Diplomarbeit

Partielle Wissenskompilation

Claudia Obermaier
obermaie@uni-koblenz.de

Gutachter:
Prof. Dr. Ulrich Furbach und
Christoph Wernhard M.A.

Institut für Informatik
Universität Koblenz-Landau
Koblenz, 29. Mai 2006

An der Universität Koblenz-Landau eingereichte Diplomarbeit.

Gutachter:
Prof. Dr. Ulrich Furbach
Christoph Wernhard M.A.

Ich erkläre hiermit, dass die vorliegende Diplomarbeit von mir selbständig verfasst wurde. Alle benötigten Texte und Quellen sind im Literaturverzeichnis aufgeführt.

Danksagung

Als erstes möchte ich mich bei meinen beiden Betreuern Prof. Dr. Ulrich Furbach und Christoph Wernhard bedanken. Christoph hat mir auch dann noch sehr geduldig geholfen, wenn ich wiederholt die selben Fragen gestellt habe und Uli hat mir immer wieder wertvolle Anregungen für diese Arbeit gegeben. Außerdem möchte ich mich bei Jan Murray für die Hilfe im Kampf gegen Ligaturen und für die seelenruhig Beantwortung meiner zahlreichen Fragen zum Thema Prolog und Latex bedanken. Besonderer Dank gilt Alexander Fuchs, der durch kritisches Korrekturlesen dieser Diplomarbeit sehr viel zur Fertigstellung dieser Arbeit beigetragen hat. Außerdem möchte ich mich bei meinen Eltern für ihre umfassende Unterstützung während meines gesamten Studiums bedanken.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Wissenskompilation in der Aussagenlogik

1.1 Einführung

Viele Probleme in der Aussagenlogik sind nur sehr aufwändig lösbar. Ist beispielsweise eine Wissensbasis gegeben, an die wir Anfragen stellen wollen, so kann dies mitunter sehr mühsam sein. Viele dieser Anfragen können nur in exponentieller Zeit beantwortet werden. Häufig entsprechen die Modelle der Wissensbasis, der Antwort auf eine Anfrage. Dabei interessieren wir uns oft besonders für minimale Modelle, also für Modelle, die keine überflüssigen Elemente enthalten. Auch das Erzeugen eines Modells für eine gegebene Wissensbasis ist sehr kostenintensiv. Eine effiziente Bearbeitung ist sowohl die für Beantwortung von Anfragen als auch für die Modellgenerierung nur in bestimmten Sonderfällen, wie z.B. wenn die Wissensbasis aus Hornklauseln besteht, möglich. Um trotzdem effizient Anfragen beantworten zu können, hat sich die Vorgehensweise der Wissenskompilation entwickelt. Dabei wird die Lösung der Aufgabe in eine Offline- und eine Online-Phase aufgeteilt. In der Offline-Phase wird die Wissensbasis präkompiliert. Dabei wird sie in eine bestimmte Form umgewandelt, auf der sich die erwarteten Anfragen effizient beantworten lassen. Diese Transformation der Wissensbasis ist meist sehr aufwändig. Allerdings muss diese Präkompilation nur einmalig durchgeführt werden. Daher relativiert sich die aufwändige Umwandlung. In der darauf folgenden Online-Phase werden Anfragen an die transformierte Wissensbasis gestellt. Diese liegt nun in einer speziellen Form vor, weshalb die anfallenden Anfragen sehr effizient beantwortet werden können. Es ist üblich, dass diese Anfragen in einer Zeit beantwortet werden können, die polynomiell oder sogar nur linear zur Größe der Wissensbasis ist.

1.2 Problemstellung

Es gibt viele verschiedene Ansätze zum Thema Wissenskompilation. Ein sehr erfolgreicher Ansatz stammt von Adnan Darwiche. Darwiche ist Leiter der Arbeitsgruppe für Automated Reasoning an der UCLA. Der von ihm untersuchten Formeltyp, genannt DNNF, ist das Ziel der Präkompilation und ermöglicht vielfältige effiziente Anfragen. Auch in dieser Diplomarbeit wird dieser Formeltyp untersucht. Allerdings ist es hier das Ziel, bereits auf *teilweise* präkompilierten Wissensbasen effizient Anfragen beantworten zu können. Dabei ist die Idee, die Präkompilation so in einzelne Schritte zu unterteilen, dass eventuell bereits nach *einem* dieser Teilschritte eine Antwort auf ei-

ne Anfrage gefunden werden kann. Liefert das Ergebnis eines Präkompilationsschrittes noch keine Antwort, so wird der nächste Präkompilationsschritt durchgeführt und die Anfrage erneut gestellt. So erspart man sich den Aufwand der vollständigen Präkompilation und ist trotzdem in der Lage, effizient Anfragen beantworten zu können. Es liegt auf der Hand, dass auf dem Ergebnis eines solchen Präkompilationsschrittes mehrere Anfragen gestellt werden können. Natürlich ist es wünschenswert, bereits nach wenigen Präkompilationsschritten eine Anfrage beantworten zu können. Um dies zu ermöglichen, wird hier vorgeschlagen, zusätzliche Informationen über die betrachtete Wissensbasis zur Steuerung der schrittweisen Präkompilation zu verwenden. Die Idee, vorhandene Informationen so einzusetzen, dass Anfragen zielgerichtet beantwortet werden können, wurde auch in [BFFN97] verfolgt.

Die Diagnose von Schaltkreisen ist eine wichtige Anwendung für Wissenskompilation. Die Aufgabe ist es dabei, aus einer gegebenen Schaltkreisbeschreibung eine Erklärung für ein beobachtetes fehlerhaftes Verhalten des Schaltkreises zu beerechnen. Typischerweise interessiert man sich dabei für minimale Erklärungen. Also Erklärungen, bei denen nicht überflüssig viele Schaltelemente als defekt angenommen werden. Wie wir in Kapitel 2 sehen werden, ist sehr einfach, Formeln in DNNF zu minimieren. Daher bietet sich die Diagnose von Schaltkreisen als Anwendung der DNNF an. Auch Adnan Darwiche führt als Beispiel seine Vorgehensweise bei der Wissenskompilation häufig die Diagnose an. Daher werden auch in dieser Arbeit viele Beispiele aus diesem Bereich stammen. Am Ende dieses Kapitels möchte ich kurz erläutern, was die Problemstellung bei der Diagnose ist.

1.3 Aufbau der Arbeit

Der Aufbau dieser Diplomarbeit sieht wie folgt aus. Im zweiten Kapitel stellen wir den oben genannten Formeltyp, die DNNF vor. Hier untersuchen wir auch einige Eigenschaften von Formeln, die in DNNF vorliegen und betrachten die Komplexität einiger Operationen auf Formeln in DNNF. Außerdem betrachten wir in diesem Kapitel einen naiven Algorithmus, der Formeln in DNNF umwandelt. Da viele Algorithmen, die Klauselmengen in DNNF umwandeln, auf DPLL basieren, betrachten wir im dritten Kapitel DPLL und einige Verzweigungsheuristiken für DPLL. Im daran anschließenden vierten-Kapitel beschäftigen wir uns nun mit einem DPLL-basierten Kalkül zur Umwandlung von Kauselmengen in DNNF. Das fünfte Kapitel stellt nun die Idee der partiellen Wissenskompilation vor. Da bei der Modellbestimmung auf partiell präkompilierten Wissensbasen auch nichtminimale Modelle berechnet werden können, spielen in diesem Kapitel auch Minimalitätstests eine Rolle. Auf die oben erwähnte informationsgesteuerte Präkompilation einer Wissensbasis gehen wir in Kapitel 6 genauer ein. Im Rahmen dieser Diplomarbeit wurde ein Prologprogramm entwickelt, dass eine Klauselmenge schrittweise in DNNF umwandelt. Auch die in Kapitel 5 vorgestellte partielle Wissenskompilation

und die Steuerung dieser Kompilation durch zusätzliche Informationen wurde implementiert. Nähere Informationen zu dieser Implementierung und zu Benchmarks, mit denen die hier vorgestellte partielle Wissenskompilation getestet wurde, finden sich in Kapitel 7. Da diese Diplomarbeit auf Deutsch verfasst wurde, findet sich am Ende dieser Arbeit ein Glossar, das die zum Teil ungewohnten deutschen Begriffe ins Englische übersetzt.

1.4 Diagnose

Unter einem Schaltnetz [SS95] versteht man die Zusammenschaltung von Schaltelementen, die eine bestimmte Schaltfunktion realisieren. Diese Schaltfunktionen weisen einer Menge von Eingabevariablen eine Menge von Ausgabevariablen zu. Dabei kann es sich um verschiedene Schaltfunktionen, z.B. das Addieren von Dualzahlen, handelt. Die dabei verwendeten Schaltelemente sind logische Gatter wie z.B. AND, OR, NOT, NAND, NOR. Die Signallaufzeiten, die in den einzelnen Schaltelementen anfallen, werden bei der Beschreibung eines Schaltnetzes nicht berücksichtigt. Daher sind die Werte der Ausgabevariablen zu einem Zeitpunkt eindeutig durch die Werte der Eingabevariablen bestimmt. Enthält das Schaltnetz zusätzlich noch Speicherbausteine, so spricht man von einem Schaltwerk. Bei Schaltwerken hängen die Werte der Ausgabevariablen zu einem bestimmten Zeitpunkt nicht nur von den Werten der Eingabevariablen, sondern auch vom inneren Zustand des Schaltwerkes ab. Im Folgenden benutzen wir den Begriff Schaltkreis als Bezeichnung für ein Schaltnetz oder ein Schaltwerk. Da Schaltkreise durch diese logischen Gatter aufgebaut werden, kann man leicht aussagenlogische Formeln angeben, die das Verhalten des Schaltkreises beschreiben. Abbildung 1.1 zeigt ein Beispiel für einen

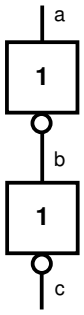


Abbildung 1.1: Beispielschaltkreis mit zwei Invertern

Schaltkreis. Die Variable a ist hier eine Eingabevariable und c ist eine Ausgabevariable. Bei den abgebildeten Schaltelementen handelt es sich um Inverter. Im Folgenden bezeichnen wir den ersten Inverter als $Inv1$ und den zweiten als $Inv2$. Dieser Schaltkreis

wird durch die folgende Klauselmenge Δ beschrieben:

$$\Delta = \{abInv1 \leftarrow a \wedge b, \tag{1.1}$$
$$abInv1 \leftarrow \neg a \wedge \neg b, \tag{1.2}$$
$$abInv2 \leftarrow b \wedge c \tag{1.3}$$
$$abInv2 \leftarrow \neg b \wedge \neg c, \} \tag{1.4}$$

Die in Δ auftretenden Variablen $abInv1$ und $abInv2$ beschreiben den Zustand der beiden Inverter. Ist $abInv1$ mit $true$ belegt, so bedeutet dies, dass der Inverter $Inv1$ defekt ist. Wird $abInv1$ jedoch mit $false$ belegt, so bedeutet dies, dass Inverter $Inv1$ korrekt funktioniert. Die Bedeutung von $abInv2$ ist analog dazu. Klausel 1.1 und 1.2 beschreiben das Verhalten des ersten Inverters. Klausel 1.1 sagt aus, dass Inverter $Inv1$ defekt ist, wenn sowohl a als auch b den Wert $true$ haben. Klausel 1.2 beschreibt, dass Inverter $Inv1$ defekt ist, wenn a und b beide den Wert $false$ haben. Klauseln 1.3 und 1.4 beschreiben, wann Inverter $Inv2$ defekt ist.

1.4.1 Verschiedene Ansätze in der Diagnose von Schaltkreisen

Unter [Sta00] werden drei verschiedene Ansätze der Diagnose von Schaltkreisen aufgeführt, die in der künstlichen Intelligenz verfolgt werden.

- Regelbasierte Diagnose
- Modellbasierte Diagnose
- Fallbasierte Diagnose

Regelbasierte Diagnose

Bei der regelbasierten Diagnose handelt es sich um den ältesten der drei Ansätze. Hier wird versucht, dass Wissen eines menschlichen Diagnoseexperten nachzubilden. Es werden Regeln angegeben, die theoretisches Wissen aber auch Erfahrungen und Daumenregeln widerspiegeln. Durch Anwendung dieser Regeln versucht nun das regelbasierte Diagnosesystem das beobachtete Schaltkreisverhalten zu erklären. Da dieser Ansatz für diese Diplomarbeit nicht relevant ist, gehen wir nicht weiter darauf ein. Der interessierte Leser findet unter [Sta00] weiterführende Informationen.

Modellbasierte Diagnose

Bei der modellbasierten Diagnose verfügt das Diagnosesystem über eine genau Beschreibung des Schaltkreises. Dabei wird genau angegeben, welche Schaltelemente vorhanden sind und wie sie funktionieren. Außerdem wird angegeben, wie die einzelnen Schaltelemete miteinander verbunden sind. Darüber hinaus wird das beobachtete fehlerhafte

Verhalten angegeben. Das modellbasierte Diagnosesystem verfügt zusätzlich über eine Diagnosemethode, mit der die defekten Komponenten abgeleitet werden. Dabei kann es sich zum Beispiel um einen Theorembeweiser handeln. Je genauer Angaben zum betrachteten Schaltkreis sind, desto mehr Verhalten kann das Diagnosesystem erklären. Da die modellbasierte Diagnose für diese Diplomarbeit relevant ist, betrachten wir nun einige Vor- und Nachteile dieses Ansatzes.

Vorteile der Modellbasierten Diagnose:

- Da das Diagnosesystem detailliertes Wissen über den Schaltkreis benutzt, kann es eine Vielzahl von Fehlern erkennen und erklären.
- Die Wissensbasis, die für die modellbasierte Diagnose erstellt wird, kann leicht nach Änderungen des Schaltkreises angepasst werden. Neu hinzugefügte Komponenten müssen z.B. einfach nur zum Modell hinzugefügt werden.

Nachteile der Modellbasierten Diagnose:

- Die Erstellung eines detaillierten Modells eines betrachteten Schaltkreises erfordert genaues Wissen über den Schaltkreis und über die Funktionsweise der einzelnen darin enthaltenen Komponenten.
- Fehler, die die Struktur des Schaltkreises verändern, wie z.B. Ausfälle in der Verkabelung, können nicht mit vertretbarem Aufwand erkannt werden.
- Im Allgemeinen ist die Berechnung von Erklärungen für ein beobachtetes fehlerhaftes Verhalten eines Schaltkreises sehr rechenintensiv.

