der Rechtfertigung und Schlussfolgerung des ersten Defaults.

Die Schlussfolgerung eines Defaults kann nicht nur inkonsistent zu anderen Defaults sein, sondern auch zur bisherigen Wissensbasis, wie das folgende Beispiel zeigt.

Beispiel 2.6

$$T = \left\langle \{y\}, \left\{ \frac{: x}{\neg y} \right\} \right\rangle$$

Durch Anwendung des Defaults würde gleichzeitig y und $\neg y$ gelten, was ein Widerspruch ist.

Wir haben nun einige Beispiele gesehen in denen Defaults zu Inkonsistenzen führen und dadurch verhindern, dass die zugehörigen Theorien eine Erweiterung haben. Man sollte jedoch beachten, dass eine Default-Theorie auch trotz eines solchen problematischen Defaults immer noch eine Erweiterung haben kann, wenn, durch Anwendung anderer Defaults, ein Zustand erreicht werden kann, in dem der problematische Default nicht mehr angewendet werden darf.

Beispiel 2.7 *Betrachten wir ein weiteres Mal das vorherige Beispiel und erweitern es um einen zweiten Default.*

$$T = \left\langle \{y\}, \left\{ \frac{: x}{\neg y}, \frac{: \neg x}{\neg x} \right\} \right\rangle$$

Auch hier führt die Anwendung des ersten Defaults zum gleichen Widerspruch wie vorher, aber wir haben nun auch die Möglichkeit den zweiten Default zuerst anzuwenden. Dadurch wird $\neg x$ zur Wissensbasis hinzugefügt, was wiederum verhindert, dass der erste Default angewendet werden kann. Wir haben also einen Zustand erreicht, in dem kein weiterer Default angewendet werden kann und damit eine Erweiterung gefunden.

Eine weitere Beobachtung ist, dass nur Defaults, in denen sich Rechtfertigung und Schlussfolgerung unterscheiden, zu Inkonsistenzen führen können. Daraus folgt, dass eine Default-Theorie die normal ist, also bei der bei allen Defaults die Rechtfertigung und Schlussfolgerung übereinstimmen, immer mindestens eine Erweiterung hat.

Das Extension Existence Problem (EXT) ist das Entscheidungsproblem für die Frage, ob eine gegebene Default-Theorie mindestens eine Erweiterung hat. Die Komplexität von EXT ist abhängig von der Klasse der gegebenen Formeln. Das Extension Existence Problem für aussagenlogische Formeln in konjunktiver Normalform ist Σ_2^{P}-vollständig [10]. Nimmt man statt beliebiger KNF-Formeln, nur Formeln in 2-KNF, ist das Problem NP-vollständig [15].

Die hohe Komplexität kommt dadurch zustande, dass zum einen die generierenden Defaults und zum anderen die Reihenfolge gefunden werden muss,

in der die Defaults angewendet werden müssen, um zur Erweiterung zu führen.

2.4 Parametrisierte Komplexität

Die parametrisierte Komplexität ist ein von Downey und Fellows [5, 4] eingeführter Ansatz, mit dem man versucht, die Komplexität schwerer Entscheidungsprobleme zu senken, indem man einen bestimmten Parameter des Problems fixiert. Die grundlegende Vorgehensweise dabei ist, zunächst einen Parameter zu suchen, der mit der Komplexität des Problems zusammenhängt und diesen zu fixieren. Anschließend wird die Komplexität des Problems in Abhängigkeit vom Parameter untersucht. Hat man einen geeigneten Parameter gewählt, sinkt die Komplexität des Problems. In der Praxis findet man für einige Probleme Parameter mit kleiner oder konstanter Größe, mit denen das Lösen dieser Probleme vereinfacht werden kann.

Beispiel 2.8 *Nehmen wir das NP-vollständige Erfüllbarkeitsproblem der Aussagenlogik (SAT) und wählen die Anzahl der Variablen einer gegebenen Formel als Parameter. Sei n die Größe der Formel und k die Anzahl der Variablen, dann lässt sich die Erfüllbarkeit in $O\left(2^k \cdot n\right)$ berechnen. Solange wir also unseren Parameter k festsetzen, haben wir eine lineare Laufzeit in n. Dieser Parameter bringt uns in der Praxis aber nur wenig, da wir uns durch das Festsetzen des Parameters auf Formeln einer bestimmten Größe einschränken.*

Definition 2.3 *Ein parametrisiertes Problem eine Sprache $L \subseteq \Sigma^* \times \mathbb{N}$, wobei Σ ein endliches Alphabet und $\mathbb{N}$ der Parameter ist. Sei C eine klassische Komplexitätsklasse, dann ist para-C die Klasse aller parametrisierter Probleme $L \subseteq \Sigma^* \times \mathbb{N}$, für die es ein Alphabet Σ', eine berechenbare Funktion $f\colon \mathbb{N} \to \Sigma'^*$ und ein Problem $L' \subseteq \Sigma^* \times \Sigma'^*$ gibt, sodass $L' \in C$ und für alle Instanzen $(x, k) \in \Sigma^* \times \mathbb{N}$ gilt, dass $(x, k) \in L \Leftrightarrow (x, f(k)) \in L'$.*

Mit anderen Worten könnte man para-C als die Klassen beschreiben, welche die Probleme enthalten, die nach einer nur vom Parameter abhängenden Vorberechnung in der Klasse C sind.

Die der Klasse P entsprechende Klasse nennen wir nicht para-P, sondern FPT. Die Probleme in FPT werden fixed-parameter tractable und ihre Laufzeit $f(k) \cdot |x|^{O(1)}$ fpt-Zeit genannt.

2.5 Backdoors in Aussagenlogik

In Zusammenhang mit dem SAT-Problem hat man bereits eine Menge verschiedener Parameter untersucht. Ein viel untersuchter Parameter sind die von Williams, Gomes und Selman [16] eingeführten Backdoors. Ein Backdoor ist eine Teilmenge der Variablen einer Formel. Die Idee von Backdoors ist, dass man statt aller Variablen zunächst nur die Variablen im Backdoor belegt. Dadurch wird die verbleibende Formel so weit vereinfacht, dass sich die Erfüllbarkeit in P berechnen lässt. Es gibt verschiedene Arten von Backdoors, aber im Rahmen dieser Arbeit konzentrieren wir uns auf starke Backdoors.

Definition 2.4 *Sei $\mathcal{F}$ eine Klasse von Formeln, φ eine Formel und B eine Teilmenge der Variablen von φ, dann ist B ein starkes $\mathcal{F}$-Backdoor, wenn für alle Belegungen $B \to \{0,1\}$ die Formel φ zur Klasse $\mathcal{F}$ gehört.*

Beispiel 2.9 *Nehmen wir die Formel*

$$\begin{aligned}\varphi := (x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_3} \vee x_4) \wedge (x_2 \vee x_4 \vee \overline{x_5}) \\ \wedge(\overline{x_1} \vee x_3 \vee x_5) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4 \vee x_5)\end{aligned}$$

und betrachten die Formeln, die durch die Belegungen der Variablen x_3 und x_4 entstehen.

$$\begin{aligned}&\varphi_1 := (x_2 \vee \overline{x_5}) \wedge (\overline{x_1} \vee x_5) && \text{für } x_3 = 0,\ x_4 = 0 \\ &\varphi_2 := (x_1 \vee x_2) \wedge (\overline{x_1} \vee x_5) && \text{für } x_3 = 0,\ x_4 = 1 \\ &\varphi_3 := 0 \wedge (x_2 \vee \overline{x_5}) \wedge (x_1 \vee \overline{x_2} \vee x_5) = 0 && \text{für } x_3 = 1,\ x_4 = 0 \\ &\varphi_4 := (x_1 \vee x_2) && \text{für } x_3 = 1,\ x_4 = 1\end{aligned}$$

Wie man leicht sehen kann, sind die vereinfachten Formeln φ_1, φ_2 und φ_4 in 2-KNF. Die Formel φ_3 enthält eine Klausel mit drei Variablen, nach weiterem Vereinfachen der Formel bleibt aber nur noch 0 über. Damit ist auch φ_3 in 2-KNF. Da also alle vier vereinfachten Formeln in 2-KNF sind, können wir sagen, dass $B = (x_3, x_4)$ ein starkes 2-KNF Backdoor für die Formel φ ist.

Dass eine der vereinfachten Formeln unerfüllbar (sogar direkt falsch) ist, stört nicht, denn es reicht, dass eine der vereinfachten Formeln erfüllbar ist, um auf Erfüllbarkeit der gesamten Formel schließen zu können. Umgekehrt wissen wir, dass, wenn alle vereinfachten Formeln unerfüllbar sind, auch die Ursprungsformel unerfüllbar ist.

Wie wir gesehen haben, führt die Anwendung eines starken $\mathcal{F}$-Backdoors der Größe k zu 2^k vereinfachten Formeln, die wir in polynomieller Zeit auf Erfüllbarkeit prüfen können, falls SAT für Formeln der Klasse $\mathcal{F}$ in P liegt. Da die Anzahl der zu prüfenden Formeln exponentiell mit der Backdoorgröße ansteigt, suchen wir Backdoors mit minimaler Größe. Dabei hängt die Komplexität der Backdoorsuche hauptsächlich von $\mathcal{F}$ ab. Für viele $\mathcal{F}$ ist die Suche nach einem minimalen Backdoor NP-vollständig, lässt sich aber in fpt-Zeit berechnen, wenn man die Backdoorgröße als Parameter hat. Wie ein Backdoor berechnet wird, wird für einige $\mathcal{F}$ in einem späteren Kapitel gezeigt.

Wir können SAT also in fpt-Zeit berechnen, wenn wir die Größe eines minimalen Backdoors als Parameter haben, indem wir zunächst mit dem Parameter ein Backdoor suchen und anschließend, mithilfe des Backdoors, die Formel auf Erfüllbarkeit prüfen.

Kapitel 3

Backdoors in Default-Logik

Fichte, Meier und Schindler [9] erweiterten das Konzept der Backdoors, sodass es auch in der Default-Logik verwendet werden kann. Dieses Kapitel basiert auf deren Arbeit und beschreibt einige der dort eingeführten Definitionen und Ansätze. Dabei wird gezeigt wie Backdoors verwendet werden können, um das Extension Existece Problem zu lösen.

