QBF-Solver

Christian Henning

31. Juli 2014

Bachelorarbeit

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

Erklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel und Quellen genutzt habe.

Hannover, den 31. Juli 2014

Christian Henning

Inhaltsverzeichnis

1. Einleitung

Im Bereich der formalen Verifikation und Modellprüfung hat sich in den letzten Jahrzehnten verstärkt durchgesetzt, Probleme mithilfe von Booleschen Formeln zu beschreiben. Mit dem Finden von Algorithmen, welche die Erfüllbarkeit solcher Booleschen Formeln möglichst effizient untersuchen, beschäftigen sich noch heute viele Menschen rund um den Globus. Dies hat zur Folge, dass es mittlerweile sehr gute Algorithmen gibt, die zwar immer noch eine exponentielle Worst-Case-Laufzeit besitzen, aber dennoch viele praktische Instanzen annähernd effizient lösen.

Doch mit steigender Komplexität der Problemstellungen explodieren die Darstellungen der Probleminstanzen als Boolesche Formeln. Der Ruf nach einer kompakteren Darstellung wird somit immer lauter. Eine Möglichkeit der kompakteren Darstellung bieten quantifizierte Boolesche Formeln. Das zugehörige Erfüllbarkeitsproblem, also das Entscheidungsproblem, ob eine quantifizierte Boolesche Formel valide ist, wird mit $\mathcal{QBF}$ bezeichnet. Doch die Darstellung von Problemen als quantifizierte Boolesche Formel (im Folgenden auch mit QBF abgekürzt) erschwert das Testen auf Validität im Vergleich zur Darstellung als gewöhnliche aussagenlogische Formel. Die Gründe dafür liegen in der Möglichkeit, Variablen existentiell bzw. universell zu quantifizieren und den sich daraus ergebenen Abhängigkeiten von unterschiedlich quantifizierten Variablen untereinander, welche bei der Zuweisung von Wahrheitswerten beachtet werden müssen.

Diese zusätzlich zu beachtenden Bedingungen bei der Lösung von QBF-Instanzen sorgen dafür, dass das Problem $\mathcal{QBF}$ in der Komplexitätsklasse $\mathcal{PSPACE}$ einzuordnen ist, also der Klasse der Probleme mit polynomiellen Platzbedarf. Wohingegen das Entscheidungsproblem über die Erfüllbarkeit von Booleschen Formeln, welches mit $\mathcal{SAT}$ bezeichnet wird, zu der Klasse der nichtdeterministisch effizient lösbaren Probleme ($\mathcal{NP}$) gehört. Des Weiteren wird in [SM73] gezeigt, dass $\mathcal{QBF}$ ein $\mathcal{PSPACE}$-vollständiges Problem ist, also alle anderen Probleme der Klasse $\mathcal{PSPACE}$ in polynomieller Zeit auf $\mathcal{QBF}$ reduziert werden können. Da die Klasse $\mathcal{NP}$ eine Teilmenge der Klasse $\mathcal{PSPACE}$ ist, ist es möglich alle Probleme, die durch Instanzen von $\mathcal{SAT}$ beschrieben werden können, auch als QBFs zu beschreiben. Aufgrund dieser Eigenschaft spielt die Suche nach möglichst effizienten Algorithmen, die $\mathcal{QBF}$ entscheiden, eine herausragende Rol-

le.

Innerhalb der letzten zwei bis drei Jahrzehnte wurden gewaltige Fortschritte bei der Suche solcher Algorithmen erzielt. Ähnlich wie bei $\mathcal{SAT}$ gab es Versuche, jährliche Wettbewerbe zu etablieren, um die Leistungsfähigkeit aktueller QBF-Solver zu bewerten ([GNPT05a]). Jedoch spielen in der Praxis QBFs noch immer eine untergeordnete Rolle ([BM08]). Die Gründe dafür können vielfältig sein. Einerseits sind aktuelle QBF-Solver noch immer ihren Rivalen, den SAT-Solvern, unterlegen. Das mag zum einen daran liegen, dass viel weniger Menschen sich an der Suche nach effizienten Lösungsalgorithmen für QBF-Instanzen beteiligen und zum anderen daran, dass alle uns bekannten Lösungsalgorithmen für QBF-Instanzen erweiterte SAT-Algorithmen sind. Andererseits ist es schwer mit der Gewohnheit zu brechen, da gewöhnliche aussagenlogische Formeln sich in vielen praktischen Bereichen etabliert haben.

Diese Arbeit baut jedoch auf der Überzeugung auf, dass die zunehmende Komplexität der zu beschreibenden Probleme die Personen, die in Bereichen solcher Problembeschreibungen involviert sind, dazu zwingen wird auf kompaktere Darstellungen umzusteigen, auch wenn diese keine höhere Ausdrucksfähigkeit (vgl. Satz 2) bieten. Somit ist eine intensive Auseinandersetzung mit dem Problem $\mathcal{QBF}$ unvermeidbar, um auch in Zukunft bestmögliche Verfahren für die heute noch mit $\mathcal{SAT}$ operierenden Anwendungsbereiche anbieten zu können.

Das Ziel dieser Arbeit ist es Algorithmen vorzustellen, welche die Erfüllbarkeit von quantifizierten Booleschen Formeln untersuchen. Dabei sollen aktuelle Ansätze in Betracht gezogen und Probleme aufgezeigt werden. Begleitend zu dieser Arbeit wurde das Framework **qbfSolve** entwickelt, welches dazu dienen soll eine einheitliche Schnittstelle für verschiedenste Algorithmen zu bieten, die $\mathcal{QBF}$ entscheiden.

Von den Lesern dieser Arbeit wird ein grundlegendes Verständnis der mathematischen Logik verlangt, sowie insbesondere umfangreiche Kenntnisse in der Aussagenlogik.

2. Grundlagen

In diesem Kapitel möchten wir uns mit den Grundlagen für eine formale Beschreibung des Problems $\mathcal{QBF}$ befassen und Eigenschaften hervorheben, die sich Lösungsalgorithmen zu Nutze machen.

Definition 1. *Eine quantifizierte Boolesche Formel (QBF) ist eine Formel der Form:*

$$Q_1 x_{1,1} Q_1 x_{1,2} \cdots Q_1 x_{1,m_1} \cdots Q_k x_{k,m_k-1} Q_k x_{k,m_k} \ \phi$$

wobei $Q_1, \ldots, Q_k \in \{\forall, \exists\}$, $Q_i \neq Q_{i+1}$ für $i \in [1, k-1]$ und ϕ ist eine gewöhnliche aussagenlogische Formel. Alle freien Variablen in ψ werden im Folgenden als existentiell quantifiziert durch den äußersten Quantor im Präfix angenommen, so dass alle in ψ vorkommenden Variablen gebunden sind.
Das zugehörige Entscheidungsproblem, ob eine QBF ψ erfüllbar ist, heißt $\mathcal{QBF}$. D. h. ψ ist erfüllbar genau dann, wenn $\psi \in \mathcal{QBF}$.

An dieser Stelle möchten wir darauf hinweisen, dass wir durch unsere Definition von QBFs die Klasse aller möglichen quantifizierten Booleschen Formeln bereits eingeschränkt haben, denn jede QBF ist nach unserer Definition bereits in pränexer Normalform. Es ist allerdings möglich eine beliebige quantifizierte Boolesche Formel in polynomieller Zeit in pränexe Normalform zu bringen ([Zol04]). Da alle hier betrachteten Lösungsalgorithmen im Allgemeinen exponentielle Laufzeit haben, können wir etwaige vorangestellte polynomielle Umformungen vernachlässigen.

Definition 2. *Eine Belegung θ ist eine Zuweisung von Variablen $x_1, \ldots, x_n$ zu den Wahrheitswerten falsch (0) oder wahr (1). Sei ψ eine QBF und $x_1, \ldots, x_n$ die in ψ vorkommenden Variablen. Für eine Belegung θ ist $\psi[\theta]$ die Formel, in der alle in der Belegung enthaltenden Variablen durch ihren zugewiesenen Wahrheitswert ersetzt wurden. Enthält θ weniger als die in ψ vorkommenden Variablen, so nennen wir θ eine partielle Belegung.*
Im Folgenden fassen wir θ als Menge von auf wahr gesetzten Literalen auf, in der jede Variable entweder positiv oder negativ enthalten sein kann. Möchten wir die Zuweisungsreihenfolge der Literale in einer Belegung $\theta = \{l_1, \ldots, l_n\}$ hervorheben, dann beziehen wir uns auf das Tupel $\tilde{\theta} = (l_1, \ldots, l_n)$, wobei $i < j$ genau dann, wenn l_i vor l_j in die Belegung aufgenommen wurde.

Direkt aus Definition 1 wird folgender Satz ersichtlich.

Satz 1. *Jede aussagenlogische Formel ϕ ist logisch äquivalent zu einer QBF ψ, d. h. $\phi \iff \psi$.*

Beweis. Sei ϕ eine aussagenlogische Formel und $x_1, \ldots, x_n$ die in ϕ vorkommenden Variablen. Dann ist die Formel

$$\psi = \exists x_1 \cdots \exists x_n \ \phi$$

äquivalent zu ϕ, denn nach der Definition von Existenzquantoren gilt für eine beliebige Belegung θ:

$$\theta \models \phi \iff \theta \models \exists x_1 \cdots \exists x_n \ \phi$$

□

Analog dazu, ist nun auch die Frage interessant, ob jede QBF auch als gewöhnliche aussagenlogische Formel dargestellt werden kann.

Satz 2. *Jede QBF ψ ist logisch äquivalent zu einer aussagenlogische Formel ϕ, d. h. $\psi \iff \phi$.*

Beweis. Sei ψ eine QBF und ϕ' eine quantorenfreie aussagenlogische Formel. Wir betrachten folgende zwei Fälle:

Fall 1: Sei $Q_k = \forall$, dann gilt folgende Äquivalenz:

$$\begin{aligned}
\psi &= Q_1 x_{1,1} Q_1 x_{1,2} \cdots Q_k x_{k,1} \cdots Q_k x_{k,m_k-1} Q_k x_{k,m_k} \ \phi' \\
&\equiv Q_1 x_{1,1} Q_1 x_{1,2} \cdots \forall x_{k,1} \cdots \forall x_{k,m_k-1} \forall x_{k,m_k} \ \phi' \\
&\equiv Q_1 x_{1,1} \cdots \forall x_{k,1} \cdots \forall x_{k,m_k-1} \ (\phi'[x_{k,m_k}] \wedge \phi'[\neg x_{k,m_k}]) \\
&\equiv Q_1 x_{1,1} \cdots \forall x_{k,1} \cdots \forall x_{k,m_k-1} \ \phi''
\end{aligned}$$

mit $\phi'' = \phi'[x_{k,m_k}] \wedge \phi'[\neg x_{k,m_k}]$

Fall 2: Sei $Q_k = \exists$, dann gilt folgende Äquivalenz:

$$\begin{aligned}
\psi &= Q_1 x_{1,1} Q_1 x_{1,2} \cdots Q_k x_{k,1} \cdots Q_k x_{k,m_k-1} Q_k x_{k,m_k} \ \phi' \\
&\equiv Q_1 x_{1,1} Q_1 x_{1,2} \cdots \exists x_{k,1} \cdots \exists x_{k,m_k-1} \exists x_{k,m_k} \ \phi' \\
&\equiv Q_1 x_{1,1} \cdots \exists x_{k,1} \cdots \exists x_{k,m_k-1} \ (\phi'[x_{k,m_k}] \vee \phi'[\neg x_{k,m_k}]) \\
&\equiv Q_1 x_{1,1} \cdots \exists x_{k,1} \cdots \exists x_{k,m_k-1} \ \phi''
\end{aligned}$$

mit $\phi'' = \phi'[x_{k,m_k}] \vee \phi'[\neg x_{k,m_k}]$

In beiden Fällen ist ϕ'' eine gewöhnliche aussagenlogische Formel. Wie man leicht erkennen kann, ist es nun möglich durch sukzessives Anwenden der beiden oberen Äquivalenzen eine QBF ψ in eine logisch äquivalente Formel ϕ umzuwandeln, die keine Quantoren mehr enthält. Die so erhaltende Formel ϕ ist somit eine gewöhnliche aussagenlogische Formel.

□

Aus Satz 1 und 2 wird sofort der große Vorteil und Nachteil von QBFs ersichtlich. Der Nachteil ist, dass mittels QBFs nur dieselbe Ausdrucksfähigkeit wie bei gewöhnlichen aussagenlogische Formeln erreicht wird. Dennoch hat die zu einer QBF äquivalente aussagenlogische Formel im Allgemeinen eine exponentielle Länge im Vergleich zur ursprünglichen QBF. Diese Tatsache vermittelt somit die Notwendigkeit sich mit der Suche nach möglichst effizienten Algorithmen, die $\mathcal{QBF}$ entscheiden, zu beschäftigen.

Um auf bestimmte Stellen innerhalb des Quantorenpräfix referenzieren zu können, benötigen wir folgende Definition:

Definition 3. *Sei ψ eine QBF:*

$$\psi = Q_1 x_{1,1} \cdots Q_1 x_{1,m_1} \cdots Q_i x_{i,1} \cdots Q_i x_{i,m_i} \cdots Q_k x_{k,m_k} \ \phi$$

Die Zahl i bezeichnen wir als das Level einer Variable respektive eines Literals. Die Variablen auf dem selben Level können wir zu disjunkten Mengen X_i zusammenfassen, so dass gilt:

$$Q_i X_i := Q_i x_{i,1} \cdots Q_i x_{i,m_i}$$

und damit:

$$\psi = Q_1 X_1 \cdots Q_i X_i \cdots Q_k X_k \ \phi$$

Die Menge X_i bezeichnen wir im Folgenden auch als die i-te Quantorenmenge.

Analog zu den eingeführten Notationen in Veröffentlichungen, die sich mit QBFs auseinander setzen (vgl. z.B. [ZM02], [LB09]), führen wir hier folgende Schreibweisen ein. Die zu einem Literal l zugehörige Variable bezeichnen wir mit $var(l)$. Das Level einer Variablen x bezeichnen wir mit $lev(x)$. Auf das Level eines Literals l greifen wir analog mit $lev(l) := lev(var(l))$ zu. Die Mengen der existenzquantifizierten und universellquantifizierten Variablen bezeichnen wir wie folgt:

$$X_\exists := \bigcup_{Q_i = \exists} X_i$$

$$X_\forall := \bigcup_{Q_i = \forall} X_i$$

Um die Quantifizierung eines Literals zu verdeutlichen, schreiben wir auch $l \in X_Q := var(l) \in X_Q$ mit $Q = \{\exists, \forall\}$. Den Wert einer Variable x, also dessen aktuelle Belegung, erhalten wir durch $val(x)$ mit $val(x) \in \{0, 1, X\}$. Auf den Wert eines Literals l greifen wir analog zu:

$$val(l) := \begin{cases} val(var(l)), & \text{falls } l = var(l), \\ \neg val(var(l)), & \text{sonst} \end{cases}$$

Des Weiteren führen wir den unären Operator $\sim$ für Literale ein:

$$\sim l := \begin{cases} x, & \text{falls } var(l) = x \text{ und } l = \neg x, \\ \neg x, & \text{falls } var(l) = x \text{ und } l = x \end{cases}$$

Somit ist sichergestellt, dass $\sim l$ ebenfalls ein Literal ist, also eine positive oder negative Variable. Dies gilt für $\neg l$ nicht notwendigerweise, denn falls $l = \neg x$ folgt $\neg l = \neg\neg x$.

Im Bereich der SAT-Community hat sich durchgesetzt, aussagenlogische Formeln in konjunktiver Normalform (KNF) zu betrachten und die entsprechenden SAT-Solver dementsprechend zu spezialisieren. Eine aussagenlogische Formel ist in KNF, wenn sie eine Konjunktion von Klauseln ist. Eine Klausel ist eine Menge von Literalen, die disjunktiv miteinander verknüpft sind.

Auch bei der Entwicklung von QBF-Solvern wird diese Einschränkung fast ausschließlich übernommen. Wobei die erweiterte Syntax der QBFs einer erweiterten Normalform bedarf:

Definition 4. *Eine QBF ψ ist in pränexer konjunktiver Normalform (PKNF), wenn sie in pränexer Normalform ist*

$$\psi = Q_1 X_1 \cdots Q_k X_k \ \phi$$

und die aussagenlogische Formel ϕ in KNF ist. Da ϕ somit als Menge von Klauseln aufgefasst werden kann, schreiben wir auch

$$\psi = Q_1 X_1 \cdots Q_k X_k \ \{K_1, \cdots, K_n\}$$

Die Zugehörigkeit einer Klausel K zu einer QBF ψ notieren wir auch mit $K \in \psi$.

Da diese Arbeit sich ausschließlich mit Möglichkeiten beschäftigt $\mathcal{QBF}$ zu entscheiden, ist es ausreichend eine erfüllbarkeitsäquivalente QBF in PKNF zu einer beliebigen QBF zu betrachten. Tseitin-Transformation bietet eine Möglichkeit, eine beliebige QBF in polynomieller Zeit in eine erfüllbarkeitsäquivalente QBF in PKNF umzuwandeln ([Tse83], [GNT06, S. 374]). Im Folgenden werden wir deswegen nur noch QBFs in

PKNF betrachten, **d. h. wir schränken den Begriff QBF nun ein auf alle QBFs in PKNF**.

Für die weiteren Betrachtungen müssen wir noch den Begriff der Tautologie einführen. Eine Formel heißt *Tautologie*, wenn sie durch jede Belegung Ihrer Literale zu wahr evaluiert. Eine Klausel ist genau dann tautologisch, wenn sie ein Literal sowohl positiv als auch negativ enthält.

In [GNT06, S. 374f.] wurde gezeigt, dass jede beliebige QBF, die keine tautologischen Klauseln enthält, in linearer Zeit zu einer QBF umgewandelt werden kann, deren Variablen auf dem höchsten Level existenzquantifiziert sind. Also kann jede QBF zu einer QBF umgewandelt werden, die die Form

$$Q_1 X_1 \cdots Q_i X_i \cdots \exists X_k \ \phi \text{ besitzt.}$$

Der Grund für die Korrektheit dieser Umwandlung ist, dass die Literale der innersten Quantorenmenge X_k erst dann belegt werden dürfen, wenn alle anderen Literale dieser Klausel bereits belegt sind. Betrachten wir eine solche Klausel $K = \{l_1, \ldots, l_m, l_{m+1}, \ldots, l_n\}$ mit $l_1, \ldots, l_m \in X_1 \cup \cdots \cup X_{k-1}$ und $l_{m+1}, \ldots, l_n \in X_k$. Wenn die innerste Quantorenmenge nun universell ist, also $Q_k = \forall$, dann gibt es eine Belegung der Literale $l_{m+1}, \ldots, l_n$, so dass alle diese Literale zu falsch evaluieren. Da die Literale $l_1, \ldots, l_m$ bereits belegt sein müssen, kann eine solche Klausel nur dann zu wahr evaluieren, wenn es bereits ein Literal in $l_1, \ldots, l_m$ gibt, das mit wahr belegt ist. Es ist also ausreichend, die Erfüllbarkeit der Klausel $K' = \{l_1, \ldots, l_m\}$ zu untersuchen, anstatt der von Klausel K. Diese Überlegungen wurden in [GNT06, S. 374] erweitert zu folgender Definition:

Definition 5. *Eine Klausel ist in minimaler Form, falls ihre Literale mit dem höchsten Level existentiell sind. Die zu einer Klausel K minimale Form notieren wir mit $min(K)$. Eine QBF ist minimal, falls alle ihre Klauseln minimal sind.*

Wie in [GNT06, S. 374f.] bewiesen, ist jede QBF ψ äquivalent zu einer QBF ψ', wobei ψ' minimal ist. Aus ψ kann ψ' in polynomieller Zeit erhalten werden, indem alle tautologischen Klauseln aus ψ entfernt werden und anschließend alle universellen Literale auf dem höchsten Level einer Klausel entfernt werden.

Im Folgenden möchten wir ein paar Besonderheiten von QBFs betrachten, die von vielen QBF-Solvern ausgenutzt werden.

2.1. Einheitsklausel

In gewöhnlichen aussagenlogischen Formeln ist eine Einheitsklausel eine Klausel, die nur ein Literal enthält. Eine Einheitsklausel kann also nur dann wahr werden, wenn das beinhaltende Literal auf wahr gesetzt wird. Das Vorhandensein von Einheitsklauseln ermöglicht es somit SAT-Solvern die Belegung eines Literals zu erzwingen und so den Suchbaum zu beschneiden. Dieses Prinzip machen sich auch QBF-Solver zu nutze.

Definition 6. *Eine nichttautologische Klausel $K = \{l_1, \ldots, l_n\}$ heißt Einheitsklausel, wenn $l_1 \in X_\exists$, $l_2, \ldots, l_n \in X_\forall$ und $lev(l_1) < lev(l_i)$ für alle $i \geq 2$.*

Mithilfe von Definition 5 und den darauf folgenden Bemerkungen wird ersichtlich, dass eine Einheitsklausel K wie in Definition 6 äquivalent zu einer gewöhnlichen Einheitsklausel mit nur einem Literal ist.

Mittels Einheitsresolution (engl. *unit propagation*) können wir also Belegungen von Variablen erzwingen und somit den Suchbaum eines Lösungsalgorithmus beschneiden. Dazu weisen wir dem Literal l_1 (vgl. Def. 6) den Wert 1 für jede Einheitsklausel K in einer QBF ψ zu.

2.2. Monotone Literale

Wie schon im Abschnitt 2.1 erweitern wir auch in diesem Abschnitt eine Eigenschaft von aussagenlogischen Formeln, die die Suche nach erfüllenden Belegungen beschleunigen kann, auf QBFs. Monotone oder pure Literale sind in gewöhnlichen aussagenlogischen Formeln Literale, die entweder nur positiv oder nur negativ in einer Formel enthalten sind. Monotone Literale erzwingen also gewissermaßen ihre Belegung, da es nur sinnvoll ist sie mit wahr zu belegen, und beschneiden so den Suchbaum. Aufgrund der unterschiedlichen Quantifizierung von Literalen in QBFs bedarf diese Definition jedoch einer Erweiterung.

Definition 7. *Ein Literal l heißt monoton in einer QBF $\psi = Q_1 X_1 \cdots Q_k X_k\ \phi$, falls*

- *$l \in X_\exists$ und $\sim l \notin \phi$ oder*
- *$l \in X_\forall$ und $l \notin \phi$.*

Wie in Definition 7 ersichtlich wird, ändert sich für existenzielle Literale nichts im Vergleich zur ursprünglichen Definition. Bei universellen Literalen l erzwingen wir die

Belegung $val(l) = 1$, falls es nicht in der Formel vorkommt. Der Grund dafür ist, dass die zugehörige QBF nur dann erfüllbar sein kann, wenn auch bei der Belegung von l mit $val(l) = 1$ eine erfüllende Belegung für die aussagenlogische Formel ϕ gefunden wird. Mit dieser Belegung trägt l nämlich nicht zur Erfüllbarkeit der Formel bei, während eine Belegung mit $val(l) = 0$ i.A. Klauseln erfüllt.

Beispiel 1. *Betrachten wir die QBF*

$$\psi = \forall y_1 \exists x_1 \ ((y_1 \vee x_1) \wedge (y_1 \vee \neg x_1))$$

In ψ ist das universelle Literal $l := \neg y_1$ monoton, da $l \notin \psi$. Eine Belegung mit $var(l) = 1$ erfüllt also keine Klausel. Ist es nun möglich mit $var(l) = 1$ eine erfüllende Belegung θ zu finden, so ist die Belegung $(\theta \setminus \{l\}) \cup \{\sim l\}$ ebenfalls erfüllbar.

3. Algorithmen

In diesem Kapitel wollen wir uns einige Algorithmen anschauen, die $\mathcal{QBF}$ entscheiden. Die hier vorgestellten Algorithmen basieren allesamt auf Ansätzen, die bereits erfolgreich in SAT-Solvern getestet wurden.

3.1. Eine Erweiterung von DPLL auf QBFs

Einer der meist bekanntesten Algorithmen zum Lösen von aussagenlogischen Formeln ist DPLL ([DLL62]). Dies ist ein rekursiver Algorithmus, der mithilfe von chronologischen Backtracking alle Belegungen durchprobiert. Die meisten heutigen Algorithmen zur Untersuchung der Erfüllbarkeit von aussagenlogischen Formeln oder QBFs basieren auf diesem Ansatz.

Der für aussagenlogische Formeln gültige DPLL Algorithmus ist relativ einfach auf QBFs erweiterbar, indem die Quantifizierungsreihenfolge beachtet wird und untersucht wird, ob für jede Belegung einer universell quantifizierten Variablen eine erfüllende Belegung erzeugt werden kann. Betrachten wir für unsere Ausführungen folgende QBF.

$$\forall y_1 \exists x_1 \forall y_2 \exists x_2 \ ((y_1 \vee x_1 \vee \neg x_2) \wedge (\neg y_1 \vee x_1 \vee y_2) \wedge (\neg x_1 \vee \neg y_2 \vee x_2)) \tag{3.1}$$

In Abbildung 3.1 ist der vollständige Suchbaum von QBF (3.1) dargestellt. Anders als bei gewöhnlichen aussagenlogischen Formeln ist das Zertifikat, welches die Mitgliedschaft einer QBF zur Klasse der erfüllbaren QBFs $\mathcal{QBF}$ zeigt, nicht eine einzelne erfüllende Belegung, sondern eine Menge von erfüllenden Belegungen Θ. Wobei für jede Belegung $\{l_1, \ldots, l_{k-1}, l_k, l_{k+1}, \ldots, l_n\} \in \Theta$ mit $lev(l_i) \leq lev(l_j)$ für $i \leq j$ gilt, dass $\{l_1, \ldots, l_{k-1}, {\sim}l_k, l'_{k+1}, \ldots, l'_n\} \in \Theta$ ist, falls $var(l_k) \in X_\forall$. D. h. die Länge des Zertifikats ist im Allgemeinen exponentiell lang in Bezug auf die zugehörige QBF. Dies ist tatsächlich ein Problem, da für die meisten praktischen Instanzen Solver solche Zertifikate nicht speichern können und der Erfüllbarkeitstest somit nicht effizient verifizierbar ist. Ein Beispiel für ein solches Zertifikat ist in Abbildung 3.1 rot gekennzeichnet.