Fallbasierte Diagnose

In der fallbasierten Diagnose werden Fehler zusammen mit den dazu passenden Erklärungen in einer Wissensbasis abgespeichert. Bei einem neuen Problem, sucht das System die Fälle aus der Wissensbasis heraus, die dem neuen Problem am meisten ähneln. Mit diesen abgespeicherten Fehlererklärungen wird dann eine Erklärung für den neu aufgetretenen Fehler generiert. Im Gegensatz zur regelbasierten und modellbasierten Diagnose wird diese Wissensbasis automatisch durch das Bearbeiten von Fällen aufgebaut. Auch diesen Ansatz werden wir nicht weiter verfolgen. Unter [Sta00] findet man weitere Informationen zu diesem Thema.

1.4.2 Wissenskompilation in der Diagnose

Wie wir gerade erfahren haben, ist die modellbasierte Diagnose sehr rechenintensiv. Bei diesem Diagnoseansatz liegt eine detaillierte Beschreibung des Schaltkreises vor, der dann mit dem beobachteten fehlerhaften Verhalten kombiniert wird. Daher liegt es auf

der Hand, die Idee der Wissenskompilation auf die Modellbasierte Diagnose anzuwenden. Die Idee dabei ist es, die Formalisierung des Schaltkreises zuerst in eine bestimmte Form zu transformieren. Diese Präkompilation findet in einer Offline-Phase statt und muss für einen Schaltkreis nur einmal durchgeführt werden. Bei dieser Transformation wird das beobachtete fehlerhafte Verhalten des Schaltkreises noch nicht beachtet. Daher ist die durch die Präkompilation erreichte Form unabhängig vom beobachteten Fehler und kann für viele verschiedene Fehlerbeobachtungen verwendet werden. Dabei wird die Beschreibung des fehlerhaften Verhaltens - ein Eingabevektor zusammen mit einem Ausgabevektor, der nicht dem erwarteten Ausgabevektor entspricht - mit der präkompilierten Schaltkreisbeschreibung kombiniert. Daraus kann eine Erklärung des Verhaltens generiert werden. Diese Erklärung entspricht einer minimalen Menge von Schaltgliedern, die defekt sein könnten und so das beobachtete fehlerhafte Verhalten hervorrufen könnten. Das Erstellen dieser minimalen Erklärungen in der Online-Phase ist in polynomieller oder sogar in linearer Zeit möglich. Das weist direkt darauf hin, dass die Präkompilationsphase exponentiell sein muss. Im nächsten Kapitel betrachten wir einen bestimmten Formeltyp, der sich als Ziel einer solchen Präkompilation anbietet.

2 Zerlegbare Negationsnormalform

Die zerlegbare Negationsnormalform (DNNF) ist eine spezielle Normalform, die von Adnan Darwiche entwickelt wurde. Sie bietet sich als Zielform bei der Wissenskompilation an, da sie eine Vielzahl von Operationen mit polynomiellem oder sogar linearem Zeitaufwand unterstützt. Im Folgenden betrachten wir diese Art von Normalform genauer und beschäftigen uns mit den Eigenschaften von Formeln, die in DNNF vorliegen. Unter [Dar01] findet man eine ausführliche Einführung zu diesem Thema.

2.1 Begriffserklärungen

Bei den im Folgenden betrachteten Formeln handelt es sich stets um aussagenlogische Formeln, die nur die Junktoren $\wedge$, $\vee$ und $\neg$ enthalten. Da wir häufig den Begriff der Interpretation verwenden werden, definieren wir hier kurz, was wir darunter verstehen. In der folgenden Definition bezeichnet $atome(F)$ die Menge der Atome, die in der aussagenlogischen Formel F vorkommen.

Definition 2.1.1 ***(Interpretation)*** *Sei F eine aussagenlogische Formel, die aus den Atomen $atome(F)$ aufgebaut ist. Eine Interpretation I für F ist eine Menge von Atomen mit $I \subseteq atome(F)$, die mit wahr belegt wird. Alle Atome aus $atome(F)$, die nicht in I sind, werden mit falsch belegt.*

DNNF steht für Decomposable Negation Normalform. Formeln, die in DNNF vorliegen, sind also Formeln in Negationsnormalform, die zusätzlich die Zerlegbarkeitseigenschaft haben. Was diese Eigenschaft bedeutet, betrachten wir später. Definieren wir erst, wann eine Formel in Negationsnormalform vorliegt.

Definition 2.1.2 ***(Negationsnormalform)*** *Eine aussagenlogische Formel F ist in Negationsnormalform (NNF), wenn Negationen in F nur unmittelbar vor Atomen auftreten.*

Beispiel 2.1.3 ***(Negationsnormalform)*** *Die Formel $\neg(a \wedge \neg b) \vee c$ ist nicht in Negationsnormalform, da vor der Klammer $(a \wedge \neg b)$ eine Negation steht. Man kann diese Formel jedoch durch Anwendung der De Morganschen Regel in Negationsnormalform umwandeln. Das Ergebnis dieser Umwandlung ist folgende Formel: $\neg a \vee b \vee c$*

Damit eine Formel in DNNF ist, wird zusätzlich die Eigenschaft der *Zerlegbarkeit* gefordert.

Definition 2.1.4 ***(Zerlegbare Negationsnormalform (DNNF))*** *Eine Formel F liegt in DNNF vor, wenn sie in Negationsnormalform ist und sie zusätzlich zerlegbar ist. Wobei diese Zerlegbarkeitseigenschaft bedeutet, dass für jede Konjunktion $\alpha = \alpha_1 \wedge \ldots \wedge \alpha_n$, die in F vorkommt, gilt: $atome(\alpha_i) \cap atome(\alpha_j) = \emptyset$.*

Die Zerlegbarkeitseigenschaft bedeutet, dass die Konjunktionsglieder einer Konjunktion keine gemeinsamen Variablen haben.

Beispiel 2.1.5 ***(Zerlegbare Negationsnormalform)*** *Die Formel $\neg a \vee b \vee c \vee (\neg a \wedge (b \vee d \vee (a \wedge \neg b)))$ ist nicht in DNNF, da beide Konjunktionsglieder der Konjunktion $(\neg a \wedge (b \vee d \vee (a \wedge \neg b)))$ das Atom a enthalten.*

Die Formel $\neg a \vee b \vee c \vee (\neg a \wedge (b \vee d \vee (d \wedge \neg b)))$ hingegen ist in DNNF, da keine Konjunktionsglieder gemeinsame Atome aufweisen.

Diese Zerlegbarkeitseigenschaft der DNNF ist von entscheidender Bedeutung. Durch diese Eigenschaft ist es möglich, eine Vielzahl von polynomiellen oder sogar linearen Operationen auf der Formel durchzuführen, die auf einer anderen Darstellung der Formel wesentlich aufwändiger sind. Auf einige dieser Operationen gehen wir im nächsten Kapitel ein. Vorher betrachten wir jedoch eine weitere Verschärfung der DNNF.

Definition 2.1.6 ***(gleichmäßige DNNF)*** *Eine DNNF F heißt genau dann gleichmäßig, wenn für alle in F enthaltenen Disjunktionen $\alpha = \alpha_1 \vee \ldots \vee \alpha_n$ gilt: $atome(\alpha) = atome(\alpha_i)$ für alle i.*

Beispiel 2.1.7 ***(gleichmäßige DNNF)*** *Sei die Formel*

$$F = (a \vee b) \wedge c \wedge \neg d$$

gegeben. F ist in DNNF, aber F ist keine gleichmäßige DNNF, da die Disjunktionsglieder der Disjunktion $a \vee b$ nicht die gleichen Atome enthalten. Wir können F jedoch in eine gleichmäßige DNNF umwandeln. Das Ergebnis F' sieht wie folgt aus.

$$F' = ((a \wedge b) \vee (a \wedge \neg b) \vee (\neg a \wedge b)) \wedge c \wedge \neg d$$

F' erhält man aus F indem man bei der Disjunktion $a \vee b$ einfach alle möglichen Wertekombinationen für die Variablen a und b aufzählt.

Eine DNNF heißt also gleichmäßig, wenn in jeder Disjunktion alle Disjunktionsglieder die gleichen Atome enthalten. Es ist leicht zu sehen, dass sich jede DNNF in eine gleichmäßige DNNF umwandeln läßt. Hat die DNNF die Länge n und enthält sie m verschiedene Atome, so kann man sie in $O(nm)$ Zeit in eine gleichmäßige DNNF umwandeln. Wir betrachten Formeln in gleichmäßiger DNNF, weil sich einige Operationen auf Formeln in DNNF besser auf einer gleichmäßigen DNNF definieren lassen.

2.2 Operationen auf Formeln in DNNF und deren Komplexität

In [Dar01] findet man diese kurze Aufzählung der wichtigsten Operationen auf Formeln in DNNF.

1. Das Entscheidbarkeitsproblem kann für Formeln in DNNF in linearer Zeit gelöst werden.
2. Die konjunktive Verknüpfung einer Formel in DNNF mit einer Menge von Literalen kann in linearer Zeit durchgeführt werden.
3. Die Projektion auf einige Atome einer in DNNF vorliegenden Formel kann in linearer Zeit durchgeführt werden.
4. Die Berechung der minimalen Kardinalität einer DNNF kann in linearer Zeit vorgenommen werden. Die Kardinalität eines Modells ist die Anzahl der Atome, die in diesem Modell auf wahr gesetzt werden. Die minimale Kardinalität einer DNNF ist daher die minimale Kardinalität, die unter den Modellen der DNNF auftritt.
5. Eine gleichmäßige DNNF kann in linearer Zeit minimiert werden. Minimieren bedeutet Umwandeln in eine DNNF, die genau die Modelle mit minimaler Kardinalität der ursprünglichen DNNF hat.
6. Die Modelle einer gleichmäßigen DNNF können in linearer Zeit zur Größe der DNNF und quadratisch zur Anzahl der Modelle aufgezählt werden.

Aussage 1 folgt direkt daraus, dass der Erfüllbarkeitstest auf den einzelnen Teilformeln unabhängig voneinander durchgeführt werden kann. Bei einer Konjunktion von zwei Teilformeln $\alpha = \alpha_1 \wedge \ldots \wedge \alpha_n$ können sich die Interpretationen $I_1, \ldots, I_n$, die jeweils die einzelnen Teilformeln α_i erfüllen, nicht widersprechen. Das ist durch die Zerlegbarkeitseigenschaft der Formeln in DNNF sichergestellt. Daher kann aus den Interpretationen $I_1, ..., I_n$ einfach eine Interpretation I erstellen, die die gesamte Formel α erfüllt. Die Gültigkeit von 2 ist nicht direkt ersichtlich, da durch die konjunktive Verknüpfung einer DNNF mit einer Menge von Literalen die Zerlegbarkeitseigenschaft der Formel verloren geht. Unter 2.2.5 wird jedoch ein Operator vorgestellt, mit dem die Vereinigung einer Menge von Literalen mit einer DNNF in linearer Zeit so durchgeführt werden kann, dass die Ergebnisformel wieder in DNNF vorliegt.

Um die übrigen Operationen und ihren Aufwand genauer betrachten zu können, müssen wir jedoch zuerst den Begriff der O-Instanziierung einführen.

Definition 2.2.1 *(**A-Instanziierung, A-Formel**) Sei A eine Menge von aussagenlogischen Variablen $v_1 \ldots v_n$. Eine A-Formel ist eine Formel, die aus den Atomen der Menge A, true und false mit Hilfe der Disjunktion, der Konjunktion und der Negation*

aufgebaut ist. Unter einem A-Literal versteht man entweder ein positives Literal v_i oder ein negatives Literal $\neg v_i$. Eine A-Klausel ist eine Disjunktion $l_1 \vee \ldots \vee l_n$, wobei jedes l_i ein A-Literal ist. Eine A-Instanziierung ist eine Konjunktion $l_1 \wedge \ldots \wedge l_n$, wobei jedes l_i ein A-Literal ist. Die Bezeichnung $\overline{A}$-Formel ($\overline{A}$-Literal) werden wir für Formeln (Literale) benutzen, die aus Atomen aufgebaut sind, die nicht in A enthalten sind.

Definition 2.2.2 ***(Konditionierung)*** *Sei Δ eine aussagenlogische Formel, O eine Menge von aussagenlogischen Variablen. Sei α eine O-Instanziierung. Dann ist das Ergebnis der Konditionierung (Conditioning) von Δ durch α die aussagenlogische Formel, die man aus Δ erhält, indem man jedes O-Literal in Δ durch true (false) ersetzt, wenn es (in)konsistent zu α ist. Die Konditionierung von Δ durch α wird als $\Delta|\alpha$ dargestellt.*

Bei der Konditionierung einer Formel Δ durch eine O-Instanziierung α wird die Information, die in α steckt, in Δ hineinkodiert. Das folgende Beispiel verdeutlicht dies.

Beispiel 2.2.3 ***(Konditionierung)*** *Sei die aussagenlogische Formel $\Delta = (a \wedge \neg b) \vee \neg c$ und die O-Instanziierung $\alpha = \neg b \wedge c$ gegeben. Bilden wir nun die Konditionierung von Δ durch α. Wir erhalten das Ergebnis $\Delta|\alpha = (a \wedge true) \vee false = a$.*

Der zeitliche Aufwand der Konditionierung einer DNNF Δ durch eine Instanziierung α ist linear zur Länge der Formel Δ. Die Sprache der Formeln in DNNF ist gegen Konditionierung abgeschlossen. Dies ist sofort klar, da durch die Konditionierung stets alle Vorkommen der betrachteten Atome entfernt werden. Auch gegen Disjunktion sind Formeln in DNNF abgeschlossen, da die in der Formel vorhandenen Konjunktionen durch eine hinzukommende Disjunktion nicht verändert werden. Gegen Konjunktion sind Formeln in DNNF allerdings nicht abgeschlossen, wie das folgende Beispiel zeigt.

Beispiel 2.2.4 *Die Formeln $\Delta_1 = a \vee b$ und $\Delta_2 = \neg a \vee c$ liegen beide in DNNF vor. Ihre Konjunktion $\Delta_1 \wedge \Delta_2 = (a \vee b) \wedge (\neg a \vee c)$ jedoch nicht.*

Um jedoch trotzdem Formeln in DNNF durch eine Konjunktion verbinden zu können, führen wir den Conjoin Operator ein.