Das Kapitel beginnt mit einer Beschreibung der vier Formelklassen Krom, Horn, Monotone und Positive-Unit und listet die Komplexitätsklassen auf, in welchen EXT für diese Formelklassen jeweils liegt. Danach wird gezeigt, dass die Unterschiede von Aussagenlogik und Default-Logik dazu führen, dass Anpassungen an der Definition von Backdoors vorgenommen werden müssen, damit diese für Defaul-Logik und EXT verwendet werden können. Es folgen Definitionen für Erweiterte Literale, Redukte und schließlich die neue Definition für starke Backdoors in Default-Logik.

Anschließend beschäftigen wir uns mit der Suche nach Backdoors. Dabei wird die Vorgehensweise der Algorithmen erläutert, mit denen Backdoors für die vier genannten Formelklassen gesucht werden. Abschließend wird gezeigt welche Schritte durchgeführt werden müssen, um EXT mithilfe von Backdoors zu berechnen, und in welche (parametrisierte) Komplexitätsklassen EXT dadurch fällt.

3.1 Extension Existence Problem

Wie bereits im Kapitel 2.3 angesprochen wurde, geht es beim Extension Existence Problem (EXT) darum herauszufinden, ob eine Default-Theorie mindestens eine Erweiterung hat. Die Komplexität des Problems ist abhängig von

der Klasse der gegebenen Formeln. Gottlob [10] zeigte 1992 das EXT(KNF) Σ_2^{P}-vollständig ist. Schränkt man die Formeln weiter ein, so lässt sich die Komplexität senken. Im folgenden wollen wir vier Teilklassen von KNF betrachten.

- Krom-Formeln (auch 2-KNF-Formeln genannt) sind eine Einschränkung der KNF-Formeln bei denen in allen Klauseln maximal 2 Literale vorkommen dürfen.
- Bei Horn-Formeln dürfen Klauseln maximal ein positives Literal enthalten, aber beliebig viele negative Literale.
- Monotone-Formeln enthalten ausschließlich positive Literale.
- In Positive-Unit-Formeln enthalten alle Klauseln genau ein positives Literal und keine negaiven Literale.

Enthält eine Default-Theorie nur Krom-Formeln oder nur Horn-Formeln ist das entsprechende Extension Existence Problem NP-vollständig, wie Stillman [15] 1990 zeigte. EXT für Monotone-Formeln ist Δ_2^{P}-vollständig, was ebenfalls von Stillman gezeigt wurde. Am einfachsten wird EXT wenn nur Positive-Unit-Formeln verwendet werden. In dem Fall sinkt die Komplexität auf P, der Beweis dazu kam 2012 von Beyersdorff et al. [2].

3.2 Unterschiede zur Aussagenlogik

Im Kapitel 2.5 haben wir gesehen wie Backdoors (beziehungsweise die Größe des Backdoors) als Parameter genutzt werden können, um KNF-Formeln, durch Belegen der Backdoor-Variablen und anschließendes Vereinfachen der Formel, in eine Formel-Klasse überführen lässt, für die das SAT-Problem effizienter gelöst werden kann. Wie beim SAT-Problem, ist auch die Komplexität des EXT-Problem von der Klasse der verwendeten Formeln abhängig und so stellt sich die Frage, ob nicht auch beim EXT-Problem Backdoors genutzt werden können, um die Formeln zu vereinfachen.

Backdoors nach der bisherigen Definition für SAT können nicht direkt für das EXT-Problem verwendet werden. Aufgrund der Unterschiede von SAT- und EXT-Instanzen müssen zunächst einige Anpassungen an der Definition von Backdoors vorgenommen werden. Ein solcher Unterschied ist, dass wir bei SAT immer nur eine einzelne Aussagenlogische Formel haben, während bei EXT mit Default-Theorien gearbeitet wird, die aus einer Menge von Formeln bestehen. Bei EXT reicht es nicht, einzelne Formeln in die

Wunschklasse zu überführen, sondern es müssen alle Formeln, sowohl aus der Wissensbasis als auch aus Vorbedingung, Rechtfertigungen und Schlussfolgerung aller Default-Regeln, in die Klasse überführt werden. Das Backdoor muss also aus einer Menge von Variablen bestehen, sodass alle Belegungen dieser Variablen dazu führen, dass sämtliche Formeln in der Default-Theorie von der gewünschten Klasse sind.

Ein weiterer Unterschied betrifft die Belegungen der Variablen. Bei SAT sind alle Variablen zweiwertig, also entweder Wahr oder Falsch, die Default-Logik hingegen hat einen dreiwertigen Charakter, denn hier darf der Wahrheitswert einer Variable auch unbekannt sein. Der unbekannte Wahrheitswert einer Variable kann es ermöglichen Folgerungen zu ziehen, die nicht möglichen währen wenn der Wahrheitswert bekannt wäre.

Beispiel 3.1 *Betrachten wir die Default-Theorie* $T = \langle W, D\rangle$ *mit* $D = \left\{\frac{\top:X}{Y}, \frac{\top:\neg X}{Z}\right\}$. *Ist* $W = X$ *so enthalten wir eine Erweiterung* $E = \{X, Y\}$. $W = \neg X$ *hingegen führt zur Erweiterung* $E = \{X, Z\}$. *Ist* $W = \top$ *also* X *unbekannt, so enthalten wir die Erweiterung* $E = \{Y, Z\}$. *Wie man sieht führen alle drei Belegungen zu verschiedenen Erweiterungen. Insbesondere ist zu beachten, dass nur wenn* X *unbekannt ist beide Defaults angewendet werden können und damit mehr gefolgert werden kann, als wenn der Wahrheitswert von* X *bekannt wäre.*

Das bedeutet für die Backdoors, dass nicht nur die Belegung der Backdoorvariablen auf Wahr und Falsch, sondern auch auf Unbekannt dazu führen muss, dass sich die Formeln auf die gewünschte Formel-Klasse vereinfachen lassen.

Um formal die oben genannten Änderungen an der Definition von Backdoors vorzunehmen, führen wir zunächst *Erweiterte Literale* und *Redukte* ein.

Definition 3.1 *Ein Erweitertes Literal ist ein Literal oder eine neue Variable* x_ε[1]. *Weiterhin sei* $\sim \ell = x$, *wenn* $\ell = \neg x$ *und* $\sim \ell = \neg x$, *wenn* $\ell = x$ *ist. Sei* φ *eine Formel und* ℓ *ein erweitertes Literal, dann erhält man die Redukte* $\rho_\ell(\varphi)$ *folgendermaßen:*

1. *wenn* ℓ *ein Literal ist: entferne alle Klauseln von* φ, *die* ℓ *enthalten und alle Literale* $\sim \ell$ *aus den Klauseln von* φ.

2. *wenn* $\ell = x_\varepsilon$ *ist: entferne alle Literale* $\neg x, x$ *aus den Klauseln von* φ.

[1] entspricht Variable mit unbekanntem Wahrheitswert

Sei $\langle W, D\rangle$ eine Default-Theorie und ℓ ein erweitertes Literal, dann sei

$$\rho_\ell(W, D) := \left(W. \left\{ \frac{\rho_\ell(\alpha) : \rho_\ell(\beta)}{\rho_\ell(\gamma) \wedge y_i} \middle| \delta_i = \frac{\alpha : \beta}{\gamma} \in D \right\} \right),$$

wobei y_i eine neue Variable ist und $\rho_\ell(W) := \bigcup_{\omega \in W} \rho_\ell(\omega)$.

Durch das Redukt $\rho_\ell(W, D)$ erhalten wir also die Default-Theorie, welche durch Belegen einer Variable und anschließendes Vereinfachen entsteht. Das Hinzufügen der y_is in den Schlussfolgerungen hat keinen direkten Einfluss auf die Suche nach Erweiterungen; sie können daher in der Praxis auch weggelassen werden. In der Arbeit von Fichte [9] dienen die y_is einem Beweis, auf den hier allerdings nicht weiter eingegangen wird.

Definition 3.2 *Sei X eine Menge von Variablen, dann definieren wir die Menge der (dreiwertigen) Belegungen als*

$$\mathbb{T}(X) := \{\{a_1, \ldots, a_{|X|}\} | x \in X \text{ und } a_i \in \{x, \neg x, x_\varepsilon\}\}$$

Für $Y \in \mathbb{T}(X)$ ist das Redukt $\rho_Y(W, D)$ die Hintereinanderausführung aller $\rho_y(\cdot)$ für $y \in Y$ auf $\langle W, D\rangle$. Die Reihenfolge in der die Redukte $\rho_y(\cdot)$ angewendet werden spielt dabei keine Rolle.

Mit diesen Definitionen haben wir nun alles was wir brauchen, um Backdoors für Default-Logik zu definieren.

Definition 3.3 *Sei $\mathcal{F}$ eine Klasse von Formeln, $\langle W, D\rangle$ eine KNF-Default-Theorie und $B \subseteq Vars(W, D)$ eine Teilmenge der Variablen der Default-Theorie, dann ist B ein starkes $\mathcal{F}$-Backdoor, wenn für alle Belegungen $Y \in \mathbb{T}(B)$ das Redukt $\rho_Y(W, D)$ eine $\mathcal{F}$-Default-Theorie ist.*

3.3 Backdoor Suche

Um eine Default-Theorie mithilfe von Backdoors zu vereinfachen, brauchen wir natürlich zunächst ein passendes Backdoor. Die Suche nach einem starken $\mathcal{F}$-Backdoor ist dabei abhängig von der Klasse $\mathcal{F}$. Die Algorithmen, die zur Suche der Backdoors in Default-Logik verwendet werden, nutzen die selbe Vorgehensweise, die man auch für die Backdoorsuche in Aussagenlogik verwendet. Im Folgenden wird die Vorgehensweise für die Klassen Monotone, Horn, Positive-Unit und Krom vorgestellt.

3.3.1 Monotone

In Monotone-Formeln kommen alle Literale ausschließlich positiv vor. Das kleinste starke Monotone-Backdoor besteht also genau aus den Literalen, die negativ vorkommen. Es reicht also ein Durchlauf durch sämtliche Formeln, bei dem alle gefundenen negativen Literale zum Backdoor hinzugefügt werden. Damit ist die Suche nach starken Monotone-Backdoors in linearer Zeit möglich, womit sie von den betrachteten Klassen die am einfachsten zu findenden Backdoors sind.