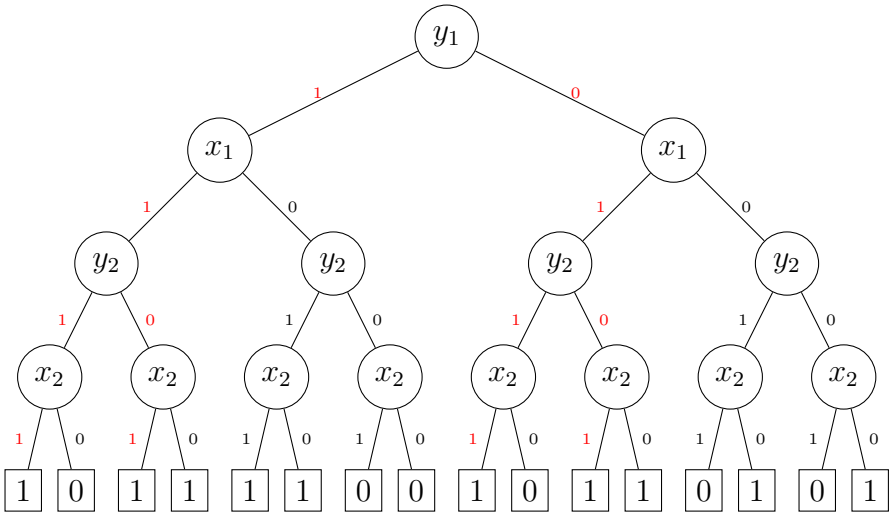


Abbildung 3.1.: Vollständiger Suchbaum für QBF (3.1) mit möglichem Zertifikat

Algorithmus 1 Q-DLL

Eingabe: QBF ψ, Belegung θ // Aufruf durch Q-DLL(ψ, $\emptyset$)
Ausgabe: True, falls $\psi[\theta]$ erfüllbar, sonst False

1: **if** θ erfüllt ψ **then return** True **end if**
2: **if** $\psi[\theta]$ ist unerfüllbar **then return** False **end if**
// Einheits-Resolution
3: **if** Literal l ist Einheitsliteral in $\psi[\theta]$ **then return** Q-DLL(ψ, $\theta \cup \{l\}$) **end if**
// Elimination monotoner Literale
4: **if** Literal l ist monoton in $\psi[\theta]$ **then return** Q-DLL(ψ, $\theta \cup \{l\}$) **end if**
5: $l \leftarrow$ ERHALTENÄCHSTESLITERAL(ψ, θ)
6: **if** $l \in X_\exists$ **then**
7: **if** Q-DLL(ψ, $\theta \cup \{l\}$) **then return** True
8: **else return** Q-DLL(ψ, $\theta \cup \{\sim l\}$) **end if**
9: **else** // $l \in X_\forall$
10: **if** Q-DLL(ψ, $\theta \cup \{l\}$) **then return** Q-DLL(ψ, $\theta \cup \{\sim l\}$)
11: **else return** False **end if**
12: **end if**

Algorithmus 1 zeigt ein Beispiel eines leicht erweiterten DPLL-Algorithmus für QBFs wie er unter anderem in [GNT06, S. 380] vorgestellt wurde. Dieser Algorithmus besitzt die Möglichkeit den Suchbaum zu beschneiden, indem er Einheits-Resolution (Zeile 3)

und Elimination monotoner Literale (Zeile 4) durchführt. Beide Erweiterungen können die Laufzeit deutlich verbessern. In Zeile 9 wird der Unterschied zu einem gewöhnlichen DPLL Algorithmus deutlich, da für universell quantifizierte Variablen beide Entscheidungen zu einer erfüllenden Belegung führen müssen. Für den Moment gehen wir nicht näher auf die Funktionsweise der Methode ERHALTENÄCHSTESLITERAL ein, setzen jedoch voraus, dass diese die Abhängigkeiten aufgrund der Quantifizierung beachtet.

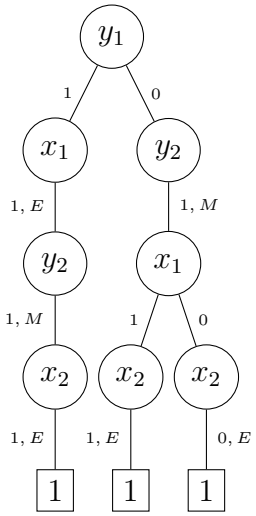


Abbildung 3.2.: Ein Beispiel für einen von Algorithmus 1 aufgespannten Suchbaum für die QBF (3.1)

In Abbildung 3.2 ist ein möglicher Suchbaum dargestellt, der den Ablauf von Algorithmus 1 am Beispiel von QBF (3.1) verdeutlichen soll. Der Suchbaum konnte offensichtlich aufgrund der beiden eingeführten Erweiterungen deutlich beschnitten werden. Belegungen, die aufgrund von Einheitsresolution erzwungen wurden, sind mit einem E markiert. Belegungen, die wiederum aufgrund dessen erzwungen wurden, dass das Literal monoton ist, sind mit einem M markiert.

3.2. Ein Versuch die Effizienz von SAT-Solvern zu integrieren

Als nächstes wollen wir den Algorithmus EVALUATE vorstellen [CSGG02]. Dieser Algorithmus nutzt ein paar nützliche Beobachtungen an QBFs aus, die den Erfüllbarkeitstest beschleunigen können. Sein Hauptanliegen ist es jedoch, das vorliegende Entscheidungsproblem auf $\mathcal{SAT}$ zu reduzieren.

Algorithmus 2 EVALUATE

Eingabe: QBF ψ, Belegung θ // Aufruf durch EVALUATE(ψ, $\emptyset$)
Ausgabe: True, falls $\psi[\theta]$ erfüllbar, sonst False
1: $\psi \leftarrow$ ENTFERNETAUTOLOGIEN(ψ, θ)
2: **if** $X_1 \subseteq X_\exists$ **then return** Σ_EVALUATE(ψ, θ)
3: **else return** Π_EVALUATE(ψ, θ) **end if**

Algorithmus 2 zeigt den Grundaufbau von EVALUATE. Dieser besteht im Wesentlichen aus den zwei Unterfunktionen Π_EVALUATE (Algorithmus 3) und Σ_EVALUATE (Algorithmus 11, Anhang A). Um die Korrektheit dieser Unterfunktionen sicherzustellen, ist es notwendig sämtliche tautologischen Klauseln aus der übergebenen QBF zu entfernen. Die Funktion Π_EVALUATE behandelt den Fall, dass die unbelegten Variablen mit dem niedrigsten Level universell sind. Die Funktion Σ_EVALUATE behandelt dementsprechend den Fall, dass diese Variablen existentiell sind. Da sich diese beiden Funktionen praktisch nur bei der Behandlung einer eventuellen Verzweigung unterscheiden, erklären wir im Folgenden deren Funktionsweise am Beispiel der Funktion Π_EVALUATE.

Algorithmus 3 Π_EVALUATE

Eingabe: QBF ψ, Belegung θ mit $\psi[\theta] = \forall X_1 \dots Q_k X_k \ \{K_1, \dots, K_n\}$
Ausgabe: True, falls $\psi[\theta]$ erfüllbar, sonst False
1: **if** $K_i \cap X_\forall = \emptyset$ für alle $i = 1 \dots n$ **then return** SAT(ψ, θ) **end if**
2: $\psi' \leftarrow$ ENTFERNEALLEUNIVERSELLENLITERALE(ψ, θ)
3: **if** SAT(ψ', θ) **then return** True **end if**
4: **while** Es gibt ein $l \in X_1$ mit $l \notin \theta$ **do**
5: **if** θ erfüllt ψ **then return** True **end if**
6: **if** $\psi[\theta]$ ist unerfüllbar **then return** False **end if**
7: **if** Es existiert $K \in \psi[\theta]$ mit $K \subseteq X_\forall$ **then return** False **end if**
8: **if** Literal l ist Einheitsliteral in $\psi[\theta]$ **then** $\theta \leftarrow \theta \cup \{l\}$
9: **else if** Literal l ist monoton in $\psi[\theta]$ **then** $\theta \leftarrow \theta \cup \{l\}$
10: **else if** Es gibt eine paarweise Widerlegung durch $K, K' \in \psi[\theta]$ **then**
11: **return** False
12: **else**
13: $l \leftarrow$ ERHALTENÄCHSTESLITERAL(ψ, θ)
14: **if** Π_EVALUATE(ψ, $\theta \cup \{l\}$) **then return** Π_EVALUATE(ψ, $\theta \cup \{\sim l\}$)
15: **else return** False **end if**
16: **end if**

17: **end while**
18: **return** Σ_EVALUATE(ψ, θ)

In Algorithmus 3 wird zuerst untersucht, ob in der Matrix der Formel $\psi[\theta]$ lediglich existentielle Literale vorkommen (Zeile 1). In diesem Fall vereinfacht sich das Problem zu $\mathcal{SAT}$ und es ist ausreichend eine erfüllende Belegung zu finden. Dies geschieht durch den Aufruf der Methode SAT, welche einen beliebigen SAT-Solver (z.B. DPLL) nutzt, um die Erfüllbarkeit der Matrix $\{K_1, \ldots, K_n\}$ zu prüfen. Anschließend wird die Formel ψ' konstruiert, indem alle Vorkommen von universellen Literalen aus der Formel ψ entfernt werden. Wenn nun bereits ψ' erfüllbar ist (Zeile 3), dann ist ψ *trivial erfüllbar* ([CSGG02, S. 107]). Eine QBF ψ heißt *trivial erfüllbar*, wenn es eine partielle Belegung $\theta \subseteq X_\exists$ gibt, die ψ erfüllt. D. h. die QBF ψ ist selbst dann noch erfüllbar, wenn alle universellen Literale mit falsch belegt werden. Der Umkehrschluss ist jedoch im Allgemeinen nicht korrekt. Z.B. ist die QBF $\forall y \exists x\ \{\{y, x\}, \{\neg y, \neg x\}\}$ erfüllbar, wohingegen $\forall y \exists x\ \{\{x\}, \{\neg x\}\}$ unerfüllbar ist.

Ob eine Belegung θ eine QBF ψ falsifiziert, also die Formel $\psi[\theta]$ unerfüllbar ist, können wir im einfachsten Fall analog zu der Unerfüllbarkeit einer aussagenlogischen Formel in KNF prüfen, d. h. $\psi[\theta]$ ist unerfüllbar, wenn eine leere Klausel $\{\}$ in $\psi[\theta]$ enthalten ist. Nun können wir aber mithilfe von Definition 5 und den anschließenden Bemerkungen ein weiteres Kriterium zur Prüfung der Unerfüllbarkeit einer QBF ψ entwickeln. Eine QBF ψ ist unerfüllbar, wenn sie eine Klausel K enthält, die lediglich universell quantifizierte Literale enthält ([CSGG02, S. 106]). In diesem Fall muss es eine Belegung dieser Literale geben, die die Klausel falsifiziert. Zu beachten ist hierbei, dass alle tautologischen Klauseln bereits zu Beginn von Algorithmus 2 entfernt wurden. Die Anwendung dieses Kriteriums findet in Zeile 7 statt.

In [CSGG02, S. 111] wurde ein weiteres Kriterium eingeführt, um die Unerfüllbarkeit einer QBF ψ zu prüfen. Dazu betrachten wir die zwei Klauseln $\{P_1, x, Q_1\}$ und $\{P_2, \neg x, Q_2\}$ mit $x \in X_\exists$, $P_1, P_2, Q_1, Q_2 \subseteq X_\forall$ und für $l \in P_1$ gilt $\sim l \notin P_2$, deren Literale aufsteigend nach ihrem Level sortiert sind. Wenn P_1 und P_2 leer sind, dann handelt es sich um zwei Einheitsklauseln, wobei eine Belegung des ersten Literals jeweils eine der beiden Klauseln falsifiziert. Falls P_1 und P_2 nicht leer sind, dann gibt es eine Belegung, die P_1 und P_2 falsifiziert, da beide ausschließlich aus universellen Literalen bestehen. In diesem Fall, erhalten wir erneut zwei Einheitsklauseln, die sich gegenseitig widerlegen. Andersherum ist möglich zu argumentieren, dass eine Belegung von x eine Klausel ausschließlich aus universellen Literalen entstehen lässt, welche nach obigen Ausführungen die Formel falsifiziert. Das Vorkommen solcher Klauseln bezeichnen wir als *paarweise Widerlegung* (Zeile 10) einer QBF ψ.

Wenn die Erfüllbarkeit von $\psi[\theta]$ noch unbekannt ist und keine Implikationen (Einheitsresolution, Elimination monotoner Literale) ausgeführt werden konnten, verzweigen wir anhand eines unbelegten Literals l. Wichtig ist hierbei, dass die Funktion ERHALTENÄCHSTESLITERAL nur Literale liefert, die zu dem niedrigsten Level mit unbelegten Literalen gehören.

Die von der Laufzeit her gesehen kritischsten Stellen dieses Algorithmus sind die Zeilen 1 und 3. Zeile 3 wird bei jedem Aufruf von Π_EVALUATE und Σ_EVALUATE ausgeführt und führt nur dann zum Erfolg, wenn das Problem auf $\mathcal{SAT}$ reduzierbar ist. Aus diesem Grund hängt die Laufzeit dieses Algorithmus massiv von der Effizienz des unterliegenden SAT-Solvers und der jeweiligen Instanz ab.

3.3. Nicht-chronologisches Backtracking mithilfe von Klausel- und Monom-Resolution

Im Folgenden wollen wir uns mit Möglichkeiten beschäftigen, den in Abschnitt 3.1 eingeführten Algorithmus zu erweitern, um unnötige Rechenschritte möglichst einzusparen. In diesem Abschnitt betrachten wir insbesondere ein in [GNT06] eingeführtes Verfahren, um ein nicht-chronologisches Backtracking (auch Backjumping genannt) in den Algorithmus 1 zu integrieren. Wir wollen die Idee zunächst anhand von Beispielen verdeutlichen.

$$\exists x_1 \forall y_1 \exists x_2 \ \{\{\neg y_1, x_2\}, \{\neg y_1, \neg x_2\}, \{x_1, y_1, x_2\}, \{\neg x_1, y_1, x_2\}\} \tag{3.2}$$

Die ersten zwei Klauseln von QBF 3.2 sind offensichtlich unerfüllbar, d. h. QBF 3.2 ist unerfüllbar, unabhängig von der Belegung der Variable x_1. Algorithmus 1 würde aufgrund der Quantifizierungsreihenfolge zuerst die Variable x_1 mit einem Wert belegen. Sobald sich herausstellt das dieser Zweig unerfüllbar ist, würde er x_1 mit einem anderen Wert belegen und den ganzen resultierenden Zweig erneut untersuchen. Es wäre also von Vorteil einen Algorithmus zu entwickeln, welcher erkennt, dass die Unerfüllbarkeit dieses Zweiges unabhängig von einer Entscheidung zu Beginn dieses Zweiges ist. Verschiedene Ansätze, um dies zu realisieren, wurden schon in vielen SAT-Solvern getestet. Bei QBFs lässt sich diese Idee sogar noch erweitern. Wenn eine Entscheidung an einem universellen Knoten zu einem erfüllbaren Zweig geführt hat, ist es notwendig die Belegung des Knotens zu negieren und den resultierenden Zweig erneut auf Erfüllbarkeit zu untersuchen. Auch hier kann es vorkommen, dass die Erfüllbarkeit des Zweiges unabhängig von der Entscheidung zu Beginn des Zweiges ist.

$$\forall y_1 \forall y_2 \exists x_1 \ \{\{\neg y_1, \neg y_2, x_1\}, \{\neg y_1, y_2, \neg x_1\}, \{y_1, \neg y_2, x_1\}, \{y_1, y_2, \neg x_1\}\} \tag{3.3}$$

QBF 3.3 ist erfüllbar durch das Zertifikat $\Theta = \{\{\neg y_2, \neg x_1\}, \{y_2, x_1\}\}$, also unabhängig von der Belegung der universellen Variable y_1. Doch Algorithmus 1 würde erneut zuerst anhand der Variable y_1 verzweigen und beide Zweige untersuchen. Es ist also notwendig ein Verfahren zu entwickeln, welches anhand bekannter Informationen entscheiden kann, ob die Erfüllbarkeit eines Zweiges bekannt ist. Wir wollen ein solches Verfahren analog zu [GNT06] basierend auf Resolution vorstellen.

Resolution ist ein Widerlegungsverfahren für aussagenlogische Formeln in KNF, d. h. die Unerfüllbarkeit von Formeln kann mithilfe von Resolution bewiesen werden [Vol13]. Die Idee dabei ist, dass die Belegung eines Literals, das in einer Klausel positiv vorkommt und in einer anderen negativ, nur eine Klausel jeweils erfüllen kann. D. h. in der Menge der übrigen Literale der beiden Klauseln muss noch mindestens ein Literal wahr werden. Ist dies nicht möglich, dann wurde die Formel widerlegt, da mindestens eine Klausel nicht erfüllt werden kann. Um also eine Formel zu widerlegen, muss sukzessive die leere Klausel $\{\}$ mithilfe von Resolution aus der ursprünglichen Klauselmenge abgeleitet werden. Dieses Verfahren wurde in [BKF95] auf QBFs erweitert. **Um die Korrektheit der folgenden Aussagen zu garantieren, betrachten wir von nun an nur noch QBFs in minimaler Form, die keine Tautologien enthalten.**

Definition 8. *Seien $K_1 \cup \{l\}$ und $K_2 \cup \{\sim l\}$ Klauseln mit $l \in X_\exists$, wobei es kein Literal l' gibt mit $l' \in K_1$ und $\sim l' \in K_2$. Die KResolution($K_1 \cup \{l\}$, $K_2 \cup \{\sim l\}$) ist gegeben als*

$$\frac{K_1 \cup \{l\} \quad K_2 \cup \{\sim l\}}{min(K_1 \cup K_2)}$$

Wie in Definition 8 zu erkennen ist, lassen wir keine Tautologien als Resolventen zu. Dies dient lediglich dem besseren Verständnis. Wie in [Zha02] bewiesen wurde, ist KResolution auch ohne diese Einschränkung widerlegungsvollständig.

Beispiel 2. *Betrachten wir eine mögliche Resolutionswiderlegung der QBF 3.2.*

$$\frac{\{\neg y_1, x_2\} \quad \{\neg y_1, \neg x_2\}}{min(\{\neg y_1\}) = \{\}}$$

Aufgrund des Vorhandenseins von universell quantifizierten Literalen, bieten QBFs auch die Möglichkeit eine neue Art der Resolution einzuführen. Dazu betrachten wir QBFs in PDNF. Eine aussagenlogische Formel ist in DNF, wenn sie eine Disjunktion von Monomen ist, wobei ein Monom eine Menge von Literalen ist, die konjunktiv

miteinander verknüpft sind. Analog zu QBFs in PKNF ist eine QBF in PDNF, wenn sie in pränexer Normalform ist und ihr präfixfreier Teil in DNF ist. Da $\mathcal{PSPACE}$ abgeschlossen ist bezüglich des Komplements, ist das Erfüllbarkeitsproblem für QBFs in PDNF auch $\mathcal{PSPACE}$-vollständig ([CmBLM05]). Betrachten wir zwei Monome M_1 und M_2, wobei ein universelles Literal l in dem einen Monom positiv und in dem anderen negativ vorkommt. Zwangsläufig ist durch eine Belegung von l eines der beiden Monome unerfüllbar. D. h. es genügt das Monom $(M_1 \cup M_2) \setminus \{l, \sim l\}$ anstelle von M_1 und M_2 zu betrachten, wenn die Erfüllbarkeit dieser Monome untersucht wird. Mithilfe dieser Beobachtungen ist es möglich, eine neue Art der Resolution zu entwickeln, sodass eine Formel erfüllbar sein muss, falls das leere Monom $\{\}$ mithilfe einer solchen Monom-Resolution abgeleitet werden kann. Bevor wir jedoch diese in [GNT06, S. 377f] eingeführte Resolution formal einführen, müssen wir uns noch ein paar Eigenheiten von Monomen ansehen. Ein Monom M heißt widersprüchlich, wenn es ein Literal l gibt, mit $l, \sim l \in M$. Ein Monom M heißt minimal, falls dessen Literale auf dem höchsten Level existenzquantifiziert sind. Auch hier gilt, dass jedes nicht widersprüchliche Monom äquivalent zu einem minimalen Monom ist ([GNT06, S. 377]), da eventuelle existentielle Literale auf dem höchsten Level eines Monoms immer mit wahr belegt werden können, wenn alle vorherigen Literale bereits mit wahr belegt sind. Und auch hier gilt analog, dass wir von nun an nur noch minimale Monome betrachten, die nicht widersprüchlich sind.

Definition 9. *Seien $M_1 \cup \{l\}$ und $M_2 \cup \{\sim l\}$ Monome mit $l \in X_\forall$, wobei es kein Literal l' gibt mit $l' \in M_1$ und $\sim l' \in M_2$. Die MResolution($M_1 \cup \{l\}$, $M_2 \cup \{\sim l\}$) ist gegeben als*

$$\frac{M_1 \cup \{l\} \quad M_2 \cup \{\sim l\}}{min(M_1 \cup M_2)}$$

MResolution angewandt auf QBFs in PDNF ist erfüllbarkeitsvollständig ([GNT06, S. 377]), d. h. eine QBF in PDNF ist genau dann erfüllbar, wenn das leere Monom $\{\}$ mittels MResolution aus ihrer ursprünglichen Monommenge abgeleitet werden kann. Doch wie können wir dieses Verfahren nutzen, um die Erfüllbarkeit von QBFs in PKNF zu untersuchen. Dazu nutzen wir das Verfahren der Modellgenerierung ([GNT06, S. 378]), d. h. wir entwickeln aus erfüllenden Belegungen Monome, so dass die Schnittmenge von allen Klauseln unserer Formel mit diesem Monom nicht leer ist.

Definition 10. *Ein Überdeckungsmonom M einer QBF*

$$\psi = Q_1 X_1 \cdots Q_k X_k \ \{K_1, \ldots, K_n\}$$

ist ein nicht widersprüchliches Monom mit der Eigenschaft $M \cap K_i \neq \emptyset$ für alle $i = 1 \ldots n$ und $var(l) \in X_\exists \cup X_\forall$ für $l \in M$. Die Erzeugung eines Überdeckungsmonoms heißt Modellgenerierung.

MResolution in Kombination mit Modellgenerierung ist erfüllbarkeitsvollständig für QBFs in PKNF ([GNT06, S. 378]).

Beispiel 3. *Betrachten wir erneut QBF (3.1) und dessen in Abbildung 3.1 markiertes mögliches Zertifikat. Da QBF (3.1) valide ist, muss aus ihr das leere Monom* $\{\}$ *mithilfe von MResolution in Kombination mit Modellgenerierung ableitbar sein. Dazu müssen wir eine Menge von Monomen generieren, die jeweils den Eigenschaften des Monoms* M *aus Definition 10 gerecht werden. Eine solche Menge leiten wir aus dem Zertifikat für QBF (3.1) ab, welches aus den Belegungen*

$$\{\{y_1, x_1, y_2, x_2\}, \{y_1, x_1, \neg y_2, x_2\}, \{\neg y_1, x_1, y_2, x_2\}, \{\neg y_1, x_1, \neg y_2, x_2\}\}$$

besteht. Wenn wir diese Literalmengen nun nicht mehr als Belegungen auffassen, sondern als Monome $M_1, \ldots, M_4$*, dann haben wir die gesuchte Menge gefunden. Denn jedes Monom* $M_1, \ldots, M_4$ *entsprach ursprünglich einer Belegung, die die Matrix von QBF (3.1) erfüllt, somit gilt für alle Klauseln* $K \in$ *QBF (3.1), dass* $M_i \cap K \neq \emptyset$ *für* $i = 1 \ldots 4$*. Mithilfe der Minimalformen von* $M_1, \ldots, M_4$ *ist nun folgende Resolutionsableitung möglich.*

$$\dfrac{\dfrac{\dfrac{\{y_1, x_1, y_2\} \quad \{y_1, x_1, \neg y_2\}}{min(\{y_1, x_1\}) = \{y_1\}} \quad \dfrac{\{\neg y_1, x_1, y_2\} \quad \{\neg y_1, x_1, \neg y_2\}}{min(\{\neg y_1, x_1\}) = \{\neg y_1\}}}{\{\}}}$$

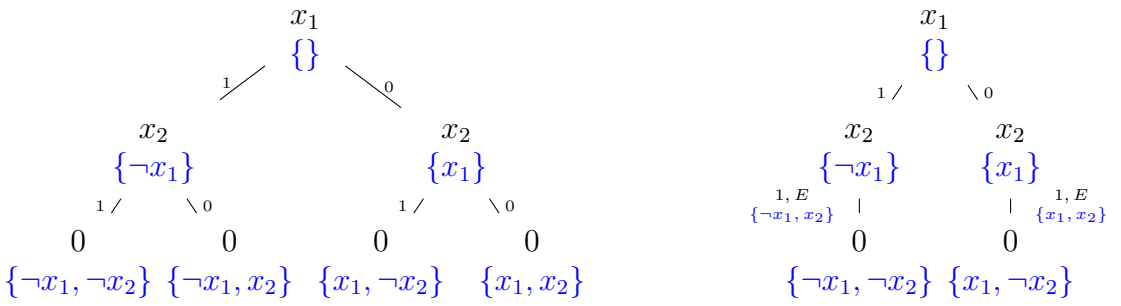
Anmerkung: QBF (3.1) ist bereits trivial erfüllbar (vgl. Abschnitt 3.2) durch die Belegung $\theta = \{x_1, x_2\}$*. Somit ist auch* $\{\{x_1, x_2\}\}$ *eine nach obigen Bemerkungen gültige Monommenge, denn* $min(\{x_1, x_2\}) = \{\}$*.*

Nun gilt es die Frage zu erläutern, wie die eingeführten Verfahren genutzt werden können, um Algorithmus 1 zu erweitern. Zur Beantwortung dieser Frage greifen wir auf die Erkenntnis zurück, dass ein Durchlauf von DPLL mit einer unerfüllbaren aussagenlogischen Formel einer baumartigen Resolutionswiderlegung entspricht ([BKS04]). Dazu assoziieren wir mit jedem Blatt im DPLL-Suchbaum eine durch die aktuelle Belegung unerfüllbare Klausel. Durch sukzessive Anwendung von Resolution können wir so jedem Knoten eine Resolvente zuweisen, so dass der Wurzel die leere Klausel genau dann zugewiesen wird, wenn die Eingabeformel unerfüllbar ist.

Beispiel 4. *Betrachten wir die unerfüllbare aussagenlogische Formel*

$$\phi = \{\{x_1, x_2\}, \{x_1, \neg x_2\}, \{\neg x_1, x_2\}, \{\neg x_1, \neg x_2\}\}$$

Aufgrund der Unerfüllbarkeit von ϕ *muss es möglich sein, die leere Klausel abzuleiten. In Abbildung 3.3(a) ist der Suchbaum von DPLL mit Eingabeformel* ϕ *abgebildet. Jedem*



(a) DPLL

(b) DPLL mit Einheitsresolution

Abbildung 3.3.: Suchbäume verschiedener DPLL-Algorithmen mit eingezeichneter Resolutionswiderlegung der in Beispiel 4 behandelten aussagenlogischen Formel

unerfüllbaren Blatt wurde eine Klausel aus der ursprünglichen Matrix zugewiesen (in blau), so dass diese unter der aktuellen Belegung widerlegt ist. Durch Resolution mit dem direkt benachbarten Knoten wird dem Elternknoten erneut eine widerlegte Klausel, die Resolvente, zugewiesen, bis die leere Klausel abgeleitet wurde. Ein Suchbaum eines erweiterten DPLL Algorithmus, der Implikationen wie Einheitsresolution zulässt, ist in Abbildung 3.3(b) dargestellt. Wenn eine Belegung durch eine Einheitsklausel erzwungen wird, wird deren ursprüngliche Klausel mit der erzwungenen Belegung assoziiert. Im linken Zweig geschieht dies zum Beispiel durch die Einheitsklausel $\{x_2\}$, welche mithilfe der aktuellen Belegung aus der ursprünglichen Klausel $K = \{\neg x_1, x_2\}$ gewonnen wurde. Dem anschließenden Blatt wird nun die unerfüllbare Klausel $K' = \{\neg x_1, \neg x_2\}$ zugewiesen. Da diese das negierte Einheitsliteral enthält, ist es notwendig K' und K zu resolvieren, um sich dieses Literals zu entledigen.

Auch ein erweiterter DPLL-Algorithmus, der Implikationen wie Einheitsresolution und Detektion monotoner Literale zulässt, mit einer unerfüllbaren Formel als Eingabeformel stellt eine Resolutionswiderlegung dar. Betrachten wir die Fälle einzeln. Angenommen eine Belegung $\theta \cup \{l\}$ wird durch eine Einheitsklausel $K[\theta] = \{l\}$ erzwungen. Dann müssen alle Literal $l \neq l' \in K$ mit falsch belegt sein ($val(l') = 0$). D. h. die Belegung $\theta \cup \{\sim l\}$ führt zu einem unerfüllbaren Blatt, welches die Klausel K zugewiesen bekommt. Aufgrund dessen assoziieren wir direkt eine Belegung θ, deren nächste Zuweisung durch eine Einheitsklausel $K[\theta]$ erzwungen wird, mit der Klausel K. Bei monotonen Literalen l nutzen wir aus, dass bei einer Belegung dieser mit $val(l) = 1$ das Literal $\sim l$ nicht Teil einer unerfüllbaren Klausel sein kann und es von daher nicht nötig ist $\sim l$ aus einer unerfüllbaren Klausel heraus zu resolvieren.