Definition 2.2.5 *(Conjoin) Sei Δ eine Formel in DNNF und α eine Instanziierung. Dann ist der Conjoin Operator definiert als:*

$$\mathit{Conjoin}(\Delta, \alpha) \stackrel{def}{=} (\Delta|\alpha) \wedge \alpha$$

Das Ergebnis dieses Operators ist eine DNNF, die äquivalent zu $\Delta \wedge \alpha$ ist.

Das die durch den Conjoin Operator hergestellte Formel tatsächlich äquivalent zu $\Delta \wedge \alpha$ ist, erkennt man leicht, wenn man bedenkt, dass eine Konjunktion genau dann wahr ist, wenn alle Konjunktionsglieder wahr sind. Die Konjunktion $\Delta \wedge \alpha$ kann also nur wahr sein,

wenn auch α wahr ist. Diese Information kann direkt mit der Hilfe der Konditionierung in die Formel Δ hineinkodiert werden. Daher kann bei Δ eine Konditionierung durch α vorgenommen werden.

Als nächstes betrachten wir die Projektion einer Formel, die in DNNF vorliegt, auf eine Menge von Atomen. Die Projektion wird in der Praxis häufig benutzt. Beispielsweise wenn man nur bestimmte Teile einer Wissensbasis betrachten möchte. Bei der Projektion will man sich auf die Betrachtung einer bestimmten Menge von Atomen einer aussagenlogischer Formel beschränken. Alle Atome, die in dieser Menge nicht enthalten sind, werden aus der Formel entfernt. Bei aussagenlogischen Formeln, die nicht in DNNF vorliegen, ist die Projektion eine aufwändige Operation. Liegt eine Formel jedoch in DNNF vor, so kann diese mit einem Zeitaufwand der linear zu Länge der betrachteten Formel ist, auf eine beliebige Menge von Atomen projiziert werden. Dabei wird jedes Literal, dessen Atom nicht in der oben genannten Menge enthalten ist, auf *true* gesetzt wird.

Definition 2.2.6 ***(Projektion)*** *Die Projektion einer aussagenlogischen Formel Δ auf eine Menge von Atomen A ist eine aussagenlogische Formel Γ, die folgende Eigenschaften hat:*

1. *Γ ist eine A-Formel.*
2. *Für jede A-Formel β gilt: $\Gamma \models \beta$ genau dann, wenn $\Delta \models \beta$.*

Betrachten wir nun, wie wir eine Formel, die in DNNF vorliegt auf eine Menge von Atomen projizieren:

Definition 2.2.7 ***(Projektion einer DNNF)*** *Sei Δ eine aussagenlogische Formel in DNNF und A eine Menge von Atomen. Dann ist $\mathsf{Project}(\Delta, A)$ wie folgt definiert:*

1. $\mathsf{Project}(\Delta, A) \stackrel{def}{=} \begin{cases} true, & wenn\ \Delta\ ein\ \overline{A} - Literal\ ist; \\ \Delta, & wenn\ \Delta\ ein\ A - Literal\ oder\ true\ oder\ false\ ist. \end{cases}$
2. $\mathsf{Project}(\Delta = \bigwedge_i \alpha_i, A) \stackrel{def}{=} \bigwedge_i \mathsf{Project}(\alpha_i, A)$.
3. $\mathsf{Project}(\Delta = \bigvee_i \alpha_i, A) \stackrel{def}{=} \bigvee_i \mathsf{Project}(\alpha_i, A)$.

Den Beweis dafür, dass die in 2.2.7 definierte Project Operation die in 2.2.6 geforderten Eigenschaften erfüllt, findet man unter [Dar01]. Betrachten wir nun ein kleines Beispiel:

Beispiel 2.2.8 ***(Projektion)*** *Sei die Formel $\Delta = e \wedge (b \vee \neg e) \wedge c$ und die Menge von Atomen $A = \{b, c\}$ gegeben. Das Ergebnis der Projektion von Δ auf A erhalten wir, indem wir alle $\overline{A}$ Literale in Δ durch true ersetzen. Dies ergibt:*

$$\Delta = true \wedge (b \vee true) \wedge c = c$$

Eine weitere wichtige Operation, die man auf Formeln in DNNF in linearer Zeit durchführen kann, ist der Erfüllbarkeitstest. Betrachten wir dafür die Operation Sat?.

Definition 2.2.9 *(Sat?) Für eine aussagenlogische Formel Δ, die in Negationsnormalform vorliegt, ist das Prädikat Sat? wie folgt definiert:*

1. $\mathsf{Sat?}(\Delta) \stackrel{def}{=} \begin{cases} true, & \text{wenn } \Delta \text{ ein Literal ist oder } \Delta \text{ wahr ist;} \\ false, & \text{wenn } \Delta \text{ false ist.} \end{cases}$

2. $\mathsf{Sat?}(\Delta = \bigwedge_i \alpha_i) \stackrel{def}{=} true \text{ gdw } \mathsf{Sat?}(\alpha_i) = true \text{ für jedes } i$

3. $\mathsf{Sat?}(\Delta = \bigvee_i \alpha_i) \stackrel{def}{=} true \text{ gdw } \mathsf{Sat?}(\alpha_i) = true \text{ für ein } i$

Die für die Auswertung dieses Prädikats benötigte Zeit ist linear zur Länge der Formel. Da die DNNF eine Unterform der Negationsnormalform ist, kann man mit dem Sat? Prädikat auch die Erfüllbarkeit von Formeln testen, die in DNNF vorliegen.

Satz 2.2.10 *Eine DNNF Δ ist genau dann erfüllbar, wenn $\mathsf{Sat?}(\Delta) = true$ ist.*

Damit liegt uns ein Erfüllbarkeitstest für Formeln in DNNF vor. Oft interessieren wir uns aber nicht nur dafür, ob eine Formel erfüllbar ist, sondern auch für ihre Modelle. Dabei sind häufig die minimalen Modelle von besonderem Interesse.

Definition 2.2.11 ***(minimales Modell)*** *Eine Interpretation M ist genau dann ein minimales Modell einer Formelmenge Σ, wenn M ein Modell für Σ ist und es kein Modell M' von Σ gibt, für das $M' \subset M$.*

Definition 2.2.12 ***(Kardinalität)*** *Unter der Kardinalität $\mathsf{Card}(M)$ eines Modells M einer erfüllbaren aussagenlogischen Formel, verstehen wir die Anzahl der Atome, die das Modell M mit true belegt. Die Kardinalität $\mathsf{Card}(F)$ einer erfüllbaren aussagenlogischen Formel F ergibt sich als:*

$$\mathsf{Card}(F) \stackrel{def}{=} min_{M \models F}\ \mathsf{Card}(M)$$

Die minimale Kardinalität einer unerfüllbaren aussagenlogischen Formel definieren wir als ∞.

Die Kardinalität $\mathsf{Card}(F)$ einer erfüllbaren aussagenlogischen Formel F entspricht dem Minimum der Kardinalitäten aller Modelle von F. Man berechnet sie, indem man zuerst alle Modelle von F bestimmt, die Kardinalitäten der einzelnen Modelle berechnet und zum Schluß das Minimum all dieser Kardinalitäten auswählt. Die minimale Kardinalität $\mathsf{MCard}(F)$ einer Formel F, die in DNNF vorliegt, kann in linearer Zeit berechnet werden. Die folgende rekursive Definition gibt eine Anleitung dafür.

Definition 2.2.13 *(MCard) Sei Δ eine Formel in DNNF. $\mathsf{MCard}(\Delta)$ ist definiert als:*

1. $\mathsf{MCard}(\Delta) \stackrel{def}{=} \begin{cases} 1, & \text{wenn } \Delta \text{ ein positives Literal oder true ist;} \\ 0, & \text{wenn } \Delta \text{ ein negatives Literal ist;} \\ \infty, & \text{wenn } \Delta = false \text{ ist.} \end{cases}$
2. $\mathsf{MCard}(\Delta = \bigvee_i \alpha_i) \stackrel{def}{=} min_i \mathsf{MCard}(\alpha_i)$.
3. $\mathsf{MCard}(\Delta = \bigwedge_i \alpha_i) \stackrel{def}{=} \sum_i \mathsf{MCard}(\alpha_i)$.

Betrachten wir ein kleines Beispiel.

Beispiel 2.2.14 *(MCard) Sei die Formel $\Delta = a \wedge ((c \wedge \neg d) \vee (c \wedge d) \vee (\neg c \wedge \neg d))$ gegeben. Wir berechnen nun $\mathsf{MCard}(\Delta)$ nach Definition 2.2.13:*

$$\begin{aligned} & \mathsf{MCard}(\Delta) \\ = \ & \mathsf{MCard}(a \wedge ((c \wedge \neg d) \vee (c \wedge d) \vee (\neg c \wedge \neg d))) \\ = \ & \mathsf{MCard}(a) + min(\mathsf{MCard}(c \wedge \neg d), \mathsf{MCard}(c \wedge d), \mathsf{MCard}(\neg c \wedge \neg d)) \\ = \ & 1 + min[(\mathsf{MCard}(c) + \mathsf{MCard}(\neg d)), (\mathsf{MCard}(c) + \mathsf{MCard}(d)), \\ & (\mathsf{MCard}(\neg c) + \mathsf{MCard}(\neg d))] \\ = \ & 1 + min[(1+0), (1+1)), (0+0)] \\ = \ & 1 \end{aligned}$$

Dieses Beispiel verdeutlicht, dass es sich bei $\mathsf{MCard}(\Delta)$ einfach nur um die minimale Anzahl von Atomen handelt, die mit *true* belegen werden müssen, damit Δ erfüllt ist.

Als nächstes wollen wir uns mit der Minimierung von Formeln, die in DNNF vorliegen, beschäftigen. Wie bereits angedeutet, soll die Minimierung einer DNNF gerade die DNNF sein, die nur durch die Modelle mit minimaler Kardinalität der ursprüglichen DNNF erfüllt wird. Diese Operation wird besonders für Beispiele aus dem Bereich der Diagnose von Bedeutung sein, da man sich hier für Erklärungen interessiert, bei denen eine minimale Anzahl von Komponenten defekt ist. Die folgende Definition gibt an, wie wir eine vorliegende DNNF minimieren können.

Definition 2.2.15 *(Minimize) Sei Δ eine gleichmäßige DNNF. Dann ist ihre Minimierung $\mathsf{Minimize}(\Delta)$ definiert durch:*

1. $\mathsf{Minimize}(\Delta) \stackrel{def}{=} \Delta$, *wenn Δ ein Literal oder true oder $false$ ist.*
2. $\mathsf{Minimize}(\Delta = \bigvee_i \alpha_i) \stackrel{def}{=} \bigvee_{\mathsf{MCard}(\alpha_i)=\mathsf{MCard}(\Delta)} \mathsf{Minimize}(\alpha_i)$.
3. $\mathsf{Minimize}(\Delta = \bigwedge_i \alpha_i) \stackrel{def}{=} \bigwedge_i \mathsf{Minimize}(\alpha_i)$.

Beispiel 2.2.16 *(Minimize) Wir wollen die Formel Δ aus Beispiel 2.2.14 minimieren:*

$$\begin{aligned}
&\mathit{Minimize}(\Delta)\\
=\ &\mathit{Minimize}(a \wedge ((c \wedge \neg d) \vee (c \wedge d) \vee (\neg c \wedge \neg d)))\\
=\ &\mathit{Minimize}(a) \wedge \mathit{Minimize}(\neg c \wedge \neg d)\\
=\ &a \wedge \mathit{Minimize}(\neg c) \wedge \mathit{Minimize}(\neg d)\\
=\ &a \wedge \neg c \wedge \neg d
\end{aligned}$$

Man sieht, dass die Formel $\mathit{Minimize}(\Delta)$ *nur noch die minimalen Modelle von Δ hat.*

Bisher haben wir zwar Operationen für den Erfüllbarkeitstest und die Minimierung, aber noch keine Operation zur Berechnung von Modellen kennen gelernt. Eine solche Operation ist die Models Operation, die wir nun definieren. Modelle werden hier als eine Menge von Zuweisungen der Form $X = v$ angesehen, wobei X ein Atom und $v \in \{true, false\}$ ist.

Definition 2.2.17 *(Models) Sei Δ eine gleichmäßige NNF. Dann ist* $\mathit{Models}(\Delta)$ *wie folgt definiert:*

1. $\mathit{Models}(\Delta) \stackrel{def}{=} \begin{cases} \{\{p = true\}\}, & \text{wenn } \Delta \text{ ein positives Literal } p \text{ ist;}\\ \{\{p = false\}\}, & \text{wenn } \Delta \text{ ein negatives Literal } \neg p \text{ ist;}\\ \{\{\}\}, & \text{wenn } \Delta = true;\\ \{\}, & \text{wenn } \Delta = false. \end{cases}$

2. $\mathit{Models}(\Delta = \bigwedge_i \alpha_i) \stackrel{def}{=} \bigcup_i \mathit{Models}(\alpha_i)$;

3. $\mathit{Models}(\Delta = \bigvee_i \alpha_i) \stackrel{def}{=} \{\bigcup_i \beta_i \mid \beta_i \in \mathit{Models}(\alpha_i)\}$.

Der folgende Satz sagt aus, dass man mit Hilfe der Models Operation die Modelle einer gleichmäßigen DNNF aufzählen kann:

Satz 2.2.18 *Sei Δ eine gleichmäßige DNNF und I eine Interpretation die zu Δ passend ist. Dann gilt $I \models \Delta$ genau dann, wenn $I \in \mathit{Models}(\Delta)$.*

Beweise für die Sätze 2.2.18 und 2.2.10 findet man unter [Dar01]

2.3 Umwandlung in DNNF

Es wurde bereits mehrmals erwähnt, dass die Umwandlung einer aussagenlogischen Formel in DNNF sehr aufwändig sein kann. Daher zahlt es sich aus, sich genauer mit Algorithmen zur Umwandlung einer Formel in DNNF zu beschäftigen. Zunächst betrachten wir einen naiven Algorithmus. Dieser Algorithmus ist zwar sehr ineffizient, ist jedoch sehr hilfreich um die Probleme bei der Umwandlung zu verdeutlichen.

2.3.1 Naiver Algorithmus zur Umwandlung von aussagenlogischen Formeln in DNNF

Betrachten wir nun einen rekursiven Algorithmus zur Umwandlung beliebiger aussagenlogischer Formeln in DNNF. Dieser Algorithmus basiert auf dem folgenden Satz.