3.3.2 Horn

Horn-Formeln bestehen aus Klauseln die maximal ein positives Literal enthalten. Von allen Klauseln aus unserer Default-Theorie die mehr als ein positives Literal enthalten, müssen alle bis auf eines dieser positiven Literale zum Backdoor hinzugefügt werden. Je nachdem welche Literale ausgewählt werden, kann die Größe des resultierenden Backdoors variieren. Um sicher zu stellen das wir ein kleinstes Horn-Backdoor finden, erstellen wir einen Graphen $G = (V, E)$ mit $V = Vars(W, D)$. Anschließend durchlaufen wir alle Klauseln der Default-Thoerie und wenn eine Klausel mehrere positiven Literale enthält, fügen wir für alle Paare dieser positiven Literale eine Kante zwischen den entsprechenden Knoten zu E hinzu. Danach suchen wir in dem Graphen ein *Vertex Cover*, also eine kleinste Teilmenge der Knoten, so dass für alle Kanten mindestens einer der Endpunkte in der Teilmenge enthalten ist. Diese Teilmenge ist das gesuchte Backdoor.

Die Berechnung eines Vertex Covers ist normalerweise ein NP-vollständiges Problem, haben wir aber die Größe des Backdoors k (und damit auch des Vertex Covers) als Parameter, so können wir die Suche einschränken, wodurch es möglich wird, den Vertex Cover, so vorhanden, in $O(1,2738^k + kn)$ [3] zu berechnen. Daraus folgt, dass sich starke Horn-Backdoors in fpt-Zeit berechnen lassen.

3.3.3 Positive-Unit

Positive-Unit-Formeln bestehen aus Klauseln, die aus genau einem positiven Literal bestehen. Die Suche eines starken Positive-Unit-Backdoors besteht aus zwei Schritten. Im ersten Schritt durchlaufen wir wie bei Monotone-Backdoors alle Formeln und fügen gefundene negative Literale zum Backdoor hinzu. Daraufhin vereinfachen wir die Formeln indem wir alle positiven und negativen Vorkommen der Literale, die wir zum Backdoor hinzugefügt

haben, aus den Klauseln entfernen. Nun bestehen die Klauseln nur noch aus positiven Literalen, wir müssen also nur noch dafür sorgen, dass alle Klauseln auf Größe eins reduziert werden. Um dies zu erreichen, führen wir im zweiten Schritt die gleiche Graphen Konstruktion durch, die wir auch für Horn-Backdoors benutzt haben. Auch hier berechnen wir ein Vertex Cover auf dem Graphen und fügen die Variablen des Vertex Covers zu dem Backdoor aus Schritt eins hinzu. Der erste Schritt läuft in linearer Zeit und der zweite kann, wie wir gesehen haben, in fpt-Zeit durchgeführt werden. Insgesamt benötigen wir also fpt-Zeit für die Suche nach Positive-Unit-Backdoors.

3.3.4 Krom

In Krom-Formeln dürfen Klauseln aus maximal zwei Literalen bestehen. In einem starken Krom-Backdoor sind also alle bis auf zwei Literale jeder Klausel enthalten. Wie bei Horn-Backdoors ist die Größe des Backdoors von der Wahl der Literale abhängig. Auch die Suche nach einem kleinsten Krom-Backdoor folgt einem ähnlichen Ansatz wie bei Horn-Backdoors. Zunächst erstellen wir einen Hypergraphen $H = (X, E)$ mit $X = Vars(W, D)$. Dann durchlaufen wir alle Klauseln der Default-Theorie. Falls eine Klausel drei oder mehr Literale enthält, bildet man Tripel aus allen Kombinationen dieser Literale. Für jedes Literal-Tripel fügt man eine Hyperkante zwischen den entsprechenden Knoten zu E hinzu. In dem so erzeugten Hypergraphen suchen wir ein *3-Hitting-Set* (eine generalisierte Form vom Vertex Cover für Hypergraphen), also eine kleinste Teilmenge der Knoten, so dass mindestens ein Knoten jeder Hyperkante in der Teilmenge enthalten ist. Diese Teilmenge ist gleichzeitig ein Krom-Backdoor.

Wie beim Vertex Cover ist auch die Berechnung vom Hitting-Set ein NP-vollständiges Problem, welches wir mithilfe der Backdoorgröße k als Parameter vereinfachen können. Durch den Parameter lässt sich ein 3-Hitting-Set in $O(2,179^k + n^3)$ [8] finden. Damit ist die Suche eines kleinsten starken Krom-Backdoors in fpt-Zeit möglich.

3.4 EXT mit Backdoors

Jetzt wo wir gesehen haben wie man Backdoors berechnet bleibt nur noch die Frage, inwieweit die Nutzung von Backdoors die Berechnung des Extension Existence Problems erleichtern kann. Wie bereits im Kapitel 3.1 erwähnt, ist die normale Berechnung von EXT(KNF) ohne Backdoors Σ_2^{P}-vollständig. Bei dem Ansatz mit Backdoors berechnen wir zunächst ein starkes $\mathcal{F}$-Backdoor

D in fpt-Zeit (falls $\mathcal{F}$ = Monotone sogar in P), sofern die Backdoorgröße k als Parameter gegeben ist. Anschließend müssen wir für die $3^k = \mathbb{T}(B)$ Belegungen das Redukt $\rho_Y(W, D)$ erzeugen. Dadurch erhalten wir $\mathcal{F}$-Default-Theorien für die wir EXT($\mathcal{F}$) berechnen.

Für $\mathcal{F}$ = Monotone haben wir also eine schnelle Backdoorsuche in P und anschließend 3^k Aufrufe von EXT(Monotone) in Δ_2^{P}. Insgesamt ist EXT(KNF $\rightarrow$ Monotone) in para-Δ_2^P. Wenn $\mathcal{F}$ Horn oder Krom ist, benötigen wir fpt-Zeit für die Suche und die 3^k Aufrufe von EXT(Horn) beziehungsweise EXT(Krom) sind in NP. Sowohl EXT(KNF $\rightarrow$ Horn) als auch EXT(KNF $\rightarrow$ Krom) liegen damit in para-NP. Auch bei $\mathcal{F}$ = Positive-Unit benötigt die Backdoorsuche fpt-Zeit und da EXT(Positive-Unit) in P liegt, lassen sich die 3^k Aufrufe davon in fpt-Zeit berechnen, womit auch EXT(KNF $\rightarrow$ Positive-Unit) in FPT liegt.

Nun könnte man denken, dass sich die Nutzung von Positive-Unit-Backdoors am meisten lohnt, da dadurch die Laufzeit nur in FPT ist. Man sollte allerdings bedenken, dass die Laufzeit exponentiell von der Backdoorgröße abhängt, und es ist zu erwarten, dass Positive-Unit-Backdoors in der Paxis meist größer sind als die anderen Backdoors.

Weiterhin bleibt unklar, ob sich die Nutzung von Backdoors in der Praxis wirklich lohnt, auch wenn die Komplexität der EXT Aufrufe sinkt. Im Fall von Krom-, Horn- und Monotone-Backdoors bleibt die Komplexität in NP oder höher, wodurch es weiterhin keine (bekannten) Algorithmen gibt um EXT effizient zu lösen. Dafür muss man abhängig von der Backdoorgröße exponentiell viele Aufrufe von EXT machen. Bei Positive-Unit-Backdoors lässt sich EXT effizient lösen, dafür hat man dort, wie bereits erwähnt, meist große Backdoors, wodurch die exponentielle Anzahl der Aufrufe von EXT umso stärker ins Gewicht fällt.

Kapitel 4

Programme

Im Rahmen dieser Arbeit wurde das Program Backdoor-QUIP [14] entwickelt, welches auf dem NML-Theorem-Prover QUIP basiert. Backdoor-QUIP ermöglicht es, für gegebene Default-Theorien Backdoors zu suchen, und dann für alle Belegungen des Backdoors die Redukte zu erzeugen, bevor mit Quip nach Erweiterungen gesucht wird.

Im Folgenden werden zunächst QUIP und das zugrunde liegende Framework beschrieben. Anschließend wird auf Backdoor-QUIP eingegangen. Dabei wird zum einen der Aufbau des Programms erläutert und zum anderen ein Überblick über mögliche Einstellungen sowie die Form der Eingabe (Problembeschreibung) gegeben. Danach folgt eine kurze Beschreibung eines Generators für Default-Theorien, mit dem die Testfälle für die in Kapitel 5 durchgeführten Tests generiert wurden.

4.1 QUIP

Egly et al. [6] entwickelten ein Framework mit dem verschiedene Probleme der Nichtmonotonen Logiken, deren Komplexität in der Polynomialzeithierarchie liegen, gelöst werden können. Ein viel verwendeter Ansatz, NP Probleme zu lösen, ist es, diese zunächst auf SAT zu reduzieren und dann mit hoch optimierten SAT-Solvern zu lösen. Demselben Ansatz folgend, reduziert Eglys Framework Probleme der Polynomialzeithierarchie auf PSPACE, indem sie in quantifizierte boolsche Formeln (QBF) umgewandelt werden, welche anschließend mit einem QBF-Reasoner gelöst werden. Vorteil dieses Ansatzes ist es, dass man für alle behandelten Probleme ein einheitliches Lösungsverfahren hat und dabei auf bereits optimierte QBF-Reasoner zurückgegriffen werden kann.

Die prototypische Umsetzung des Frameworks QUIP besteht aus drei Teilen: QUIP_filter, boole und QUIP_interpreter. Als erstes liest QUIP_filter die eingegebene Problembeschreibung und erzeugt daraus QBFs. Im Fall von EXT besteht die Problembeschreibung beispielsweise aus Definitionen von Formeln, Defaults und Default-Theorien. Die erzeugten QBFs sind dabei nicht als äquivalente Darstellung der eingegebenen Default-Theorien zu verstehen, sondern vielmehr als Repräsentation der Frage, ob diese Theorien eine Erweiterung haben. QUIP bietet für Default-Theorien zwei verschiedene Varianten für die Umwandlung in QBF an. Die erste Variante modelliert das Raten der Generierenden Defaults und deren Anwendungsreihenfolge, während die Zweite auf der Charakterisierung von Erweiterungen als *full sets* (nach Niemelä [12]) basiert. Tests haben gezeigt, dass die erste Variante die besseren Ergebnisse erzielt.