Dieses Konzept erweitern wir nun analog zu [GNT06, S. 378ff.] auf Q-DLL. Dazu weisen wir den erfüllten Blättern im Q-DLL-Suchbaum Monome und den unerfüllten Klauseln zu. Wir könnten nun zuerst denselben Ansatz wie oben machen und den unerfüllten Blättern eine Klausel zuweisen, in der jedes Literal zu falsch evaluiert. Da wir bei Klauseln jedoch nur existentielle Literale resolvieren, können wir uns zu Nutze machen, dass eine nicht-tautologische Klausel bereits dann widerlegt ist, wenn alle existentiellen Literale in ihr mit falsch belegt sind (vgl. Abschnitt 3.2). Es ist also ausreichend, ein unerfülltes Blatt mit einer falsifizierten Klausel zu belegen.

Definition 11. *Eine Klausel K heißt falsifizierte Klausel unter der Belegung θ, wenn sie minimal und nicht tautologisch ist und für alle Literale $l \in K$ gilt:*

- *falls $l \in X_\exists \Rightarrow \sim l \in \theta$ und*
- *falls $l \in X_\forall \Rightarrow l \notin \theta$.*

Somit ist implizit sichergestellt, dass eine falsifizierte Klausel K keine existentiellen Literale enthält, deren Negation als monoton zugewiesen wurden. Zur Verdeutlichung dieser Tatsache betrachten wir das monotone Literal $l \in X_\exists$ in einer QBF $\psi[\theta]$. Falls es eine Klausel $K' \in \psi$ gibt mit $\sim l \in K'$, dann muss es ein $l' \in K'$ geben, so dass $l' \in \theta$, anderenfalls wäre l nicht monoton in $\psi[\theta]$. D. h. es kann keine falsifizierte Klausel $K \in \psi[\theta']$ entstehen mit $\theta \subseteq \theta'$, $\sim l \in K$ und $l' \notin \theta$ für alle $l' \in K$. Das monotone Literal l selbst kann trivialerweise nicht Element einer falsifizierten Klausel sein, da es mit wahr belegt wird.

Nun ist es für das weitere Vorgehen, wie wir später sehen werden, essentiell zu wissen, ob eine KResolution von zwei falsifizierten Klauseln K_l, $K_{\sim l}$ anhand eines Literals l geblockt werden kann durch Literale l' mit $var(l') \neq var(l)$, $l' \in K_l$ und $\sim l' \in K_{\sim l}$.

Satz 3. *Die KResolution zweier falsifizierter Klauseln $K_1 \cup \{l\}$ und $K_2 \cup \{\sim l\}$ kann nur aufgrund von unbelegten universellen Literalen $l_1, \ldots, l_k \in K_1$ geblockt werden, wenn $\sim l_1, \ldots, \sim l_k \in K_2$ und $l_1, \ldots, l_k$ ein niedrigeres Level haben als ein existentielles Literal $\sim l_E \in K_1$, wobei l_E als Einheitsliteral zugewiesen wurde.*

Beweis. Sei θ eine partielle Belegung und K_1, K_2 zwei unter θ falsifizierte Klauseln. Betrachten wir die KResolution der Klauseln $K_1 \cup \{l\}$ und $K_2 \cup \{\sim l\}$ anhand des Literals l. Sei l' ein Literal, das diese KResolution blockt, also $var(l') \neq var(l)$, $l' \in K_1$ und $\sim l' \in K_2$. Angenommen l' wäre existentiell. Dann müsste $\sim l' \in \theta$ sein nach Definition 11. Da $\sim l' \in K_2$ ist, wäre K_2 keine falsifizierte Klausel unter der Belegung θ. Dies ist ein Widerspruch zu der Annahme, dass K_2 falsifiziert sei unter θ. Das Literal l' kann also nicht existentiell sein.

Angenommen l' sei bereits mit einem Wert belegt, also $l' \in \theta$ oder $\sim l' \in \theta$. Da $l' \in K_1$ und $\sim l' \in K_2$ wäre somit die Klausel K_1 oder K_2 zwangsläufig erfüllt. Dies ist erneut ein Widerspruch zur Annahme und zeigt, dass l' unbelegt sein muss. Da K_1 und K_2 als falsifizierte Klauseln in Minimalform sind, muss deren Literal mit dem jeweils höchsten Level existentiell sein. Betrachten wir die Klausel $K_1 = \{\ldots, l', \ldots, \sim l_E\}$. Sei $\sim l_E \in X_\exists$ das Literal mit dem höchsten Level in K_1, insbesondere also $lev(l') < lev(\sim l_E)$, da $l' \in X_\forall$. Da $\sim l_E$ existentiell ist, muss gelten, dass $l_E \in \theta$. Dies bedeutet insbesondere, dass die Belegung des Literals l_E impliziert wurde, da das Literal l' ansonsten aufgrund der Quantifizierungsreihenfolge zuerst belegt worden wäre. Da, wie oben erläutert, monotone Literale und deren Negation nicht in falsifizierten Klauseln auftauchen, muss l_E ein Einheitsliteral sein. □

Wie Satz 3 zeigt, müssen wir vor einer KResolution zweier unter der Belegung θ falsifizierter Klauseln K_1 und K_2 erst die blockierenden universellen Literale, in beispielsweise K_1, loswerden. Dies tun wir indem wir uns jeweils des Einheitsliteral auf dem höchsten Level der Klausel K_1 entledigen, solange bis alle blockierenden Literale entfernt wurden. Nehmen wir also an, dass $K_1 = \{l_1, \ldots, l_k, \ldots, \sim l_E\}$ mit $lev(l_i) \leq lev(l_j)$ für $i \leq j$, wobei l_k ein unbelegtes universelles Literal ist mit $\sim l_k \in K_2$ und $\sim l_E \in X_\exists$ mit $lev(l_i) \leq lev(\sim l_E)$. Das Literal l_E muss laut Satz 3 ein Einheitsliteral sein und damit ist $\theta = \theta' \cup \{l_E\} \cup \theta''$, wobei in θ' alle Literale sind, die vor l_E zugewiesen wurden. Sei K_E die zugehörige Einheitsklausel. Wir wissen, dass K_E unter der Belegung $\theta' \cup \{\sim l_E\}$ eine falsifizierte Klausel sein muss. Außerdem wissen wir, dass es in K_E keine unbelegten universellen Literale l' mit $lev(l') \leq lev(l_E)$ gibt (vgl. Def. 6). Sollte es blockierende universelle Literale l' mit $lev(l') \geq lev(l_E)$ geben, dann muss es ein existentielles Literal l'' geben mit $lev(l') \leq lev(l'')$, dessen Belegung auch durch Einheitsresolution impliziert wurde (monotone Literale sind in falsifizierten Klauseln nicht enthalten; alle betrachteten Klauseln sind in Minimalform), d. h. aber auch, dass es eine erste solche Einheitsklausel geben muss, so dass keine Blockierungen bei einer KResolution mit dieser auftreten. Durch sukzessive Resolution können somit die Blockierungen bei der KResolution von K_1 und K_E aufgelöst werden. Da $K_E \setminus \{l_E\}$ bereits unter der Belegung θ' falsifiziert ist, ist der Resolvent $K_1' := min((K_1 \cup K_E) \setminus \{l_E, \sim l_E\})$ unter θ ebenfalls eine falsifizierte Klausel. Falls es in K_1' noch immer existentielle Literale l_E' mit $lev(l_k) < lev(l_E')$ gibt, dann entledigen wir uns dieser existentiellen Literale l_E' auf derselben Weise, bis alle solche Literale entfernt sind und das blockierende Literal l_k aufgrund der Minimalform ebenfalls herausfällt. Dieses Verfahren ist in Algorithmus 4 dargestellt ([GNT06, S. 388]).

Algorithmus 4 ERWEITERTEKRESOLUTION

Eingabe: QBF ψ, Belegung θ, Klausel K_1, Klausel K_2
Ausgabe: Resolvent K von K_1 und K_2 unter Berücksichtigung blockierender Literale

// K_1 und K_2 müssen unter der Belegung θ falsifizierte Klauseln sein
1: **if** Es existiert $l \in K_1$ mit $l \in X_\forall$ und $\sim l \in K_2$ **then**
2: $l \leftarrow$ ERHALTELITERALMITHÖCHSTENLEVELINKLAUSEL(K_1)
3: $\theta' \leftarrow$ ERHALTEBELEGUNGSPÄFIX(θ, $\sim l$)
4: $K_E \leftarrow$ ERHALTEZUGEHÖRIGEEINHEITSKLAUSEL(ψ, θ', $\sim l$)
5: $K_1' \leftarrow$ KRESOLUTION(K_1, K_E)
6: **return** ERWEITERTEKRESOLUTION(ψ, θ, K_1', K_2)
7: **else**
8: **return** KRESOLUTION(K_1, K_2)
9: **end if**

In Zeile 1 von Algorithmus 4 wird überprüft, ob es ein blockierendes Literal gibt. Falls ja, dann muss es nach Satz 3 ein existentielles Literal l in K_1 geben, das ein höheres Level als alle anderen Literale in K_1 hat und dessen Negation als Einheitsliteral zugewiesen wurde. Auf dieses Literal l greifen wir mittels der Funktion ERHALTELITERALMITHÖCHSTENLEVELINKLAUSEL zu. Um dessen zugehörige Einheitsklausel K_E zu finden, benötigen wir die Belegung θ', in der alle Literale sind, die vor l_E zugewiesen wurden (Zeile 3). Des Literals l entledigen wir uns anschließend mithilfe von K_E in Zeile 5.

Wir haben nun alle benötigten Grundlagen, um Q-DLL um KResolution zu erweitern. Dazu weisen wir jedem unerfüllten Blatt eine falsifizierte Klausel nach Definition 11 aus der ursprüngliche Matrix zu. Betrachten wir einen inneren Knoten p_θ im Q-DLL-Suchbaum der QBF ψ. Der Knoten p_θ besitzt die partielle Belegung θ und ist der Entscheidungsknoten des Literals l, er besitzt also die beiden Kinderknoten $p_{\theta\cup\{l\}}$ und $p_{\theta\cup\{\sim l\}}$. Nehmen wir an, dem Knoten $p_{\theta\cup\{l\}}$ sei die falsifizierte Klausel K_l und dem Knoten $p_{\theta\cup\{\sim l\}}$ die falsifizierte Klausel $K_{\sim l}$ zugewiesen. Nach welchen Regeln können wir nun dem Knoten p_θ eine unter der Belegung θ falsifizierte Klausel K zuweisen?

K1: Sei $\sim l \notin K_l$:
Da K_l eine falsifizierte Klausel unter der Belegung $\theta \cup \{l\}$ ist und $\sim l \notin K_l$, muss K_l auch eine falsifizierte Klausel unter der Belegung θ sein. Wir weisen also dem Knoten p_θ die Klausel K_l zu.
(Analog: Wenn $l \notin K_{\sim l}$, dann ist $K := K_{\sim l}$ eine gültige Zuweisung.)

K2: Sei $\sim l \in K_l$ und $l \in X_\forall$:
In diesem Fall müssen wir beachten, dass für alle universellen Literale l' in einer falsifizierten Klausel K' unter der Belegung θ' gelten muss, dass $l' \notin \theta'$. Für das Literal $\sim l$ in der Klausel K_l gilt diese Bedingung sowohl unter der Belegung $\theta\cup\{l\}$, als auch unter der Belegung θ. Somit ist es erneut ausreichend, dem Knoten p_θ die Klausel K_l zu zuweisen.

K3: Sei $l \in X_\exists$ und l monoton in $\psi[\theta]$:
Nach den obigen Bemerkungen kann $\sim l$ nicht in der falsifizierten Klausel K_l enthalten sein. Damit behandeln wir diesen Fall analog zu Fall K1 und weisen p_θ die Klausel K_l zu.

K4: Sei $\sim l \in K_l$, $l \in K_{\sim l}$, $l \in X_\exists$ und sei die Belegung von l nicht impliziert:
Wir weisen dem Knoten p_θ den Resolventen der falsifizierten Klauseln seiner Kinderknoten zu, also $K :=$ ERWEITERTEKRESOLUTION$(K_l, K_{\sim l})$. Damit ist sichergestellt, dass K ebenfalls unter der Belegung θ falsifiziert ist, da $K \subseteq (K_l \cup K_{\sim l}) \setminus \{l, \sim l\}$ und $K_l \setminus \{\sim l\}$ (bzw. $K_{\sim l} \setminus \{l\}$) unter der Belegung θ falsifiziert ist.

K5: Sei $\sim l \in K_l$, $l \in X_\exists$ und l ist Element einer Einheitsklausel $K'[\theta] \in \psi[\theta]$:
Sei $K'[\theta] = \{l, l_1, \ldots, l_n\}$ eine Einheitsklausel nach Definition 6. Die Belegung $\theta \cup \{\sim l\}$ führt zwangsläufig zu einer Falsifikation der Klausel K'. Wir können also direkt dem Knoten $p_{\theta \cup \{\sim l\}}$ die Klausel $K_{\sim l} := min(K')$ zuweisen. Um sich nun des Literals $\sim l$ der Klausel K_l zu entledigen, resolvieren wir K_l mit $K_{\sim l}$ und weisen den Resolventen dem Knoten p_θ zu, also $K :=$ ERWEITERTEKRESOLUTION$(K_l, K_{\sim l})$. Der Resolvent K ist eine falsifizierte Klausel (vgl. K4).

Hinweis: Zu beachten ist, dass $l \notin K_l$ und $\sim l \notin K_{\sim l}$, da beide Klauseln unter den Belegungen $\theta \cup \{l\}$ bzw. $\theta \cup \{\sim l\}$ falsifiziert sind.

Damit haben wir Q-DLL um ein widerlegungsvollständiges Resolutionsverfahren erweitert, d. h. ein so modifizierter Algorithmus liefert genau dann *Falsch* als Ausgabe, wenn er aus den falsifizierten Blättern die leere Klausel $\{\}$ ableiten konnte ([GNT06, S. 394]).

Beispiel 5. *Betrachten wir erneut QBF (3.2). In Abbildung 3.4 ist der* Q-DLL-*Suchbaum von QBF (3.2) eingezeichnet. Jedem unerfüllten Blatt wurde eine falsifizierte Klausel zugewiesen. Mit den Regeln K1 - K5 konnte dadurch jedem unerfüllten Knoten eine falsifizierte Klausel zugewiesen werden. Betrachten wir die falsifizierte Klausel* $K = \{\neg y_1, \neg x_2\}$ *an dem unerfüllten Blatt mit der Belegung* $\theta = (x_1, y_1, x_2)$*. Das Literal, welches an dem zugehörigen Zweig entschieden wurde, ist* x_2*. Da* $x_2 \in X_\exists$, $\neg x_2 \in K$ *und* x_2 *ein Einheitsliteral ist, müssen wir Regel K5 anwenden. Die zugehörige Einheitsklausel ist* $K_E = \{\neg y_1, x_2\}$*. Dem Knoten mit der partiellen Belegung* (x_1, y_1) *weisen wir also die Klausel* $min((K \cup K_E) \setminus \{x_2, \neg x_2\}) = min((\{\neg y_1, \neg x_2\} \cup \{\neg y_1, x_2\}) \setminus \{x_2, \neg x_2\}) = \{\}$ *zu. Da wir nun bereits die leere Klausel abgeleitet haben, wird an allen Elternknoten die Regel K1 angewendet, d. h. allen Elternknoten wird ebenfalls die leere Klausel zugewiesen. Durch KResolution mit falsifizierten Klauseln wäre somit bereits nach der Untersuchung des linken Teilbaums klar, dass QBF (3.2) unerfüllbar ist.*

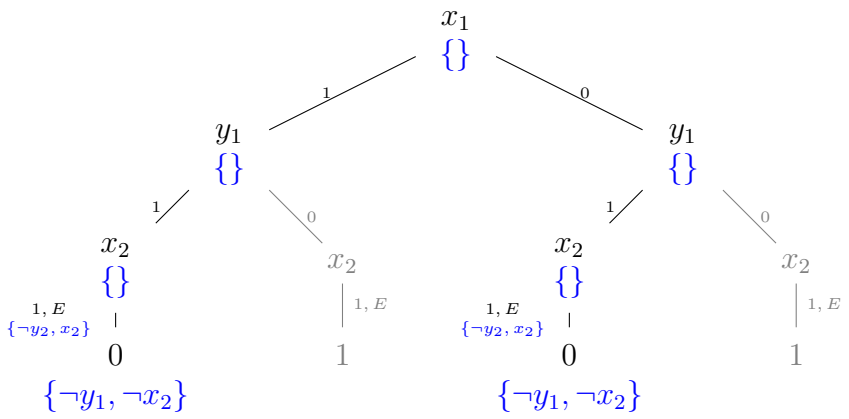


Abbildung 3.4.: Der Suchbaum von Q-DLL mit QBF (3.2) als Eingabeformel. In blau ist eine Resolutionswiderlegung eingezeichnet. Die grauen Pfade werden von Q-DLL übersprungen.

Nachdem wir uns nun ausgiebig damit beschäftigt haben, wie wir mittels KResolution im Zusammenspiel mit Q-DLL die Unerfüllbarkeit einer QBF (oder eines Teil-Suchbaums) zeigen können, wollen wir uns nun damit beschäftigen, wie wir die Erfüllbarkeit einer QBF (oder eines Teil-Suchbaums) mittels MResolution und Q-DLL zeigen können. Dazu gehen wir analog vor und überlegen uns zuerst, welche Monome wir erfüllten Blättern zuweisen.

Definition 12. *Ein Monom M heißt verifiziertes Monom unter der Belegung θ, wenn es minimal und nicht widersprüchlich ist und für alle Literale $l \in M$ gilt:*

- *falls $l \in X_\forall \Rightarrow l \in \theta$ und l wurde nicht als monoton zugewiesen und*
- *falls $l \in X_\exists \Rightarrow \sim l \notin \theta$.*

Die interessante Frage ist nun, ob an jedem erfüllten Blatt mittels Modellgenerierung ein Überdeckungsmonom erzeugt werden kann, welches immer auch ein verifiziertes Monom nach Definition 12 ist. Die Antwort lautet Ja. Sei ψ eine QBF mit den Klauseln $K_1, \ldots, K_n$ und θ eine erfüllende Belegung für ψ. Da θ eine erfüllende Belegung ist, muss für θ die Bedingung gelten, dass $\theta \cap K_i \neq \emptyset$ für alle $i = 1, \ldots, n$. Somit resultiert bereits die Umwandlung von θ in ein Monom M in einem gültigen Überdeckungsmonom, welches für alle $l \in M$ die Bedingungen $l \in \theta$ für $l \in X_\forall$ und $\sim l \notin \theta$ für $l \in X_\exists$ (da $l \in \theta$ für $l \in X_\exists$) aus Definition 12 erfüllt. Zusätzlich fordert Definition 12 aber noch die Einschränkung, dass M keine universellen Literale enthält, die als monoton zugewiesen wurden. Allerdings muss M auch nach der Entfernung aller monotonen universellen Literale ein Überdeckungsmonom sein, denn ein universelles Literal l ist

nur dann monoton in einer QBF $\psi[\theta']$, wenn $l \notin \psi[\theta']$. D. h. alle Klauseln $K \in \psi$ mit $l \in \psi$ müssen bereits durch θ' erfüllt sein, also $\theta' \cap K \neq \emptyset$. Da $\theta' \subseteq \theta$ gilt auch $(\theta \setminus \{l\}) \cap K_i \neq \emptyset$ für alle $i = 1, \ldots, n$.

Doch wozu wird die Einschränkung benötigt, dass M keine monotonen universellen Literale enthält? Der Grund ist, dass implizierte Literale in verifizierten Monomen M_1 und M_2 erneut zu einer Blockierung der MResolution führen können. Denn die MResolution von verifizierten Monomen kann nur durch unbelegte existentielle Literale geblockt werden (Beweis analog zu dem von Satz 3). Da M_1 und M_2 minimal sind, kann es ein unbelegtes existentielles Literal l nur dann geben, wenn es zwei implizierte universelle Literale $l_1 \in M_1$ und $l_2 \in M_2$ gibt mit $lev(l_j) > lev(l)$ für $j \in \{1, 2\}$. Bislang haben wir aber lediglich eine Möglichkeit kennen gelernt, um die Belegung von universellen Literalen zu implizieren, nämlich durch monotone Literale. Da solche Literale jedoch nicht Teil eines verifizierten Monoms sind, können Blockierungen bei der MResolution mit verifizierten Monomen nicht auftreten. Genau genommen könnten wir die zweite Einschränkung für Literale l eines verifizierten Monoms von Definition 12 also verschärfen zu $l \in \theta$, falls $l \in X_\exists$, da unbelegte existentielle Literale nicht in verifizierten Monomen auftreten können. Allerdings müssten wir dann eine Monom-Resolution auch für existentielle Literale einführen, wie wir später in der Resolutionsregel M2 sehen werden. Außerdem werden wir in Abschnitt 3.4 noch eine weitere Implikation für universelle Literale einführen.

Wir weisen nun also jedem erfüllten Blatt im Q-DLL-Suchbaum ein verifiziertes Monom zu, welches mittels Modellgenerierung erzeugt wurde. Nach welchen Regeln können wir nun eine baumartige Resolutionsableitung erstellen? Betrachten wir dazu erneut den inneren Knoten p_θ im Q-DLL-Suchbaum der QBF ψ mit den Kinderknoten $p_{\theta \cup \{l\}}$ und $p_{\theta \cup \{\sim l\}}$. Nehmen wir an, dem Knoten $p_{\theta \cup \{l\}}$ sei das verifizierte Monom M_l und dem Knoten $p_{\theta \cup \{\sim l\}}$ das verifizierte Monom $M_{\sim l}$ zugewiesen. Wir nutzen die folgenden Auswahlregeln, um p_θ ein verifiziertes Monom M zu zuweisen.

M1: Sei $l \notin M_l$:
Da M_l ein verifiziertes Monom unter der Belegung $\theta \cup \{l\}$ ist und $l \notin M_l$, muss M_l auch ein verifiziertes Monom unter der Belegung θ sein. Wir weisen also dem Knoten p_θ das Monom M_l zu.
(Analog: Wenn $\sim l \notin M_{\sim l}$, dann ist $M := M_{\sim l}$ eine gültige Zuweisung.)

M2: Sei $l \in M_l$ und $l \in X_\exists$:
In diesem Fall müssen wir beachten, dass für alle existentiellen Literale l' in einem verifiziertem Monom M' unter der Belegung θ' gelten muss, dass $\sim l' \notin \theta'$. Für das Literal l in der Klausel K_l gilt diese Bedingung sowohl unter der Belegung $\theta \cup \{l\}$, als auch unter der Belegung θ. Somit ist es erneut ausreichend, dem Knoten p_θ

das Monom M_l zu zuweisen.

M3: Sei $l \in X_\forall$ und l monoton in $\psi[\theta]$:
Nach Definition 12 kann l nicht im verifiziertem Monom M_l enthalten sein. Damit behandeln wir diesen Fall analog zu Fall M1 und weisen p_θ das Monom M_l zu.

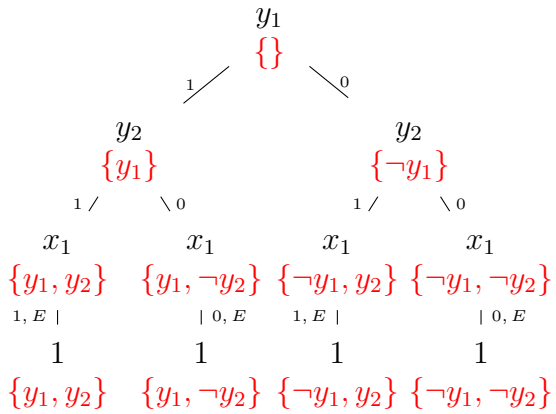
M4: Sei $l \in M_l$, $\sim l \in M_{\sim l}$, $l \in X_\forall$ und sei die Belegung von l nicht impliziert:
Wir weisen dem Knoten p_θ den Resolventen der verifizierten Monome seiner Kinderknoten zu, also $M := \text{MRESOLUTION}(M_l, M_{\sim l})$. Damit ist sichergestellt, dass M ebenfalls unter der Belegung θ ein verifiziertes Monom ist, da $M \subseteq (M_l \cup M_{\sim l}) \setminus \{l, \sim l\}$ und $M_l \setminus \{l\}$ (bzw. $M_{\sim l} \setminus \{\sim l\}$) unter der Belegung θ ein verifiziertes Monom ist.

Hinweis: Zu beachten ist, dass $\sim l \notin M_l$ und $l \notin M_{\sim l}$, da beide Monome unter den Belegungen $\theta \cup \{l\}$ bzw. $\theta \cup \{\sim l\}$ verifizierte Monome sind.

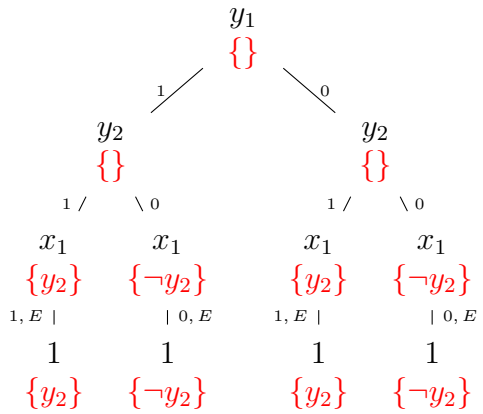
In diesen Auswahlregeln wird noch einmal die Schwierigkeit von MResolution mittels Modellgenerierung gegenüber KResolution in Q-DLL deutlich. Die Effizienz des Backjumpings hängt maßgeblich von den verifizierten Monomen ab, die den erfüllten Blättern zugewiesen wurden. Sollte ein solches verifiziertes Monom M einfach aus der Minimalform der jeweiligen Belegung des Blattes gewonnen werden, indem monotone universelle Literale entfernt werden, dann sind alle universellen Literale, die nicht impliziert wurden, von der Wurzel bis zum Blatt in dem Monom enthalten. Die Auswahlregel M1 würde also keine Anwendung für solche Literale finden. Stattdessen müsste für diese Literale immer Auswahlregel M4 angewendet werden, welche die Untersuchung beider Zweige des betrachteten Knotens verlangt. Backjumping durch MResolution findet in diesem Fall also keine Anwendung. Eine Möglichkeit diesem Problem zu begegnen, ist M erneut aus der Belegung zu erzeugen und es vor der Umwandlung in ein verifiziertes Monom zu reduzieren, so dass alle universellen Literale $l \in M$ entfernt werden, falls $M \setminus \{l\}$ noch immer ein gültiges Überdeckungsmonom ist ([GNT06, S. 401]).

Damit haben wir Q-DLL schließlich auch um ein erfüllbarkeitsvollständiges Resolutionsverfahren erweitert, d. h. ein so modifizierter Algorithmus liefert genau dann *Wahr* als Ausgabe, wenn er aus den verifizierten Blättern das leere Monom {} ableiten konnte ([GNT06, S. 394]).

Beispiel 6. *Betrachten wir erneut QBF (3.3). In Abbildung 3.5 ist der* Q-DLL-*Suchbaum von QBF (3.3) mit eingezeichneter MResolution für zwei unterschiedliche Arten der Modellgenerierung dargestellt. In Abbildung 3.5(a) wurde jedem Blatt ein verifiziertes Monom zugewiesen, welches aus der jeweiligen Belegung θ gewonnen wurde, indem diese Literalmenge in Minimalform überführt wurde (implizierte monotone Literale sind nicht enthalten). Wie zu erkennen ist, könnte ein so durch MResoluti-*



(a) Modellgenerierung durch Umwandlung der erfüllenden Belegung in ein verifiziertes Monom



(b) Modellgenerierung durch Umwandlung einer reduzierten erfüllenden Belegung in ein verifiziertes Monom

Abbildung 3.5.: Q-DLL-Suchbäume mit QBF (3.3) als Eingabeformel. Jedem Knoten der Suchbäume sind verifizierte Monome zugewiesen (in rot).

on mit Modellgenerierung erweiterter Q-DLL *Algorithmus keinen Vorteil gegenüber gewöhnlichen* Q-DLL *liefern, da alle eingezeichneten Pfade untersucht werden müssen. In Abbildung 3.5(b) wurden die den Blättern zugewiesenen Monome zusätzlich reduziert. Betrachten wir dazu das Blatt mit der erfüllenden Belegung* $\theta = \{y_1, y_2, x_1\}$. *Offensichtlich bildet diese Literalmenge ein Überdeckungsmonom* M*:*

$$\{\neg y_1, \neg y_2, x_1\}, \{\neg y_1, y_2, \neg x_1\}, \{y_1, \neg y_2, x_1\}, \{y_1, y_2, \neg x_1\}$$

In diesem Überdeckungsmonom ist das Literal y_1 *jedoch redundant, denn das Monom* $\{y_2, x_2\}$ *ist ebenfalls ein Überdeckungsmonom:*

$$\{\neg y_1, \neg y_2, x_1\}, \{\neg y_1, y_2, \neg x_1\}, \{y_1, \neg y_2, x_1\}, \{y_1, y_2, \neg x_1\}$$

Das Literal y_2 *in* M *ist wiederum nicht redundant, denn es gibt kein anderes Literal in* M*, das die Klausel* $\{\neg y_1, y_2, \neg x_1\}$ *erfüllt.*

Durch Entfernen dieser Redundanzen ist es uns somit möglich, den Blättern verifizierte Monome zuzuweisen, die ein späteres Backjumping erlauben. Im Fall von Abbildung 3.5(b) ist Erfüllbarkeit bereits nach der Untersuchung des linken Teilbaums bekannt. Am Wurzelknoten kann Regel M1 angewendet werden und somit die Untersuchung des rechten Teilbaums ausgelassen werden.