Satz 2.3.1 *Seien Δ_1 und Δ_2 aussagenlogische Formeln in DNNF. Sei $X = atome(\Delta_1) \cap atome(\Delta_2)$. Sei $\Delta = \bigvee_\beta (\Delta_1|\beta) \wedge (\Delta_2|\beta) \wedge \beta$, wobei es sich bei β um alle möglichen X-Instanziierungen handelt. Dann ist Δ in DNNF und Δ ist äquivalent zu $\Delta_1 \wedge \Delta_2$.*

Dieser Satz beschreibt, wie man aus zwei Formeln Δ_1 und Δ_2, die in DNNF vorliegen, die DNNF zu $\Delta_1 \wedge \Delta_2$ erhält. Man kann diesen Satz allerdings auch andersherum verwenden. Ist man eine Formel Δ der Form $\Delta = \Delta_1 \wedge \Delta_2$ gegeben, so sagt uns Satz 2.3.1 wie wir Δ in DNNF umwandeln können. Dazu müssen wir nur die Menge X der gemeinsamen Atome der Teilformeln Δ_1 und Δ_2 bilden und dann erhält man eine zu Δ äquivalente DNNF wie folgt:

$$\Delta = \bigvee_\beta ((\Delta_1|\beta) \wedge (\Delta_2|\beta) \wedge \beta)$$

wobei β eine X-Instanziierung ist.

Mit diesem Satz als Grundlage können wir einen rekursiven Algorithmus angeben, mit sich aussagenlogische Formeln in DNNF umwandeln lassen. Die betrachtete aussagenlogische Formel muss in Klauselnormalform gegeben sein. Sei Δ diese Klauselmenge. Dann erhält man die zu Δ äquivalente DNNF als:

1. Besteht Δ nur aus einer einzigen Klausel α, dann gilt $dnnf(\Delta) = \alpha$.
2. Sonst ist $dnnf(\Delta) = \bigvee_\beta dnnf(\Delta_1|\beta) \wedge dnnf(\Delta_2|\beta) \wedge \beta$. Dabei bilden Δ_1 und Δ_2 eine Partition der Klauselmenge Δ und β ist eine Instanziierung der Atome, die sowohl in Δ_1 als auch in Δ_2 vorkommen.

Da eine Klausel eine Disjunktion von Literalen ist, liegt eine einzige Klausel trivialerweise immer in DNNF vor. Diesen Sachverhalt beschreibt 1. In 2. wird angegeben, wie man eine Klauselmenge, die aus mehr als einer Klausel besteht, in DNNF umwandelt. Dabei geht man so vor, dass man die Klauselmenge zuerst in zwei Mengen Δ_1 und Δ_2 unterteilt. Wie diese Partition auszusehen hat, wird nicht angegeben. Die Wahl der Partition beeinflusst die Komplexität der Umwandlung in DNNF jedoch stark. Dies werden wir später noch an einem Beispiel verdeutlichen. Für die so gebildete Partition von Δ in Δ_1 und Δ_2 bilden wir nun die in Satz 2.3.1 angegebene Menge der gemeinsamen Atome:

$$X = atome(\Delta_1) \cap atome(\Delta_2)$$

Diese Atome gehören zu den Atomen, die verhindern, dass Δ in DNNF vorliegt. Daher entfernen wir diese Atome durch Konditionierung aus der Klauselmenge. Um keine

Information der Klauselmenge zu verlieren, wird die Konditionierung bezüglich aller möglichen Instanziierungen der Atome in X durchgeführt. Man erkennt leicht, dass dieser Algorithmus sehr aufwändig ist. Besonders die Bildung aller Instanziierungen der Atome in X und die Konditionierung von Δ durch all diese Instanziierungen vergrößert die DNNF. Befinden sich in der Menge der gemeinsamen Atome von Δ_1 und Δ_2 n Atome, so werden 2^n Instanziierungen gebildet. Betrachten wir hierzu ein Beispiel.

Beispiel 2.3.2 ***(naiver Umwandlungsalgorithmus)*** *Betrachten wir erneut den Schaltkreis aus Abbildung 1.1. Die folgende Klauselmenge Δ beschreibt diesen Schaltkreis:*

$$\begin{aligned} F = \{&abInv1 \vee \neg a \vee \neg b,\\ &abInv1 \vee a \vee b,\\ &abInv2 \vee \neg b \vee \neg c,\\ &abInv2 \vee b \vee c\} \end{aligned}$$

Diese Klauselmenge liegt nicht in DNNF vor, da z.B. das Atom b in allen Klauseln vorkommt. Wandeln wir also Δ mit dem oben beschriebenen naiven Algorithmus in DNNF um. Da Δ mehr als eine Klausel enthält, müssen wir Δ zuerst in zwei Klauselmengen Δ_1 und Δ_2 aufteilen. Eine Möglichkeit dafür ist diese Aufteilung:

$$\begin{aligned} \Delta_1 = \{&abInv1 \vee \neg a \vee \neg b,\\ &abInv1 \vee a \vee b\} \end{aligned}$$

$$\begin{aligned} \Delta_2 = \{&abInv2 \vee \neg b \vee \neg c,\\ &abInv2 \vee b \vee c\} \end{aligned}$$

Daraus ergibt sich die Menge X der gemeinsamen Atome als:

$$X = atome(\Delta_1) \cap atome(\Delta_2) = \{b\}$$

Mögliche X-Instanziierungen sind also $\beta_1 = b$ und $\beta_2 = \neg b$. Mit diesen X-Instanziierungen können wir nun eine zur betrachteten Klauselmenge gehörige DNNF bilden.

$$\begin{aligned} dnnf(F) =& (\Delta_1|b \wedge \Delta_2|b \wedge b) \vee (\Delta_1|\neg b \wedge \Delta_2|\neg b \wedge \neg b)\\ =& [(abInv1 \vee \neg a) \wedge (abInv2 \vee \neg c) \wedge b] \vee\\ & [(abInv1 \vee a) \wedge (abInv2 \vee c) \wedge \neg b] \end{aligned}$$

Bei diesem Beispiel war die Umwandlung der Klauselmenge in DNNF trotz des naiven Algorithmus wenig aufwändig. Das liegt einerseits daran, dass wir nur eine sehr kleine

Klauselmenge betrachtet haben. Andererseits hätte schon eine weniger geschickte Wahl der Mengen Δ_1 und Δ_2 zu einer wesentlich umständlicheren Umwandlung geführt. Das nächste Beispiel zeigt, dass schon kleine Beispiele bei der Umwandlung mit Hilfe des naiven Algorithmus sehr rechenintensiv werden können. Es sei im Voraus bemerkt, dass es sich auch bei dem nächsten Beispiel noch um ein sehr kleines Beispiel handelt. Die in der Praxis auftretenden Klauselmengen sind um ein Vielfaches größer.

Beispiel 2.3.3 *Sei der folgende Schaltkreis gegeben: Die folgende Klauselmenge* Δ *be-*

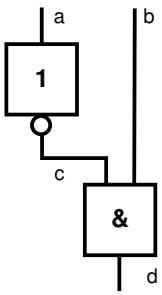


Abbildung 2.1: Beispielschaltkreis mit einem Inverter und einem AND-Gatter

schreibt das Verhalten dieses Schaltkreises:

$$\begin{aligned}\Delta = \{&abInv \vee \neg a \vee \neg c,\\ &abInv \vee a \vee c,\\ &abAnd \vee \neg c \vee \neg b \vee d,\\ &abAnd \vee c \vee \neg d,\\ &abAnd \vee b \vee \neg d\}\end{aligned}$$

Benutzen wir nun den naiven Algorithmus um diese Klauselmenge in DNNF umzuwandeln. Dazu teilen wir Δ *nun in disjunkte Teilmengen* Δ_1 *und* Δ_2 *auf. Eine Möglichkeit dies zu tun, sieht wie folgt aus:*

$$\begin{aligned}\Delta_1 = \{&abInv \vee \neg a \vee \neg c,\\ &abInv \vee a \vee c\}\end{aligned}$$

$$\begin{aligned}\Delta_2 = \{&abAnd \vee \neg c \vee \neg b \vee d,\\ &abAnd \vee c \vee \neg d,\\ &abAnd \vee b \vee \neg d\}\end{aligned}$$

Damit ergibt sich die Menge X der gemeinsamen Atome von Δ_1 und Δ_2 als:

$$X = \{c\}$$

Daraus ergibt sich durch den naiven Umwandlungsalgorithmus:

$$\begin{aligned} dnnf(\Delta) &= (dnnf(\Delta_1|c) \wedge dnnf(\Delta_2|c) \wedge c) \vee (dnnf(\Delta_1|\neg c) \wedge dnnf(\Delta_2|\neg c) \wedge \neg c) \\ &= (dnnf(\{abInv \vee \neg a\}) \wedge dnnf(\{abAnd \vee \neg b \vee d, abAnd \vee b \vee \neg d\}) \wedge c) \vee \\ &\quad (dnnf(\{abInv \vee a\}) \wedge dnnf(\{abAnd \vee \neg d, abAnd \vee b \vee \neg d\}) \wedge \neg c) \\ &= ((abInv \vee \neg a) \wedge dnnf(\{abAnd \vee \neg b \vee d, abAnd \vee b \vee \neg d\}) \wedge c) \vee \\ &\quad ((abInv \vee a) \wedge dnnf(\{abAnd \vee \neg d, abAnd \vee b \vee \neg d\}) \wedge \neg c) \end{aligned}$$

In einer Nebenrechnung berechnen wir die noch nicht in DNNF vorliegenden Teilformeln $dnnf(\{abAnd \vee \neg b, abAnd \vee b \vee \neg d\}) \wedge c)$ und $dnnf(\{abAnd \vee \neg d, abAnd \vee b \vee \neg d\}) \wedge \neg c)$. Zur Vereinfachung verwenden wir folgende Bezeichnungen:

$$\begin{aligned} \Gamma &= \{abAnd \vee \neg b \vee d, abAnd \vee b \vee \neg d\} \\ \Theta &= \{abAnd \vee \neg d, abAnd \vee b \vee \neg d\} \end{aligned}$$

Berechnung von $dnnf(\Gamma) = dnnf(\{abAnd \vee \neg b \vee d, abAnd \vee b \vee \neg d\})$: Als erstes Teilen wir Γ in zwei disjunkte Teilmenen auf. Die einzige Möglichkeit hierfür ist die folgende.

$$\begin{aligned} \Gamma_1 &= abAnd \vee \neg b \vee d \\ \Gamma_2 &= abAnd \vee b \vee \neg d \end{aligned}$$

Die Menge X_Γ der gemeinsamen Atome von Γ_1 und Γ_2 ergibt sich als $X_\Gamma = \{abAnd, b, d\}$. Daraus erhalten wir die möglichen Instanziierungen:

$$\begin{aligned} \beta_1 &= abAnd \wedge b \wedge d, \\ \beta_2 &= abAnd \wedge b \wedge \neg d, \\ \beta_3 &= abAnd \wedge \neg b \wedge d, \\ \beta_4 &= abAnd \wedge \neg b \wedge \neg d \\ \beta_5 &= \neg abAnd \wedge b \wedge d, \\ \beta_6 &= \neg abAnd \wedge b \wedge \neg d, \\ \beta_7 &= \neg abAnd \wedge \neg b \wedge d, \\ \beta_8 &= \neg abAnd \wedge \neg b \wedge \neg d \end{aligned}$$

Damit können wir eine DNNF für Γ bestimmen.

$$\begin{aligned}
dnnf(\Gamma) &= \bigvee_{\beta_i}(dnnf(\Gamma_1|\beta_i) \wedge dnnf(\Gamma_2|\beta_i) \wedge \beta_i) \\
&= (true \wedge true \wedge abAnd \wedge b \wedge d) \vee \\
&\quad (true \wedge true \wedge abAnd \wedge b \wedge \neg d) \vee \\
&\quad (true \wedge true \wedge abAnd \wedge \neg b \wedge d) \vee \\
&\quad (true \wedge true \wedge abAnd \wedge \neg b \wedge \neg d) \vee \\
&\quad (true \wedge true \wedge \neg abAnd \wedge b \wedge d) \vee \\
&\quad (false \wedge true \wedge \neg abAnd \wedge b \wedge \neg d) \vee \\
&\quad (true \wedge false \wedge \neg abAnd \wedge \neg b \wedge d) \vee \\
&\quad (true \wedge true \wedge \neg abAnd \wedge \neg b \wedge \neg d) \\
&= (abAnd \wedge b \wedge d) \vee (abAnd \wedge b \wedge \neg d) \vee \\
&\quad (abAnd \wedge \neg b \wedge d) \vee (abAnd \wedge \neg b \wedge \neg d) \vee \\
&\quad (\neg abAnd \wedge b \wedge d) \vee (\neg abAnd \wedge \neg b \wedge \neg d)
\end{aligned}$$

Berechnen wir nun eine DNNF von $\Theta = \{abAnd \vee \neg d, abAnd \vee b \vee \neg d\}$. Dazu unterteilen wir Θ in zwei disjunkte Teilmengen. Auch in diesem Fall gibt es dfür nur eine Möglichkeit.

$$\begin{aligned}
\Theta_1 &= abAnd \vee \neg d \\
\Theta_2 &= abAnd \vee b \vee \neg d
\end{aligned}$$

Daraus können wir die Menge $X_\Theta = \{abAnd, d\}$ der gemeinsamen Atome von Θ_1 und Θ_2 bestimmen. Die möglichen Instanziierungen bezeichen wir wieder mit β:

$$\begin{aligned}
\beta_1 &= abAnd \wedge d, \\
\beta_2 &= abAnd \wedge \neg d, \\
\beta_3 &= \neg abAnd \wedge d, \\
\beta_4 &= \neg abAnd \wedge \neg d
\end{aligned}$$

Mit diesen Informationen können wir nun die Teilformel Θ in DNNF umwandlen. Wir erhalten:

$$\begin{aligned}
dnnf(\Theta) &= \bigvee_{\beta_i}(dnnf(\Theta_1|\beta_i) \wedge dnnf(\Theta_2|\beta_i) \wedge \beta_i) \\
&= (true \wedge true \wedge abAnd \wedge d) \vee (true \wedge true \wedge abAnd \wedge \neg d) \vee \\
&\quad (false \wedge b \wedge \neg abAnd \wedge d) \vee (true \wedge true \wedge \neg abAnd \wedge \neg d) \\
&= (abAnd \wedge d) \vee (abAnd \wedge \neg d) \vee (\neg abAnd \wedge \neg d)
\end{aligned}$$

Nun haben wir alle Teile für die DNNF unserer Ausgangsklauselmenge Δ berechnet und können die Ergebnisse von $dnnf(\Gamma)$ und $dnnf(\Theta)$ oben einsetzen.

$$\begin{aligned} dnnf(\Delta) = ((&abInv \vee \neg a) \wedge ((abAnd \wedge b \wedge d) \vee (abAnd \wedge b \wedge \neg d) \vee \\ &(abAnd \wedge \neg b \wedge d) \vee (abAnd \wedge \neg b \wedge \neg d) \vee \\ &(\neg abAnd \wedge b \wedge d) \vee (\neg abAnd \wedge \neg b \wedge \neg d)) \wedge c) \vee \\ &((abInv \vee a) \wedge ((abAnd \wedge d) \vee (abAnd \wedge \neg d) \vee (\neg abAnd \wedge \neg d)) \wedge \neg c) \end{aligned}$$

Schon dieses sehr kleine Beispiel verdeutlicht die Probleme des naiven Umwandlungsalgorithmus. Man kann sich leicht vorstellen, dass die Berechnung der DNNF bei ungeschickter Wahl von Δ_1 und Δ_2 wesentlich komplexer wird. Daher macht es Sinn, sich Gedanken über andere Algorithmen zur Gewinnung der DNNF zu machen. Bevor wir uns jedoch mit Alternativen [DH05] zum naiven Algorithmus beschäftigen, betrachten eine Unterform der DNNF.