Die erzeugten QBFs werden an boole übergeben. boole (BDDlib [1]) ist ein öffentlich verfügbarer QBF-Reasoner, der auf binären Entscheidungsdiagrammen basiert. Ein Vorteil von boole ist es, dass damit beliebige QBFs gelöst werden können. QUIP erzeugt aus den verschiedenen Problemen häufig große und komplexe QBFs, weshalb es praktisch ist, dass diese nicht zusätzlich noch in eine bestimmte Form umgewandelt werden müssen, um sie zu lösen. Die Ausgabe von boole besteht aus Formeln in disjunktiver Normalform (DNF), die an QUIP_interpreter übergeben werden. QUIP_interpreter verwendet das von QUIP_filter verwendete Mapping von Problem auf QBF um die DNF-Formeln zu interpretieren und in eine dem Problem entsprechenden Ausgabe umzuwandeln. Für EXT besteht die Ausgabe aus den möglichen Erweiterungen sofern es welche gibt, und ansonsten aus der Aussage, dass es keine gibt. Alle Teile von QUIP sind in C geschrieben.

In einem Vergleich zwischen QUIP und einigen etablierten NML-Theorem-Provern konnte QUIP mit den spezialisierten Programmen, welche für ihre Aufgabe gut optimiert wurden, nicht mithalten, was auch zu erwarten war. Gegen andere Programme, die wie QUIP nur zum Testen ihres entsprechenden Ansatzes entwickelt wurden, konnte QUIP allerdings sich durchaus durchsetzen.

4.2 Backdoor-QUIP

Backdoor-Quip ist in Java geschrieben und besteht aus den drei Klassen QUIP_Backdoor, Parser und BackdoorDetect. QUIP_Backdoor enthält die

main-Methode und ist für die Ein- sowie Ausgabe zuständig. Der Parser verarbeitet die Eingabe und ruft für gelesene Default-Theorien BackdoorDetect auf. Die Klasse BackdoorDetect sucht für die übergebenen Default-Theorien nach Backdoors, erzeugt damit die Redukte und gibt diese zurück an den Parser. Soll kein Backdoor gesucht werden, oder wird kein Backdoor der angegebenen Größe gefunden, so wird die Default-Theorie unverändert zurückgegeben.

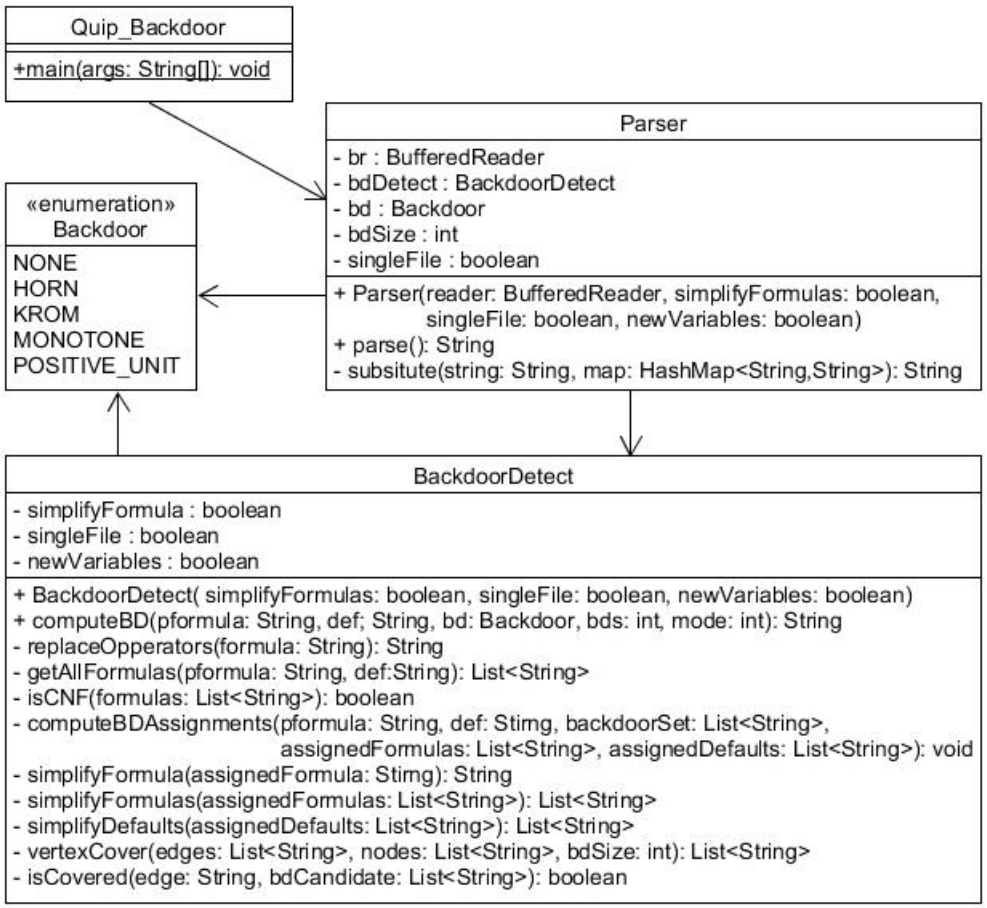

Abbildung 4.1: Klassendiagramm von Backdoor-QUIP

4.2.1 QUIP_Backdoor

Die Klasse QUIP_Backdoor enthält die main-Methode. Der Aufruf erwartet vier Argumente: den Namen der Datei, die die Problembeschreibung enthält,

und drei weitere Parameter die entweder 0 oder 1 sein dürfen.

QUIP_Backdoor *Dateiname P1 P2 P3*

P1: Der erste Parameter gibt an, ob die Formeln in den Redukten vereinfacht werden sollen. Ist der Wert 0, werden die Literale nur auf Wahr oder Falsch gesetzt, ist der Wert 1, werden zusätzlich erfüllte Klauseln und auf Falsch gesetzte Literale aus den Formeln entfernt.

P2: Der zweite Wert gibt an, ob die Ausgabe in eine einzelne Datei geschrieben werden soll (1), oder ob jedes Redukt als eigene Datei ausgegeben wird (0).

P3: Der dritte Wert steht für das Hinzufügen neuer Variablen zu den Schlussfolgerungen der Defaults (die y_is aus Definition 3.1). Bei 1 werden die Variablen hinzugefügt, bei 0 nicht.

Alternativ ist es auch möglich nur die Datei zu übergeben. In diesem Fall werden die Formeln vereinfacht, keine Variablen hinzugefügt und das Ergebnis in einer Datei ausgegeben.

Nach dem Aufruf wird die Datei geöffnet und an den Parser übergeben. Dieser verarbeitet die Eingabe und gibt das Ergebnis als String zurück. Daraufhin wird die Ausgabedatei erstellt und der String in diese geschrieben. Wird die Einstellung für mehrere Ausgabedateien verwendet, wird für jede Zeile des Strings eine eigene Datei erstellt. Die Namen der Ausgabedateien folgen dabei dem Muster: 'Name_bd' bei einer Datei und 'Name_Nummer_bd' bei mehreren Dateien.

4.2.2 Parser

Der Parser ist für die Verarbeitung der Eingabe zuständig. In diesem Abschnitt liegt der Fokus auf dem Ablauf des Parsens, eine genaue Beschreibung der Eingabe folgt im Abschnitt 4.2.5. Die Eingabe wird zeilenweise gelesen, wobei jede Zeile eine Definition, einen Befehl oder eine Einstellung enthalten darf. Als erstes wird geprüft ob eine Zeile einen Kommentar (durch % eingeleitet) enthält, in dem Fall betrachtet der Parser nur den Teil vor dem %, der Rest wird ignoriert. Kommentare werden nicht in die Ausgabe übernommen. Jede Anweisung, mit Ausnahme der Definition von aussagenlogischen Formeln, startet mit einem eigenen Schlüsselwort, dass mit @ beginnt. Anhand dieser Schlüsselwörter entscheidet der Parser, was mit der Zeile gemacht wird.

Definitionen von aussagenlogischen Formeln und Defaults (@D) werden vom Parser gespeichert, um diese, wenn sie in Default.Theorien verwendet werden, in diese einsetzen zu können. Aussagenlogische Formeln werden, für den Fall, dass sie auch für andere (nicht Default-Logik) Probleme benötigt werden, zusätzlich in die Ausgabe übernommen. Die Schlüsselwörter @DL, @DL1 und @DL2 stehen für Default-Theorien, wobei sich die Schlüsselwörter in der von QUIP verwendeten Methode zur Umwandlung in QBF unterscheiden. Bei Default-Theorie-Definitionen ersetzt der Parser zunächst alle Identifier durch vorher definierte Formeln und Defaults und übergibt sie dann an BackdoorDetect für die Backdoorsuche. Die Rückgabe wird dann zur Ausgabe hinzugefügt.

Wenn für eine Default-Theorie ein Backdoor gesucht wird, gibt BackdoorDetect eine Default-Theorie für jede Belegung des Backdoors zurück. In der Ausgabe ist nicht mehr zu erkennen, dass diese Theorien zusammengehören, daher wird empfohlen, nur eine Default-Theorie in der Eingabe zu definieren.

Zeilen, die mit @BD beginnen, bestimmen ob Backdoors gesucht werden sollen und, wenn ja, von welchem Typ. Standardmäßig werden keine Backdoors gesucht. Mit @BDS kann die Größe von gesuchten Backdoors eingestellt werden. Bei 0 (der Standardeinstellung) werden von eins beginnend alle Größen durchprobiert, bis ein Backdoor gefunden wird.

Zusätzlich zu den genannten Schlüsselwörtern, akzeptiert der Parser auch alle anderen von QUIP verwendeten Schlüsselwörter. Zeilen mit diesen Schlüsselwörtern werden unverändert in die Ausgabe übernommen. Wird die Einstellung für Ausgabe in mehreren Dateien verwendet, werden nur noch Default-Theorien ausgegeben. Wenn die gesamte Eingabe abgearbeitet ist, wird der Ausgabestring an QUIP_Backdoor zurückgegeben.

4.2.3 BackdoorDetect

Die Klasse BackdoorDetect ist für die Backdoorsuche und anschließende Erzeugung der Belegungen zuständig. Dazu erhält sie vom Parser eine Default-Theorie, den Typ und die Größe des zu suchenden Backdoors. Soll kein Backdoor gesucht werden, wird die Default-Theorie direkt an den Parser zurückgegeben. Bei der Definition von Default-Theorien können für Operatoren und Konstanten verschiedene Schreibweisen verwendet werden. Um die weitere Bearbeitung zu erleichtern, werden diese so ersetzt, dass eine einheitliche Schreibweise verwendet wird. Als nächstes wird geprüft, ob alle Formeln in KNF sind. Da die Algorithmen für die Backdoorsuche von KNF-Formeln

ausgehen, bricht die Backdoorsuche mit einer entsprechenden Fehlermeldung ab, wenn nicht alle Formeln der Default-Theorien in KNF sind.