Algorithmus 5 Q-DLL-BJ

Eingabe: QBF ψ in minimaler Form, Belegung θ // Aufruf durch Q-DLL-BJ(ψ, $\emptyset$)
Ausgabe: Monom M, falls $\psi[\theta]$ erfüllbar, sonst Klausel K

1: **if** θ erfüllt ψ **then return** GENERIEREMODELL(ψ, θ) **end if**
2: **if** Klausel $K \in \psi$ widerlegt $\psi[\theta]$ **then return** K **end if**
3: **if** Literal l ist Einheitsliteral in $\psi[\theta]$ **then**
4: $K \leftarrow$ ERHALTEZUGEHÖRIGEEINHEITSKLAUSEL(ψ, θ, l)
5: $T \leftarrow$ Q-DLL-BJ(ψ, $\theta \cup \{l\}$)
6: **if** T ist ein Monom **or** $\sim l \notin T$ **then return** T **end if**
7: **return** ERWEITERTEKRESOLUTION(ψ, θ, T, K)
8: **end if**
9: **if** Literal l ist monoton in $\psi[\theta]$ **then return** Q-DLL-BJ(ψ, $\theta \cup \{l\}$) **end if**
10: $l \leftarrow$ ERHALTENÄCHSTESLITERAL(ψ, θ)
11: $T_1 \leftarrow$ Q-DLL-BJ(ψ, $\theta \cup \{l\}$)
12: **if** $l \in X_\exists$ **and** (T_1 ist ein Monom **or** $\sim l \notin T_1$) **then return** T_1 **end if**
13: **if** $l \in X_\forall$ **and** (T_1 ist eine Klausel **or** $l \notin T_1$) **then return** T_1 **end if**
14: $T_2 \leftarrow$ Q-DLL-BJ(ψ, $\theta \cup \{\sim l\}$)
15: **if** $l \in X_\exists$ **and** (T_2 ist ein Monom **or** $l \notin T_2$) **then return** T_2 **end if**
16: **if** $l \in X_\forall$ **and** (T_2 ist eine Klausel **or** $\sim l \notin T_2$) **then return** T_2 **end if**
17: **if** $l \in X_\exists$ **then return** ERWEITERTEKRESOLUTION(ψ, θ, T_1, T_2) **end if**
18: **return** MRESOLUTION(T_1, T_2)

Die bisher gesammelten Erkenntnisse sind in Algorithmus 5 ([GNT06, S. 393]) zusammengetragen. Dieser erweitert Algorithmus 1 um KResolution und MResolution. Es wird also jedem Knoten im Suchbaum eine falsifizierte Klausel oder ein verifiziertes Monom unter Verwendung der Resolutionsregeln K1 - K5 und M1 - M4 zugewiesen. Wir wollen abschließend kurz die Anwendung dieser Regeln in Algorithmus 5 anhand der Implikationen verdeutlichen und zusätzlich die wesentlichen Verbesserungen dieses Algorithmus im Vergleich zu Q-DLL hervorheben. Betrachten wir zuerst die Umsetzung der Einheitsresolution. In Zeile 5 erhalten wir den Term T des direkten Kinderknotens, welcher aus der erzwungenen Belegung $\theta \cup \{l\}$ resultiert. Den Begriff *Term* verwenden wir als Oberbegriff für Disjunktionsterme (Klauseln) und Konjunktionsterme (Monome). Wenn T ein Monom ist, dann wenden wir Regel M2 (bzw. M1, falls $l \notin T$) an und weisen dem aktuellen Knoten das Monom T zu. Ist T jedoch eine Klausel, so müssen wir zwei Fälle unterscheiden. Im ersten Fall ist $\sim l \notin T$ und wir wenden K1 an und weisen somit dem aktuellen Knoten erneut T zu. Im zweiten Fall, also $\sim l \in T$, müssen

wir Regel K5 anwenden. D. h. wir weisen dem aktuellen Knoten den Resolventen aus T und der entsprechenden Einheitsklausel K zu.

Da monotone Literale nicht Bestandteil eines verifizierten Monoms oder einer falsifizierten Klausel sind, können wir in Zeile 9 dem aktuellen Knoten den Term zuweisen, der mit dem Kinderknoten assoziiert ist (Regel M3 oder K3).

Sind keine Implikationen möglich, so verzweigen wir anhand eines unbelegten Literals l unter Beachtung der Quantifizierungsreihenfolge. An dieser Stelle befindet sich der wesentliche Unterschied zu Algorithmus 1. Angenommen l ist existentiell. Wenn der Zweig $\theta \cup \{l\}$ unerfüllbar ist, dann untersucht Algorithmus 1 immer auch den Zweig $\theta \cup \{\sim l\}$. Algorithmus 5 tut dies nur, wenn die Unerfüllbarkeit des Zweiges $\theta \cup \{l\}$ abhängig von der Belegung von l ist, also wenn $\sim l \in T$. Ansonsten springen wir direkt zu dem Elternknoten (Backjumping) und ignorieren den Zweig $\theta \cup \{\sim l\}$. Der Fall, dass l ein universelles Literal ist und der Zweig $\theta \cup \{l\}$ erfüllbar ist, wird analog behandelt. Wir untersuchen den Zweig $\theta \cup \{\sim l\}$ nur, wenn die Erfüllbarkeit des Zweiges $\theta \cup \{l\}$ von der Belegung von l abhängig ist. Ansonsten springen wir zum Elternknoten zurück.

3.4. Lernen von falsifizierten Klauseln und verifizierten Monomen

In Abschnitt 3.3 hatten wir mühselig mittels Klausel- und Monom-Resolution jedem Knoten im Q-DLL-Suchbaum einen Term zugewiesen. Doch die Informationen, die uns dieser Term über die Erfüllbarkeit bzw. Unerfüllbarkeit der Eingabeformel oder Teile dieser verriet, nutzten wir nur an genau dieser Stelle im Suchbaum und verwarfen sie anschließend. Dies wollen wir nun ändern. Wir werden die Eingabeformel ψ um die abgeleiteten Terme erweitern und diese nutzen, um den Suchbaum weiter zu beschneiden. Dabei folgen wir im Wesentlichen den Überlegungen aus [GNT06, S. 394ff.].

Beispiel 7. *Betrachten wir die folgende QBF ψ:*

$$\begin{aligned}\forall y_1 \exists x_1 \exists x_2 \exists x_3 \ \{&\{y_1, x_1, x_3\}, \{\neg y_1, x_1, x_2\}, \{\neg x_1, x_2, x_3\}, \\ &\{\neg x_1, x_2, \neg x_3\}, \{\neg x_1, \neg x_2, x_3\}, \{\neg x_1, \neg x_2, \neg x_3\}\}\end{aligned} \tag{3.4}$$

Der Suchbaum von Algorithmus 5 mit ψ als Eingabeformel ist in Abbildung 3.6 dargestellt. Dieser Suchbaum enthält zwei unerfüllbare Teilbäume Π_1 und Π_2, deren Wurzel bei den partiellen Belegungen $\{y_1, x_1\}$ und $\{\neg y_1, x_1\}$ liegt. Die Resolutionsableitung von Π_1 endet mit der falsifizierten Klausel $\{\neg x_1\}$. D. h. die Unerfüllbarkeit von Π_1 ist nur

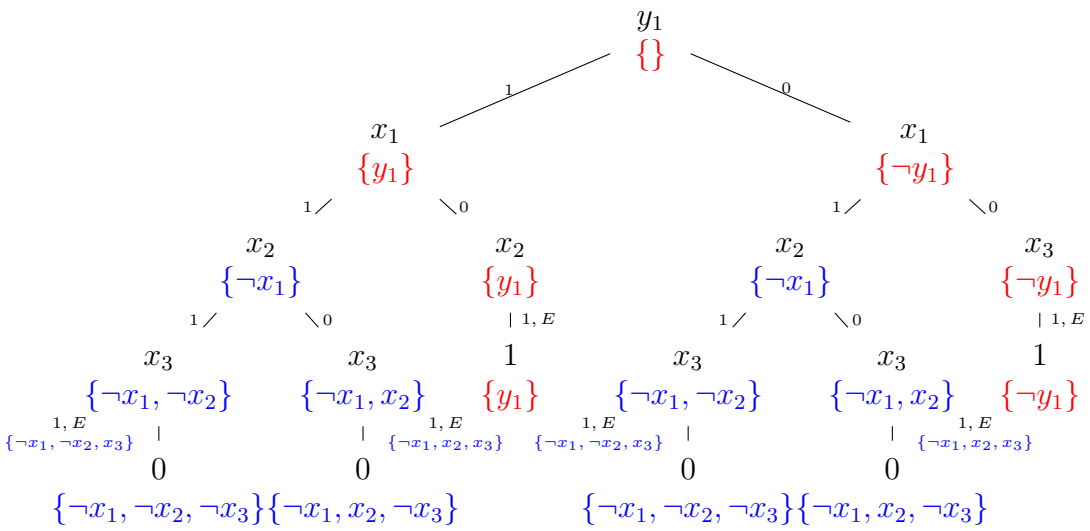


Abbildung 3.6.: Der Suchbaum von Q-DLL-BJ mit QBF (3.4) als Eingabeformel.

abhängig von der Belegung $val(x_1) = 1$ *und unabhängig von allen anderen bereits belegten Variablen (in diesem Fall nur* y_1*). Wenn wir uns die falsifizierte Klausel* $\{\neg x_1\}$ *merken würden, könnten wir den Baum* Π_2 *überspringen, da dieser erneut durch eine Belegung von* x_1 *mit* $val(x_1) = 1$ *erzeugt wird.*

Zunächst gilt es die Frage zu beantworten, wie wir eine QBF ψ um die abgeleiteten Terme erweitern können, ohne die Erfüllbarkeit von ψ zu beeinflussen. Dazu fügen wir die abgeleiteten falsifizierten Klauseln konjunktiv zur Matrix von ψ hinzu. Dies beeinflusst die Erfüllbarkeit von ψ nicht, da eine solche hinzugefügte Klausel nur dann unter einer Belegung θ' falsifiziert, wenn nicht für alle aufgrund der Quantifizierung benötigten Belegungen $\theta \supseteq \theta'$ gilt, dass alle Klauseln $\mathbf{K}$ aus der ursprünglichen Matrix, aus denen K abgeleitet wurde, erfüllt sind. Genauso muss K erfüllt sein, wenn alle Klauseln in $\mathbf{K}$ erfüllt sind und diese nicht durch eine Belegung $\tilde{\theta} = (l_1, \ldots, l_k, l, \ldots)$ mit $l \in X_\forall$ erfüllt sind, bei der es keine Belegung $\tilde{\theta}' = (l_1, \ldots, l_k, \neg l, \ldots)$ gibt mit $\theta' \models \mathbf{K}$ (siehe Anhang B, Satz 4). Analog können wir aus der Matrix von ψ abgeleitete verifizierte Monome disjunktiv mit der Matrix verknüpfen. Denn falls ein solches Monom erfüllt ist durch eine Belegung θ', dann gibt es auch eine Menge von aufgrund der Quantifizierung benötigten Belegungen $\theta \supseteq \theta'$, die die Matrix von ψ erfüllen (siehe Anhang B, Satz 5). Diese Überlegungen führen uns also zu einer erweiterten Definition von QBFs ([ZM02, S. 203], [GNT06, S. 395]).

Definition 13. *Sei* ψ *eine QBF*

$$Q_1 X_1 \cdots \exists X_i \; \Phi$$

und Φ *die Matrix von* ψ*. Eine Erweiterte QBF (EQBF)* ψ^* *von* ψ *ist eine QBF in der Form:*

$$Q_1 X_1 \cdots \exists X_i \ ((\Phi \wedge \Psi) \vee \Omega)$$

wobei

- Ψ *eine Menge von falsifizierten Klauseln ist, die aus der Matrix* ψ *abgeleitet wurden und*
- Ω *eine Menge von verifizierten Monomen ist, die mittels Modellgenerierung aus* ψ *abgeleitet wurden.*

Das Ziel dieses Abschnitts ist es somit, eine gegebene QBF in eine EQBF umzuwandeln, mit dem Wunsch, die Unerfüllbarkeit einer partiellen Belegung schneller zu erkennen durch die größere Anzahl an Klauseln, die falsifizieren können, respektive deren Erfüllbarkeit frühzeitig zu erkennen durch die Verifikation eines hinzugefügten Monoms. Doch die Erweiterung einer gewöhnlichen QBF in PKNF in Definition 13 zu einer EQBF verlangt nach einer erneuten Prüfung der Definitionen 6 und 7. Einheitsresolution mit existentiellen Literalen ist tatsächlich leicht auf EQBFs übertragbar. Angenommen eine minimale Einheitsklausel $\{l\}$ ist in $\Phi \cup \Psi$ enthalten, dann führt eine Belegung $val(l) = 0$ zur Falsifikation dieser Klausel. Die Belegung des Literals l wird damit erneut durch Einheitsklauseln erzwungen. Zusätzlich haben wir nun aber auch die Möglichkeit, Einheitsresolution auf universelle Literale zu erweitern. Angenommen in Ω gibt es ein minimales *Einheitsmonom* $\{l\}$, dann kann eine Belegung mit $val(l) = 1$ zu eine erfüllende Belegung erweitert werden. Da dieser Zweig nun offensichtlich erfüllbar ist, kann direkt der Zweig mit $val(l) = 0$ untersucht werden.

Definition 14. *Ein Term* $T = \{l_1, \ldots, l_n\}$ *mit* $lev(l_1) < lev(l_i)$ *für alle* $i \geq 2$ *heißt Einheitsterm, wenn*

- $l := l_1 \in X_\exists$*,* $l_2, \ldots, l_n \in X_\forall$ *und* $T \in \Phi \cup \Psi$ *oder*
- $\sim l := l_1 \in X_\forall$*,* $l_2, \ldots, l_n \in X_\exists$ *und* $T \in \Omega$*.*

Das Literal l *ist das zugehörige Einheitsliteral.*

Bei monotonen Literalen ist die Situation leicht komplizierter ([GNT04]), da ein monotones Literal nach Definition 7 die Korrektheit des von uns in Abschnitt 3.3 erarbeiteten Resolutionssystems zunichte macht. Betrachten wir z.B ein existentielles Literal l, welches nach Definition 7 monoton in $\Phi[\theta']$ ist. Es könnte somit durchaus eine Klausel $K \in \Psi$ geben, sodass $\sim l \in K[\theta']$. Wenn eine Belegung $\theta \supseteq \theta' \cup \{l\}$ nun zu einer Falsifikation der Klausel K führt, dann enthält die falsifizierte Klausel ein Literal, dessen

Negation als monoton zugewiesen wird. Somit wäre Satz 3 nicht mehr korrekt und die Resolution kann blockieren. In [GNT06, S. 396] werden zwei Lösungen für dieses Problem vorgeschlagen. Eine Möglichkeit wäre alle Klauseln K in Ψ (resp. Monome M in Ω) vor der Zuweisung eines monotonen Literals l zu löschen, falls $\sim l \in K$ (resp. $l \in M$). Doch wir verwenden die zweite Lösung und erweitern unsere bisherige Definition von monotonen Literalen, um sicherzustellen, dass falsifizierte Klauseln bzw. verifizierte Monome keine monotonen Literale oder deren Negation enthalten.

Definition 15. *Ein Literal l heißt monoton in einer EQBF, falls*

- $l \in X_\exists$ *und* $\sim l \notin \Phi \cup \Psi$ *oder*
- $l \in X_\forall$ *und* $l \notin \Phi \cup \Omega$.

Durch die Einführung der Einheitsresolution anhand von universellen Literalen besteht nun die Möglichkeit, dass die MResolution von zwei verifizierten Monomen durch unbelegte existentielle Literale geblockt wird (vgl. Abschnitt 3.3). Wir lösen das Problem analog zu dem Fall, dass eine KResolution von unbelegten universellen Literalen geblockt wird. Seien $M_1 = \{l_1, \ldots, l_k, \ldots, l_E\}$ und M_2 unter der Belegung θ erfüllte verifizierte Monome mit $lev(l_i) \leq lev(l_j)$ für $i \leq j$, wobei l_k ein unbelegtes existentielles Literal ist mit $\sim l_k \in M_2$ und $l_E \in X_\forall$ mit $lev(l_i) \leq lev(l_E)$. Da l_k ein unbelegtes Literal mit einem geringeren Level als l_E ist, muss l_E impliziert worden sein. Nun sind monotone Literale aber nicht in verifizierten Monomen enthalten, also ist l_E ein Einheitsliteral und damit ist $\theta = \theta' \cup \{l_E\} \cup \theta''$, wobei in θ' alle Literale sind, die vor l_E zugewiesen wurden. Sei M_E das zugehörige Einheitsmonom. Wir wissen, dass M_E unter der Belegung $\theta' \cup \{\sim l_E\}$ ein verifiziertes Monom sein muss. Außerdem wissen wir, dass es in M_E keine unbelegten existentiellen Literale l' mit $lev(l') \leq lev(l_E)$ gibt (vgl. Def. 14), d. h. es gibt keine blockierenden Literale bei einer MResolution von M_1 und $min(M_E)$. Da $M_E \setminus \{\sim l_E\}$ bereits unter der Belegung θ' verifiziert ist, ist der Resolvent $M_1' := min((M_1 \cup min(M_E)) \setminus \{l_E, \sim l_E\})$ unter θ ebenfalls ein verifiziertes Monom. Falls es in M_1' noch immer universelle Literale l_E' mit $lev(l_k) < lev(l_E')$ gibt, dann entledigen wir uns dieser Literale l_E' auf derselben Weise, bis alle solche Literale entfernt sind und das blockierende Literal l_k aufgrund der Minimalform ebenfalls herausfällt. Dieses Verfahren ist in Algorithmus 6 dargestellt ([GNT06, S. 396]).

Algorithmus 6 ERWEITERTEMRESOLUTION

Eingabe: QBF ψ, Belegung θ, Monom M_1, Monom M_2
Ausgabe: Resolvent M von M_1 und M_2 unter Berücksichtigung blockierender Literale

```
                    // M1 und M2 müssen unter der Belegung θ verifizierte Monome sein
1: if Es existiert l ∈ M1 mit l ∈ X∃ und ∼l ∈ M2 then
2:     l ← erhalteLiteralMitHöchstenLevelInMonom(M1)
```

3: $\theta' \leftarrow$ erhalteBelegungspäfix(θ, l)
4: $M_E \leftarrow$ erhalteZugehörigesEinheitsmonom(ψ, θ', l)
5: $M_1' \leftarrow$ MResolution(M_1, M_E)
6: **return** erweiterteMResolution(ψ, θ, M_1', M_2)
7: **else**
8: **return** MResolution(M_1, M_2)
9: **end if**

Die Funktionsweise von Algorithmus 6 ist analog zu der von Algorithmus 4. Die vorkommenden Funktionen erfüllen demzufolge auch gleichartige Aufgaben (vgl. Abschnitt 3.3).

Bevor wir Algorithmus 5 jedoch um die Verwendung von EQBFs erweitern, müssen wir sicherstellen, dass unsere Auswahlregeln zur Assoziation eines Terms mit einem inneren Knoten des Q-DLL-Suchbaums noch immer korrekt sind. Für die Auswahlregeln K1 - K5 trifft dies tatsächlich zu. Wir können jedem unerfüllten Blatt eine falsifizierte Klausel aus $\Phi \cup \Psi$ zuweisen und mithilfe von K1 - K5 auch jedem unerfüllten inneren Knoten. Die Regeln M1 - M4 sind wiederum nicht mehr allgemeingültig, da wir nun eine weitere Möglichkeit kennen, um die Belegung universeller Literale zu implizieren. Dadurch ergibt sich einmal die Besonderheit, dass die Resolution in Regel M4 durch unbelegte existentielle Literale blockiert werden kann. Dementsprechend muss M4 erweitert werden, sodass eine erweiterteMResolution durchgeführt wird. Für den Fall eines inneren Knotens p_θ mit der Belegung θ, der einen Kinderknoten $p_{\theta \cup \{l\}}$ mit dem assoziierten verifizierten Monom M_l und der Belegung $\theta \cup \{l\}$ hat, wobei l ein impliziertes universelles Einheitsliteral ist, haben wir bislang noch keine Resolutionsregel. Dies wollen wir nun ändern und analog zu K5 eine Regel für Knoten einführen, die ihren Kinderknoten durch Zuweisung eines universellen Einheitsliterals erzeugen.

M5: Sei $l \in M_l$, $l \in X_\forall$ und $\sim l$ ist Element eines Einheitsmonoms $M'[\theta] \in \psi[\theta]$:
Sei $M'[\theta] = \{\sim l, l_1, \ldots, l_n\}$ ein Einheitsmonom nach Definition 14. Die Belegung $\theta \cup \{\sim l\}$ führt zwangsläufig zu einer Verifikation des Monoms M'. Wir können also direkt dem Knoten $p_{\theta \cup \{\sim l\}}$ das Monom $M_{\sim l} := min(M')$ zuweisen. Um sich nun des Literals l des Monoms M_l zu entledigen, resolvieren wir M_l mit $M_{\sim l}$ und weisen den Resolventen dem Knoten p_θ zu, also $M :=$ erweiterteMResolution(M_l, $M_{\sim l}$). Der Resolvent M ist ein verifiziertes Monom (vgl. M4).

Nun haben wir also zusätzlich ein vollständiges Verfahren, um das leere Monom $\{\}$ aus einer erfüllbaren EQBF abzuleiten. Dazu weisen wir zunächst jedem erfüllten Blatt ein verifiziertes Monom zu, welches entweder mittels Modellgenerierung aus der aktuellen Belegung gewonnen wird oder aus der Menge Ω stammt. Anschließend assoziieren wir unter Ausnutzung der Regeln M1 - M5 mit jedem erfüllten inneren Knoten ein

verifiziertes Monom, soweit dies möglich ist.

Algorithmus 7 Q-DLL-LN

Eingabe: QBF ψ in minimaler Form, Belegung θ // Aufruf durch Q-DLL-BJ(ψ, $\emptyset$)
Ausgabe: Monom M, falls $\psi[\theta]$ erfüllbar, sonst Klausel K

1: **if** $\theta = \emptyset$ **then** ENTFERNEALLEGELERNTENTERME **end if**
2: **if** Es gibt Monom $M \in \Omega$ mit $min(M[\theta]) = \{\}$ **then return** M **end if**
3: **if** θ erfüllt $\Phi \cup \Psi$ **then return** GENERIEREMODELL(ψ, θ) **end if**
4: **if** Es gibt Klausel $K \in \Phi \cup \Psi$ mit $min(K[\theta]) = \{\}$ **then return** K **end if**
5: **if** Literal l ist Einheitsliteral in $\Omega[\theta] \cup \Phi[\theta] \cup \Psi[\theta]$ **then**
6: $T_E \leftarrow$ ERHALTEZUGEHÖRIGENEINHEITSTERM(ψ, θ, l)
7: $T \leftarrow$ Q-DLL-LN(ψ, $\theta \cup \{l\}$)
8: **if** $l \in X_\exists$ **and** (T ist ein Monom **or** $\sim l \notin T$) **then return** T **end if**
9: **if** $l \in X_\forall$ **and** (T ist eine Klausel **or** $l \notin T$) **then return** T **end if**
10: **if** $l \in X_\exists$ **then** $T \leftarrow$ ERWEITERTEKRESOLUTION(ψ, θ, T, T_E)
11: **else** $T \leftarrow$ ERWEITERTEMRESOLUTION(ψ, θ, T, T_E) **end if**
12: LERNETERM(T, θ)
13: **return** T
14: **end if**
15: **if** Literal l ist monoton in $\Omega[\theta] \cup \Phi[\theta] \cup \Psi[\theta]$ **then return** Q-DLL-LN(ψ, $\theta \cup \{l\}$) **end if**
16: $l \leftarrow$ ERHALTENÄCHSTESLITERAL(ψ, θ)
17: $T_1 \leftarrow$ Q-DLL-LN(ψ, $\theta \cup \{l\}$)
18: **if** $l \in X_\exists$ **and** (T_1 ist ein Monom **or** $\sim l \notin T_1$) **then return** T_1 **end if**
19: **if** $l \in X_\forall$ **and** (T_1 ist eine Klausel **or** $l \notin T_1$) **then return** T_1 **end if**
20: $T_2 \leftarrow$ Q-DLL-LN(ψ, $\theta \cup \{\sim l\}$)
21: **if** $l \in X_\exists$ **and** (T_2 ist ein Monom **or** $l \notin T_2$) **then return** T_2 **end if**
22: **if** $l \in X_\forall$ **and** (T_2 ist eine Klausel **or** $\sim l \notin T_2$) **then return** T_2 **end if**
23: **if** $l \in X_\exists$ **then** $T \leftarrow$ ERWEITERTEKRESOLUTION(ψ, θ, T_1, T_2)
24: **else** $T \leftarrow$ ERWEITERTEMRESOLUTION(ψ, θ, T_1, T_2) **end if**
25: LERNETERM(T, θ)
26: **return** T

Algorithmus 7 zeigt einen erweiterten Q-DLL-Algorithmus, der sowohl Backjumping als auch das Lernen abgeleiteter Klauseln bzw. Monome beherrscht [GNT06, S. 398]. Das Lernen in Algorithmus 7 wird durch Umwandlung der Eingabeformel in eine EQBF realisiert. Dazu gibt es zwei Mengen Ψ und Ω, wie in Definition 13 eingeführt, mit globaler Sichtbarkeit. Diese müssen beim ersten Aufruf der Funktion, also beim Aufruf

mit der leeren Belegung, als leere Mengen initialisiert werden (Zeile 1). Eine EQBF ist erfüllt, wenn alle Klauseln in $\Psi \cup \Phi$ erfüllt sind (Zeile 3) oder ein Monom aus Ω erfüllt ist (Zeile 2). Sie ist wiederum unerfüllt, wenn eine Klausel in $\Psi \cup \Phi$ falsifiziert ist (Zeile 4). Eine weitere Neuerung gegenüber Algorithmus 5 ist die Einführung von Einheitsresolution anhand von universellen Literalen, wobei auch die neu eingeführte Regel M5 in Zeile 11 ihre Anwendung findet.

Immer dann, wenn mit einem inneren Knoten ein Term assoziiert wird, der nicht von einem Kinderknoten übernommen wurde, und somit mittels Resolution gewonnen wurde (Anwendung von K4 oder K5 bzw. M4 oder M5), dann speichern wir diesen Term. Dies geschieht durch die Methode LERNETERM, die ein Monom zu der Menge Ω und eine Klausel zu der Menge Ψ hinzufügt. Tatsächlich speichern wir mittels dieser Strategie im Allgemeinen exponentiell viele Terme, was in den meisten Fällen dazu führen wird, dass der entstandene Berechnungsmehraufwand durch die Miteinbeziehung der Mengen Ψ und Ω den Nutzen überwiegt. Es gibt verschiedene Heuristiken, um diesen Problem zu begegnen ([GNT06, S. 399, 401f.]). Der Einfachheit halber unterstellen wir der Methode LERNETERM eine einfache Heuristik, die periodisch die Mengen Ψ und Ω durchsucht und die Terme löscht, die eine vordefinierte Anzahl an unbelegten Literalen überschreiten. Bei diesem Ansatz gehen wir also davon aus, dass ein Term in Ψ oder Ω, der unter der aktuellen Belegung viele unbelegte Literale enthält, schwer zu falsifizieren bzw. verifizieren ist.