2.4 d-DNNF

Eine Verschärfung der DNNF stellt die d-DNNF [Dar00] dar. Dabei handelt es sich um Formeln, die in DNNF vorliegen und zusätzlich noch die Determinismuseigenschaft erfüllen.

Definition 2.4.1 ***(d-DNNF)*** *Eine Formel F liegt in d-DNNF vor, wenn F in DNNF vorliegt und zusätzlich noch die Determinismuseigenschaft hat. Das heißt, dass die Disjunktionsglieder jeder Disjunktion paarweise logisch inkonsistent sind.*

Die logische Inkonsistenz der einzelnen Disjunktionsglieder bedeutet, dass die konjunktive Verknüpfung dieser Disjunktionsglieder unerfüllbar ist. Es gibt also kein Modell, dass mehrere Disjunktionsglieder gleichzeitig wahr macht.

Beispiel 2.4.2 ***(d-DNNF)*** *Die Formel*

$$(\neg a \wedge \neg b) \vee a$$

ist nicht in d-DNNF, weil die beiden Disjunktionsglieder $\neg a \wedge \neg b)$ und a nicht logisch inkonsistent sind. Die folgende Formel dagegen liegt in d-DNNF vor:

$$(\neg a \wedge \neg b) \vee (a \wedge b)$$

2.5 Anwendungsbeispiel: DNNF in der Diagnose

Es wurde bereits angedeutet, dass sich die DNNF als Ziel der Präkompilation eignet. Eine mögliche Anwendung hierfür stellt die Diagnose dar, mit der sich auch Adnan Darwiche

[Dar98] umfassend beschäftigt hat. Im Folgenden wollen wir, an Hand von einem Beispiel aus der Diagnose, den Einsatz der DNNF und der besprochenen Operationen auf Formeln in DNNF betrachten. Führen wir jedoch zuerst einige Bezeichnungen ein.

Definition 2.5.1 *Eine Systembeschreibung (Δ, A, O) ist ein Tripel, bestehend aus der Klauselmenge Δ, die den Schaltkreis beschreibt, der Menge der Schaltglieder A und der Menge der Ein- und Ausgabevariablen O. Eine Beobachtung α ist eine O-Instanziierung und beschreibt das Verhalten des Schaltkreises. Häufig wird die Beobachtung auch als fehlerhaftes Verhalten bezeichnet. Eine Diagnose einer Systembeschreibung (Δ, A, O) und einer Beobachtung α ist eine A-Instanziierung, die konsistent zu $(\Delta \cup \{\alpha\})$ ist.*

Bezogen auf Beispiel 2.1 besteht die Menge A aus $A = \{abInv, abAnd\}$ und die Menge O aus $O = \{a, b, d\}$. Betrachten wir nun, was eine kompilierte Systembeschreibung ist:

Definition 2.5.2 ***(kompilierte Systembeschreibung)*** *Sei eine Systembeschreibung (Δ, A, O) gegeben. Eine kompilierte Systembeschreibung $KSB(\Delta, A, O)$ ist eine $A \cup O$-Formel, die die folgenden Eigenschaften erfüllt:*

1. *$KSB(\Delta, A, O)$ ist eine gleichmäßige DNNF.*
2. *für alle $A \cup O$-Formeln β gilt: $\Delta \models \beta$ gdw $KSB(\Delta, A, O) \models \beta$*

Diese Definition beschreibt, wie Darwiche bei der Diagnose vorgeht. In der Offline-Phase wandelt er die Systembeschreibung zuerst in DNNF um und entfernt dann alle Atome, die nicht in $A \cup O$ enthalten sind. Dies geschieht mittels der Projektion. Zum Schluss wird das Ergebnis der Projektion noch in eine gleichmäßige DNNF umgewandelt. Durch das Durchführen dieser Operationen ist sichergestellt, dass die 2. Eigenschaft aus Definition 2.5.2 erfüllt ist. Dass man diese kompilierten Systembeschreibung tatsächlich für die Diagnose verwenden kann, besagt der folgende Satz.

Satz 2.5.3 *Sei (Δ, A, O) eine Systembeschreibung und α ein beobachtetes Verhalten des entsprechenden Schaltkreises. Dann gilt:*

$$Diagnoses(\Delta \cup \alpha) = Diagnoses(KSB(\Delta, A, O) \wedge \alpha)$$

In der Online-Phase kann die KSB dazu verwendet werden, Erklärungen für ein fehlerhaftes Verhalten des Schaltkreises zu generieren. Da die Online-Phase nur die bereits vorgestellten Operationen auf der DNNF benutzt, kann die Erklärung in polynomieller Zeit erstellt werden. Der folgende Satz beschreibt, wie man aus der vorliegenden KSB und einer Beobachtung α eine Erklärung erhält.

Satz 2.5.4 *Sei $\Gamma = KSB(\Delta, A, O)$ und α eine beobachtetes Verhalten des Schaltkreises gegeben. Dann gilt:*

- *$MCard(Project(\Gamma|\alpha, A))$ gibt die minimale Fehleranzahl des Schaltkreises an.*

- $\mathsf{Models}(\mathsf{Minimize}(\mathsf{Project}(\Gamma|\alpha, A)))$ *liefert die minimalen Diagnosen des Systems.*

Wir erhalten also alle minimalen Erklärungen für das beobachtete Verhalten, indem wir $KSB(\Delta, A, O)$ durch α konditionieren, anschließend auf A projizieren und danach die Minimize und die Models Operation anwenden. Bei diesen Operationen handelt es sich tatsächlich maximal um polynomielle Operationen. Daher können alle minimalen Erklärungen in polynomieller Zeit berechnet werden

Betrachten wir nun, wie dieser Diagnoseansatz bei einem konkreten Beispiel funktioniert. Im Folgenden bezeichnet $\mathsf{smooth}(F)$ das Ergebnis der Umwandlung der DNNF F in eine gleichmäßige DNNF.

Beispiel 2.5.5 *Sei der Schaltkreis aus Beispiel 2.3.3 und die Beobachtung $\alpha = \neg a \wedge b \wedge \neg d$ gegeben. Die Menge der Schaltelemente ist $A = \{abInv, abAnd\}$ und die Menge der Ein- und Ausgabevariablen ist als $O = \{a, b, d\}$ gegeben. Damit können wir nun $KSB(\Delta, A, O)$ bilden. In Beispiel 2.3.3 haben wir Δ bereits in DNNF umgewandelt. Das Ergebnis war:*

$$\begin{aligned} dnnf(\Delta) = {} & ((abInv \vee \neg a) \wedge ((abAnd \wedge b \wedge d) \vee (abAnd \wedge b \wedge \neg d) \vee \\ & (abAnd \wedge \neg b \wedge d) \vee (abAnd \wedge \neg b \wedge \neg d) \vee \\ & (\neg abAnd \wedge b \wedge d) \vee (\neg abAnd \wedge \neg b \wedge \neg d)) \wedge c) \vee \\ & ((abInv \vee a) \wedge ((abAnd \wedge d) \vee (abAnd \wedge \neg d) \vee (\neg abAnd \wedge \neg d)) \wedge \neg c) \end{aligned}$$

Diese DNNF müssen wir nun auf die Menge $(A \cup O)$ projizieren:

$$\begin{aligned} & Project(dnnf(\Delta), A \cup O) \\ = {} & ((abInv \vee \neg a) \wedge ((abAnd \wedge b \wedge d) \vee (abAnd \wedge b \wedge \neg d) \vee \\ & (abAnd \wedge \neg b \wedge d) \vee (abAnd \wedge \neg b \wedge \neg d) \vee \\ & (\neg abAnd \wedge b \wedge d) \vee (\neg abAnd \wedge \neg b \wedge \neg d))) \vee \\ & ((abInv \vee a) \wedge ((abAnd \wedge d) \vee (abAnd \wedge \neg d) \vee (\neg abAnd \wedge \neg d))) \end{aligned}$$

Im nächsten Schritt wird das Ergebnis von $\mathsf{Project}(dnnf(\Delta), A \cup O)$ in eine gleichmäßige DNNF umgewandelt.

$$\begin{aligned} & smooth(\mathsf{Project}(dnnf(\Delta), A \cup O)) \\ = {} & (((abInv \wedge \neg a) \vee (abInv \wedge a) \vee (\neg abInv \wedge \neg a)) \wedge \\ & ((abAnd \wedge b \wedge d) \vee (abAnd \wedge b \wedge \neg d) \vee \\ & (abAnd \wedge \neg b \wedge d) \vee (abAnd \wedge \neg b \wedge \neg d) \vee \\ & (\neg abAnd \wedge b \wedge d) \vee (\neg abAnd \wedge \neg b \wedge \neg d))) \vee \\ & (((abInv \wedge a) \vee (abInv \wedge \neg a) \vee (\neg abInv \wedge a)) \wedge \\ & ((abAnd \wedge d) \vee (abAnd \wedge \neg d) \vee (\neg abAnd \wedge \neg d))) \end{aligned}$$

Damit haben wir $KSB(\Delta, A, O) = \mathsf{smooth}(\mathsf{Project}(dnnf(\Delta), A \cup O))$ *gebildet. Im Folgenden bezeichnet* Γ *die kompilierte Systembeschreibung* $KSB(\Delta, A, O)$*. Die Off line-Phase ist abgeschlossen und wir generieren nun in der Online-Phase alle minimalen Erklärung für die Beobachtung* $\alpha = \neg a \wedge b \wedge \neg d$*. Dafür müssen wir* $\mathsf{Models}(\mathsf{Minimize}(\mathsf{Project}(\Gamma|\alpha, A)))$ *bilden.:*

Im ersten Schritt der Online-Phase bilden wir:

$$\begin{aligned}\Gamma|\alpha = &((abInv \vee \neg abInv) \wedge abAnd) \vee \\ &(abInv \wedge (abAnd \vee \neg abAnd)) \\ = &(abAnd \vee abInv)\end{aligned}$$

Da das Ergebnis bereits nur noch Atome aus A enthält, können wir uns die Projektion auf A sparen. Um im nächsten Schritt die Minimize *Operation anwenden zu können, wenden wir* $\Gamma|\alpha$ *in eine gleichmäßige DNNF um:*

$$\mathsf{smooth}(\Gamma|\alpha) = (abAnd \wedge abInv) \vee (abAnd \wedge \neg abInv) \vee (\neg abAnd \wedge abInv)$$

Nun können wir die Minimize *Operation anwenden, um sicherzustellen, dass wir später nur minimale Erklärungen erhalten.*

$$\mathsf{Minimize}(\mathsf{smooth}(\Gamma|\alpha)) = (abAnd \wedge \neg abInv) \vee (\neg abAnd \wedge abInv)$$

Im letzten Schritt bestimmen wir die Erklärungen und erhalten:

$$\begin{aligned}\mathsf{Models}(\mathsf{Minimize}(\mathsf{smooth}(\Gamma|\alpha))) = &\{\{abAnd = true, abInv = false\}, \\ &\{abAnd = false, abInv = true\}\}\end{aligned}$$

Bei diesen beiden Erklärungen handelt es sich tatsächlich um die minimalen Erklärungen für das beobachtete Verhalten. Man erkennt leicht, dass die Minimize*-Operation gerade die nichtminimale Erklärung* $\{abAnd = true, abInv = true\}$ *entfernt hat.*

3 Davis-Putman-Logemann-Loveland

3.1 Zusammenhang zwischen DPLL und DNNF

Viele Algorithmen, mit denen eine gegebene Klauselmenge in DNNF umgewandelt werden kann, basieren auf dem Davis-Putman-Logemann-Loveland Algorithmus. Daher wird der DPLL-Algorithmus in diesem Kapitel genauer betrachtet. Unter [DH05] und [Dar04] findet man weiterführende Informationen zur DPLL-basierten Umwandlung von Klauselmengen in DNNF. Außerdem untersuchen wir einige Verzweigungsheuristiken für den DPLL-Algorithmus. Diese Verzweigungsheuristiken werden später auch bei der Umwandlung von Klauselmengen in DNNF eine Rolle spielen.

3.2 DPLL Algorithmus

Der Davis-Putnam-Logemann-Loveland Algorithmus [DP60], [DLL62] ist ein Entscheidungsverfahren für das Erfüllbarkeitsproblem in der Aussagenlogik. Der vollständige Algorithmus basiert auf Backtracking und arbeitet auf Klauselmengen. Obwohl er bereits 1962 vorgestellt wurde, ist er auch für aktuelle SAT-Solver noch von großer Bedeutung.