Nun folgt die eigentliche Backdoorsuche, die abhängig von dem gesuchten Backdoortypen ist. Die Suche folgt dabei den in Kapitel 3.3 beschriebenen Vorgehensweisen. Dabei ist zu beachten, dass der verwendete Vertex Cover Algorithmus (auch für Hitting Set verwendet) nach einem Backdoor mit genau der angegebenen Größe sucht. Also selbst wenn ein kleineres Backdoor möglich wäre, wird dieses nicht gefunden, sondern nur ein Backdoor der angegebenen Größe. Ist hingegen die angegebene Größe zu klein und es wird kein Backdoor in der Größe gefunden, so wird die Suche mit einer entsprechenden Fehlermeldung abgebrochen. Wird im Parser keine Backdoorgröße angegeben oder es wird 0 als Größe eingestellt, dann wird von eins angefangen alle Größen durchprobiert, bis ein Backdoor gefunden wird. Auf diese Weise wird immer ein kleinstes Backdoor gefunden, dafür hat dieses Brute Force Verfahren eine höhere Laufzeit als die Parametrisierte Variante.

Nachdem ein Backdoor gefunden wurde, werden, je nach gewählter Einstellung, die Schlussfolgerungen der Defaults um frische Variablen erweitert oder nicht. Anschließend werden die Default-Theorien erzeugt, die durch Belegen der Backdoorvariablen entstehen. Gegebenenfalls werden nun die Formeln der erzeugten Default-Theorien vereinfacht bevor sie an den Parser zurückgegeben werden. Bei der Einstellung, dass die Ausgabe in einer einzelnen Datei erfolgt, wird zusätzlich ein Kommentar mit dem Gefundenen Backdoor zur Ausgabe hinzugefügt.

4.2.4 Scripte

Für die Ausführung von Backdoor-QUIP stehen mehrere Scripte zur Verfügung.

- **BD_Suche** führt QUIP_Backdoor aus und gibt als Ausgabe die Datei mit den Redukten zurück. Das gefundene Backdoor wird als Kommentar vor den Redukten angegeben.
- **BD_QUIP** führt erst QUIP_Backdoor aus und übergibt die Ausgabe direkt an QUIP. Das Ergebnis der Erweiterungssuche wird dann von QUIP interpretiert und auf der Konsole ausgegeben.
- **BD_QUIP2** hat den gleichen Ablauf wie BD_QUIP aber nutzt als Zwischenausgabe mehrere Dateien. Dies führt bei großen Backdoors zu besseren Laufzeiten, aber dafür wird die Ausgabe unübersichtlicher.

Sowohl BD_QUIP als auch BD_QUIP löschen alle während der Ausführung erstellten Dateien. Die Scripte benötigen den Namen der Eingabedatei als Argument. Der Aufruf von BD_QUIP sieht beispielsweise so aus:

BD_QUIP *Dateiname*

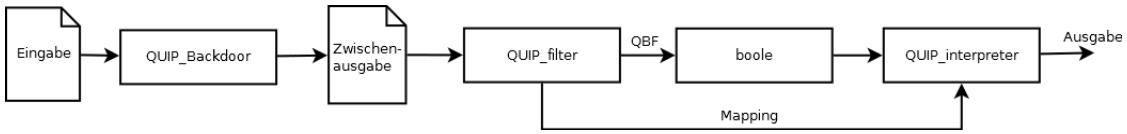

Abbildung 4.2: Ablauf von BD_QUIP

Zusätzlich wurden zwei weitere Scripte für die durchgeführten Tests erstellt. Der Ablauf von Test_Script1 entspricht dem von BD_Suche und Test_Script2 ist wie BD_QUIP2, allerdings wird hier QUIP_interpreter nicht ausgeführt. Beide Scripte haben keine Ausgabe und löschen während der Ausführung erstellte Dateien. Die Scripte messen die Zeit vor und nach den Aufrufen von QUIP_Backdoor und QUIP und schreiben die Differenzen in eine Logdatei. Die Testscripte können mehrere Eingabedateien gleichzeitig bearbeiten, dazu werden die Namen der Eingabedateien aus einer separaten Textdatei eingelesen. Der Aufruf der Testscripte benötigt daher keine Argumente.

4.2.5 Eingabe

In diesem Abschnitt wird die Syntax der Eingabe von BackdoorQUIP beschrieben. Diese unterscheidet sich nur minimal von der von QUIP verwendeten Syntax. Da wir uns in dieser Arbeit aber nur für Default-Logik interessieren, werden an dieser Stelle nur der Teile der Syntax vorgestellt, die für Default-Logik benötigt werden. Grundsätzlich werden aber auch die für andere Logiken benötigten Definitionen und Befehle vom Parser akzeptiert und in die Ausgabe übernommen, damit ein anschließender Aufruf von QUIP diese verwenden kann. Die vollständige QUIP-Syntax kann im Users Guide [17] nachgelesen werden

Die Beschreibung der Syntax erfolgt in Grammatik-Regeln, dabei schreiben wir *Nonterminale* in kursiver Schrift und Terminalsymbole werden unterstrichen. Zusätzlich verwenden wir folgende Metasymbole:

- $\rightarrow$ trennt die linke Seite (ein Nonterminal) einer Produktionsregel von der Rechten (Folge von Nonterminalen und/oder Terminalsymbolen).
- [Ausdr] Eckige Klammern geben an, dass der Ausdruck im inneren entweder einmal oder gar nicht auftritt.
- {Ausdr} Ausdrücke in geschweiften Klammern können beliebig oft auftreten.
- (Ausdr) Runde Klammern werden zum Gruppieren von Ausdrücken verwendet.
- $\text{Ausdr}_1|\text{Ausdr}_2$ gibt an, dass entweder der erste oder der zweite Ausdruck auftritt.

Bei der Eingabe unterscheiden wir zunächst zwischen drei Arten von *Anweisungen*: *Definitionen*, *Befehle* und *Einstellungen*. Zur ersten Gruppe gehören Definitionen von aussagenlogischen Formeln, Defaults und Variablen, wobei es nicht notwendig ist Variablen zu definieren, da Identifier, die nicht anders definiert wurden, wie Variablen behandelt werden. Definitionen von Defaul-Theorien sind Befehle. Befehle unterscheiden sich von Definitionen darin, dass sie von QUIP in QBFs umgewandelt und anschließend ausgewertet werden. Mit Einstellungen können Art (@BD) und Größe (@BDS) der gesuchten Backdoors festgelegt werden. Einstellungen gelten bis zum Ende der Eingabe oder bis sie geändert werden.

Anweisungen werden durch Zeilenumbrüche getrennt, sodass immer nur eine Anweisung in jeder Zeile steht. Zeilenumbrüche innerhalb von Anweisungen sind nicht erlaubt. Dies stellt einen Unterschied zur QUIP-Syntax dar, da dort Zeilenumbrüche beliebig zum Formatieren der Eingabe verwendet werden dürfen.

$$
\begin{array}{lll}
\mathit{Start} & \rightarrow & \{\mathit{Anweisung}\} \\
\mathit{Anweisung} & \rightarrow & \mathit{Definition} \mid \mathit{Befehl} \mid \mathit{Einstellung} \\
\mathit{Definition} & \rightarrow & \mathit{Formel\_def} \mid \mathit{Default\_def} \mid \mathit{Var\_def} \\
\mathit{Befehl} & \rightarrow & \mathit{Default\_Theorie} \\
\mathit{Einstellung} & \rightarrow & \underline{\text{@BD}}\ (\underline{\text{NONE}} \mid \underline{\text{HORN}} \mid \underline{\text{KROM}} \mid \underline{\text{MONOTONE}} \\
 & & \mid \underline{\text{POSITIVE\_UNIT}}) \\
 & \mid & \underline{\text{@BDS}}\ (\underline{0} \mid \ldots \mid \underline{9})\ \{\underline{0} \mid \ldots \mid \underline{9}\}
\end{array}
$$

Definitionen und Befehle können wie folgt definiert werden:

$$
\begin{array}{lll}
Formel\_def & \rightarrow & f\_ident \underline{:=} Theorie \\
Default\_def & \rightarrow & \underline{@D}\ d\_ident \underline{:=} Default\_Set \\
Var\_def & \rightarrow & \underline{@VAR}\ v\_idents
\end{array}
$$

$$
Default\_Theorie \rightarrow (\underline{@DL} \mid \underline{@DL1} \mid \underline{@DL2})\ \underline{(}\ Theorie\ \underline{;}\ Default\_Set\ \underline{)}
$$

Defaults und Defaultmengen setzen sich auf folgende Weise zusammen:

$$
\begin{array}{lll}
Default\_Set & \rightarrow & [\underline{\{}]\ Default \mid \underline{@D}\ d\_ident \mid \underline{@D}\ \underline{\{}\ d\_idents\ \underline{\}} \\
 & & \{\underline{,}\ (Default \mid \underline{@D}\ d\_ident \mid \underline{@D}\ \underline{\{}\ d\_idents\ \underline{\}})\}\ [\underline{\}}] \\
Default & \rightarrow & Formel\ \underline{:}\ Rechtfertigung\ \underline{|}\ Formel \\
Rechtfertigung & \rightarrow & \underline{\{\}} \mid Formel\ \{\underline{,}\ Formel\}
\end{array}
$$

Aussagenlogische Formeln und Theorien werden wie folgt aufgebaut:

$$
\begin{array}{lll}
Theorie & \rightarrow & \underline{\{\}} \mid [\underline{\{}]\ Formel\ \{\underline{,}\ Formel\}\ [\underline{\}}] \\
Formel & \rightarrow & Formel\ bin\_Op\ Formel \\
 & \mid & un\_Op\ Formel \\
 & \mid & \underline{(}\ Formel\ \underline{)} \\
 & \mid & \underline{[}\ Formel\ \underline{]} \\
 & \mid & v\_ident \mid f\_ident \mid Konstante
\end{array}
$$

Diese Operatoren und Konstanten können verwendet werden:

$$
\begin{array}{lll}
bin\_Op & \rightarrow & \underline{*} \mid \underline{\&} \mid \underline{AND} \\
 & \mid & \underline{+} \mid \underline{OR} \\
 & \mid & \underline{=} \mid \underline{EQ} \\
 & \mid & \underline{\hat{}} \mid \underline{XOR} \\
 & \mid & \underline{=>} \mid \underline{->} \mid \underline{IMPL} \\
un\_Op & \rightarrow & \underline{!} \mid \underline{NOT} \\
Konstante & \rightarrow & \underline{1} \mid \underline{TRUE} \mid \underline{0} \mid \underline{FALSE}
\end{array}
$$

Bei der Backdoorsuche werden gleichwertige Operetoren und Konstanten so ersetzt, dass nur noch *, +, !, 1 und 0 verwendet werden

Identifier verwenden folgende Syntax:

$$
\begin{array}{ll}
d\_idents & \rightarrow\ idents \\
v\_idents & \rightarrow\ idents \\
f\_ident & \rightarrow\ ident \\
d\_ident & \rightarrow\ ident \\
v\_ident & \rightarrow\ ident \\
idents & \rightarrow\ ident\ \{\underline{,}\ ident\} \\
ident & \rightarrow\ (\underline{A} \mid \ldots \mid \underline{Z} \mid \underline{a} \mid \ldots \mid \underline{z}) \\
 & \quad \{\underline{A} \mid \ldots \mid \underline{Z} \mid \underline{a} \mid \ldots \mid \underline{z} \mid \underline{0} \mid \ldots \mid \underline{9} \mid \underline{\_}\}
\end{array}
$$

Dabei dürfen die reservierten Wörter AND, OR, EQ, XOR, IMPL, TRUE und FALSE nicht als Identifier benutzt werden. Ebenfalls verboten ist die Verwendung mehrerer Unterstriche nacheinander (z.B. X_ _1).