3.5. Ein iterativer Ansatz zur algorithmischen Umsetzung der bisherigen Erkenntnisse

In Abschnitt 3.4 haben wir ein konflikt- bzw. lösungsbasiertes Lernverfahren eingeführt, um den von Q-DLL durchgeführten Erfüllbarkeitstest zu beschleunigen. Im Folgenden wollen wir einen Algorithmus vorstellen, der ein ähnliches Verfahren iterativ umsetzt. Der in diesem Abschnitt behandelte Algorithmus wurde in [ZM02] und [Zha02] vorgestellt.

Bevor wir uns dem Algorithmus ansehen können, benötigen wir jedoch eine genaue Vorstellung, wie das Backtracking in einem iterativen Algorithmus durchgeführt werden kann.

Definition 16. *Im Folgenden weisen wir jedem Literal ein Entscheidungslevel d in Abhängigkeit vom Suchbaum des zugehörigen Lösungsalgorithmus zu. Das Entscheidungslevel d wird mit 0 initialisiert. Jedes Mal, wenn die Methode* ERHALTENÄCHSTESLITERAL *aufgerufen wird, wird d inkrementiert. Das zurückgegebene Literal l be-*

kommt nun den neuen Wert von d als Entscheidungslevel zugewiesen. Die Variable $var(l)$ heißt Entscheidungsvariable des Entscheidungslevels d. Die Entscheidung von l ermöglicht eventuell Implikationen. Alle Literale l', deren Belegung vor dem nächsten Aufruf von ERHALTENÄCHSTESLITERAL *impliziert wurden, erhalten ebenfalls das Entscheidungslevel d mit der Entscheidungsvariable $var(l)$.*

Ein Backtracking innerhalb des Entscheidungslevels d ist nur sinnvoll, wenn alle Literale $(l, l'_1, \ldots, l'_k)$ mit dem Entscheidungslevel d aus der aktuellen Belegung $\tilde{\theta} = (l_1, \ldots, l_n, l, l'_1, \ldots, l'_k)$ entfernt werden und die Belegung der zugehörigen Entscheidungsvariable $var(l)$ revidiert wird. Da die Belegung $(l_1, \ldots, l_n, l)$ noch kein ausreichendes Ergebnis erzielt hat und somit die Untersuchung der Belegung $(l_1, \ldots, l_n, \sim l)$ quasi erzwungen wird, ist es sinnvoll und notwendig dem Literal $\sim l$ das Entscheidungslevel $d - 1$ zu zuweisen. Eine erneute Zuweisung von $var(\sim l)$ als Entscheidungsvariable würde in einer Endlosschleife beim chronologischen Backtracking führen. Eine Möglichkeit unter der Belegung $(l_1, \ldots, l_n)$ die Belegung von $\sim l$ nach dem Backtracking zu erzwingen, ist das Hinzufügen eines Terms, welcher unter der Belegung $(l_1, \ldots, l_n)$ zum Einheitsterm wird und damit die Belegung von $\sim l$ als als Einheitsliteral erzwingt. Dieses Verfahren ist analog auf Backjumping übertragbar, wo im Gegensatz zum Backtracking nicht auf das Entscheidungslevel $d - 1$ zurückgesprungen wird, sondern auf ein beliebiges Entscheidungslevel $d - s$ mit $s > 0$. Um die Assoziation eines Literals mit einem Entscheidungslevel besser verdeutlichen zu können, erweitern wir Definition 2 wie folgt.

Definition 17. *Sei θ eine Belegung nach Definition 2. Mit $\hat{\theta}$ notieren wir im Folgenden eine Menge von Tupeln, wobei für jedes $l \in \theta$ genau ein Tupel $(l, d) \in \hat{\theta}$ existiert, indem $d \in \mathbb{N}_0$ das mit l assoziierte Entscheidungslevel ist.*

Algorithmus 8 QUAFFLE

Eingabe: QBF ψ mit Matrix Φ in minimaler Form
Ausgabe: True, falls ψ erfüllbar, sonst False

```
1: entferneAlleGelerntenTerme                          // Ψ ← ∅, Ω ← ∅
2: d ← 0                                               // initiales Entscheidungslevel
3: θ̂ ← ∅                                               // aktuelle Belegung
4: while True do
5:     while True do
                                   // Solange die Erfüllbarkeit von ψ[θ] unbekannt ist
6:         while {} ∉ min(Ψ[θ] ∪ Φ[θ]) and {} ∉ min(Ω[θ]) and Ψ[θ] ∪ Φ[θ] ≠ ∅ do
7:             if Literal l ist Einheitsliteral in Ω[θ] ∪ Φ[θ] ∪ Ψ[θ] then θ̂ ← θ̂ ∪ {(l, d)}
8:             else if Literal l ist monoton in Ω[θ] ∪ Φ[θ] ∪ Ψ[θ] then θ̂ ← θ̂ ∪ {(l, d)}
9:             else break end if
10:        end while
```

11: **if** $\{\} \in min(\Psi[\theta] \cup \Phi[\theta])$ **then**
12: $d' \leftarrow$ ANALYSIEREKONFLIKT(ψ, $\hat{\theta}$)
13: **if** $d' < 0$ **then return** False
14: **else** BACKJUMP(ψ, $\hat{\theta}$, d') **end if**
15: **else if** $\{\} \in min(\Omega[\theta])$ **or** $\Psi[\theta] \cup \Phi[\theta] = \emptyset$ **then**
16: $d' \leftarrow$ ANALYSIERELÖSUNG(ψ, $\hat{\theta}$)
17: **if** $d' < 0$ **then return** True
18: **else** BACKJUMP(ψ, $\hat{\theta}$, d') **end if**
19: **else break end if**
20: **end while**
21: $d \leftarrow d + 1$
22: $l \leftarrow$ ERHALTENÄCHSTESLITERAL(ψ, θ)
23: $\hat{\theta} \leftarrow \hat{\theta} \cup \{(l, d)\}$
24: **end while**

Anmerkung: Ein Algorithmus, der dem hier vorgestellten Algorithmus 8 sehr ähnlich ist, wurde zuerst in dem bekannten QBF-Solver *Quaffle* [Yu] implementiert. Um die ursprünglichen Autoren des Algorithmus zu würdigen, nennen wir den Algorithmus QUAFFLE.

Algorithmus 8 zeigt den grundsätzlichen Aufbau eines iterativen Lösungsalgorithmus für QBFs. Wie in Zeile 1 zu erkennen ist, wollen wir uns erneut die in Abschnitt 3.4 eingeführten EQBFs zu Nutze machen, um ein lernbasiertes Lösungsverfahren zu entwickeln. Der Algorithmus besteht im Wesentlichen aus zwei Endlosschleifen, wobei die erste (Zeile 4) erst dann verlassen wird, wenn die Erfüllbarkeit der Eingabeformel bekannt ist (Zeile 13 bzw. 17). Die zweite Schleife (Zeile 5) wird immer dann abgebrochen, wenn ein neues Entscheidungslevel betreten werden muss, weil keine weiteren Implikationen möglich sind und die Erfüllbarkeit der Eingabeformel unter der aktuellen Belegung unbekannt ist. Der ursprüngliche Algorithmus führt als einzige Implikationsmöglichkeit Einheitsresolution aus (Zeile 7). Wir haben den Algorithmus in Zeile 8 leicht erweitert, um die Detektion monotoner Literale zu ermöglichen. Diese Erweiterung beeinflusst nicht die Korrektheit des Algorithmus, solange die in Abschnitt 3.3 und 3.4 gemachten Bemerkungen und insbesondere Definition 15 berücksichtigt werden. Das Hinzufügen dieser weiteren Möglichkeit Belegungen zu implizieren, führt allerdings nicht zwangsläufig zu einer Effizienzsteigerung. Die Detektion monotoner Literale ist im Allgemeinen sehr rechenaufwändig und stark abhängig von den verwendeten Datenstrukturen. Die in dem von uns entwickelten Framework **qbfSolve** verwendeten Datenstrukturen erlauben hingegen ein sehr schnelles Erkennen monotoner Literale, wodurch das Hinzufügen von deren Erkennung zu Algorithmus 8 keine negativen Laufzeitauswirkungen haben sollte.

Sind alle möglichen Implikationen ausgeführt und die Erfüllbarkeit unter der aktuellen Belegung noch unbekannt, dann wird die innere Endlosschleife in Zeile 19 abgebrochen und eine Entscheidung über die Belegung eines Literals l getroffen (Zeile 22). Die Variable $var(l)$ ist dann die Entscheidungsvariable des in Zeile 21 berechneten Entscheidungslevels.

Das Kernstück des Algorithmus liegt jedoch in den Zeilen 12 und 16. Die Methoden ANALYSIEREKONFLIKT und ANALYSIERELÖSUNG legen das Entscheidungslevel fest, auf das nach einem Konflikt bzw. nach einer Lösung zurückgesprungen wird. Der Rücksprung passiert in der Methode BACKJUMP, welche auch die Variablen d und $\hat{\theta}$ aktualisiert (nicht im Algorithmus erkennbar). Angenommen v sei die Entscheidungsvariable des Entscheidungslevels d und das Literal l mit $var(l) = v$ sei in der aktuellen Belegung ($l \in \theta$). Damit nach einem Rücksprung auf (beispielsweise) das Entscheidungslevel $d - 1$ nicht erneut l in die Belegung aufgenommen wird, sondern $\sim l$, müssen die Methoden ANALYSIEREKONFLIKT und ANALYSIERELÖSUNG, die das Rücksprunglevel $d - 1$ festlegen, sicherstellen, dass eine Aufnahme von $\sim l$ in die Belegung nach dem Rücksprung impliziert wird.

Eine simple Implementierung der Methode ANALYSIEREKONFLIKT wäre ein Rücksprung auf ein Entscheidungslevel, so dass die letzte Entscheidung über die Belegung eines existentiellen Literals revidiert wird. Respektive sollte die Methode ANALYSIERELÖSUNG die letzte Entscheidung über die Belegung eines universellen Literals korrigieren. Dies könnte ermöglicht werden durch das Hinzufügen von Termen, die aus der aktuellen Belegung gewonnen werden. Sei beispielsweise l das Literal, dessen Belegung korrigiert werden soll, und die aktuelle Belegung sei $\tilde{\theta} = (l_1, \ldots, l_k, l, \ldots)$, wobei l das Entscheidungslevel d besitzt mit der Entscheidungsvariable $var(l)$. Wenn eine Klausel (ein Monom) $\{l_1, \ldots, l_k, \sim l\ (l)\}$ zu der Formel hinzugefügt wird, dann ist nach einem Rücksprung auf das Entscheidungslevel $d - 1$ das Literal $\sim l$ ein Einheitsliteral. In einer solchen Implementierung wäre das Verhalten von QUAFFLE analog zu dem von Q-DLL und der weitere Suchprozess hätte keinen Nutzen von den gespeicherten Termen. Im weiteren Verlauf wollen wir jedoch eine intelligentere Implementierung dieser Methoden ausarbeiten, so dass der Suchbaum durch ein auf Resolution basiertem nicht-chronologischen Backtracking mittels des Speicherns von abgeleiteten Informationen weiter beschnitten wird.

Betrachten wir zuerst den Konfliktfall, also das Entstehen einer falsifizierten Klausel K unter der aktuellen Belegung θ. In diesem Fall kann ein Rücksprung nur dann sinnvoll sein, wenn wir die Entscheidung über die Belegung eines existentiellen Literals korrigieren. Zusätzlich wollen wir die Rücksprungentscheidung aber abhängig von der falsifizierten Klausel K machen. Etwas präziser ausgedrückt, wollen wir mittels KResolution eine Klausel erzeugen, die nach einem Rücksprung eine Einheitsklausel ist und

somit den Suchprozess von dem aktuellen Konflikt wegführt. Da K aufgrund der aktuellen Belegung falsifizierte, muss das zuletzt zugewiesene Literal l aus K existentiell sein, es sei denn, die Eingabeformel ist trivial unerfüllbar und K besteht somit nur aus universellen Literalen (in diesem Fall springen wir direkt zu dem Entscheidungslevel -1 zurück). Sei d das Entscheidungslevel von l. Angenommen es gibt mehrere existentielle Literale mit dem Entscheidungslevel d in K und K wäre somit keine Einheitsklausel nach einem Rücksprung in das Level $d-1$. Da l zuletzt zugewiesen wurde, ist dessen Belegung impliziert. In diesem Fall muss $\sim l$ ein Einheitsliteral sein, da monotone Literale nicht in falsifizierten Klauseln enthalten sind (vgl. Abschnitt 3.3). Wir können also die Regel K5 anwenden, um uns des Literals l zu entledigen. Da die entstehende Klausel erneut unter der Belegung $\theta \setminus \{\sim l\}$ falsifiziert ist und wir auf jeden Fall auf ein Level $< d$ zurückspringen (d. h. $\sim l$ ist in dieser Belegung nicht enthalten), können wir dieses Vorgehen wiederholen und sukzessive die zuletzt zugewiesenen existentiellen Literale entfernen, solange, bis es nur noch ein existentielles Literal l mit dem Entscheidungslevel d in der entstandenen Klausel gibt. Es könnte aber dennoch sein, dass die Entscheidungsvariable von d universell ist und somit der entstandene Teilbaum seit der letzten Entscheidung eines existentiellen Literals unerfüllbar ist. Die Belegung von l muss damit impliziert sein, da $var(l)$ nicht die Entscheidungsvariable von d sein kann. Deswegen können wir uns erneut mittels K5 des Literals l entledigen. Wir wissen also nun, wie wir eine Klausel K' aus der Konfliktklausel erzeugen, deren existentielles Literal l' mit dem höchsten Entscheidungslevel d das Rücksprunglevel $d-1$ bestimmt. Es könnte aber passieren, dass K' keine Einheitsklausel nach dem Rücksprung ist, da K' noch immer unbelegte universelle Literale oder universelle Literale mit demselben Entscheidungslevel enthalten kann, die ein geringeres Level als l' besitzen. Da die Methode ERHALTENÄCHSTESLITERAL die Quantifizierungsreihenfolge beachtet und $var(l)$ somit keine Entscheidungsvariable sein kann, muss die Belegung des Literals l erneut impliziert worden sein. Wir entledigen uns also erneut mittels Resolution des Literals l und springen auf ein noch kleineres Entscheidungslevel zurück. Dadurch dass die Belegung von l nach dem Rücksprung durch die erhaltene Klausel K'' impliziert wird, ist sichergestellt, dass der aktuelle Pfad nicht noch einmal durchsucht wird. Wir wollen noch einmal anmerken, dass das Hinzufügen der Klausel K'' die Erfüllbarkeit der Formel nicht verändert (vgl. Abschnitt 3.4), da diese mittels KResolution von falsifizierten Klauseln an unerfüllten Blättern abgeleitet wurde. Es kann also nicht passieren, dass nach einem Rücksprung erfüllende Pfade, die für den Validierungsprozess erforderlich sind, übersprungen werden. Wir wollen aus diesen Überlegungen nun ein Stopp-Kriterium für den Erzeugungsprozess der zu lernenden Klausel aus einer falsifizierten Klausel aufstellen ([ZM02, S. 209]). Sei K^* die Klausel, die wir aus der falsifizierten Klausel mittels der Regel K5 abgeleitet haben. Wenn für K^* gilt, dass

SK1: K^* enthält keine existentiellen Literale,

dann haben wir mittels KResolution die leere Klausel {} abgeleitet und somit die Unerfüllbarkeit der Eingabeformel bewiesen. Wir springen also in das Entscheidungslevel -1 zurück und geben *False* aus. Ansonsten müssen, wie oben erläutert, die folgenden Bedingungen erfüllt sein.

SK2: Sei d das höchste Entscheidungslevel eines existentiellen Literals in K^*. Dann gibt es nur ein existentielles Literal $l \in K^*$ mit dem Entscheidungslevel d und

SK3: die Entscheidungsvariable des Entscheidungslevels d ist existentiell und

SK4: es gibt keine unbelegten universellen Literale oder belegte universelle mit dem Entscheidungslevel d, die ein geringeres Level als l besitzen.

Algorithmus 9 ANALYSIEREKONFLIKT

Eingabe: QBF ψ und eine Belegung $\hat{\theta}$, wobei $\psi[\theta]$ unerfüllt ist
Ausgabe: Entscheidungslevel d auf das zurückgesprungen werden sollte

```
1:  K ← ERHALTEKONFLIKTKLAUSEL(ψ, θ)
2:  while Solange SK1 or SK2 - SK4 nicht erfüllt sind für K do
3:      l ← ERHALTEZULETZTZUGEWIESENESEXISTENTIELLESLITERAL(K, θ)
4:      K_E ← ERHALTEZUGEHÖRIGENEINHEITSTERM(ψ, θ, l)
5:      K ← ERWEITERTEKRESOLUTION(ψ, θ, K, K_E)
6:  end while
7:  LERNETERM(K, θ)
8:  if K ist die leere Klausel then return −1
9:  else
10:     l ← ERHALTEZULETZTZUGEWIESENESEXISTENTIELLESLITERAL(K, θ)
11:     return ERHALTEENTSCHEIDUNGSLEVEL(l, θ̂) −1
12: end if
```

Algorithmus 9 zeigt die Implementierung der Methode ANALYSIEREKONFLIKT aus Algorithmus 8 nach dem oben beschriebenen Vorgehen. In Zeile 1 wählen wir eine Klausel aus $\Phi \cup \Psi$, die unter der aktuellen Belegung falsifiziert ist. Solange diese Klausel das oben definierte Stopp-Kriterium nicht erfüllt, entfernen wir das zuletzt zugewiesene existentielle Literal. Da dessen Belegung impliziert sein muss, nutzen wir dazu lediglich Regel K5. Sobald das Stopp-Kriterium erreicht wurde, lernen wir die erhaltene Klausel in Zeile 7. Wenn wir die leere Klausel bereits ableiten konnten, dann geben wir als Rücksprunglevel -1 an. Ansonsten springen wir in das Vorgängerlevel des zuletzt zugewiesenen existentiellen Literals in der abgeleiteten Klausel. Wir wollen an dieser Stelle bemerken, dass das Literal l in Zeile 10 tatsächlich dem *First UIP* (*Unique Implication Point*), wie in [ZMMM01] anhand von Implikationsgraphen eingeführt, entspricht ([Zha02]). Vereinfacht gesagt ist ein *UIP* ein Literal l, das zu einer unerfüllten

Belegung θ hinzugefügt wurde, so dass alle Pfade ab dem Präfix $(\ldots, l)$ von $\tilde{\theta}$ zu dem bestehenden Konflikt geführt hätten. Der *First UIP* ist das letzte solche Literal in $\tilde{\theta}$.

Analog möchten wir jetzt den Fall betrachten, dass wir eine erfüllende Belegung gefunden haben. In diesem Fall können wir mittels Modellgenerierung ein verifiziertes Monom M erzeugen oder dieses existiert bereits in der Menge Ω. Mittels MResolution leiten wir erneut ein Monom M^* ab, welches wir zu der Menge Ω hinzufügen. Wenn M^* das leere Monom $\{\}$ ist, dann können wir direkt auf das Entscheidungslevel -1 zurückspringen und damit *True* ausgeben.

SM1: M^* enthält keine universellen Literale

Ansonsten wählen wir das zuletzt zugewiesene universelle Literal l aus M mit dem Entscheidungslevel d. Gibt es mehrere universelle Literale in M mit dem Entscheidungslevel d, dann muss l ein Einheitsliteral sein, da monotone Literale nicht in verifizierten Monomen enthalten sind. Wir wenden also Regel M5 an, um uns solcher Literale zu entledigen. Außerdem wollen wir mindestens soweit zurückspringen, dass wir die letzte Entscheidung eines universellen Literals revidieren können, da dessen momentane Belegung offenbar einen erfüllbaren Teilbaum hervorruft. Des Weiteren müssen wir auch hier wieder berücksichtigen, dass unbelegte existentielle Literale oder belegte mit dem Entscheidungslevel d dafür sorgen können, dass $\sim l$ kein Einheitsliteral nach dem Rücksprung ist, wenn diese ein geringeres Level als l besitzen. Damit ergibt sich für das abgeleitete Monom M^* analog zu SK2 - SK4 noch das zusätzliche Stopp-Kriterium ([ZM02, S. 210]).

SM2: Sei d das höchste Entscheidungslevel eines universellen Literals in M^*. Dann gibt es nur ein universelles Literal $l \in M^*$ mit dem Entscheidungslevel d und

SM3: die Entscheidungsvariable des Entscheidungslevels d ist universell und

SM4: es gibt keine unbelegten existentiellen Literale oder belegte existentielle mit dem Entscheidungslevel d, die ein geringeres Level als l besitzen.

In Algorithmus 10 wurden diese Überlegungen umgesetzt.

Algorithmus 10 ANALYSIERELÖSUNG

Eingabe: QBF ψ und eine Belegung $\hat{\theta}$, wobei $\psi[\theta]$ erfüllt ist
Ausgabe: Entscheidungslevel d auf das zurückgesprungen werden sollte
1: **if** Es gibt Monom $M' \in \Omega$ mit $min(M'[\theta]) = \{\}$ **then** $M \leftarrow M'$
2: **else** $M \leftarrow$ GENERIEREMODELL(ψ, θ) **end if**
3: **while** Solange SM1 **or** SM2 - SM4 nicht erfüllt sind für M **do**

```
    l ← erhalteZuletztZugewiesenesUniversellesLiteral(M, θ)
    M_E ← erhalteZugehörigenEinheitsterm(ψ, θ, l)
    M ← erweiterteMResolution(ψ, θ, M, M_E)
end while
lerneTerm(M, θ)
if M ist das leere Monom then return −1
else
    l ← erhalteZuletztZugewiesenesUniversellesLiteral(M, θ)
    return erhalteEntscheidungslevel(l, θ̂) −1
end if
```

In Quaffle lernen wir damit höchstens einen Term pro Konflikt bzw. Lösung[1]. Da dies aber immer noch exponentiell viele sind, ist es weiterhin notwendig die Mengen der gelernten Terme periodisch unter der Verwendung einer Heuristik zu dezimieren (vgl. Abschnitt 3.4).

[1] Die Methode lerneTerm ist auch im Fall von Algorithmus 7 soweit erweiterbar, dass maximal ein Term pro Konflikt/Lösung gelernt wird [GNT06, S. 402]. Auch **qbfSolve** nutzt die Erkennung des *First UIP* bei der Speicherung von Termen.

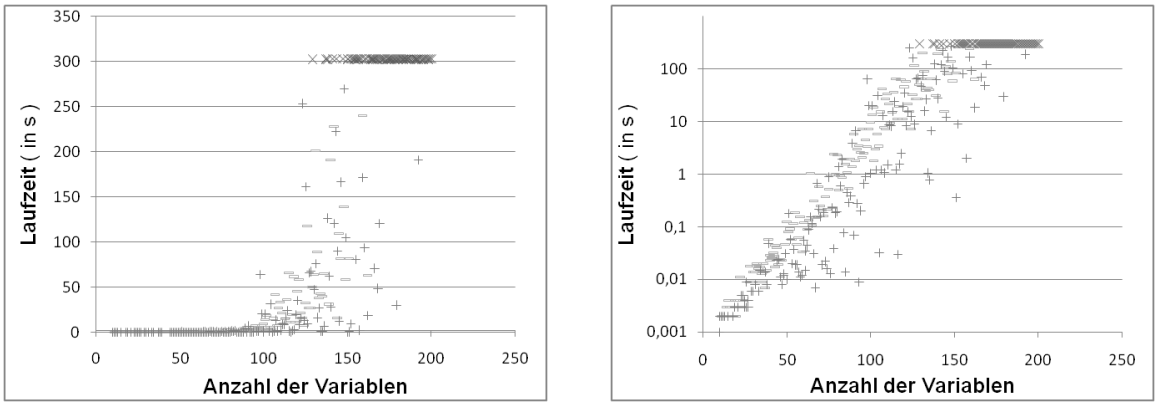
4. Experimentelle Laufzeituntersuchung

Im Folgenden werden wir die Laufzeiten, der in Kapitel 3 beschriebenen Algorithmen, untersuchen und vergleichen. Alle Tests wurden auf demselben Testsystem durchgeführt. Bei dem Testsystem selbst handelt es sich um eine lokale Recheneinheit mit dem Prozessor *AMD Athlon 64 X2 5200+*, 2GB Arbeitsspeicher und dem Betriebssystem *Xubuntu 14.04* (Kernel: *3.13.0-30-generic*). Als Probleminstanzen wurden zum einen Pseudozufallsinstanzen verwendet, die mit dem Formelgenerator von **qbfSolve** erstellt wurden und zum anderen Instanzen aus der *QBFLIB* ([GNPT05c]). Um Trivialfälle zu vermeiden, haben wir bei der Erstellung der Pseudozufallsinstanzen eine zusätzliche Einschränkung in den Formelgenerator eingebaut, so dass jede Klausel in den erstellten Formeln mindestens zwei existentielle Literale enthält. Bei den Instanzen aus der *QBFLIB* verwendeten wir ausschließlich aus der Praxis stammende Instanzen, hauptsächlich Instanzen aus dem Formelsatz *TOILET*. Alle getesteten Instanzen lagen im *QDIMACS* Format vor (vgl. Anhang C), d. h. alle Tests wurden mit QBFs in PKNF durchgeführt. Der Zeitüberschreitungswert lag bei allen Tests bei 300 Sekunden. Wenn also die Erfüllbarkeit der Eingabeinstanz nach 300 Sekunden noch immer unbekannt ist, wurde der Erfüllbarkeitstest abgebrochen. Sofern nicht anders angegeben, wurden die für **qbfSolve** angegeben Standardwerte für Optionen verwendet (vgl. Anhang D).

4.1. Laufzeitinterpolation

Eines der Ziele dieser Arbeit war, die Laufzeiten der behandelten Algorithmen zu interpolieren, um diese somit zur Durchführung direkter Vergleiche nutzen zu können. Dabei wollten wir analog zu den Auswertungen vieler SAT-Algorithmen vorgehen und eine Menge von Testergebnissen einer exponentiellen Regressionsanalyse unterziehen. Der typische Verlauf von Testresultaten bei der Untersuchung eines SAT-Solvers ist in Abbildung 4.1 dargestellt[1]. Besonders in Abbildung 4.1(b) wird der exponentielle

[1]Der dabei verwendete SAT-Solver implementiert den Algorithmus Meta-DPLL ([Mei14]), der den gewöhnlichen DPLL-Algorithmus um die Möglichkeit, Belegungen zu implizieren, erweitert. Dieser

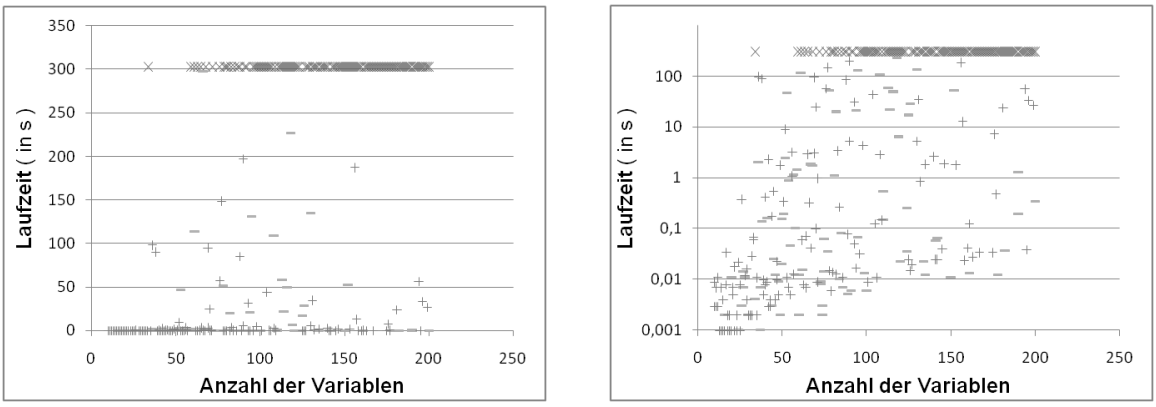


(a) lineare Zeitskala

(b) logarithmische Zeitskala

Abbildung 4.1.: Typischer Verlauf der Testresultate eines SAT-Solvers. Verwendeter Algorithmus: Meta-DPLL ([Mei14]); erfüllbare Formeln (+), unerfüllbare Formeln (−), Zeitüberschreitung (×)

Charakter der Laufzeit deutlich, so dass bei hinreichend vielen Testresultaten eine taugliche Laufzeit interpoliert werden kann. Bei der Auswertung der Testergebnisse, die mit **qbfSolve** erzielt wurden, stellte sich jedoch heraus, dass deren typischer Verlauf eher dem in Abbildung 4.2 entspricht. Eine vollständige und schlüssige Erklärung für dieses Phänomen können wir allerdings nicht bieten. Es sollte dennoch klar sein, dass sich es sich bei allen Algorithmen aus Kapitel 3 um Exponentialzeitalgorithmen handelt, wie auch der Algorithmus DPLL, auf dem diese Algorithmen basieren. Zusätzlich muss die Laufzeit entscheidend von der Anzahl der Variablen der Eingabeformel abhängen, da mit dieser die Anzahl der Belegungen exponentiell wächst, die im schlimmsten Fall zu überprüfen sind. D. h. wir können ohne weiteren Beweis zumindest feststellen, dass die Laufzeit aller Algorithmen im Bereich von $\mathcal{O}(2^n)$ liegt, wobei n für die Anzahl der Variablen in der Probleminstanz steht. Wir vermuten, dass bei einer ausreichend großen Testdatenmenge und einem großzügig gewählten Zeitüberschreitungswert sich auch für einen QBF-Solver ein Muster, ähnlich dem von Abbildung 4.1, einstellen wird. Nichtsdestotrotz wollen wir einige der Testresultate nutzen, um Laufzeitunterschiede der unterschiedlichen Algorithmen zu verdeutlichen.