3.2.1 Algorithmus

Der DPLL-Algorithmus ist ein tableaubasierter Algorithmus. Während der Berechnungen ist stets nur ein Zweig aktiv. Da alle anderen Zweige nicht gespeichert werden müssen, ist der DPLL-Algorithmus sehr speichersparend. Seine wichtigste Regel ist die Verzweigungsregel. Dabei wird ein Atom ausgewählt und ihm ein Wahrheitswert zugewiesen. Die betrachtete Klauselmenge wird dieser Wahrheitsbelegung entsprechend vereinfacht. Dabei wird jede Klausel, die durch die Wahrheitsbelegung wahr wird, gelöscht. Außerdem wird jedes Literal, das durch die aktuelle Wahrheitsbelegung falsch wird, entfernt. Die so entstehende Klauselmenge wird auf Erfüllbarkeit überprüft. Ist die vereinfachte Klauselmenge erfüllbar, so auch die ursprüngliche Klauselmenge. Entsteht durch das Löschen von Literalen eine leere Klausel, so ist die Klauselmenge unerfüllbar. Dann wird der gegenteilige Wahrheitswert für das betrachtete Atom angenommen. Da die Anzahl der Verzweigungen durch die Anzahl der in der Klauselmenge vorkommenden Atome begrenzt ist, ist sichergestellt, dass der DPLL-Algorithmus terminiert. Findet man eine Belegung aller Variablen, die alle Klauseln wahr macht, so hat man ein Modell

der Klauselmenge gefunden. Die Unerfüllbarkeit einer Klauselmenge kann jedoch nur durch Konstruktion des gesamten Tableaus nachgewiesen werden.

Darüber hinaus benutzt DPLL die Weitergabe von Einerklauseln (Unit Propagation) und die Elimination von pur auftretenden Literalen (Pure Literal Elimination).

Eine Einerklausel ist eine Klausel, die nur ein einziges Literal enthält. Da eine Klauselmenge einer konjunktiven Verknüpfung von Klauseln entspricht, muss jede einzelne Klausel wahr werden, damit die gesamte Klauselmenge wahr wird. Dies gilt insbesondere auch für Einerklauseln. Da diese nur aus einem einzigen Literal bestehen, muss dieses Literal zwingenderweise wahr sein. Daher kann die betrachtete Klauselmenge vereinfacht werden, indem das entsprechende Literal in der gesamten Klauselmenge auf wahr gesetzt wird. Bei diesen Literalen ist eine Fallunterscheidung also unnötig. Das beschriebene Vorgehen wird als Weitergabe von Einerklauseln bezeichnet.

Beispiel 3.2.1 *Sei folgende Klauselmenge gegeben:*

$$\{\neg a \vee b \vee c, \neg b \vee d, a \vee \neg c, b\}$$

Dabei ist die Klausel b eine Einerklausel. Daher ist b in jedem Modell der Klauselmenge wahr. Die obige Klauselmenge kann dementsprechend vereinfacht werden. Wir belegen b mit wahr und vereinfachen die Klauselmenge. Die erste Klausel enthält b und wird daher wahr. In der zweiten Klausel ist die Negation von b enthalten. Da b mit wahr belegt ist, ist die Negation von b falsch und kann daher aus der Klausel gelöscht werden. Die dritte Klausel bleibt unverändert. Damit ergibt sich:

$$\{d, a \vee \neg c, b\}$$

Durch die Weitergabe von Einerklauseln werden unnötige Verzweigungen eingespart. Daher ist es eine sinnvolle Erweiterung des DPLL-Algorithmus.

Ein Literal heißt pur, wenn es in der gesamten Klauselmenge entweder nur positiv oder nur negativ vorkommt. Belegt man dieses Literal mit wahr, kann man alle Klauseln, die dieses Literal enthalten, löschen. Diese Regel bezeichnet man als Elimination von pur auftretenden Literalen.

Beispiel 3.2.2 *Sei folgende Klauselmenge gegeben:*
$\{\neg a \vee b \vee c, \neg b \vee d, d \vee c, b\}$
Hier ist das Literal c pur, da es nur positiv vorkommt. Daher kann man c mit wahr belegen und die Klauselmenge vereinfachen. Dabei fallen die erste und die dritte Klausel weg. Damit ergibt sich:
$\{\neg b \vee d, b\}$

Die so entstandene Klauselmenge ist erfüllbarkeitsäquivalent zur ursprünglichen Klauselmenge.

3.3 Verzweigungsheuristiken für DPLL

Es ist klar, dass die Wahl der Atome, nach denen bei DPLL die Fallunterscheidung vorgenommen wird, die Größe des Suchraumes beeinflusst. Daher wurde eine Vielzahl von Heuristiken entwickelt, die versuchen die Wahl des Verzweigungsliterals zu optimieren. Wir wollen an dieser Stelle nur kurz auf drei dieser Heuristiken eingehen. Der interessierte Leser findet unter [DGP04] und [ZM02] weitere Verzweigungsheuristiken. Dort findet man unter anderem auch Heuristiken, die auf bestimmte Anwendungsdomänen zugeschnitten sind. Solche Heuristiken wollen wir hier nicht betrachten. Im Folgenden wollen wir einige syntaktische Verzweigungsheuristiken für DPLL betrachten. Informationen hierzu findet man unter [Zha].

3.3.1 Einfaches Zählen von Literalen

Die einfachste Verzweigungsheuristik für DPLL erhält man durch Zählen von Literalen. Man sieht sofort, dass es Sinn macht Literale für eine Fallunterscheidung auszuwählen, die besonders häufig in der Klauselmenge vorkommen. Bei dieser Heuristik ermitteln wir pro Atom zwei Werte:

$$CP(X) = \textit{ Anzahl der Vorkommen von } X \textit{ in nichtsubsumierten Klauseln}$$
$$CN(X) = \textit{ Anzahl der Vorkommen von } \neg X \textit{ in nichtsubsumierten Klauseln}$$

Die Werte für $CN(X)$ und für $CP(X)$ können sehr leicht berechnet werden. Diese Werte benutzen wir nun, um das nächste Literal zu bestimmen, nach dem die Fallunterscheidung vorgenommen werden soll. Hier gibt es mehrere Möglichkeiten.

DLCS:

Dies steht für Dynamic Largest Combined Sum. Man berechnet für jede in der betrachteten Klauselmenge enthaltene aussagenlogische Variable die kombinierte Summe $CN(X) + CP(X)$. Für die Fallunterscheidung wählt man die Variable mit der höchsten kombinierten Summe. Außerdem regelt diese Heuristik noch, welcher Zweig der Fallunterscheidung zuerst untersucht wird. Ist $CP(X) \geq CN(X)$, so betrachten wir den Zweig für X zuerst. Sonst den Zweig für $\neg X$. DLCS wählt also das Atom aus, das in der betrachteten Klauselmenge am häufigsten vorkommt. Das Vorzeichen der Vorkommen ist nur für die Wahl des Zweiges der zuerst betrachtet wird, von Interesse.

DLIS:

Ist die zweite Möglichkeit, die ermittelten Werte $CP(X)$ und $CN(X)$ zu benutzen. Dies steht für Dynamic Largest Individual Sum. Hier wird die Variable für die nächste Fallunterscheidung ausgewählt, die den höchsten CN oder den höchsten CP Wert hat. Welcher Zweig zuerst untersucht wird, wird wie bei DLCS bestimmt.

Obwohl es sich bei diesen Heuristiken um sehr einfache Heuristiken handelt, sind sie doch sehr effektiv. Das werden wir auch in einem später betrachteten Beispiel sehen.

3.3.2 MOMS

Die nächste Heuristik, die wir hier betrachten ist die MOMS Heuristik [CA93]. Die Bezeichnung MOMS steht für Maximum Occurrences in Clauses of Minimum Size und weist direkt auf die Art der Heuristik hin. Hier werden die Variablen für Fallunterscheidungen bevorzugt behandelt, die am häufigsten in kleinen Klauseln vorkommen. Man wählt das Literal aus, das den folgenden Wert maximiert:

$$2^k(f^*(l) + f^*(\neg l)) + f^*(l)f^*(\neg l)$$

Dabei bezeichnet $f^*(l)$ die Anzahl der Vorkommen des Atoms l in den kürzesten nichtresolvierten Klauseln. Bei k handelt es sich um einen Tuningparameter. Die Idee hinter dieser Heuristik ist die Tatsache, dass eine Fallunterscheidung nach den Variablen, die am häufigsten in kleinen Klauseln vorkommen, sehr häufig Einerklauseln erzeugt oder zur leeren Klausel führt. Da die Weitergabe von Einerklauseln den DPLL Baum klein hält, ist es wünschenswert, dies möglichst häufig durchzuführen. Die MOMS Heuristik ist sehr leicht zu berechnen und sorgt somit nur für einen geringen Overhead.

3.3.3 Einerklausel-Heuristik

Diese Heuristik [CA93] nutzt die Stärke der Weitergabe von Einerklauseln noch stärker aus. Da die Anwendung dieser Regel diese Klauselmenge stark vereinfacht, ist es sinnvoll, die Literale für die Fallunterscheidung zu wählen, die viele Einerklauseln erzeugen. Daher bestimmt diese Heuristik die genaue Anzahl der Einerklauseln, die das Splitting nach den einzelnen Literalen hervorruft und wählt dann das Literal mit der höchsten Anzahl aus. Leider ist die Bestimmung der genauen Anzahl von entstehenden Einerklauseln sehr aufwändig. Daher wird diese Heuristik eher selten benutzt.

3.3.4 Backbone-Heuristik

Bei der Backbone-Heuristik [DD01] werden die Atome für die Fallunterscheidung ausgewählt, die sehr wahrscheinlich im Backbone der betrachteten Klauselmenge liegen. Der Backbone einer Klauselmenge ist die Menge an Atomen, die in jedem Modell der Klauselmenge wahr sind. Verzweigt man nach Atomen, die zum Backbone der betrachteten Klauselmenge gehören, so führt dies dazu, dass in einem der ersten Zweige ein Modell gefunden wird. Diese Idee ist mit der Idee der initialen Interpretation verwandt, die wir in einem späteren Kapitel untersuchen werden.

3.3.5 BOHM's Heuristik

Diese Heuristik wurde nach einem Theorembeweiser benannt, den M. Böhm entworfen hat. Dabei wird das Literal l für die nächste Verzweigung ausgewählt, dass den folgenden Vektor maximiert.

$$H_i(l) = \alpha \cdot max\{h_i(l), h_i(\bar{l})\} + \beta \cdot min\{h_i(l), h_i(\bar{l})\}$$

$h_i(l)$ bezeichnet die Anzahl der nichtresolvierten Klauseln der Länge i, in denen l vorkommt. Die Länge einer Klausel wird hier durch die Anzahl der verbleibenden Literale dieser Klausel bestimmt. Die Werte für α und β werden mit 1 und 2 vorgeschlagen. Bei der BOHM's Heuristik werden Literale vorgezogen, die kleinere Klauseln wahr machen oder die kleinere Klauseln weiter verkleinern.

3.3.6 Jeroslow-Wang Heuristik

Bei der Jeroslow-Wang Heuristik wird für jedes Literal l der Wert

$$J(l) = \sum 2^{-|\omega|} \tag{3.1}$$

bestimmt. Dabei wird über die Klauseln ω summiert, in denen l enthalten ist. Anhand dieses J Wertes wird nun die Variable ausgesucht, nach der die nächste Fallunterscheidung vorgenommen wird.

Einseitige Jeroslow-Wang

Hier wird das Literal mit dem höchsten J-Wert ausgewählt. Literale, die oft in kleinen Klauseln vorkommen, werden bevorzugt für die nächste Verzweigung ausgewählt.

Zweiseitige Jeroslow-Wang

Hier wird die Variable X mit dem höchsten Wert für $J(X) + J(\neg X)$ ausgewählt. Auch diese Heuristik bewertet die Vorkommen von Atomen in kleineren Klauseln höher und zieht somit Atome vor, die häufig in kleineren Klauseln vorkommen. Bei der zweiseitigen Jeroslow-Wang Heuristik fließen jedoch Vorkommen von X und von $\neg X$ in die Auswahl des Atoms für die nächste Verzweigung ein. Dies sind nur einige der vielen

Verzweigungsheuristiken für DPLL. Im nächsten Kapitel untersuchen wir eine weitere Möglichkeit, Klauselmengen in DNNF umzuwandeln. Der dort vorgestellt Algorithmus baut auf dem DPLL-Algorithmus auf. Daher können wir im Anschluss auch den Nutzen der oben besprochen Heuristiken bei der Umwandlung einer Formel in DNNF untersuchen.

4 Umwandlung in DNNF mit DPLL-FTab

Unter [Wer06a] findet man die Beschreibung des DPLL-FTab Kalküls. Dabei handelt es sich um einen auf DPLL basierenden Tableaukalkül, mit dem sich unter anderem Klauselmengen in DNNF umwandeln lassen. Wie auch DPLL bearbeitet dieser Kalkül zu einem Zeitpunkt immer nur einen Zweig. Das hat den Vorteil, dass nur der aktuelle Zweig gespeichert werden muss. Der jeweils aktive Zweig wird im Folgenden durch Unterstreichen des zum aktiven Zweig gehörigen Blattes markiert. Dieser Kalkül kann auch für andere Dinge, z.B. für die Aufzählung aller Modelle einer Klauselmenge, benutzt werden. Wir werden jedoch die Regeln des Kalküls in der Form betrachten, wie sie zur Berechnung der DNNF nötig sind. Im Folgenden bezeichnet Δ die Klauselmenge, die in DNNF umgewandelt werden soll.

4.1 Beschreibung des Kalküls

Definition 4.1.1 ***(D-Tableau)*** *Ein D-Tableau ist ein geordneter Baum mit einer Beschriftungsfunktion, der die folgenden Bedingungen erfüllt:*

- *Jeder Knoten hat maximal zwei Nachfolgeknoten und*
- *die beiden Kinder eines Knotens sind stets dual zueinander.*

Die oben genannte Labelingfunktion weist jedem Knoten eine Literalbeschriftung zu. Diese Literalbeschriftung ist ein Literal, wahr oder falsch.