Zur Formatierung dürfen eine beliebige Anzahl an Leerzeichen und Tabulatoren benutzt werden. Dabei sollte beachtet werden, dass auf Schlüsselwörter (mit @) immer ein Leerzeichen folgt. Identifier und die aus Buchstaben bestehenden Varianten der Operatoren und Konstanten sollten durch ein Leerzeichen getrennt werden. Weiterhin zur Formatierung erlaubt sind Leerzeilen oder Zeilen die nur einen Kommentar enthalten. Kommentare werden durch das Prozentzeichen (%) eingeleitet und gelten bis zum Ende der Zeile. Leerzeichen und Tabulatoren dürfen beliebig zur Formatierung der Eingabe verwendet werden.

4.3 Testfall Generator

Damit wir Testen können wie schnell Backdoor-QUIP Backdoors finden kann und ob die Suche nach Erweiterungen mit QUIP durch Backdoors beschleunigt werden kann, benötigen wir eine ganze Menge geeigneter Default-Theorien als Testfälle. Dabei können wir nicht beliebige Default-Theorien verwenden, sondern wir benötigen Theorien mit kleinen Backdoors. Theorien mit großen Backdoors würden aufgrund der exponentiell vielen Belegungen eine hohe Laufzeit verursachen und sind für die Tests daher ungeeignet. Aber im Bereich der kleinen Backdoors benötigen wir Backdoors, verschiedener Größe

um sehen zu können, bis zu welcher Größe Backdoors in der Praxis einen Nutzten bringen. Zusätzlich wollen wir Backdoors für vier verschiedene Formelklassen untersuchen und benötigen daher für jede dieser Klassen eine eigene Reihe von Default-Theorien.

Um Testfälle mit diesen Anforderungen zu erzeugen, wurde ein Generator entwickelt. Die Idee bei der Generierung ist es, zunächst Formeln der gewünschten Formelklasse zu erzeugen, um diese dann so um weitere Variablen (die zu suchenden Backdoorvariablen) zu erweitern, dass die Formeln anschließend nicht mehr zur ursprünglichen Formelklasse gehören. Auf die Weise ist Sichergestellt, dass ein Backdoor der gewünschten Formelklasse und Größe vorhanden ist.

Zusätzlich zu einem Namen für die Ausgabedatei und der Formelklasse benötigt der Generator noch folgende Eingaben: die Anzahl verschiedener Variablen, wie viele davon Backdoorvariablen sind, die maximale Anzahl von Literalen pro Klausel, die maximale Anzahl von Klauseln pro Formel, die Größe der Wissensbasis und die Menge der Defaults. Da sich die Formelklassen nur im Aufbau der Klauseln unterscheiden, ist der Ablauf des Generators bis zum Erstellen der Klauseln immer gleich. Zunächst werden für jeden Default drei Formeln erzeugt (Vorbedingung, Rechtfertigung und Schlussfolgerung). Für jede dieser Formel wird per Zufall bestimmt aus wie vielen Klauseln sie besteht. Ebenfalls per Zufall bestimmt ist die Anzahl der Literale pro Klausel. Beim Füllen der Klauseln wird für jedes Literal ausgelost, ob es eine 'normale' Variable oder eine Backdoorvariable wird. Normale Variablen folgen dabei den Einschränkungen der Formelklasse (z.B. bei Monotone sind sie immer positiv), während Backdoorvariablen gegen diese Einschränkungen verstoßen können. Zum Abschluss wird zufällig die Nummer der Variable gewählt und ob sie als positives oder negatives Literal vorkommt (sofern von der Formelklasse erlaubt).

Die Formeln der Wissensbasis sollten ursprünglich genauso aufgebaut werden wie die der Defaults, doch das führte dazu, dass praktisch nie die Vorbedingungen eines Defaults erfüllt wurden. Daher werden nun nur noch einzelne Literale für diese Formeln verwendet. Dadurch kommt es deutlich häufiger dazu, dass die Vorbedingung eines oder mehrerer Defaults erfüllt wird.

Nachdem die Default-Theorie generiert wurde, werden drei Ausgabedateien erstellt. Alle enthalten die Default-Theorie, aber mit verschiedenen Einstellungen für die Backdoorsuche. Eine Variante ist ohne Backdoorsuche, eine

mit Suche aber ohne Parameter und die letzte ist mit Backdoorsuche und der Backdoorgröße als Parameter. Da die Formeln zufällig generiert werden, ist es möglich, wenn auch unwahrscheinlich, dass die Default-Theorie ein kleineres Backdoor hat als angegeben wurde.

Die Vorgehensweise der Formelgenerierung funktioniert für Horn- und Monotone-Backdoors gut, allerdings hat sie bei Krom- und Positive-Unit-Backdoors leider einen unerwünschten Nebeneffekt. Da bei diesen Formelklassen die Anzahl der Literale pro Klausel beschränkt ist, werden bei größeren Klauseln alle überzähligen Literale mit Backdoorvariablen gefüllt. Dies führt dazu das der Anteil an Backdoorvariablen in den Formeln größer ist. Beim Bilden der Redukte nach der Backdoorsuche werden die Backdoorvariablen belegt und durch anschließende Vereinfachung der Formeln entfernt. Ein größerer Anteil an Backdoorvariablen führt also dazu, dass die resultierenden Redukte kleiner sind, was wiederum eine verkürzte Laufzeit bei der Erweiterungssuche zur Folge hat. Dadurch werden Vergleiche zwischen den Formelklassen erschwert und die einzigen Klassen, die wir bedenkenlos vergleichen können, sind Horn und Monotone.

Kapitel 5

Tests

In diesem Kapitel werden die mit Backdoor-QUIP durchgeführten Tests beschrieben. Dabei wird nach einer Beschreibung der Testdaten zunächst eine Reihe kleinerer Tests und deren Ergebnisse besprochen. Danach wird auf drei Tests ausführlicher eingegangen. Der erste Test ist zur Backdoorsuche, der zwiete vergleicht die Laufzeiten für EXT von Default-Theorien der verschiedenen Formelklassen und der dritte Test untersucht, wie sich das Lösen von EXT durch Backdoors beeinflussen lässt.

Die Tests wurden auf einem Intel Core i3 370M mit 4 GB Arbeitsspeicher durchgeführt. Um Schwankungen in der Laufzeit herauszurechnen, wurden alle Tests mehrfach durchgeführt und die Zeiten gemittelt. Für die Tests wurden die in Kapitel 4.2.4 beschriebenen Testscripte verwendet.

5.1 Testdaten

Als Testfälle für die folgenden Tests wurden mit dem in Kapitel 4.3 beschriebenen Generator Default-Theorien erzeugt. Diese Theorien bestehen jeweils aus 15 Defaults und fünf Formeln, wobei die Formeln nur aus genau einem Literal bestehen. Dadurch wird erreicht, dass am Anfang meistens einige Vorbedingungen der Defaults erfüllt sind, damit es überhaupt nichttriviale Erweiterungen geben kann. Voraussetzungen, Rechtfertigungen und Schlussfolgerungen der Defaults bestehen jeweils aus einer KNF-Formel mit ein bis zu vier Klauseln, welche wiederum ein bis vier Literale enthalten. Jede Default-Theorie verwendet 50 verschiedene Variablen.

Da die Anzahl der Literale und Klauseln der generierten Formeln vom Zufall abhängig ist, gab es einige Default-Theorien die deutlich größer oder

kleiner als der Durchschnitt waren. Um dadurch entstandene Ausreißer in den Testergebnissen zu vermeiden, wurden für alle Theorien die Zeit für einen einfachen QUIP Durchlauf (ohne Backdoors) gemessen. Default-Theorien, die mehr als 50 Prozent vom Durchschnitt der entsprechenden Formelklasse abwichen, wurden verworfen. Bei einem Durchschnitt von ungefähr 15 Sekunden benötigten einige dieser Theorien nur wenige Millisekunden, andere hingegen mehrere Minuten.

5.2 Erste Tests

Bevor wir zu den eigentlichen Tests kommen, sollen die Ergebnisse einiger kleinerer vorangegangener Testversuche erwähnt werden. Daran wird erkennbar, warum die für die Tests benutzten Einstellungen von Backdoor-QUIP verwendet wurden.

Bei den ersten Versuchen fielen einige Default-Theorien dadurch auf, dass die Berechnung mit Backdoors weit mehr als die 3^k-fache Laufzeit hatte als ohne Backdoors. Das würde bedeuten, dass die Berechnung jeder einzelnen Belegung länger dauert als die der ursprünglichen Formel, was seltsam wäre. Der Versuch die Belegungen einzeln zu prüfen zeigte, dass diese schneller zu lösen waren als die Ursprungsformel. Wie sich herausgestellt hat, hat QUIP Probleme mit zu großen Eingaben. Die exponentiell mit der Größe der Backdoors steigende Anzahl der Belegungen, die getestet werden müssen, führt schon bei recht kleinen Backdoors dazu, dass die Eingabe zu groß wird. Um dieses Problem zu vermeiden, wurde Backdoor-QUIP die Möglichkeit gegeben, die Ausgabe in mehrere Dateien aufzuteilen. Statt einer großen Datei werden QUIP also nacheinander viele kleine Dateien übergeben. Auf die Art und Weise kann QUIP auch große Eingaben ohne Leistungseinbrüche lösen.