(a) lineare Zeitskala (b) logarithmische Zeitskala

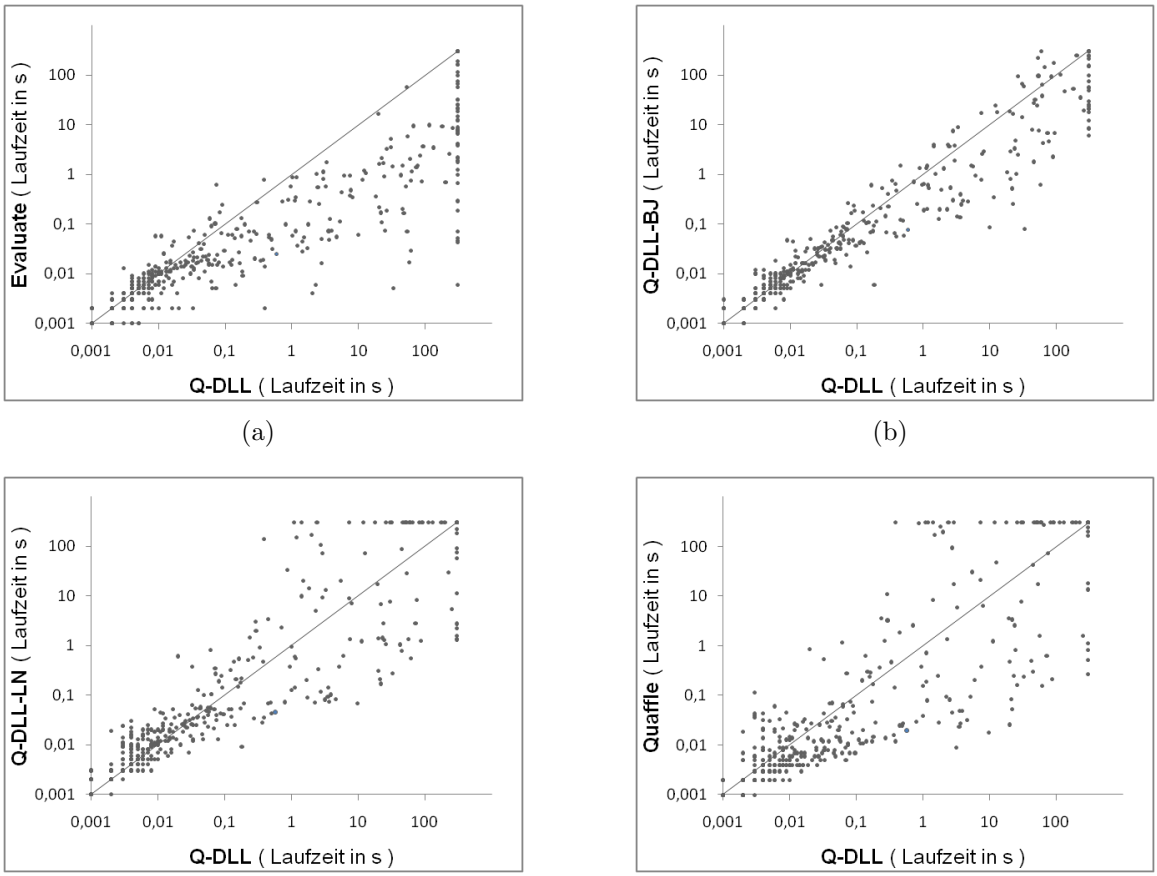
Abbildung 4.2.: Typischer Verlauf der Testresultate eines QBF-Solvers. Verwendeter Algorithmus: Q-DLL; erfüllbare Formeln (+), unerfüllbare Formeln (−), Zeitüberschreitung (×)

4.2. Laufzeitvergleich

In diesem Abschnitt wollen wir die Laufzeiten der Algorithmen aus Kapitel 3 anhand eines Beispieltestdatensatzes vergleichen. Der Datensatz für die Gegenüberstellungen in diesem Abschnitt, sofern nicht anders angegeben, besteht aus Pseudozufallsinstanzen mit einer festen Anzahl an Quantorenmengen (jeweils 5 Stück), einer pseudozufälligen Anzahl an Variablen (zwischen 10 und 200), einer pseudozufälligen Anzahl an Klauseln (zwischen dem vier- und achtfachen der Variablenanzahl), sowie einer festen Klauselgröße (5 Literale pro Klausel). Untersuchungen mit einer pseudozufälligen Anzahl an Quantorenmengen oder einer deutlich höheren Anzahl an Quantorenmengen, abhängig von der jeweiligen Variablenanzahl, ergaben ähnliche Resultate. Die feste Klauselgröße von 5 haben wir gewählt, da diese zu den ausgeglichensten Verhältnis zwischen erfüllbaren und unerfüllbaren Formeln führt. Eine zu klein gewählte Klauselgröße führt schnell zur Unerfüllbarkeit der Formel. Entsprechend führt eine zu groß gewählte Klauselgröße zu einer Mehrheit von erfüllbaren Formeln.

Betrachten wir zuerst Abbildung 4.3, in der die Algorithmen aus Abschnitt 3.2 bis 3.5 mit Algorithmus 1 (Q-DLL) verglichen werden. In Abbildung 4.3(a) fällt sofort auf, dass Evaluate unerwartet gute Resultate im Vergleich zu Q-DLL aufweist. Überraschenderweise erwies sich Evaluate in unseren Testreihen oft tatsächlich als

Algorithmus wird auch von **qbfSolve** verwendet, beispielsweise als SAT-Algorithmus für Evaluate.



(a) (b)

(c) (d)

Abbildung 4.3.: Gegenüberstellung der Laufzeiten von Q-DLL zu Evaluate (a), Q-DLL-BJ (b), Q-DLL-LN (c) und Quaffle (d)

der Algorithmus, der mitunter die besten Ergebnisse lieferte. Dennoch ist es schwierig einen Datensatz zu finden, der die Stärken und Schwächen aller Algorithmen gleichzeitig hervorhebt. Das überproportional gute Abschneiden von Evaluate ist in diesem Fall wohl der Tatsache geschuldet, dass jede in einer Pseudozufallsinstanz enthaltende Klausel mindestens zwei existentielle Literale enthält. Abbildung 4.4 erlaubt eine differenziertere Sichtweise auf die Stärken von Evaluate. In diesem Fall handelt es sich vollständig um Instanzen, die aus der Praxis stammen. Es wird deutlich, dass Evaluate sowohl von großem Vorteil sein kann, nämlich dann, wenn die Probleminstanz auf SAT reduzierbar ist oder eines der (Un-)Erfüllbarkeitskriterien aus Abschnitt 3.2 eintritt. Gleichzeitig kann dieser aber auch sehr nachteilig wirken, wenn keines dieser Kriterien gilt. Die übrigen Diagramme in Abbildung 4.3 entsprechen im Gegensatz zu Abbildung 4.3(a) dem allgemeinen Erscheinungsbild bei Pseudozufallsinstanzen oder

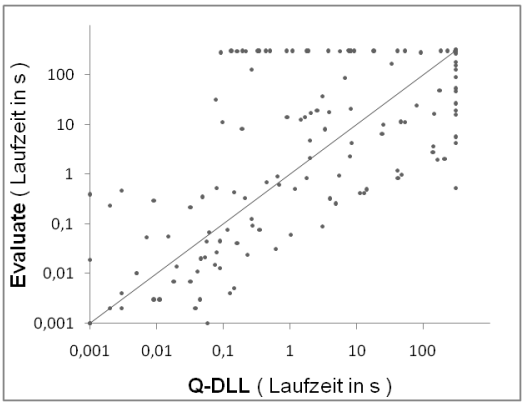


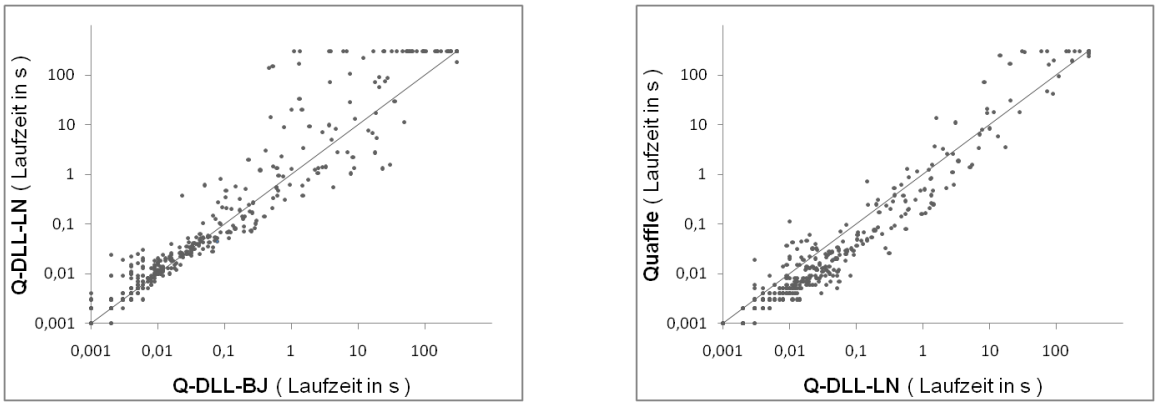
Abbildung 4.4.: Gegenüberstellung der Laufzeiten von Q-DLL zu EVALUATE mit Testinstanzen, die aus der Planung stammen

praktischen Instanzen.

In Abbildung 4.3(b) ist zu erkennen, dass die Laufzeit von Q-DLL-BJ in einigen Fällen der von Q-DLL deutlich überlegen ist. Bei den Instanzen wiederum, bei denen Q-DLL eine bessere Laufzeit aufweist, können wir davon ausgehen, dass Backjumping in keinem bedeutenden Maß angewendet werden konnte. Dennoch ist die Laufzeit in diesen Fällen nie gravierend schlechter zu der von Q-DLL, da der Berechnungmehraufwand in Q-DLL-BJ sich proportional zu dem Berechnungsaufwand von Q-DLL verhält, solange Backjumping keine Anwendung findet. Dieser Algorithmus lässt sich also ohne das Risiko anwenden, dass eine wesentlich schlechtere Laufzeit als die von Q-DLL erzielt wird.

Bei den Lernalgorithmen in Abbildung 4.3(c) und 4.3(d) ist das Verhältnis von Gewinn und Verlust an Laufzeit wesentlich ausgeglichener. Dies ist durch die Gefahr des exponentiellen Lernens von Termen (vgl. Abschnitt 3.4 und 3.5) erklärbar. Instanzen, bei denen die gelernten Terme für den weiteren Suchprozess von größerem Nutzen sein können, haben augenscheinlich eine deutlich bessere Laufzeit als Q-DLL. Können die gelernten Terme jedoch nicht bzw. kaum dazu beitragen den Suchbaum zu beschneiden, wächst der Berechnungsmehraufwand exponentiell an. Trotz der großen Ähnlichkeiten dieser Diagramme scheint der Algorithmus QUAFFLE dennoch leicht bessere Laufzeiten aufzuweisen. Diese Beobachtung wird zusätzlich noch einmal anhand von Abbildung 4.5(b) ersichtlich. Es gibt zwar auch einige wenige Instanzen, bei denen QUAFFLE die Erfüllbarkeit im Gegensatz zu Q-DLL-LN nicht vor Erreichung des Zeitüberschreitungswert feststellen konnte, aber der Großteil der Instanzen wurde von QUAFFLE schneller bearbeitet als von Q-DLL-LN.

Versuchen wir abschließend noch zu klären, ob die Erweiterung von Q-DLL-BJ in



(a) (b)

Abbildung 4.5.: Gegenüberstellung der Laufzeiten von Q-DLL-BJ zu Q-DLL-LN (a) und Q-DLL-LN zu Quaffle (b)

Abschnitt 3.4 zu einem Lernalgorithmus gewinnbringend war. In Abbildung 4.5(a) ist dieser Vergleich beispielhaft dargestellt. Wie zu erkennen ist, führt der Berechnungmehraufwand in Q-DLL-LN in vielen Fällen zu schlechteren Laufzeiten. Dieser Trend lässt sich an den meisten Datensätzen beobachten, die wir untersucht haben. Es gibt aber auch immer wieder einzelne Instanzen bei denen die Laufzeit von Q-DLL-LN der von Q-DLL-BJ weit überlegen ist, was in dieser Abbildung nicht deutlich zum Vorschein tritt. Dennoch müssen wir an dieser Stelle feststellen, dass unsere experimentellen Ergebnisse sich an dieser Stelle deutlich von denen aus [GNT06] unterscheiden. Dies mag daran liegen, dass einzelne Datensätze tatsächlich ein konträres Bild zeichnen. Wir haben allerdings versucht den von uns beobachteten allgemeinen Trend wiederzugeben und dieser zeigt leider deutlich, dass Lernalgorithmen nicht den erhofften Laufzeitgewinn bieten können.

5. Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurden verschiedene Algorithmen vorgestellt, die das Problem $\mathcal{QBF}$ entscheiden. Alle Algorithmen aus Kapitel 3 basierten auf Ansätzen, die bereits eine erfolgreiche Anwendung in Lösungsalgorithmen für quantorenfreie aussagenlogische Formeln fanden. Nach näherer Betrachtung lässt sich sogar feststellen, dass alle diese Algorithmen letztendlich DPLL-basiert sind. Bei unserer Ausarbeitung haben wir dennoch verschiedene Herangehensweisen betrachtet, wie ein möglichst laufzeitoptimaler Lösungsalgorithmus hergeleitet werden kann.

Dabei haben wir zuerst in Algorithmus 1 (Q-DLL) den zu DPLL gänzlich analogen Fall betrachtet, indem wir einen Algorithmus vorstellten, der den Suchbaum mittels chronologischer Rückschritte abarbeitet, bis die Erfüllbarkeit der Eingabeformel feststeht. In Algorithmus 2 (Evaluate) versuchten wir sowohl die Besonderheiten von QBFs zu berücksichtigen, als auch die Effizienz existierender SAT-Solver zu integrieren. Anschließend entwickelten wir Algorithmen, deren Ausführung einer Ableitung der leeren Klausel bzw. des leeren Monoms mittels QBF-spezifischer Resolutionsverfahren entsprach. In Algorithmus 5 (Q-DLL-BJ) nutzten wir dies, um die chronologischen Rückschritte in Q-DLL zu durchbrechen und den Suchbaum durch solche Rücksprünge zu verkleinern. Dies erweiterten wir in Algorithmus 7 (Q-DLL-LN) und 8 (Quaffle) dahingehend, dass wir gewonnene Informationen über die Eingabeformel speicherten und diese somit auch an anderen Stellen des Suchbaums nutzen konnten, um Pfade, deren Erfüllbarkeit dadurch bekannt war, zu überspringen. Mit Quaffle stellten wir zusätzlich einen iterativen Lösungsalgorithmus vor.

Wie Kapitel 4 zeigt, hat sich der Berechnungsmehraufwand der Algorithmen aus Abschnitt 3.2 bis 3.5 häufig gegenüber Q-DLL bezahlt gemacht. Vor allem Q-DLL-BJ weist einen scheinbar kontinuierlichen Vorteil oder höchstens einen kaum spürbaren Nachteil gegenüber Q-DLL auf. Die lernbasierten Algorithmen hingegen haben den großen Nachteil, dass sie exponentiell viel Information speichern können, wodurch im Gegensatz zum Algorithmus Q-DLL-BJ auch Fälle auftreten, in denen ein immenser Laufzeitnachteil gegenüber Q-DLL auftreten kann. Weitere Forschung in diesem Bereich ist somit unumgänglich, damit Möglichkeiten gefunden werden, die ein effizientes Lernen und Vergessen von Termen erlauben.

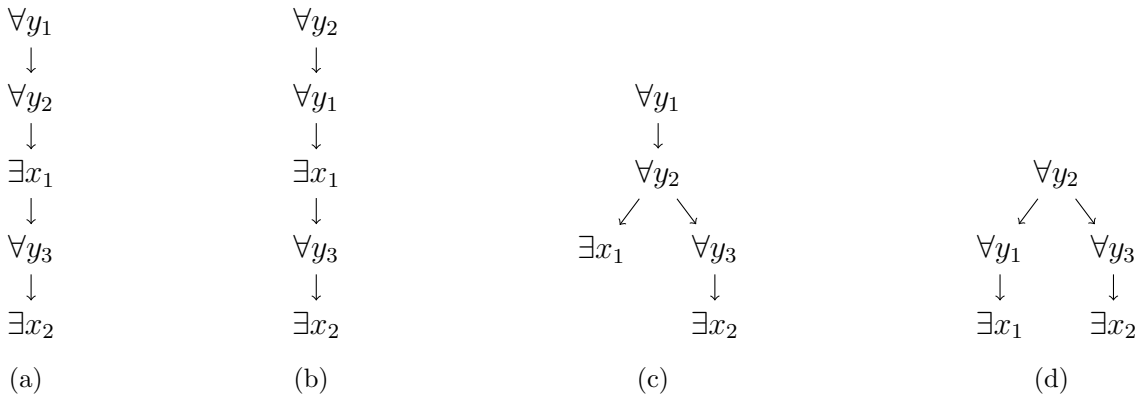


Abbildung 5.1.: Verschiedene Abhängigkeitsschemata für die in Beispiel 8 betrachtete QBF ψ.

Der Umfang dieser Arbeit erlaubte es nicht, einen vollständigen Überblick über die aktuelle Forschung in diesem Bereich zu bieten. Wir möchten es dennoch nicht unterlassen, auf weiterführende Literatur und Themen hinzuweisen.

Zum einen haben wir bislang kein Wort darüber verloren, nach welchen Kriterien die Methode ERHALTENÄCHSTESLITERAL (vgl. Kapitel 3) ein unbelegtes Literal auswählt. Hierbei sind bekannte Heuristiken aus dem SAT-Umfeld, wie *DLIS*, *DLCS*, *MOM*, *Böhm*, *Jeroslaw-Wang* und *Kürzeste Klausel* ([Mei14]), leicht auf QBFs übertragbar. Durch die Berücksichtigung der Abhängigkeiten aufgrund der Quantifizierung ergibt sich aber auch hier ein deutlicher Berechnungsmehraufwand. Dennoch können einfache aber effektive Heuristiken wie *Kürzeste Klausel* auch in QBF-Solvern Vorteile bringen (vgl. Abbildung D.1), indem zusätzlich vorab die Menge der Literale berechnet wird, deren Zuweisung unter der aktuellen Belegung keine Abhängigkeiten verletzt. Doch auch den Begriff *Abhängigkeiten* haben wir im Rahmen dieser Arbeit nicht ausreichend gewürdigt. Unter der Phrase *"Berücksichtigung der Abhängigkeiten"* verstanden wir stets ein triviales Abhängigkeitsschema, das sich lediglich an den Quantifizierungslevels orientierte. Ausgeklügeltere Abhängigkeitsschemata werden in [LB09] und [LB10] eingeführt. Um uns eine Vorstellung anderer Abhängigkeitsschemata zu machen, wollen wir folgendes Beispiel betrachten.

Beispiel 8. *Betrachten wir die QBF ψ:*

$$\forall y_1 \forall y_2 \exists x_1 \forall y_3 \exists x_2 \ ((y_1 \vee \neg y_2 \vee x_1) \wedge (y_2 \vee y_3 \vee x_2) \wedge (y_2 \vee \neg y_3 \vee x_2))$$

Nach einem trivialen Abhängigkeitsschema hätte die Methode ERHALTENÄCHSTESLITERAL *immer ein Literal ausgewählt, welches sich auf dem niedrigsten Level mit noch*

unbelegten Literalen befindet. In diesem Beispiel wären somit die beiden Abhängigkeitsschemata in Abbildung 5.1(a) und 5.1(b) möglich.

Betrachten wir ψ nach folgender Umwandlung:

$$\forall y_1 \forall y_2 \ (\exists x_1 (y_1 \vee \neg y_2 \vee x_1) \wedge \forall y_3 \exists x_2 ((y_2 \vee y_3 \vee x_2) \wedge (y_2 \vee \neg y_3 \vee x_2)))$$

Durch die Umwandlung wird deutlich, dass auch die in Abbildung 5.1(c) und 5.1(d) gezeigten Abhängigkeitsschemata gültig sind für ψ. Beispielsweise kann das Literal y_3 vor dem Literal x_1 gewählt werden, da keine Klausel in ψ existiert, die eine Abhängigkeit dieser Literale untereinander herstellt.

Nähere Informationen zur Konstruktion solcher Quantorenbäume können in [Ben05b] gefunden werden.

Wir möchten aber auch nicht unerwähnt lassen, dass es auch grundlegend unterschiedliche Ansätze gibt für QBF-Lösungsverfahren. Zum einen bietet sich die Möglichkeit an, eine gegebene QBF in eine quantorenfreie aussagenlogische Formel umzuwandeln und diese mit bekannten SAT-Solvern zu lösen [LB08]. Dies kann auch umgesetzt werden in einem Verfahren, welches zuerst sämtliche existentielle Literale entfernt, indem die Eingabeformel in Skolemform gebracht und diese anschließend wieder in eine quantorenfreie aussagenlogische Formel umwandelt wird ([Ben05a], [Ben05c]).

Eine weitere Möglichkeit die Effizienz von QBF-Solvern zu steigern, bieten inkrementelle QBF-Solver ([LE14]). Dabei wird ausgenutzt, dass Instanzen, die aus der Planung, formalen Verifikation, Modellprüfung u. a. stammen, häufig Gemeinsamkeiten aufzeigen, die in herkömmlichen QBF-Solvern ignoriert werden. Ein inkrementeller QBF-Solver erhält eine Menge von Eingabeformeln und versucht gelernte Terme nicht nach dem Erfüllbarkeitstest einer Formel zu löschen, sondern diese für die nächste Formel erneut zu nutzen, solange die Korrektheit des Ergebnisses davon nicht beeinträchtigt wird. Dieser Ansatz stammt erneut aus dem SAT-Umfeld und seine Effizienz hängt immens von der eingegebenen Formelmenge ab. Bei Instanzen, die große Gemeinsamkeiten aufweisen, können dennoch spürbare Verbesserungen mittels dieser Methode erzielt werden.

Bislang haben wir den Eindruck erhalten, dass viele Konzepte, die für die Lösung aussagenlogischer Formeln entwickelt wurden, in modifizierter Weise auch für QBF-Solver anwendbar sind. Dies trifft für einen großen Teil der SAT-Lösungsalgorithmen jedoch nicht zu, nämlich den probabilistischen SAT-Algorithmen. Dies mag auf den ersten Blick verwunderlich sein, da die Untersuchung von probabilistischen Algorithmen einen beachtlichen Anteil der Forschung im Bereich des Erfüllbarkeitsproblems

$\mathcal{SAT}$ einnimmt. Dabei muss jedoch berücksichtigt werden, dass ein essentieller Unterschied bei den Zertifikaten von $\mathcal{QBF}$- und $\mathcal{SAT}$-Instanzen existiert. Das Zertifikat einer QBF wächst exponentiell mit dessen Formellänge (vgl. Abschnitt 3.1). Daraus wird schnell ersichtlich, warum Ansätze aus SAT-Solvern in diesem Fall praktisch nicht zu übernehmen sind. Betrachten wir dazu einen probabilistischen Algorithmus, der eine erfüllende Belegung θ einer QBF ψ errät. Anschließend müsste beginnend bei dem ersten universellen Literal l in $\tilde{\theta} = (l_1, \ldots, l_k, l, \ldots)$ eine erfüllende Belegung für den Präfix $(l_1, \ldots, l_k, \neg l)$ gefunden werden. Dies müsste rekursiv wiederholt werden für alle übrigen universellen Literale in diesen und den entstehenden Belegungen. Sollte dies jedoch misslingen, dann wäre die Unerfüllbarkeit von ψ noch nicht bewiesen. Eine simple Übernahme probabilistischer Ansätze scheint also wenig vielversprechend zu sein, was die Dominanz von deterministischen Algorithmen in QBF-Solvern zu erklären scheint. Dennoch müssen die Forschungsergebnisse aus diesem Fundus der SAT-Algorithmen nicht komplett verworfen werden. Eine womöglich lohnenswerte Untersuchung wäre die Eingliederung von probabilistischen SAT-Algorithmen zum Finden von existentiellen Teilbelegungen. Der einzige uns bekannte probabilistische QBF-Algorithmus wurde in [GHRS03] vorgestellt, welcher auf stochastischer lokaler Suche basiert und die oben beschriebenen Einschränkungen zum Finden eines Zertifikats einhält.

Abschließend wollen wir noch ein paar Bemerkungen zur Integration der aktuell technischen Möglichkeiten in QBF-Solvern machen. Alle in dieser Arbeit vorgestellten Algorithmen arbeiten sequentiell. Nun ist es aber so, dass in den letzten Jahren die Leistungssteigerung von einzelnen CPU-Kernen stagniert. Stattdessen sind moderne CPUs immer mehr darauf spezialisiert Aufgaben parallel zu verarbeiten und auch die Vernetzung räumlich separierter Recheneinheiten nimmt stetig zu. Es scheint also neben dem Finden immer ausgeklügelterer Algorithmen auch sinnvoll zu sein, die technischen Fähigkeiten vollends auszunutzen. Eine Möglichkeit, beispielsweise Q-DLL zu parallelisieren, ist die Erzeugung von Threads an Entscheidungsstellen im Suchbaum, wobei die zwei entstehenden Teilbäume parallel abgearbeitet werden. Dabei muss eine zusätzliche Kontrolle eingerichtet werden, die entscheidet wann eine solche Aufspaltung sinnvoll ist. Ansonsten werden exponentiell viele Threads erzeugt, was selbst bei unbegrenzten Speicher zu weit mehr Verwaltungsaufwand als Nutzen führen würde. Eine einfache Möglichkeit wäre beispielsweise genau so viele aktive Threads zu zulassen wie CPU-Kerne vorhanden sind. Bei Lernalgorithmen entsteht zusätzlich das Problem des gemeinsamen Speicherzugriffs. Lösungsansätze für das Problem der Parallelisierung von einem QBF-Lösungsalgorithmus, ähnlich dem aus Abschnitt 3.4, an lokalen Recheneinheiten oder in verteilten Systemen, werden in [LMS+09] präsentiert.