Definition 4.1.2 ***(Zustand, aktiver Zweig)*** *Ein Zustand ist ein Paar $\langle T, N\rangle$, wobei T ein D-Tableau und N ein aktives Blatt ist. Der zu diesem aktiven Blatt gehörige Zweig heißt aktiver Zweig.*

Definition 4.1.3 ***(geschlossener Zweig)*** *Ein Zweig ist genau dann geschlossen, wenn er einen mit false beschrifteten Knoten enthält. Ein Literal L heißt in einem Zweig definiert, wenn es einen Knoten in diesem Zweig gibt, der mit L oder $\neg L$ beschriftet ist.*

Definition 4.1.4 ***(offener Zweig, offener Knoten)*** *Ein nicht geschlossener Zweig heißt offen. Einen Knoten nennen wir genau dann offen, wenn er ein Blatt ist und der zu ihm gehörige Zweig offen ist.*

Definition 4.1.5 ***(Init)*** *Der initiale Zustand Init ist das D-Tableau, das nur aus einem einzigen Knoten besteht, der mit true beschriftet ist. Dieses Blatt ist auch gleichzeitig der aktive Zweig.*

Definition 4.1.6 ***(Verzweigungsregel)***
Vorbedingungen:

- *B ist der aktive Zweig,*
- *B ist nicht geschlossen,*
- *L ist ein Literal, dass in B nicht vorkommt und*
- *L kommt in einer Klausel C vor, für die $B \rightarrow C$ nicht gilt.*

Operation: *Hänge an das aktive Blatt zwei Kindknoten an. Der linke Kindknoten wird mit L und der rechte mit $\neg L$ beschriftet. Markiere den linken Kindknoten als das aktive Blatt.*

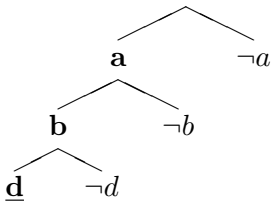
Diese Regel entspricht der Verzweigungsregel bei DPLL und nimmt somit eine Fallunterscheidung vor.

Definition 4.1.7 ***(Schließen)***
Vorbedingungen:

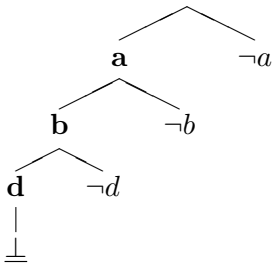
- *B ist der aktive Zweig,*
- *B ist nicht geschlossen und*
- *es gibt eine Klausel C in der Klauselmenge Δ, so dass $B \rightarrow \neg C$ gilt.*

Operation: *Hänge an das aktive Blatt einen Folgeknoten an, der mit false beschriftet ist. Markiere den neuen Knoten als aktiven Knoten.*

Beispiel 4.1.8 ***(Schließen)*** *Sei die Klausel $C = \neg a \vee \neg b \vee \neg d$ und der folgende Baum gegeben.*



Der linke Zweig sei der aktive Zweig. Die Beschriftung des aktiven Zweiges ist $a \wedge b \wedge d$. Diese Beschriftung macht die oben erwähnte Klausel C falsch. Daher kann der aktive Zweig geschlossen werden, indem false angehängt wird.



Definition 4.1.9 ***(Wechselregel)***
Vorbedingungen:

- *B ist der aktive Zweig,*
- *B ist nicht geschlossen,*
- *$\Delta|B$ ist in DNNF[1] ,*
- *B hat die Form $B_1 N_L B_2$,*
- *N_L hat einen offenen rechten Geschwisterknoten und*
- *kein Knoten von B_2 hat einen offenen rechten Geschwisterknoten.*

Operation: *Kennzeichne den offenen rechten Geschwisterknoten von N_L als aktiv.*

Diese Regel sorgt dafür, dass wir, nachdem wir die DNNF an einem Zweig erreicht haben, die anderen Zweige bearbeiten. Eine solche Regel gibt es bei DPLL nicht. DPLL testet Klauseln nur auf Erfüllbarkeit und terminiert, nachdem das erste Modell gefunden wurde. Die Wechselregel sorgt also dafür, dass wir den nächsten Zweig bearbeiten, wenn der aktive Zweig fertig ist. Wobei wir einen Zweig B als fertig bezeichnen, wenn $\Delta|B$ bereits in DNNF ist. Nun benötigen wir noch eine Regel, die es uns erlaubt, an einem anderen Zweig weiterzuarbeiten, wenn der aktuelle Zweig geschlossen ist. Diese Regel ist die Rücksetzen-Regel.

[1] $\Delta|B$ bezeichnet hier das Ergebnis der Konditionierung der Klauselmenge Δ durch die Beschriftung des Zweiges B. Siehe hierzu auch Definition 2.2.2 auf Seite 10.

Definition 4.1.10 ***(Rücksetzen)***
Vorbedingungen:

- *B ist der aktive Zweig,*
- *B ist geschlossen,*
- *B hat die Form $B_1 N_L B_2$,*
- *N_L hat einen offenen rechten Geschwisterknoten und*
- *kein Knoten von B_2 hat einen offenen rechten Geschwisterknoten.*

Operation: *Kennzeichne den offenen rechten Geschwisterknoten von N_L als aktiv.*

Mit diesen vier Regeln kann man jede beliebige Klauselmenge in DNNF umwandeln. Betrachten wir hierzu ein Beispiel:

Beispiel 4.1.11 *Wir wandeln die Klauselmenge Δ mit den oben beschriebenen Regeln in DNNF um. Δ ist gegeben durch:*

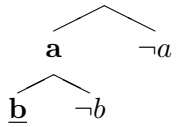
$$\begin{aligned}\Delta = \{&a \vee b \vee c,\\ &\neg a \vee \neg b \vee c,\\ &a \vee c \vee \neg d,\\ &b \vee \neg d \vee e,\\ &e \vee b\}\end{aligned}$$

Wir starten mit dem leeren Tableau und wenden als erstes die Verzweigungsregel an. Dafür wählen wir uns ein Atom. Hier verzweigen wir zuerst mit dem Atom a. Das liefert das folgende Tableau, in dem der linke Zweig aktiv ist:

Um zu überprüfen, ob unser aktiver Zweig bereits fertig ist, bilden wir $\Delta|a$ und erhalten die Folgende Klauselmenge:

$$\begin{aligned}\Delta|a = \{&\neg b \vee c,\\ &b \vee \neg d \vee e,\\ &e \vee b\}\end{aligned}$$

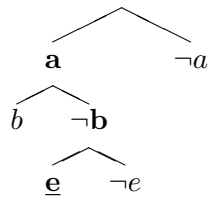
Diese Klauselmenge ist jedoch noch nicht in DNNF. Daher müssen wir am aktiven Zweig erneut die Verzweigungsregel anwenden. Diesmal wählen wir das Atom b für die Verzweigung.



Im nächsten Schritt überprüfen wir wieder, ob wir am aktiven Zweig fertig sind, indem wir $\Delta|\{a, b\} = c$ *bilden. Diese Klauselmenge ist in DNNF. Daher haben wir den aktiven Zweig fertiggestellt und können mit der Wechselregel am nächsten offenen Zweig weiterarbeiten. Der aktive Zweig ist nun der Zweig, der mit* $a, \neg b$ *beschriftet ist. Überprüfen wir zunächst wieder, ob wir am aktiven Zweig bereits fertig sind:*

$$\Delta|\{a, \neg b\} = \{\neg d \vee e, \\ e\}$$

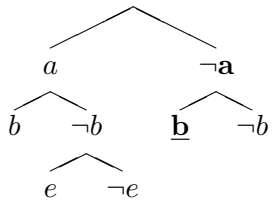
Diese Klauselmenge liegt noch nicht in DNNF vor. Wir benutzen erneut die Verzweigungsregel und verzweigen mit dem Atom e:



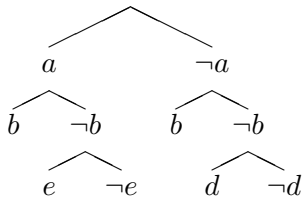
Der aktive Zweig ist mit $a, \neg b$ *und e beschriftet und ist bereits fertiggestellt, denn* $\Delta|\{a, \neg b, e\}$ *ist die leere Klauselmenge und ist damit bereits in DNNF. Durch Anwendung der Wechselregel wechseln wir zum nächsten Zweig. Nun ist der Zweig mit der Beschriftung* $a, \neg b$ *und* $\neg e$ *aktiv. An diesen Zweig können wir mit der Schließen-Regel false anhängen, da dieser Zweig die Klausel* $e \vee b$ *aus* Δ *falsch macht. Somit ist der aktive Zweig geschlossen. Mit der Rücksetzen-Regel wechseln wir zum nächsten offenen Zweig. Nun ist der Zweig, der mit* $\neg a$ *beschriftet ist, aktiv. Zur Überprüfung, ob dieser Zweig bereits fertiggestellt ist, bilden wir:*

$$\Delta|\neg a = \{b \vee c, \\ c \vee \neg d, \\ b \vee \neg d \vee e, \\ e \vee b\}$$

Da diese Klauselmenge nicht in DNNF ist, verzweigen wir erneut mit der Verzweigungsregel. Dafür wählen wir das Atom b und erhalten folgendes Tableau:



Nun sind wir mit dem derzeit aktiven Zweig fertig, denn $\Delta|\{\neg a, b\} = c \vee \neg d$ ist bereits in DNNF. Mit der Wechselregel wechseln wir erneut den aktiven Zweig. Wendet man diese Regeln weiter nacheinander an, so erhält man das folgende Tableau.



Wie man aus diesem Tableau die DNNF abliest, werden wir im nächsten Abschnitt betrachten.

4.2 Vom Tableau zur DNNF

Im vorigen Abschnitt haben wir gesehen, wie man aus einer gegebenen Klauselmenge ein Tableau erstellt. Es stellt sich nun die Frage, wie man aus dem Tableau die zur Klauselmenge gehörige DNNF abliest. Betrachten wir zunächst, wie man grundsätzlich eine Formel aus einem Baum ablesen kann. Die mit einem Baum assoziierte Formel ist rekursiv wie folgt definiert.

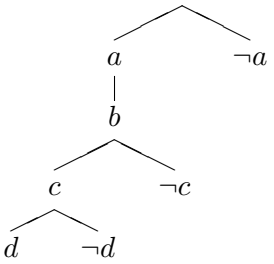
Definition 4.2.1 ***(die mit einem Knoten/Zweig/Tableau assoziierte Formel)***

- *Für alle Knoten N ist die mit N assoziierte Formel $formel(N)$ definiert als:*
  - *Ist N ein Blatt, so ist $formel(N) = literal(N)$, wobei $literal(N)$ das Literal ist, mit dem das Blatt N beschriftet ist.*
  - *Sonst gilt $formel(N) = literal(N) \wedge \bigvee_{N' \text{ ist Kindknoten von } N} formel(N')$*
- *Für alle Tableaux T ist die mit T assoziierte Formel $formel(T)$ über den formel Operator der Knoten definiert: $formel(T) = formel(root(T))$*

- *Für alle Zweige B ist die mit B assoziierte Formel $formel(B)$ definiert als:* $formel(B) = \bigwedge_{N \in B} literal(N)$

Die zu einem Blatt gehörige Formel entspricht also gerade dem Literal, das das Blatt beschriftet. Die Formel, die zu einem Zweig gehört, erhält man als eine Konjunktion der einzelnen Literale entlang des Zweiges. Und die zu einem Teilbaum gehörige Formel erhält man als Disjunktion der zu den einzelnen Zweigen gehörigen Formeln.

Beispiel 4.2.2 *Sei der folgende Baum gegeben:*



Die mit dem linken Zweig assozierte Formel ist:

$$a \wedge b \wedge c \wedge d$$

Die mit dem gesamten Tableau assoziierte Formel ergibt sich als:

$$(a \wedge (b \wedge ((c \wedge (d \vee \neg d)) \vee \neg c)) \vee \neg a$$

Alternativ kann man die Formel auch als Disjunktion aller Zweigformeln bilden:

$$(a \wedge b \wedge c \wedge d) \vee (a \wedge b \wedge c \wedge \neg d) \vee (a \wedge b \wedge \neg c) \vee \neg a$$

Damit wissen wir nun, wie man die zu einem beliebigen Tableau gehörige Formel erhält. Wir interessieren uns jedoch für die DNNF einer gegebenen Klauselmenge. Der oben beschriebene Kalkül wandelt eine Klauselmenge in ein D-Tableau um. Betrachten wir nun, wie man aus einem solchen D-Tableau die zur betrachteten Klauselmenge gehörige DNNF ablesen kann. Wenn im Folgenden aus dem Zusammenhang ersichtlich ist, dass wir von der zum Zweig B gehörigen Formel $formel(B)$ und nicht vom Zweig selbst sprechen, schreiben wir auch kurz B anstelle von $formel(B)$.

Definition 4.2.3 ***(Die mit einem D-Tableau assozierte Formel)*** *Sei Δ eine Klauselmenge und T ein D-Tableau, dass man durch die Anwendung der oben beschriebenen*

Regeln aus Δ erhält. Seien $B_1, \ldots, B_n$ die Zweige von T. Dann ist die mit diesem D-Tableau assoziierte Formel dnnf_{FTab} definiert durch:

$$\begin{aligned}
\mathsf{dnnf}_{FTab}(\Delta) &\equiv \Delta \wedge formel(T) \\
&\equiv \Delta \wedge \bigvee_{i \in \{1,\ldots,n\}} B_i \\
&\equiv \bigvee_{i \in \{1,\ldots,n\}} (\Delta \wedge B_i) \\
&\equiv \bigvee_{i \in \{1,\ldots,n\}} (B_i \wedge \Delta | B_i)
\end{aligned}$$

Satz 4.2.4 *Sei Δ eine Klauselmenge und T ein D-Tableau, dass man durch die Anwendung der oben beschriebenen Regeln aus Δ erhält. Und sei $\mathsf{dnnf}_{FTab}(\Delta)$ die mit T assoziierte Formel. Dann ist $\mathsf{dnnf}_{FTab}(\Delta)$ eine DNNF zu Δ.*

Im nächsten Beispiel werden wir die zur Klauselmenge Δ aus Beispiel 4.1.11 gehörige DNNF ablesen.