Ein weiteres Problem, welches zu höheren Laufzeiten von QUIP führt, entsteht durch das Hinzufügen von frischen Variablen (y_is) zu den Schlussfolgerungen der Defaults. Der Grund für die höhere Laufzeit wurde nicht gefunden, aber es scheint nicht (alleine) an der höheren Anzahl verschiedener Variablen oder geringfügig größeren Formeln zu liegen. Da das Hinzufügen der Variablen in der Praxis nicht relevant ist, wurden sie in den weiteren Tests weggelassen.

Es stellte sich die Frage, ob sich das Vereinfachen der Formeln lohnt. Bei der Backdoorsuche wird dadurch mehr Zeit benötigt, insbesondere bei größeren Backdoors, da hier sehr viele Default-Theorien zu vereinfachen sind.

QUIP selbst vereinfacht die Formeln der eingegebenen Default-Theorien nicht, sondern wandelt sie direkt in QBF um. Tests haben ergeben, dass sich die vereinfachten QBFs meistens deutlich schneller lösen lassen. Auch hier ist der Unterschied besonders bei großen Backdoors zu erkennen, da dort durch das Belegen der Variablen mehr Klauseln erfüllt werden und somit die Formeln stärker schrumpfen. Im Normalfall war die Zeitersparnis beim Lösen der QBFs größer als die Dauer des Vereinfachens. Deshalb wurden bei den folgenden Tests die Formeln nach der Backdoorsuche vereinfacht.

Wie bereits in Kapietel 4.1 erwähnt, bietet QUIP die Möglichkeit Default-Theorien auf zwei verschiedene Weisen in QBFs zu transformieren. Dabei stellte Egly et al. fest, dass die zweite Variante (full set Charakterisierung) schlechtere Ergebnisse liefert. Unsere Tests führten zum selben Ergebnis, daher wurde im Weiteren nur die erste Variante benutzt.

5.3 Backdoor Suche

Der erste richtige Test beschäftigte sich mit der Backdoorsuche und dem anschließenden Erzeugen und Vereinfachen der Belegungen. Für jede der vier Formelklassen wurden Default-Theorien mit Backdoors der Größen eins bis acht getestet. Die Suche wurde für jede Default-Theorie, sowohl mit als auch ohne Parameter durchgeführt. Ein Aufruf von QUIP_Backdoor mit deaktivierter Backdoorsuche benötigt für die verwendeten Default-Theorien ca. 0,12 Sekunden. Mit Backdoorsuche wurden folgende Zeiten gemessen:

Backdoorgröße		1	2	3	4	5	6	7	8
Horn	Ohne Param.	0,437	0,548	0,708	1,098	1,88	5,011	27,548	274,875
	Mit Param.	0,438	0,547	0,707	1,05	1,828	4,84	25,201	220,538
Krom	Ohne Param.	0,342	0,556	0,889	1,6	7,826	76,683	831,224	5735,863
	Mit Param.	0,345	0,549	0,864	1,456	6,32	56,282	546,838	4044,426
Monotone	Ohne Param.	0,373	0,533	0,602	0,974	1,755	4,721	22,192	157,404
	Mit Param.	0,364	0,517	0,615	0,981	1,757	4,716	22,175	157,258
Positive Unit	Ohne Param.	0,447	0,552	0,71	0,942	1,466	3,112	13,904	81,351
	Mit Param.	0,451	0,564	0,7	0,949	1,471	3,124	13,904	81,431

Tabelle 5.1: Zeiten für Backdoorsuche (in Sekunden)

Man sieht sehr gut, dass die Nutzung des Parameters bei Monotone-Theorien keinen Unterschied macht, was daran liegt, dass der Algorithmus den Parameter nicht nutzt. Bei den Positive-Unit-Theorien zeigt sich ebenfalls kein Unterschied. Das kommt daher, das die vom Generator erzeugten Positive-Unit-Formeln einen großen Anteil an Backdoorvariablen haben, die dadurch praktisch immer auch negiert vorkommen. Daher werden alle Backdoorvariablen schon im ersten Schritt des Algorithmus gefunden, wodurch

auch hier der Parameter keinen Einfluss hat. Bei Horn- und Krom-Theorien hingegen macht sich der Parameter schon bemerkbar, allerdings erst bei höheren Laufzeiten, die dann selbst mit Parameter noch sehr hoch sind.

Ebenfalls gut sichtbar ist, dass die Laufzeit exponentiell mit der Backdoorgröße ansteigt, selbst bei den Monotone- und Positive-Unit-Theorien deren Suchalgorithmen nur lineare Zeit benötigen. Das liegt zum einen an der steigenden Anzahl der Belegungen die vereinfacht werden müssen und zum anderen am Schreiben der Ausgabe.

Die Backdoorsuche verwendet für Krom- und Horn-Formeln einen sehr ähnlichen Algorithmus, trotzdem sind die Laufzeiten bei Krom deutlich höher. Das liegt vermutlich daran, dass dort beim Aufbau des Graphen mehr Knoten mit Kanten verbunden werden und damit mehr verschiedene Variablen als Kandidaten für das Backdoor in Frage kommen.

5.4 Vergleich der Formelklassen

Im zweiten Test wurde getestet, wie schnell QUIP Defaul-Theorien der verschiedenen Formelklassen lösen kann. Dafür wurde für jede Formelklasse drei Default-Theorien generiert, die auch ohne Backdoor von der Klasse sind. Zum Vergleich wurden zusätzlich drei Default-Theorien mit normalen KNF-Formeln erstellt. Der Test lieferte das folgende Ergebnis:

Nr.	1	2	3
KNF	17,138	14,705	16,081
Horn	11,243	13,773	9,247
Krom	1,573	1,226	1,213
Monotone	13,858	10,367	9,484
Positive Unit	0,156	0,17	0,183

Tabelle 5.2: Testergebnisse des Formelklassenvergleichs (in Sekunden)

Der Test zeigt, dass Default-Theorien mit Horn- oder Monotone-Formeln von QUIP ungefähr gleich schnell bearbeitet werden und dabei etwas schneller sind, als normale KNF-Formeln. Krom- und Positive-Unit-Theorien waren nochmals deutlich schneller, was aber zum Teil daran liegt, das die Formeln kleiner sind, da die Klauseln nur zwei beziehungsweise ein Literal enthalten, anstatt bis zu vier Literalen, wie bei den anderen Formelklassen. Wie viel der höheren Geschwindigkeit von der geringeren Größe kommt und wie viel

von der Klasse ist deshalb schwer abzuschätzen.

Auch wenn sich Horn- und Monotone-Theorien schneller berechnen ließen als KNF-Theorien, so muss man bedenken, dass selbst für aus einer einzelnen Variable bestehende Backdoors schon drei solcher Theorien gelöst werden müssen. Da der Zeitvorteil zu gering ausfällt um die höhere Anzahl der Berechnungen auszugleichen, kann man bereits erahnen, dass sich Horn- und Monotone-Backdoors nicht eignen um EXT zu lösen.

5.5 EXT mit Backdoors

Im letzten Test behandeln wir unsere ursprüngliche Frage, ob die Nutzung von Backdoors die Berechnung von EXT beschleunigen kann. Als Testfälle wurden für jede Formelklasse für die Backdoorgrößen eins bis fünf jeweils zwei Default-Theorien erzeugt. Da die Positive-Unit-Theorien sehr niedrige Zeiten benötigten, wurde eine zweite Testreihe mit größeren Positive-Unit-Theorien, die jeweils 25 Defaults statt 15 haben, erstellt. Alle Default-Theorien wurden sowohl mit, als auch ohne Backdoorsuche getestet. Dabei wurde Für die Suche die Backdoorgröße als Parameter verwendet.

Backdoorgröße		1		2		3	
Nr.		1	2	3	4	5	6
Horn	Ohne BD	11,118	11,156	10,496	10,115	10,6	14,548
	Mit BD	29,396	26,413	36,388	75,458	148,521	57,939
Krom	Ohne BD	2,432	1,758	2,555	3,215	3,679	2,772
	Mit BD	4,326	2,491	41,122	9,874	17,76	9,527
Monotone	Ohne BD	8,98	9,298	9,325	10,895	13,403	10,163
	Mit BD	16,069	26,592	35,807	71,677	164,476	55,122
Positive Unit	Ohne BD	0,191	0,224	0,197	0,216	0,248	0,181
	Mit BD	0,516	0,526	0,857	0,961	1,954	1,492
Positive Unit (groß)	Ohne BD	1,095	1,062	1,741	1,207	3,151	2,68
	Mit BD	2,071	3,163	7,16	6,517	17,551	10,494

Backdoorgröße		4		5	
Nr.		7	8	9	10
Horn	Ohne BD	15,262	12,143	16,147	13,706
	Mit BD	111,248	462,467	417,443	364,58
Krom	Ohne BD	3,498	4,202	6,511	5,465
	Mit BD	55,366	17,704	59,475	97,614
Monotone	Ohne BD	13,149	13,544	14,266	14,631
	Mit BD	259,916	485,077	1084,227	168,018
Positive Unit	Ohne BD	0,198	0,173	0,329	0,694
	Mit BD	3,62	3,371	9,369	13,159
Positive Unit (groß)	Ohne BD	3,89	5,763	6,682	5,63
	Mit BD	16,129	24,262	47,78	33,575

Tabelle 5.3: Testergebnisse von EXT mit Backdoors (in Sekunden)

Die Ergebnisse des Tests zeigen, dass bei allen Formelklassen und Backdoorgrößen die Berechnung von EXT mit Backdoors länger dauert als ohne. Die Zeiten die für die Berechnungen mit Backdoor benötigt werden schwanken dabei selbst bei gleicher Formelklasse und Backdoorgröße stark. Dies sieht man beispielsweise bei den Monotone-Default-Theorien Neun und Zehn. Beide Theorien haben ein Backdoor der Größe fünf, doch die neunte Theorie benötigt mehr als sechsmal soviel Zeit wie die Zehnte. Die dritte Krom-Theorie zeigt, dass es teilweise dazu kommen kann, dass QUIP länger für ein einzelnes Redukt benötigt, als für die ursprüngliche Theorie

Auch wenn die Gesamtzeit durch die Nutzung von Backdoors ansteigt, lassen sich die einzelnen Redukte schneller bearbeiten als die Ursprungstheorie. Da EXT gelöst ist, wenn auch nur für ein Redukt eine Erweiterung gefunden wird, könnte man theoretisch die Suche beim ersten Fund abbrechen. In dem Fall würde die Nutzung von Backdoors immerhin dann einen Vorteil bringen, wenn bei den ersten untersuchten Redukten eine Erweiterung gefunden wird.