Dies war eine Auswahl an Forschungsfeldern aus diesem Bereich. Wir hoffen, dass wir verdeutlichen konnten, dass der Forschungsbereich, der sich mit dem Problem $\mathcal{QBF}$ befasst, stetig wächst. Dennoch steht die Forschung in diesem Bereich noch immer am

Anfang, weshalb es nicht sonderlich verwunderlich ist, dass QBF-Solver noch lange nicht die Praxisrelevanz wie SAT-Solver erreichen konnten. Diese Feststellung sollte aber nicht die Bedeutung der Entwicklung von laufzeitoptimierten QBF-Solvern schmälern. Viel mehr sollte sie als Ansporn dienen, sich des komplexeren Problems $\mathcal{QBF}$ anzunehmen und zukunftsorientierte Forschungsarbeit zu leisten.

A. Der Algorithmus Σ_Evaluate

Ergänzend möchten wir nun den Algorithmus Σ_Evaluate präsentieren, der eine wesentliche Unterfunktion des in Abschnitt 3.2 vorgestellten Algorithmus Evaluate darstellt. Im Gegensatz zu Algorithmus 3 behandelt der Algorithmus Σ_Evaluate den Fall, dass die unbelegten Literale mit dem niedrigsten Level existentiell sind.

Algorithmus 11 Σ_Evaluate

Eingabe: QBF ψ, Belegung θ mit $\psi[\theta] = \exists X_1 \dots Q_k X_k \ \{K_1, \dots, K_n\}$
Ausgabe: True, falls $\psi[\theta]$ erfüllbar, sonst False

1: **if** $K_i \cap X_\forall = \emptyset$ für alle $i = 1 \dots n$ **then return** SAT(ψ, θ) **end if**
2: $\psi' \leftarrow$ entferneAlleUniversellenLiterale(ψ, θ)
3: **if** SAT(ψ', θ) **then return** True **end if**
4: **while** Es gibt ein $l \in X_1$ mit $l \notin \theta$ **do**
5: **if** θ erfüllt ψ **then return** True **end if**
6: **if** $\psi[\theta]$ ist unerfüllbar **then return** False **end if**
7: **if** Es existiert $K \in \psi[\theta]$ mit $K \subseteq X_\forall$ **then return** False **end if**
8: **if** Literal l ist Einheitsliteral in $\psi[\theta]$ **then** $\theta \leftarrow \theta \cup \{l\}$
9: **else if** Literal l ist monoton in $\psi[\theta]$ **then** $\theta \leftarrow \theta \cup \{l\}$
10: **else if** Es gibt eine paarweise Widerlegung durch $K, K' \in \psi[\theta]$ **then**
11: **return** False
12: **else**
13: $l \leftarrow$ erhalteNächstesLiteral(ψ, θ)
14: **if** Σ_Evaluate(ψ, $\theta \cup \{l\}$) **then return** True
15: **else return** Σ_Evaluate(ψ, $\theta \cup \{\sim l\}$) **end if**
16: **end if**
17: **end while**
18: **return** Π_Evaluate(ψ, θ)

B. Korrektheit der Klausel- und Monom-Erweiterung einer EQBF

Wir wollen uns im Folgenden von der Korrektheit, der in Definition 13 gemachten Erweiterungen, überzeugen. Dazu gilt hier erneut die Einschränkung, dass wir nur QBFs betrachten, die in minimaler Form sind und keine tautologischen Klauseln enthalten. Außerdem wird für alle hier betrachteten Belegungen θ angenommen, dass diese wie alle Belegungen in Kapitel 3 unter Beachtung der Abhängigkeiten aufgrund der Quantifizierung zustande gekommen sind.

Definition 18. *Sei θ eine Belegung der QBF ψ. Wir nennen eine Menge Θ von Belegungen eine erfüllende Erweiterung von θ, wenn für alle $\theta' \in \Theta$ gilt, dass*

- *$\tilde{\theta}$ ein Präfix von $\tilde{\theta'}$ ist und*
- *für jedes $l \notin \theta$, $l \in \theta'$ und $l \in X_\forall$ gibt es ein $\theta'' \in \Theta$ mit $\sim l \in \theta''$, wobei θ' und θ'' bis zu dem Literal l den selben Präfix haben und*
- *$\theta' \models \Phi$.*

Bemerkung: Eine erfüllende Erweiterung der leeren Belegung $\theta = \emptyset$ ist ein Zertifikat der QBF ψ.

Satz 4. *Sei ψ eine QBF mit der Matrix Φ und K eine mittels einer baumartigen KResolution aus Φ abgeleitete Klausel, also $\Phi \vdash_{KRes} K$. Sei θ eine Belegung, die die Klausel K gerade so falsifiziert[1], dann gibt es keine erfüllende Erweiterung von θ. Des Weiteren gilt, dass wenn es eine Belegung θ gibt, die Φ erfüllt und die Klausel K falsifiziert, dann gibt es einen Präfix von $\tilde{\theta}$, der keine erfüllende Erweiterung besitzt.*

Beweis: Wir beweisen den Satz mittels Induktion über die Anzahl der Resolutionsschritte i.

[1] D. h. es gibt keine Belegung $\theta' \subsetneq \theta$, die K falsifiziert.

Induktionsanfang: $i = 0$
Sei $K \in \Phi$ und θ eine Belegung, die K falsifiziert. Offensichtlich gilt die Behauptung.

Induktionsvoraussetzung:
Sei $\mathbf{K} \subseteq \Phi$ und K eine aus $\mathbf{K}$ mittels baumartiger KResolution in i-Schritten gewonnene Klausel, d. h. $\mathbf{K} \vdash_{KRes,i} K$. Sei θ eine Belegung, die die Klausel K gerade so falsifiziert. Dann gibt es keine erfüllende Erweiterung von θ, so dass $\mathbf{K}$ und damit Φ immer erfüllt sind. Des Weiteren gilt, dass wenn es eine Belegung θ gibt, die $\mathbf{K}$ erfüllt und die Klausel K falsifiziert, dann gibt es einen Präfix von $\tilde{\theta}$, der keine erfüllende Erweiterung besitzt.

Induktionsschritt: $i \to i + 1$
Seien $\mathbf{K_1}, \mathbf{K_2} \subseteq \Phi$ und K_1, K_2 Klauseln mit $\mathbf{K_j} \vdash_{KRes,i} K_j$ für $j \in \{1, 2\}$. Wir betrachten die Klausel K, die mittels einer baumartigen KResolution in einem Schritt aus K_1 und K_2 abgeleitet wurde ($\{K_1, K_2\} \vdash_{KRes,1} K$). Sei l das existentielle Literal mit $l \in K_1$ und $\sim l \in K_2$. Nach Definition 8 ist $l, \sim l \notin K$. Eine Belegung des Literals l kann also entweder K_1 oder K_2 erfüllen, aber nicht beide.

Angenommen es gäbe eine Belegung θ, die K gerade so falsifiziert, aber eine erfüllende Erweiterung Θ besitzt, die K_1 und K_2 immer erfüllt. Da K alle existentiellen Literale aus $K_1 \setminus \{l\}$ und $K_2 \setminus \{\sim l\}$ enthält und deren Negation bereits in θ enthalten sein muss, kann K_1 bzw. K_2 nur durch ein universelles Literal erfüllt werden. Allerdings enthält K auch alle universellen Literale aus K_1 und K_2, bis auf die Literale $\mathbf{l}_\forall$, die jeweils ein höheres Level als alle existentiellen Literale in $K_1 \setminus \{l\}$ und $K_2 \setminus \{\sim l\}$ haben. D. h. K_1 und K_2 können nur dann erfüllt werden, wenn mindestens eines der Literale aus $\mathbf{l}_\forall$ mit wahr belegt sind. Mit der Annahme, dass wir keine Einheitsresolution an universellen Literalen durchführen, können wir davon ausgehen, dass alle Literale in $\mathbf{l}_\forall$ unbelegt oder mit falsch (als monoton impliziert; können auch als wahr impliziert worden sein, siehe unten) belegt sein müssen, bevor alle existentiellen Literale aus K in die Belegung aufgenommen wurden. Daraus ergibt sich, dass es eine Belegung $\theta' \in \Theta$ geben muss, in der alle Literale in $\mathbf{l}_\forall$ mit falsch belegt sind. Da l jeweils nur eine der Klauseln K_1 und K_2 erfüllen kann, ist θ' keine erfüllende Belegung und damit Θ keine erfüllende Erweiterung.

Tatsächlich kann es auch vorkommen, dass ein Literal $l_m \in \mathbf{l}_\forall$ mit wahr belegt ist, da dessen Negation als monotones Literal detektiert wurde. Sei o. B. d. A. $l_m \in K_1$. Ein solcher Fall kann auftreten, wenn $K_1 \in \Psi$ ist und die Klausel K_m aus $\mathbf{K_1}$, die l_m enthält, bereits erfüllt ist. Dieser Fall ist für uns jedoch nicht weiter interessant, da in einem solchen Fall immer eine Klausel aus $\mathbf{K_1}$ oder $\mathbf{K_2}$ falsifiziert sein muss, wenn K falsifiziert ist. Sei $\theta_m \subseteq \theta$ die Belegung, unter der l_m monoton ist in $\Phi \cup \Omega$ (vgl. Definition 15). Da $K_m \in \Phi$, muss $K_m[\theta_m]$ bereits erfüllt sein. Da nun aber K

falsifiziert ist unter θ, muss K_m durch eines der existentiellen Literale[2] erfüllt sein, derer wir uns bei der Ableitung von K mittels Resolution entledigt haben. Da diese Literale immer nur eine der beiden Ausgangsklauseln erfüllen und K_m schon eine dieser erfüllten Ausgangsklauseln ist, gibt es eine falsifizierte Klausel in $\mathbf{K_1}$ oder $\mathbf{K_2}$.

Nun ist es aber so, dass wir Einheitsresolution anhand universeller Literale nach Definition 14 zulassen. Allerdings muss es dafür ein Monom $M \in \Omega$ geben, so dass eine Belegung $\theta'' \subseteq \theta$ dieses Monom in ein Einheitsmonom umwandelt. Die interessante Frage ist nun, ob es so ein solches Monom geben kann. Die Antwort lautet Nein. Sei $l' \in \mathbf{l}_\forall$ und $M[\theta''] = \{\sim l'\}$, so dass das Einheitsliteral l' bereits K_1 oder K_2 erfüllt. Ein Überdeckungsmonom M', das $\sim l'$ enthält, muss auch mindestens ein Literal l'' aus $(K_1 \cup K_2) \setminus \{l, \sim l\}$ enthalten, um Φ zu erfüllen. Wenn dieses Literal l'' aus K stammt, dann muss dieses noch in M enthalten sein, da es ein geringeres Level als $\sim l'$ besitzt und somit nicht vor $\sim l'$ in einer baumartigen Resolution wie in Kapitel 3 aus M heraus resolviert werden kann ($\sim l'$ konnte an dieser Stelle noch nicht mittels Einheitsresolution impliziert sein). Da K unter θ falsifiziert ist, evaluiert M zu falsch und ist somit kein Einheitsmonom. Die andere Möglichkeit ist, dass $l'' \in \mathbf{l}_\forall$ und $l'' \neq l'$ ist. In diesem Fall kann $M[\theta'']$ erneut kein Einheitsmonom sein, denn um l'' aus M' heraus zu resolvieren wird ein anderes Überdeckungsmonom benötigt, dass $\sim l''$ und nicht l' enthält. Ein solches Monom würde wieder ein Literal aus $(K_1 \cup K_2) \setminus \{l, \sim l\}$ benötigen, um Φ zu erfüllen. Ist dieses Literal wieder aus $\mathbf{l}_\forall$ tritt dasselbe Problem auf, d. h. wir können uns dieser Literale nicht mittels Resolution entledigen ohne ein Überdeckungsmonom zu nutzen, welches ein Literal aus K enthält, wodurch M wiederum unter der Belegung θ zu falsch evaluiert (s. o.).

Damit haben wir einen Widerspruch zur Annahme und damit bewiesen, dass es keine erfüllende Erweiterung Θ von θ gibt, die K_1 und K_2 immer erfüllt. Nach der Induktionsvoraussetzung führt eine Falsifikation von K_1 oder K_2 immer auch zu der Falsifikation einer Klausel in $\mathbf{K_1}$ oder $\mathbf{K_2}$.

Nun ist noch zu zeigen, dass eine Belegung θ, die K_1 und K_2 erfüllt und K falsifiziert, einen Präfix $\tilde{\theta'}$ von $\tilde{\theta}$ besitzt, für den keine erfüllende Erweiterung existiert. Seien l und $\mathbf{l}_\forall$ wie oben definiert. Da eine Belegung von l entweder K_1 oder K_2 erfüllen kann und K falsifiziert ist, muss ein Literal $l' \in \mathbf{l}_\forall$ mit wahr belegt sein, also $\tilde{\theta} = (l_1, \ldots, l_n, l', \ldots)$. Da diese Belegung nach den obigen Ausführungen nicht impliziert worden sein kann, muss K auch unter der Belegung $\tilde{\theta'} = (l_1, \ldots, l_n)$ falsifiziert sein. D. h. es gibt keine Belegung θ, die K_1, K_2 erfüllt und K gerade so falsifiziert. Die Klausel K muss also schon vorher durch einen Präfix $\tilde{\theta'}$ von $\tilde{\theta}$ falsifiziert worden sein.

[2] Ein bereits mit wahr belegtes universelles Literal müsste auch in K enthalten sein, da K gerade so falsifiziert ist, oder es wurde erneut als monoton impliziert, wobei es wieder ein anderes Literal in K_m geben muss, dass diese Klausel bereits vorher erfüllte.

Wie oben bewiesen, gibt es für einen solchen Präfix $\tilde{\theta}'$, der K gerade so falsifiziert, keine erfüllende Erweiterung. □

Satz 5. *Sei ψ eine QBF mit der Matrix Φ und M ein mittels einer baumartigen MResolution in Kombination mit Modellgenerierung aus Φ abgeleitetes Monom, also $\Phi \vdash_{MGen,MRes} M$. Sei θ eine Belegung, die das Monom M gerade so verifiziert*[3]*, dann gibt es eine erfüllende Erweiterung von θ.*

Beweis: Analog zu dem Beweis von Satz 4. □

[3] D. h. es gibt keine Belegung $\theta' \subsetneq \theta$, die M verifiziert.

C. Parsen einer QBF im QDIMACS Format

Im Folgenden wollen wir kurz umreißen wie **qbfSolve** Eingabeformeln im *QDIMACS* Format parst. Dazu entwickeln wir einen *Recursive Descent Parser*, um den eventuellen Berechnungsmehraufwand eines tabellenbasierten Parsers bei kleinen Grammatiken zu umgehen.

C.1. Ein Format zum Speichern und zum Austausch von QBFs

QDIMACS ist ein Format für quantifizierte Boolesche Formeln (QBFs) in PKNF. Dieses Format basiert auf dem im SAT-Umfeld bekannten Format *DIMACS* zur Repräsentation aussagenlogischer Formeln in KNF und ist abwärtskompatibel zu diesem. Eine Formel im *QDIMACS* Format besteht aus drei Bestandteilen: der Dateikopf (engl. *header*), der Präfix und die Matrix. Der Dateikopf enthält zum einen beliebige Kommentare, die der Formel hinzugefügt werden können, und eine Problemzeile, die die Anzahl der Variablen und Klauseln beinhaltet. Der Präfix enthält die Quantorenmengen nach aufsteigendem Level sortiert und die Matrix enthält sämtliche zur Formel gehörigen Klauseln. Bevor wir einen Parser für dieses Format ableiten wollen, möchten wir uns das Format an einem Beispiel verdeutlichen.

Beispiel 9. *Betrachten wir dazu die folgende QBF:*

$$\exists x_1 \exists x_2 \forall x_3 \exists x_4 \; ((x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)) \tag{C.1}$$

Die QBF (C.1) umgewandelt in das QDIMACS *Format sieht wie folgt aus:*

```
c Die erste Kommentarzeile
c Eine weitere Kommentarzeile
```

```
p  cnf  4  3
e  1  2  0
a  3  0
e  4  0
1  3  -4  0
-2  4  0
-1  2  -3  0
```

Die ersten drei Zeilen gehören zum Dateikopf. Im Allgemeinen besteht der Dateikopf aus einer beliebigen Anzahl an Kommentarzeilen gefolgt von einer Problemzeile. Die Problemzeile beinhaltet immer zwei Nummern. Die erste Nummer v in der Problemzeile entspricht der maximalen Anzahl an Variablen, die in der Formel enthalten sein dürfen. Variablennamen selbst werden mit Nummern von 1 *bis v bezeichnet. Negationen werden durch ein Minuszeichen kenntlich gemacht. Die zweite Nummer entspricht der Anzahl an Klauseln, die in der Matrix enthalten sind.*

Zeilen, die mit e oder a beginnen, gehören zum Präfix. Eine Präfixzeile, die mit e anfängt, deklariert die darauf folgenden Variablen als existentiell quantifiziert. Dementsprechend gehören die Variablen, die in einer Zeile beginnend mit a auftauchen, zu einer universellen Quantorenmenge. Die auf dem Präfix folgenden Zeilen gehören zur Matrix. Dabei entspricht jede Zeile einer Klausel. Jede Zeile des Präfix oder der Matrix wird mit dem Zeichen 0 *terminiert.*

C.2. Erzeugung einer LL(1)-Grammatik für das QDIMACS Format

Um einen *Recursive Descent Parser* implementieren zu können, benötigen wir eine *LL(1)*-Grammatik. Diese werden wir aus der folgenden BNF-Grammatik für *QDIMACS*-Dateien ableiten ([GNPT05b]):

<input> ::= <preamble> <prefix> <matrix> EOF

<preamble> ::= [<comment_lines>] <problem_line>
<comment_lines> ::= <comment_line> <comment_lines> | <comment_line>
<comment_line> ::= c <text> EOL
<problem_line> ::= p cnf <pnum> <pnum> EOL

<prefix> ::= [<quant_sets>]

<quant_sets> ::= <quant_set> <quant_sets> | <quant_set>
<quant_set> ::= <quantifier> <atom_set> 0 EOL
<quantifier> ::= e | a
<atom_set> ::= <pnum> <atom_set> | <pnum>

<matrix> ::= <clause_list>
<clause_list> ::= <clause> <clause_list> | <clause>
<clause> ::= <literal> <clause> | <literal> 0 EOL
<literal> ::= <num>

<text> ::= {Eine Folge von ASCII-Zeichen (keine Steuerzeichen)}
<num> ::= {Eine vorzeichenbehaftete Integer-Zahl ungleich 0}
<pnum> ::= {Eine vorzeichenbehaftete Integer-Zahl größer als 0}

Wie wir sehen, enthält die Grammatik keine Linksrekursionen. Dennoch gibt es Mehrdeutigkeiten, die wir entfernen müssen. Doch bevor wir uns dieser annehmen, werden wir Abkürzungen für alle Nichtterminale einführen, um die Lesbarkeit der folgenden Ausführungen zu verbessern:

I := <input>	QSS := <quant_sets>
H := <preamble>	QS := <quant_set>
P := <prefix>	Q := <quantifier>
M := <matrix>	AS := <atom_set>
CLS := <comment_lines>	CL := <clause_list>
CL := <comment_line>	C := <clause>
PL := <problem_line>	L := <literal>
T := <text>	N := <num>
PN := <pnum>	

Zusätzlich werden wir im Folgenden die beiden Terminale EOL und EOF in Kleinbuchstaben schreiben, damit nicht die Gefahr besteht, dass diese mit Nichtterminalen verwechselt werden. In der späteren Implementierung werden diese Terminale jeweils ein Zeilenende (eol) bzw. das Dateiende (eof) repräsentieren.

Damit haben wir nun folgende Grammatik erhalten:

$$\begin{aligned} G &= (N, \Sigma, P, S) \\ &= (\{I, H, P, M, CLS, CL, PL, T, PN, QSS, QS, Q, AS, CL, C, L, N\}, \\ &\quad \{c, p, cnf, a, e, 0, string, long\ int, eol, eof\}, P, I) \end{aligned}$$

wobei

$$
\begin{aligned}
P = \{ & I \to H\ P\ M\ eof; & & Q \to e; \\
& H \to CLS\ PL; & & AS \to PN\ AS; \\
& H \to PL; & & AS \to PN; \\
& CLS \to CL\ CLS; & & M \to CS; \\
& CLS \to CL; & & CS \to C\ CS; \\
& CL \to c\ T\ eol; & & CS \to C; \\
& PL \to p\ cnf\ PN\ PN\ eol; & & C \to L\ C; \\
& P \to QSS; & & C \to L\ 0\ eol; \\
& P \to \epsilon; & & L \to N; \\
& QSS \to QS\ QSS; & & T \to string; \\
& QSS \to QS; & & N \to long\ int; \\
& QS \to Q\ AS\ 0\ eol; & & PN \to long\ int\} \\
& Q \to a;
\end{aligned}
$$

Als erstes müssen wir die gemeinsamen Präfixe entfernen, d. h. jede Produktion der Form $A \to \alpha\beta_1 | \ldots | \alpha\beta_r | \gamma_1 | \ldots | \gamma_2$ mit $A \in N$ und $\alpha, \beta_i, \gamma_j \in (N \cup \Sigma)^*$ wird umgewandelt zu

$A \to \alpha A' | \gamma_1 | \ldots | \gamma_2$ und
$A' \to \beta_1 | \ldots | \beta_r$,

wobei es sich bei A' um ein neues Nichtterminalsymbol handelt (vgl. [PB12, S. 70]).

Diese Regel angewandt auf unsere Menge der Produktionen P ergibt:

$$
\begin{aligned}
CLS \to CL\ CLS \mid CL \quad &\Rightarrow \quad \begin{matrix} CLS \to CL\ CLS' \\ CLS' \to CLS \mid \epsilon \end{matrix} \\
QSS \to QS\ QSS \mid QS \quad &\Rightarrow \quad \begin{matrix} QSS \to QS\ QSS' \\ QSS' \to QSS \mid \epsilon \end{matrix} \\
AS \to PN\ AS \mid PN \quad &\Rightarrow \quad \begin{matrix} AS \to PN\ AS' \\ AS' \to AS \mid \epsilon \end{matrix} \\
CS \to C\ CS \mid C \quad &\Rightarrow \quad \begin{matrix} CS \to C\ CS' \\ CS' \to CS \mid \epsilon \end{matrix}
\end{aligned}
$$

$$C \to L\ C \mid L\ 0\ eol \quad \Rightarrow \quad \begin{matrix} C \to L\ C' \\ C' \to C \mid 0\ eol \end{matrix}$$

Damit haben wir eine neue Grammatik

$$G' = (N', \Sigma, P', S) = (N \cup \{CLS', QSS', AS', CS', C'\}, \Sigma, P', I)$$

generiert mit

$$\begin{aligned} P = \{ & I \to H\ P\ M\ eof; \\ & H \to CLS\ PL; \\ & H \to PL; \\ & CLS \to CL\ CLS'; \\ & CLS' \to CLS; \\ & CLS' \to \epsilon; \\ & CL \to c\ T\ eol; \\ & PL \to p\ cnf\ PN\ PN\ eol; \\ & P \to QSS; \\ & P \to \epsilon; \\ & QSS \to QS\ QSS'; \\ & QSS' \to QSS; \\ & QSS' \to \epsilon; \\ & QS \to Q\ AS\ 0\ eol; \\ & Q \to a; \\ & Q \to e; \\ & AS \to PN\ AS'; \\ & AS' \to AS; \\ & AS' \to \epsilon; \\ & M \to CS; \\ & CS \to C\ CS'; \\ & CS' \to CS; \\ & CS' \to \epsilon; \\ & C \to L\ C'; \\ & C' \to C; \\ & C' \to 0\ eol; \\ & L \to N; \\ & T \to string; \\ & N \to long\ int; \\ & PN \to long\ int\} \end{aligned}$$

Abschließend müssen wir noch sicherstellen, dass die erzeugte Grammatik eine *LL(1)*-Grammatik ist. Dazu berechnen wir die Mengen *First* und *Follow* und prüfen für jede Produktionsregel der Form $A \to \alpha_1 | \ldots | \alpha_n$ aus P' mithilfe dieser die folgenden Bedingungen ([PB12]):

1. $First(\alpha_i) \cap First(\alpha_j) = \emptyset$, falls $i \neq j$
2. Wenn $\epsilon \in First(\alpha_j)$, dann $Follow(A) \cap First(\alpha_i) = \emptyset$, falls $i \neq j$

Um die *First* und *Follow* Mengen zu berechnen, benutzen wir einen Ansatz aus [PB12], genannt *Parchman-Graphen*. Zuerst berechnen wir die Menge N_ϵ, welche alle Nichtterminalsymbole enthält aus denen das leere Wort ϵ abgeleitet werden kann:

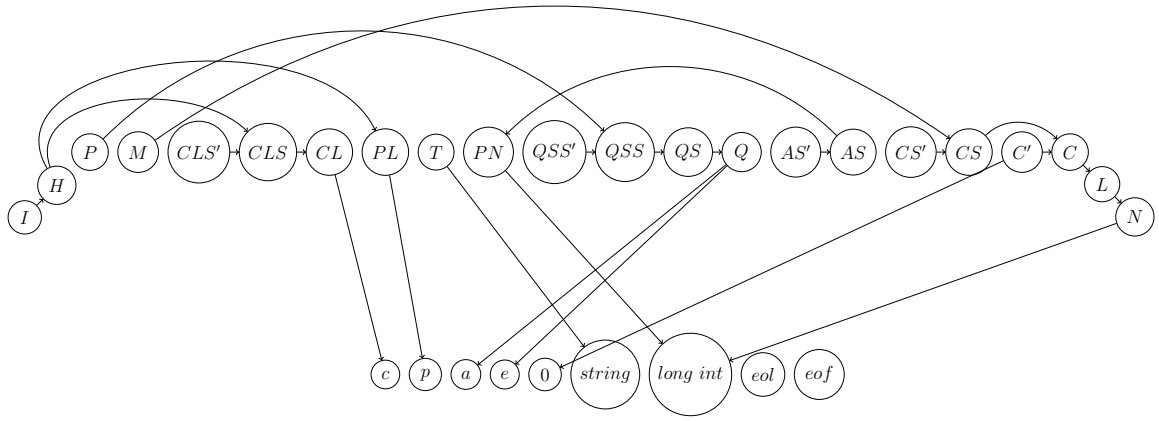


Abbildung C.1.: Der Graph $\Gamma_{first}(G')$ der Grammatik G'

$N_\epsilon = \{P,\ CLS',\ QSS',\ AS',\ CS'\}$

Nun sind wir in der Lage die *Parchman-Graphen* $\Gamma_{first}(G')$ (Abbildung C.1) und $\Gamma_{follow}(G')$ (Abbildung C.2) zu konstruieren.

Im nächsten Schritt werden wir die *First* Mengen aller Nichtterminalen erstellen, indem wir die Terminale hinzufügen, von denen ein Pfad zu dem jeweils betrachteten Nichtterminalen N in $\Gamma_{first}(G')$ (vgl. Abbildung C.1) existiert. Zusätzlich fügen wir das leere Wort ϵ hinzu, falls $N \in N_\epsilon$:

$$
\begin{array}{ll}
First(I) = \{c, p\} & First(QSS) = \{a, e\} \\
First(H) = \{c, p\} & First(QS) = \{a, e\} \\
First(P) = \{\epsilon, a, e\} & First(Q) = \{a, e\} \\
First(M) = \{long\ int\} & First(AS') = \{\epsilon, long\ int\} \\
First(CLS') = \{\epsilon, c\} & First(AS) = \{long\ int\} \\
First(CLS) = \{c\} & First(CS') = \{\epsilon, long\ int\} \\
First(CL) = \{c\} & First(CS) = \{long\ int\} \\
First(PL) = \{p\} & First(C') = \{0, long\ int\} \\
First(T) = \{string\} & First(C) = \{long\ int\} \\
First(PN) = \{long\ int\} & First(L) = \{long\ int\} \\
First(QSS') = \{\epsilon, a, e\} & First(N) = \{long\ int\}
\end{array}
$$

Auf analoge Weise erstellen wir die *Follow* Mengen aller Nichtterminalen. Zur *Follow* Menge eines Nichtterminalsymbols N fügen wir alle Terminalen bzw. das Endsymbol \$ hinzu, wenn diese über einen Pfad von N in $\Gamma_{follow}(G')$ (vgl. Abbildung C.2) erreicht werden können:

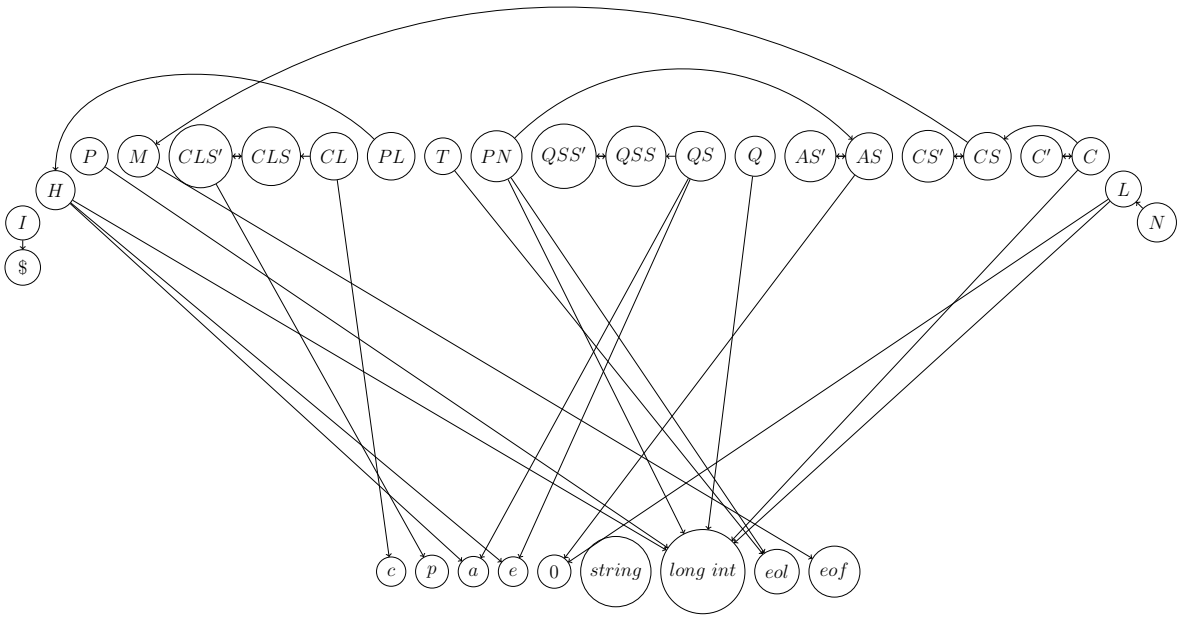


Abbildung C.2.: Der Graph $\Gamma_{follow}(G')$ der Grammatik G'

$Follow(I) = \{\$\}$	$Follow(QSS) = \{long\ int\}$
$Follow(H) = \{a, e, long\ int\}$	$Follow(QS) = \{long\ int, a, e\}$
$Follow(P) = \{long\ int\}$	$Follow(Q) = \{long\ int\}$
$Follow(M) = \{eof\}$	$Follow(AS') = \{0\}$
$Follow(CLS') = \{p\}$	$Follow(AS) = \{0\}$
$Follow(CLS) = \{p\}$	$Follow(CS') = \{\$\}$
$Follow(CL) = \{c, p\}$	$Follow(CS) = \{\$\}$
$Follow(PL) = \{a, e, long\ int\}$	$Follow(C') = \{long\ int, \$\}$
$Follow(T) = \{eol\}$	$Follow(C) = \{long\ int, \$\}$
$Follow(PN) = \{0, eol\}$	$Follow(L) = \{long\ int, 0\}$
$Follow(QSS') = \{long\ int\}$	$Follow(N) = \{long\ int, 0\}$

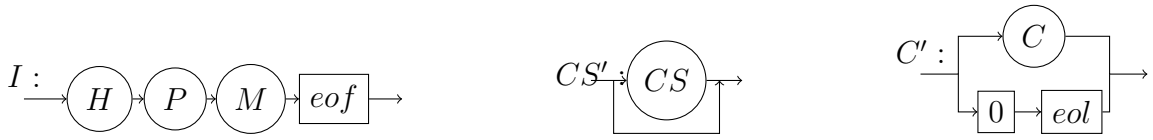
Damit können wir nun die Bedingungen (1) und (2) überprüfen:

$H \to CLS\ PL | PL$
1) $First(CLS\ PL) \cap First(PL) = \{c\} \cap \{p\} = \emptyset$

$Q \to a | e$
1) $First(a) \cap First(e) = \{a\} \cap \{e\} = \emptyset$

$CLS' \to CLS | \epsilon$
1) $First(CLS) \cap First(\epsilon) = \{c\} \cap \{\epsilon\} = \emptyset$

$AS' \to AS | \epsilon$
1) $First(AS) \cap First(\epsilon) = \{long\ int\} \cap \{\epsilon\} = \emptyset$



(a) $I \to H\ P\ M\ eof$ (b) $CS' \to CS|\epsilon$ (c) $C' \to C|0\ eof$

Abbildung C.3.: Beispiele für Syntaxdiagramme einzelner Produktionsregeln

2) $Follow(CLS') \cap First(CLS)$
$= \{p\} \cap \{c\} = \emptyset$

2) $Follow(AS') \cap First(AS)$
$= \{0\} \cap \{long\ int\} = \emptyset$

$P \to QSS|\epsilon$
1) $First(QSS) \cap First(\epsilon)$
$= \{a, e\} \cap \{\epsilon\} = \emptyset$
2) $Follow(P) \cap First(QSS)$
$= \{long\ int\} \cap \{a, e\} = \emptyset$

$CS' \to CS|\epsilon$
1) $First(CS) \cap First(\epsilon)$
$= \{long\ int\} \cap \{\epsilon\} = \emptyset$
2) $Follow(CS') \cap First(CS)$
$= \{\$\} \cap \{long\ int\} = \emptyset$

$QSS' \to QSS|\epsilon$
1) $First(QSS) \cap First(\epsilon)$
$= \{a, e\} \cap \{\epsilon\} = \emptyset$
2) $Follow(QSS') \cap First(QSS)$
$= \{long\ int\} \cap \{a, e\} = \emptyset$

$C' \to C|0\ eol$
1) $First(C) \cap First(0)$
$= \{long\ int\} \cap \{0\} = \emptyset$

Wie wir sehen, erfüllt jede Produktionsregel unserer Grammatik G' die Bedingungen einer *LL(1)*-Grammatik. Damit haben wir also eine gültige *LL(1)*-Grammatik erzeugt, welche wir im Folgenden für die Konstruktion eines Parsers verwenden wollen.

C.3. Konstruktion von Syntaxdiagrammen

Syntaxdiagramme sind eine Möglichkeit, kontextfreie Grammatiken zu repräsentieren. In dem Fall von *LL(1)*-Grammatiken handelt es sich dabei um deterministische Syntaxdiagramme ([PB12, S. 80]), die wir somit verwenden können um einen Algorithmus abzuleiten.

Beispiele für Syntaxdiagramme einzelner Produktionsregeln sind in Abbildung C.3 dargestellt. Durch Verschachtelung der Syntaxdiagramme aller Produktionsregeln erhalten wir ein einzelnes Syntaxdiagramm, welches alle Ableitungen vom Startsymbol I aus beschreibt (siehe Abbildung C.4). Da keine Nichtterminalsymbole mehr in diesem

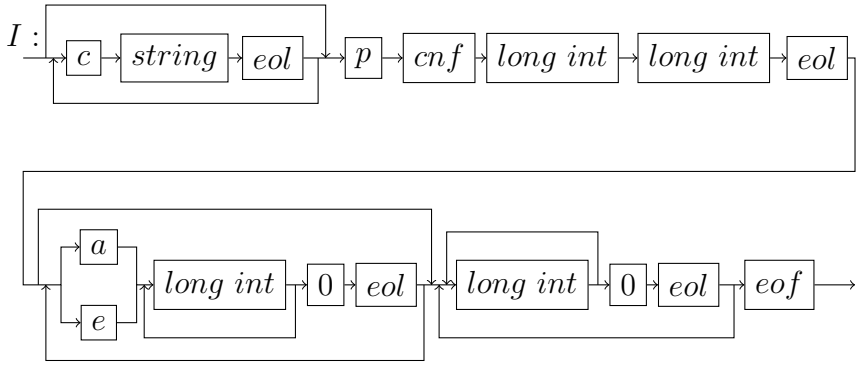


Abbildung C.4.: Das Syntaxdiagramm der Grammatik G'

Syntaxdiagramm enthalten sind, kann auch ein iterativer Parser für diese Grammatik erzeugt werden. Dieses Diagramm kann nun in einen Algorithmus übersetzt werden.

C.4. Zusätzliche Bedingungen des QDIMACS Format

Einige Bedingungen werden nicht durch die Grammatik ausgedrückt und werden daher an dieser Stelle hervorgehoben. Über die Bedeutung der beiden positiven Zahlen in der Problemzeile ($PL \rightarrow p\ cnf\ PN\ PN\ eol$) berichteten wir bereits im Abschnitt C.1. Die positiven Zahlen in Präfixzeilen ($QS \rightarrow Q\ AS\ 0\ eol$) entsprechen Variablennamen. Variablen dürfen in höchstens einer Präfixzeile vorkommen. Variablen, die in der Matrix enthalten sind, nicht aber im Präfix, werden als existentiell quantifiziert mit dem niedrigsten Level angenommen (vgl. Definition 1). Der innerste Quantor einer QBF im QDIMACS Format muss existentiell sein, was jedoch keine wirkliche Einschränkung darstellt (vgl. Definition 5). Laut dem Standard *QDIMACS ver. 1.1* gibt es eine weitere Einschränkung, nämlich dass alle Variablen, die im Präfix erscheinen, auch in der Matrix als Literal enthalten sein müssen. Diese Einschränkungen wird aber von den meisten öffentlich auffindbaren Instanzen nicht eingehalten (z. B. [GNPT05c]). Aus diesem Grund vernachlässigen wir diese Bedingung ebenfalls. Weitere Informationen über die genannten Einschränkungen können in [GNPT05b] nachgelesen werden.

D. Benutzungshinweise zu qbfSolve

Bei dem Framework **qbfSolve** handelt es sich um eine Konsolenanwendung, die verschiedene Algorithmen zur Entscheidung des Problems $\mathcal{QBF}$ bereitstellt. Das Programm selbst ist so strukturiert, dass sich beliebige Lösungsalgorithmen für das Problem $\mathcal{QBF}$ hinzufügen lassen. In den folgenden Benutzungshinweisen beziehen wir uns jedoch auf die Version von **qbfSolve**, die zum Zeitpunkt der Erstellung dieser Arbeit vorlag und die in Kapitel 3 behandelten Algorithmen umfasst.

Der allgemeine Aufruf des Programms aus der Konsole heraus sieht wie folgt aus:

```
qbfSolve [Optionen] (Eingabedatei|-verzeichnis) [Ausgabeverzeichnis]
```

D.1. Übersicht über die Optionen und Argumente von qbfSolve

`-h, --help`

Durch Setzen dieser Option werden nach dem Start von **qbfSolve** die Benutzungshinweise angezeigt. Diese werden auch dann angezeigt, wenn **qbfSolve** ohne Argumente aufgerufen wird.

`-a NUM, --algorithm=NUM`

Mittels dieser Option kann der Algorithmus ausgewählt werden, der zum Lösen der Eingabeinstanz verwendet werden soll. Zur Auswahl stehen die folgenden Algorithmen (vgl. Kapitel 3):

1. Q-DLL
2. Evaluate
3. Q-DLL-BJ

4. Q-DLL-LN
5. Quaffle

Voreingestellt bei dieser Option ist der Algorithmus Q-DLL.

`-o[NAME], --output[=NAME]`

Damit die Ergebnisse der Erfüllbarkeitsuntersuchung nicht verloren gehen, kann **qbfSolve** für jede untersuchte Instanz eine Ausgabedatei erstellen, welche die Ergebnisse zusammenfasst. Das Format dieser Ausgabedatei unterliegt dem Standard nach *QDIMACS ver. 1.1* ([GNPT05b]).

Dieser Option kann optional der Dateiname dieser Ausgabedatei übergeben werden. Wenn in dem aktuellen Programmlauf mehr als eine Instanz untersucht wird, dann werden die Ausgabedateien aufsteigend durchnummeriert. Wurde kein Dateiname angegeben, dann wird der Name der untersuchten Instanz übernommen. Die Dateiendung dieser Ausgabedateien ist *.out*.

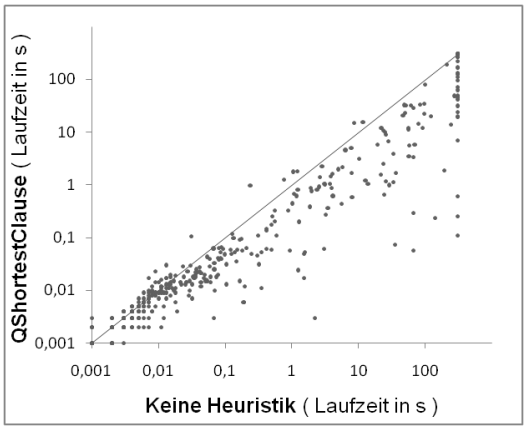
`-c[NAME], --csv[=NAME]`

Besonders wenn viele Instanzen in einem Programmdurchlauf untersucht wurden, ist es oft hilfreich die Ergebnisse aller Instanzen in einer Datei zu haben. Deswegen bietet **qbfSolve** die Möglichkeit diese Ergebnisse in einer CSV-Datei zusammenzufassen. Die CSV-Datei besteht aus vier Spalten. Jede Zeile in dieser Datei entspricht einer untersuchten Instanz. Die erste Spalte enthält die Anzahl der Variablen und die zweite die Anzahl der Klauseln der jeweiligen Instanz. Die dritte Spalte enthält das Ergebnis, welches entweder erfüllbar (*1*), unerfüllbar (*0*) oder unbekannt (*-1*) ist. In der letzten Spalte sind die Zeiten notiert, die für die Untersuchung der Instanzen benötigt wurden. Das Trennzeichen dieser CSV-Datei ist der Tabulator.

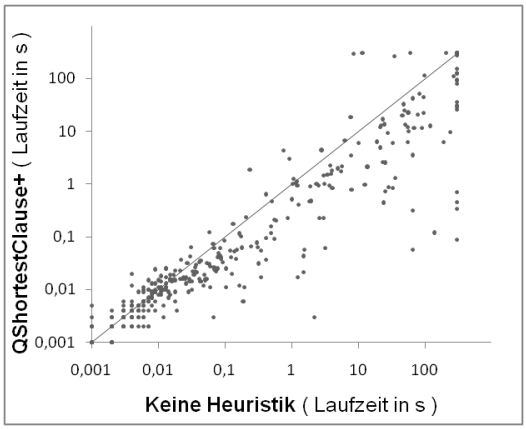
Die Angabe eines Dateinamens für diese CSV-Datei ist erneut optional. Der voreingestellte Dateiname lautet *qbfSolve_results.csv*.

`-t NUM, --timeout=NUM`

Häufig ist es dem Benutzer nicht möglich die Laufzeit eines Solvers abzuschätzen, obwohl dieser dennoch die Erfüllbarkeit von möglichst vielen Instanzen aus einer Eingabemenge innerhalb eines bestimmten Zeitraums erfahren möchte. Dazu bietet **qbfSolve** die Möglichkeit einen Zeitüberschreitungswert zu spezifizieren. Sobald dieser Wert bei der Untersuchung einer Instanz überschritten wird, bricht **qbfSolve** dessen Untersuchung ab und fährt mit der nächsten Eingabeinstanz



(a)



(b)

Abbildung D.1.: Gegenüberstellung der Laufzeiten von Q-DLL bei Verwendung verschiedener Heuristiken

fort.

`-g NUM:NUM:NUM:NUM[:NUM], --generator=NUM:NUM:NUM:NUM[:NUM]`

Zum Testen verschiedener Algorithmen bzw. Solver kann es hilfreich sein, einen Generator für pseudozufällige Instanzen zu haben. Ein solcher Generator kann mittels dieser Option aktiviert werden. Als Argument dieser Option wird eine Reihe von Zahlen erwartet, die durch einen Doppelpunkt voneinander getrennt werden. Diese Zahlenreihe hat das Format n:q:v:c:k, wobei k optional ist. Die Zahl n bestimmt die Anzahl der zu erzeugenden Formeln, q die maximale Anzahl an Quantorenmengen, v die Anzahl der Variablen, c die Anzahl der Klauseln und k die Anzahl der Literale pro Klausel. Wenn k nicht angegeben wurde, hat jede Klausel eine pseudozufällige Länge zwischen 2 und 10. Außerdem enthält jede erzeugte Klausel mindestens ein existentielles Literal, um die triviale Unerfüllbarkeit der erzeugten Formeln zu vermeiden.

Wenn diese Option gewählt wird, ist es notwendig bei den weiteren Argumenten ein Eingabeverzeichnis anzugeben, in welches die erstellten Formeln geschrieben werden. Die Dateinamen der generierten Formeln haben das Format

$$cnf\_i\_q\_v\_c[\_k].qdimacs,$$

wobei i zwischen 1 und n liegt und die i-te generierte Formel kenntlich macht. Die generierten Formeln werden anschließend auf Erfüllbarkeit untersucht.

`-e NUM, --heuristic=NUM`

Mit dieser Option kann eine Heuristik ausgewählt werden, anhand derer die nächste Entscheidungsvariable ausgewählt wird. Zur Auswahl stehen die folgenden Heuristiken:

1. (Voreinstellung) Keine spezielle Heuristik. Es wird das erste gefundene unbelegte Literal gewählt, welches keine Abhängigkeiten verletzt.
2. *QShortestClause*: Diese Heuristik ähnelt der SAT-Heuristik *Kürzeste Klausel* ([Mei14]). Sei V die Menge der Literale, die belegt werden können ohne Abhängigkeiten zu verletzen. Dann wird das Literal aus V gewählt, welches in der kürzesten von all den Klauseln, die Literale aus V enthalten, vorkommt. Der Zweig in dem das gewählte Literal auf falsch gesetzt ist, wird zuerst untersucht.
3. *QShortestClause+*: Diese Heuristik beachtet mehr die Besonderheiten von QBFs als die Heuristik *QShortestClause*. Sei V erneut die Menge der Literale, die belegt werden können ohne Abhängigkeiten zu verletzen. Dann wählt diese Heuristik das Literal aus V, welches in der Klausel mit den wenigsten existentiellen Literalen von all den Klauseln, die Literale aus V enthalten, vorkommt. Der Zweig in dem das gewählte Literal auf falsch gesetzt ist, wird zuerst untersucht. Wenn das gewählte Literal existentiell ist, ist die Wahrscheinlichkeit in einer solchen Klausel höher, dass diese falsifiziert. Ist das gewählte Literal wiederum universell, dann ist die Wahrscheinlichkeit höher, dass eine Einheitsklausel erzeugt wird.

Die Laufzeitvorteile, die sich durch die Verwendung einer bestimmten Heuristik ergeben, sind in Abbildung D.1 anhand des Beispielformelsatzes aus Abschnitt 4.2 dargestellt.

`-s NUM[:NUM:NUM], --sat_heuristic=NUM[:NUM:NUM]`

Mit dieser Option kann eine Heuristik ausgewählt werden, anhand derer die nächste Entscheidungsvariable in dem unterliegenden SAT-Solver[1] ausgewählt wird. Dieser SAT-Solver wird von manchen Algorithmen (z. B. EVALUATE) benötigt. Zu beachten ist, dass manchen Heuristiken optionale Parameter übergeben werden können. Diese werden dann einfach mit einem Doppelpunkt an die ausgewählte Heuristik angehängt. Zur Auswahl stehen die folgenden Heuristiken ([Mei14]):

[1] Aktuell verwendet dieser SAT-Solver einen erweiterten DPLL Algorithmus, welcher Einheitsresolution und Detektion monotoner Literale beherrscht.

1. (Voreinstellung) Keine spezielle Heuristik. Es wird das erste gefundene unbelegte Literal gewählt.
2. *Dynamic Largest Individual Sum (DLIS)*
3. *Dynamic Largest Clause Sum (DLCS)*
4. *Maximum Occurence in Minimal Size Clauses (MOM)*
5. *Boehm-Heuristic*: Dieser Heuristik können die beiden Parameter p_1 und p_2 ([Mei14]) hinzugefügt werden. Die Voreinstellung lautet $p_1 = p_2 = 1$.
6. *Jeroslaw-Wang*
7. *Jeroslaw-Wang 2*
8. *Shortest Clause*

`-r NUM`

Der wesentliche Nachteil von lernbasierten Lösungsalgorithmen ist, dass diese im Allgemeinen exponentiell viele Terme lernen. Trotz einiger Heuristiken, um die Anzahl der zu lernenden Terme einzuschränken, kann das Problem nur durch eine Heuristik gelöst werden, die gelernte Terme auch wieder löschen kann. Wir lösen dies mit der in Abschnitt 3.4 vorgestellten Heuristik, indem wir periodisch die Mengen der gelernten Terme durchsuchen und alle Terme löschen, die eine definierte Anzahl n an unbelegten Literalen überschreiten.

Diese Option bestimmt wie oft die Mengen der gelernten Terme abgesucht werden sollen. Voreingestellt ist ein Wert von 5000, also alle 5000 Rekursionen (resp. Iterationen) werden die Mengen der gelernten Terme aufgeräumt.

`-n NUM`

Mit dieser Option wird die maximale Anzahl von unbelegten Literalen in einem gelernten Term bestimmt, so dass dieser Term während des Aufräumvorgangs (vgl. Option `-r`) erhalten bleibt. Voreingestellt ist ein Wert von 20.

`Eingabedatei|-verzeichnis`

An dieser Stelle kann entweder eine Datei oder ein Verzeichnis angegeben werden. Die angegebene Datei muss eine Probleminstanz im *QDIMACS* Format enthalten. Wenn ein Verzeichnis angegeben wurde, dann werden alle Dateien aus diesem Verzeichnis mit der Dateiendung *.qdimacs* untersucht.

`Ausgabeverzeichnis`

Dieses Verzeichnis kann optional angegeben werden. Die Dateien, die durch die Optionen `-o` bzw. `-c` erstellt werden, werden dann in diesem Verzeichnis abgelegt.

Literaturverzeichnis

[Ben05a] M. Benedetti. Evaluating qbfs via symbolic skolemization. In Franz Baader and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 3452 of *Lecture Notes in Computer Science*, pages 285–300. Springer Berlin Heidelberg, 2005.

[Ben05b] M. Benedetti. Quantifier trees for qbfs. In *In Proc. of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT05*. Springer Verlag, 2005.

[Ben05c] M. Benedetti. skizzo: A suite to evaluate and certify qbfs. In Robert Nieuwenhuis, editor, *Automated Deduction – CADE-20*, volume 3632 of *Lecture Notes in Computer Science*, pages 369–376. Springer Berlin Heidelberg, 2005.

[BKF95] H.K. Buning, M. Karpinski, and A. Flogel. Resolution for quantified boolean formulas. *Information and Computation*, 117(1):12 – 18, 1995.

[BKS04] P. Beame, H. Kautz, and A. Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, 2004.

[BM08] M. Benedetti and H. Mangassarian. Qbf-based formal verification: Experience and perspectives. *JSAT*, 5(1-4):133–191, 2008.

[CmBLM05] S. Coste-marquis, D. Le Berre, F. Letombe, and P. Marquis. Propositional fragments for knowledge compilation and quantified boolean formulae. In *In Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, pages 288–293, 2005.

[CSGG02] M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. An algorithm to evaluate quantified boolean formulae and its experimental evaluation. *Journal of Automated Reasoning*, 28(2):101–142, 2002.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, July 1962.

[GHRS03] I. P. Gent, H. H. Hoos, A. G. D. Rowley, and K. Smyth. Using stochastic local search to solve quantified boolean formulae. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming – CP 2003*, volume 2833 of *Lecture Notes in Computer Science*, pages 348–362. Springer Berlin Heidelberg, 2003.

[GNPT05a] E. Giunchiglia, M. Narizzano, L. Pulina, and A. Tacchella. Qbf solver evaluation portal. `http://www.qbflib.org/index_eval.php`, 2005. Stand 28. Mai 2014.

[GNPT05b] E. Giunchiglia, M. Narizzano, L. Pulina, and A. Tacchella. Qdimacs standard. `http://www.qbflib.org/qdimacs.html#input`, December 2005. Accessed April 2, 2014.

[GNPT05c] E. Giunchiglia, M. Narizzano, L. Pulina, and A. Tacchella. Quantified boolean formulas satisfiability library (qbflib). `www.qbflib.org`, 2005. Stand April 14, 2014.

[GNT04] E. Giunchiglia, M. Narizzano, and A. Tacchella. Monotone literals and learning in qbf reasoning. In Mark Wallace, editor, *Principles and Practice of Constraint Programming – CP 2004*, volume 3258 of *Lecture Notes in Computer Science*, pages 260–273. Springer Berlin Heidelberg, 2004.

[GNT06] E. Giunchiglia, M. Narizzano, and A. Tacchella. Clause/term resolution and learning in the evaluation of quantified boolean formulas. *J. Artif. Intell. Res. (JAIR)*, 26:371–416, 2006.

[LB08] F. Lonsing and A. Biere. Nenofex: Expanding nnf for qbf solving. In Hans Kleine Büning and Xishun Zhao, editors, *Theory and Applications of Satisfiability Testing – SAT 2008*, volume 4996 of *Lecture Notes in Computer Science*, pages 196–210. Springer Berlin Heidelberg, 2008.

[LB09] F. Lonsing and A. Biere. A compact representation for syntactic dependencies in qbfs. In *5584 of LNCS*, pages 398–411. Springer, 2009.

[LB10] F. Lonsing and A. Biere. Integrating dependency schemes in search-based qbf solvers. In O. Strichman and S. Szeider, editors, *Theory and Applications of Satisfiability Testing – SAT 2010*, volume 6175 of *Lecture Notes in Computer Science*, pages 158–171. Springer Berlin Heidelberg, 2010.

[LE14] F. Lonsing and U. Egly. Incremental qbf solving. *CoRR*, abs/1402.2410, 2014.

[LMS+09] Matthew Lewis, Paolo Marin, Tobias Schubert, Massimo Narizzano, Bernd Becker, and Enrico Giunchiglia. Paqube: Distributed qbf solving with advanced knowledge sharing. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009*, volume 5584 of *Lecture Notes in Computer Science*, pages 509–523. Springer Berlin Heidelberg, 2009.

[Mei14] A. Meier. SAT-Algorithmen. Vorlesungsskript, 2014.

[PB12] R. Parchmann and Hans H. Brüggemann. Programmiersprachen und Übersetzer. Vorlesungsskript, 2012.

[SM73] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time(preliminary report). In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC '73, pages 1–9, New York, NY, USA, 1973. ACM.

[Tse83] G. S. Tseitin. On the complexity of derivation in propositional calculus. In G. Wrightson J. Siekmann, editor, *Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970*, pages 466–483. Springer, Berlin, Heidelberg, 1983.

[Vol13] H. Vollmer. Logik und formale Systeme. Vorlesungsskript, 2013.

[Yu] Y. Yu. Quaffle - quantified boolean formula evaluator with learning. `https://www.princeton.edu/~chaff/quaffle.html`. Stand 08. Juli 2014.

[Zha02] L. Zhang. Conflict driven learning in a quantified boolean satisfiability solver. In *in ICCAD '02: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 442–449. ACM Press, 2002.

[ZM02] L. Zhang and S. Malik. Towards a symmetric treatment of satisfaction and conflicts in quantified boolean formula evaluation. In Pascal Hentenryck, editor, *Principles and Practice of Constraint Programming - CP 2002*, volume 2470 of *Lecture Notes in Computer Science*, pages 200–215. Springer Berlin Heidelberg, 2002.

[ZMMM01] L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedings of the 2001 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '01, pages 279–285, Piscataway, NJ, USA, 2001. IEEE Press.

[Zol04] M. Zolda. Comparing Different Prenexing Strategies for Quantified Boolean Formulas. Diplomarbeit, Technische Universität Wien, 2004.