Eine weitere Beobachtung ist, dass Default-Theorien mit kleinen Krom- oder Positive-Unit-Backdoors von QUIP schneller bearbeitet werden, auch ohne das Backdoor zu nutzen. Das könnte daran liegen, dass bei diesen Default-Theorien ein deutlich größerer Anteil der Literale von den wenigen verschiedenen Backdoorvariablen beansprucht wird. Allerdings sollte dadurch auch die Redukte beim Vereinfachen deutlich schrumpfen, weshalb es überrascht, dass hier die Nutzung der Backdoors zum gleichen Zeitanstieg führt, wie bei den anderen Klassen.

Gerade bei Positive-Unit-Theorien, bei denen die Redukte in P lösbar sind, überrascht es, dass Backdoors keinen größeren Vorteil bringen. Dies führt uns zu der Vermutung, dass QUIP die Strukturen der Klassen (insbesondere von Positive-Unit) nicht (optimal) ausnutzt, auch wenn der zweite Test zeigte, dass QUIP Default-Theorien der untersuchten Formelklassen etwas schneller lösen kann.

Kapitel 6

Fazit und Ausblick

Die in Kapitel 5 durchgeführten Tests haben gezeigt, dass das in dieser Arbeit entwickelte Programm Backdoor-QUIP in der Lage ist, kleine Backdoors schnell zu finden. Dabei benötigte die Suche nach Krom-Backdoors die meiste Zeit, während Positive-Unit-Backdoor am schnellsten zu finden waren. Das lag allerdings an den generierten Testdaten, bei denen alle Backdoorvariablen in mindestens einer Formel negiert auftraten. Deshalb konnten die Positive-Unit-Backdoors bereits im ersten Schritt des Suchalgorithmus gefunden werden, welcher Identisch mit dem Suchalgorithmus für Monotone-Backdoors ist und in linearer Zeit läuft. Der aufwendige zweite Schritt mit dem Vertex-Cover Algorithmus kam hier also nicht zum Einsatz.

Obwohl Monotone- und in unserem Fall auch Positive-Unit-Backdoorts in linearer Zeit gefunden werden konnten, führten das Erstellen und Vereinfachen der Redukte sowie die Ausgabe dennoch auch bei diesen Backdoors zu exponentiell mit der Backdoorgröße steigender Laufzeit. Eine direkte Implementierung der Backdoorsuche in den Parser von QUIP könnte also durch den Wegfall der Zwischenausgabe und das doppelte Parsen der Eingabe nochmal etwas schneller sein.

Während die Backdoorsuche zufriedenstellende Ergebnisse erzielte, stellte sich die Anwendung von Backdoors für das Lösen von EXT als ineffizient heraus. Egal welche Backdoorgröße oder Formelklasse verwendet wurde, es hatte in allen Fällen einen deutlichen Anstieg der Laufzeit zur Folge. Überraschend war, das Backdoors aller Formelklassen zu einem vergleichbaren Anstieg der Laufzeit führten, wobei zu erwarten war, dass zumindest Positive-Unit-Backdoors bessere Zeiten erzielten. Dies führte zu der Vermutung, dass QUIP die Strukturen der Formelklassen nicht vollständig ausnutzt.

Es bleibt also unklar, ob die Herangehensweise, Backdoors für das Lösen von EXT zu nutzen, grundsätzlich keinen Geschwindigkeitsvorteil bringt, oder ob QUIP ungeeignet ist, die Vorteile der Backdoors zu nutzen. Daher wäre es aufschlussreich, vergleichbare Tests mit einem anderen Theorem-Prover durchzuführen, der Default-Theorien der Formelklassen erkennt und mit Algorithmen nach Erweiterungen sucht, die für die Klasse angepasst und optimiert sind.

Ein weiterer interessanter Ansatz wäre es, die Anwendung der Backdoors zu parallelisieren. Dabei eignen sich vermutlich besonders Monotone-Backdoors, da diese schnell gefunden werden können. Nachdem ein Backdoor gefunden ist, lassen sich die Erstellung und Vereinfachung der Redukte sowie das anchließende Lösen parallelisieren. Dadurch sollte die Bckdoorsuche auch bei größeren Backdoors noch in akzeptabler Zeit möglich sein. Die durchgeführten Tests haben gezeigt, dass sich einzelne Redukte deutlich schneller lösen lassen als die ursprüngliche Default-Theorie. Dies gilt insbesondere für größere Backdoors, da sich die Redukte dann stärker vereinfachen lassen. Die hohe Laufzeit der Auswertung entstand hauptsächlich dadurch, dass alle Redukte nacheinander ausgewertet werden mussten, was sich durch Parallelisierung vermeiden lässt. Es lassen sich also alle Schritte, die zu einer hohen Laufzeit geführt haben durch Parallelisierung beschleunigen. Gleichzeitig sollte es dadurch auch möglich werden, Erweiterungen für Default-Theorien mit größeren Backdoors als bisher zu suchen, ohne das die Laufzeit dabei zu stark ansteigt.

Abbildungsverzeichnis

Literaturverzeichnis

[1] *BDDlib*. http://www.cs.cmu.edu/~modelcheck/bdd.html

[2] Beyersdorff, Olaf ; Meier, Arne ; Thomas, Michael ; Vollmer, Heribert: The complexity of reasoning for fragments of default logic. In: *J. Log. Comput.* 22 (2012), Nr. 3, 587–604. http://dx.doi.org/10.1093/logcom/exq061. – DOI 10.1093/logcom/exq061

[3] Chen, Jianer ; Kanj, Iyad A. ; Xia, Ge: Improved upper bounds for vertex cover. In: *Theor. Comput. Sci.* 411 (2010), Nr. 40-42, 3736–3756. http://dx.doi.org/10.1016/j.tcs.2010.06.026. – DOI 10.1016/j.tcs.2010.06.026

[4] Downey, Rodney G. ; Fellows, Michael R.: *Parameterized Complexity*. Springer, 1999 (Monographs in Computer Science). http://dx.doi.org/10.1007/978-1-4612-0515-9. http://dx.doi.org/10.1007/978-1-4612-0515-9. – ISBN 978–1–4612–6798–0

[5] Downey, Rodney G. ; Fellows, Michael R.: *Fundamentals of Parameterized Complexity*. Springer, 2013 (Texts in Computer Science). http://dx.doi.org/10.1007/978-1-4471-5559-1. http://dx.doi.org/10.1007/978-1-4471-5559-1. – ISBN 978–1–4471–5558–4

[6] Egly, Uwe ; Eiter, Thomas ; Tompits, Hans ; Woltran, Stefan: Solving Advanced Reasoning Tasks Using Quantified Boolean Formulas. In: Kautz, Henry A. (Hrsg.) ; Porter, Bruce W. (Hrsg.): *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA.*, AAAI Press / The MIT Press, 2000. – ISBN 0–262–51112–6, 417–422

[7] Eiter, Thomas ; Gottlob, Georg: The Complexity of Logic-Based Abduction. In: Enjalbert, Patrice (Hrsg.) ; Finkel, Alain (Hrsg.) ; Wagner, Klaus W. (Hrsg.): *STACS 93, 10th Annual Symposium on*

Theoretical Aspects of Computer Science, Würzburg, Germany, February 25-27, 1993, Proceedings Bd. 665, Springer, 1993 (Lecture Notes in Computer Science). – ISBN 3–540–56503–5, 70–79

[8] Fernau, Henning: A Top-Down Approach to Search-Trees: Improved Algorithmics for 3-Hitting Set. In: *Algorithmica* 57 (2010), Nr. 1, 97–118. http://dx.doi.org/10.1007/s00453-008-9199-6. – DOI 10.1007/s00453–008–9199–6

[9] Fichte, Johannes K. ; Meier, Arne ; Schindler, Irina: Strong Backdoors for Default Logic. In: Creignou, Nadia (Hrsg.) ; Berre, Daniel L. (Hrsg.): *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings* Bd. 9710, Springer, 2016 (Lecture Notes in Computer Science). – ISBN 978–3–319–40969–6, 45–59

[10] Gottlob, Georg: Complexity Results for Nonmonotonic Logics. In: *J. Log. Comput.* 2 (1992), Nr. 3, 397–425. http://dx.doi.org/10.1093/logcom/2.3.397. – DOI 10.1093/logcom/2.3.397

[11] Moore, Robert C.: Semantical Considerations on Nonmonotonic Logic. In: *Artif. Intell.* 25 (1985), Nr. 1, 75–94. http://dx.doi.org/10.1016/0004-3702(85)90042-6. – DOI 10.1016/0004–3702(85)90042–6

[12] Niemelä, Ilkka: Towards Efficient Default Reasoning. In: *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, Morgan Kaufmann, 1995, 312–318

[13] Reiter, Raymond: A Logic for Default Reasoning. In: *Artif. Intell.* 13 (1980), Nr. 1-2, 81–132. http://dx.doi.org/10.1016/0004-3702(80)90014-4. – DOI 10.1016/0004–3702(80)90014–4

[14] Schulz, Daniel B.: *Backdoor-QUIP*. https://dumbledore.thi.uni-hannover.de/gitlab/danielschulz/ma/commit/b8e402bf

[15] Stillman, Jonathan: It's Not My Default: The Complexity of Membership Problems in Restricted Propositional Default Logics. In: Shrobe, Howard E. (Hrsg.) ; Dietterich, Thomas G. (Hrsg.) ; Swartout, William R. (Hrsg.): *Proceedings of the 8th National Conference on Artificial Intelligence. Boston, Massachusetts, USA, July 29 - August 3, 1990, 2 Volumes.*, AAAI Press / The MIT Press, 1990. – ISBN 0–262–51057–X, 571–578

[16] Williams, Ryan ; Gomes, Carla P. ; Selman, Bart: Backdoors To Typical Case Complexity. In: Gottlob, Georg (Hrsg.) ; Walsh, Toby (Hrsg.): *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, Morgan Kaufmann, 2003, 1173–1178

[17] Woltran, Stefan: *Users Guide to QUIP*. 2000

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit und die zugehörige Implementierung selbstständig verfasst und dabei nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Hannover, 24. Oktober 2018

Daniel Benjamin Schulz