Beispiel 4.2.5 *Sei das zur Klauselmenge Δ gehörige D-Tableau wie am Ende von Beispiel 4.1.11 auf Seite 36 gegeben. Wir lesen nun die mit dem D-Tableau assoziierte*

DNNF ab.

$$
\begin{aligned}
\mathit{dnnf}_{FTab}(\Delta) \equiv & \bigvee_{i\in\{1,\ldots,6\}} (B_i \wedge \Delta|B_i) \\
\equiv\ & a \wedge b \wedge \Delta|\{a,b\}\vee \\
& a \wedge \neg b \wedge e \wedge \Delta|\{a,\neg b,e\}\vee \\
& a \wedge \neg b \wedge \neg e \wedge \Delta|\{a,\neg b,\neg e\}\vee \\
& \neg a \wedge b \wedge \Delta|\{\neg a,b\}\vee \\
& \neg a \wedge \neg b \wedge d \wedge \Delta|\{\neg a,\neg b,d\}\vee \\
& \neg a \wedge \neg b \wedge \neg d \wedge \Delta|\{\neg a,\neg b,\neg d\} \\
\equiv\ & a \wedge b \wedge c\vee \\
& a \wedge \neg b \wedge e\vee \\
& a \wedge \neg b \wedge \neg e \wedge false\vee \\
& \neg a \wedge b \wedge (c \vee \neg d)\vee \\
& \neg a \wedge \neg b \wedge d \wedge c \wedge e\vee \\
& \neg a \wedge \neg b \wedge \neg d \wedge c \wedge e \\
\equiv\ & a \wedge b \wedge c\vee \\
& a \wedge \neg b \wedge e\vee \\
& \neg a \wedge b \wedge (c \vee \neg d)\vee \\
& \neg a \wedge \neg b \wedge d \wedge c \wedge e\vee \\
& \neg a \wedge \neg b \wedge \neg d \wedge c \wedge e
\end{aligned}
$$

Mit Hilfe des oben beschriebenen Kalküls können wir die zu einer Klauselmenge gehörige DNNF recht einfach erzeugen. Nicht festgelegt ist allerdings, welche Atome für die Verzweigungsregel benutzt werden sollen. Bevor wir uns jedoch mit der Wahl der Atome für die Verzweigungsregel beschäftigen, wollen wir nun einige Erweiterungen des Kalküls betrachten.

4.3 Erweiterungen des DPLL-FTab Kalküls

Wir wollen uns nun mit einigen Erweiterungen des oben beschriebenen DPLL-FTab Kalküls beschäftigen. Diese Erweiterungen verkleinern das berechnete Tableau. Zuerst betrachten wir die Weitergabe von Einerklauseln. Danach werden wir auf eine spezielle Rücksprungregel eingehen. Die letzte Erweiterung beschreibt die Integration der Projektion in die Umwandlung einer Klauselmenge in DNNF.

4.3.1 Weitergabe von Einerklauseln

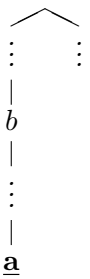
Bereits in Beispiel 4.1.11 ist dem aufmerksamen Leser etwas aufgefallen. Bei einem Schritt der Tableaukonstruktion hatten wir den aktiven Zweig, der mit a und $\neg b$ beschriftet war. Die Klauselmenge $\Delta|\{a, \neg b\} = \{\neg d \wedge e, e\}$ enthält die Einerklausel e. Im nächsten Schritt der Tableaukonstruktion haben wir die Verzweigungsregel mit dem Atom e angewandt. Eigentlich ist diese Verzweigung jedoch unnötig. Die Klauselmenge $\Delta|\{a, \neg b\}$ kann nur wahr werden, wenn e mit wahr belegt wird. Kommen in einer Klauselmenge Einerklauseln vor, so kann diese Klauselmenge nur dann wahr werden, wenn auch alle Einerklauseln wahr sind. Wir müssen also in diesem Fall nicht verzweigen und können die Klauselmenge durch Weitergabe von Einerklauseln vereinfachen.

Definition 4.3.1 ***(Einerklauselregel)***
Vorbedingungen:

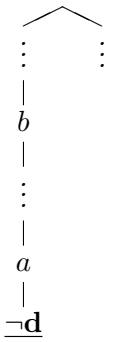
- *B ist der aktive Zweig im Tableau,*
- *es existiert eine Klausel $C \vee L$ in Δ,*
- *es gilt $B \Rightarrow \neg C$ und*
- *L kommt in B nicht vor.*

Operation: *Hänge an das aktive Blatt einen Nachfolgerknoten an, der mit L beschriftet ist. Kennzeichne dieses neue Blatt als das aktive Blatt.*

Beispiel 4.3.2 *Eine Klauselmenge enthält die Klausel $\neg a \vee \neg b \vee \neg d$ und der aktuelle Zweig enthält die Literale a und b aber nicht die Variable d.*



Dann kann das Literal $\neg d$ nach der Einerklauselregel an den aktuellen Zweig angehängt werden, da $(a \wedge b) \rightarrow (\neg a \vee \neg b)$ gilt.



4.3.2 Rücksprung

Bei dieser Erweiterung betrachten wir geschlossene Zweige. Häufig ist nur ein Teil der im geschlossenen Zweig enthaltenen Literale für das Schließen des Zweiges verantwortlich. Die Idee bei der nächste Erweiterung ist es nun, die Literale wegzulassen, die für das Schließen des Zweiges unerheblich sind. In der folgenden Definition bezeichnet L_N das Literal, dass den Knoten N beschriftet.

Definition 4.3.3 ***(Rücksprung)***
Vorbedingungen:

- *Der aktive Zweig B_N hat die Form $B_1 B_2 N' B_3$,*
- *B_1 darf nicht leer sein,*
- *B_2 und B_3 dürfen leer sein (ist B_2 nicht leer, so hat der erste Knoten von B_2 einen offenen Geschwisterknoten),*
- *B_N ist geschlossen,*
- *N' hat einen offenen Geschwisterknoten und*
- *es gilt: $\Delta \wedge B_1 \Rightarrow \neg L_{N'}$.*

Operation: *Sei N_1 der letzte Knoten von B_1. Wenn B_2 leer ist, sei $N_2 = N_1$, ansonsten sei N_2 der letzte Knoten von B_2. Sei N'' der Geschwisterknoten von N'. Daher gilt $L_{N''} = \neg L_{N'}$.*

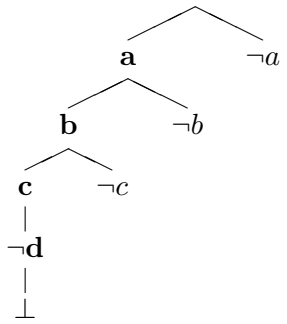
- *Entferne die aus N_1 ausgehenden Kanten,*

- *füge die Kanten, die aus N_2 ausgehen, zu N_1 hinzu,*
- *entferne die aus N'' ausgehenden Kanten und*
- *markiere N'' als aktives Blatt und*

Beispiel 4.3.4 ***(Rücksprung)****Sei die folgende Klauselmenge F gegeben:*

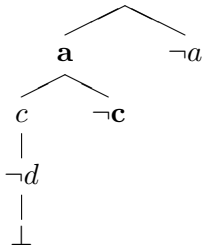
$$F = \{\neg a \vee \neg c \vee \neg d,\\ \neg a \vee \neg c \vee d,\\ b \vee e \vee f\}$$

Außerdem sei der folgende Baum gegeben, bei dem der linke Zweig aktiv und geschlossen ist.



Die Beschriftung des aktiven Zweiges ist $a \wedge b \wedge c \wedge \neg d$. Dieser Zweig ist geschlossen, weil er die zweite Klausel der Klauselmenge falsch macht. Allerdings ist das Literal b im aktiven Zweig nicht am Widerspruch zur zweiten Klausel beteiligt. Daher ist dieses Literal unerheblich und kann weggelassen werden. Überlegen wir uns nun, wie in diesem Beispiel die Bezeichnungen B_1, B_2, N' und B_3 aussehen. Der Knoten N' ist in unserem Beispiel mit c beschriftet. N' hat wie gefordert einen Geschwisterknoten, der nicht geschlossen ist. B_1 ist der mit a beschriftete Teilzweig. Überprüfen wir nun, ob die Vorbedingungen aus 4.3.3 erfüllt sind. Da $B_1 = a$ ist, ist B_1 wie gefordert nicht leer. Darüber hinaus muss $F \wedge B_1 \rightarrow \neg L_{N'}$, also $F \wedge B_1 \rightarrow \neg c$ gelten. Dass dies erfüllt ist, sieht man leicht, da $F \wedge a \wedge c$ unerfüllbar ist. Weiter ist $B_2 = b$ und $B_3 = \neg d$. Damit sind die Voraussetzungen für die Rücksprungregel erfüllt. Bestimmen wir nun N_1 als den letzten Knoten von B_1, also $N_1 = a$ und N_2 als den letzten Knoten von B_2, also $N_2 = b$. Nun werden alle aus $N_1 = a$ ausgehenden Kanten entfernt und die aus $N_2 = b$ herausgehenden Kanten an

N_1 angehängt. Wir entfernen die aus $N'' = \neg c$ herausgehenden Kanten und markieren N'' als den aktiven Knoten. Das Ergebnis sehen wir im folgenden Tableau.



Überlegen wir nun, welche Idee hinter der Rücksprungregel steckt. Ein Zweig wird genau dann geschlossen, wenn seine Beschriftung eine Klausel C aus der der Klauselmenge falsch macht. Daher macht es Sinn, diese Klausel genauer zu betrachten. Man kann alle Literale aus dem aktuellen Zweig, die nicht in der Klausel C und nicht in B_1 enthalten sind, entfernen. Das verkleinert das erstellte Tableau erheblich, da alle überflüssigen Verzweigungen aus dem geschlossenen Zweig entfernt werden.

4.3.3 Integration der Projektion in die Berechnung der DNNF

In [Wer06b] wird eine weitere Erweiterung des Kalküls vorgeschlagen. Oft wird die erstellte DNNF direkt nach der Umwandlung auf eine Menge von Atomen projiziert. Die Idee ist nun, diese Projektion in die Berechnung der DNNF zu integrieren. Dazu gibt es verschiedene Möglichkeiten. Vorab sei bemerkt, dass die Projektion dual zum Vergessen ist. Das bedeutet, dass die Projektion einer Formel F auf eine Menge von Atomen A das gleiche Ergebnis liefert, wie das Vergessen aller Atome aus $\overline{A}$. Wobei $\overline{A}$ das Komplement von A bezüglich der in F vorkommenden Atome ist. Für eine genaue Definition der Projektion und des Vergessens sei der Leser auf Definition 5.2.2 verwiesen.

Beseitigung von puren Literalen

Die erste Möglichkeit ist Beseitigung von pur auftretenden Literalen. Unter einem puren Literal versteht man ein Literal, dass in der betrachteten Klauselmenge nur mit einer Polarität vorkommt. Kommt nun ein Literal, das wir vergessen wollen, pur vor, so können wir alle Klauseln, in denen es vorkommt, streichen. Danach tritt dieses Atom nicht mehr in der Klauselmenge auf. Dies hat nicht nur eine Verkleinerung der Klauselmenge zur Folge, sondern kann auch dazu führen, dass gerade die Klauseln wegfallen, die der Zerlegbarkeitseigenschaft der Klauselmenge im Wege stehen.

Isol-Regel

Bei der zweiten Möglichkeit handelt es sich um die Isol-Regel [Bib92]. Tritt ein zu vergessendes Atom a nur in zwei Klauseln C und C' auf mit $C = C_1 \vee a$ und $C' = C_2 \vee \neg a$, so können diese beiden Klauseln aus der Klauselmenge entfernt werden und durch ihre Resolvente $C_1 \vee C_2$ ersetzt werden. Durch diese Regel wird das Atom a aus der Klauselmenge entfernt. Außerdem kann diese Regel dazu führen, dass Atome, die in C und C' vorkommen, später nur noch in der Resolvente vorkommen und somit nicht mehr gegen die Zerlegbarkeitsforderung verstoßen.

Isol*-Regel

Die dritte Möglichkeit ist die Isol*-Regel [Bib92]. Dabei handelt es sich um eine Verallgemeinerung der Isol-Regel. Angenommen ein zu vergessendes Atom a kommt in der Klauselmenge beinah pur vor. Beinah pur bedeutet, dass a in mehreren Klauseln positiv vorkommt aber in genau einer Klausel negativ auftritt oder umgekehrt. Auch in diesem Fall können Alle Klauseln, in denen a vorkommt, entfernt werden und durch alle Resolventen dieser Klauseln ersetzt werden.

Wir haben nun einen Kalkül zur Umwandlung von Klauselmengen in DNNF kennengelernt. Dieser Kalkül gibt jedoch nicht an, welches Atom für die Verzweigung bei der Verzweigungsregel ausgewählt werden soll. Es ist jedoch klar, dass die Wahl des Atoms nicht nur die Größe der berechneten DNNF, sondern auch den Aufwand der Berechnung beeinflusst. Die geschickte Wahl eines Atoms für die Verzweigung kann also den Aufwand der Berechnung der DNNF verringern. Daher ist es sinnvoll, sich mit Heuristiken für die Wahl der Verzweigungsatome zu beschäftigen.

4.4 Heuristiken für die Umwandlung in DNNF

Wie bereits oben erwähnt, beeinflusst die Wahl des Atoms bei der Verzweigungsregel den Aufwand der Umwandlung einer Klauselmenge in DNNF. Eine geschickte Wahl der Atome kann zu einem kleineren Tableau und somit zu einer weniger aufwändigen Berechnung führen. Es ist klar, dass die Anzahl der offenen Zweige die Komplexität der Umwandlung in DNNF bestimmt. Bei Anwendung der Verzweigungsregel entsteht stets ein neuer Zweig. Daher ist es sinnvoll die Einerklauselregel der Verzweigungsregel wenn möglich vorzuziehen. Für DPLL gibt es eine Vielzahl von Verzweigungsheuristiken. Einige davon haben wir bereits kennen gelernt. Der verbleibende Teil dieses Abschnittes befasst sich mit der Anwendung einiger dieser DPLL Verzweigungsheuristiken auf den oben beschriebenen Kalkül.

Anwendung von einfachem Zählen von Atomen

Für jedes Atom, das in der Klauselmenge vorkommt, wird die Anzahl der Klauseln, in denen das Atom auftritt bestimmt. Für die Verzweigungsregel wird das Atom, das am häufigsten auftritt, vorgezogen.

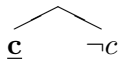
Beispiel 4.4.1 *Sei die folgende Klauselmenge gegeben.*

$$\begin{aligned}\Delta = & abInv \vee \neg a \vee \neg c,\\ & abInv \vee a \vee c,\\ & abAnd \vee \neg c \vee \neg b \vee d,\\ & abAnd \vee c \vee \neg d,\\ & abAnd \vee b \vee \neg d\end{aligned}$$

Als erstes bestimmen wir die Anzahl der Klauseln, in denen die einzelnen Atome vorkommen. Dabei entspricht der Wert unter einer Schaltvariablen X dem Wert von $CP(X) + CN(X)$.

$abInv$	a	c	$abAnd$	b	d
2	2	4	3	2	3

Entsprechend dieser Anzahlen wählen wir die Atome für die Verzweigungsregel aus. Die Idee ist nun, dass wir nach jedem Schritt $\Delta|B_i$ bilden, wobei B_i der aktuelle Zweig ist. Ist $\Delta|B_i$ nicht in DNNF und es ist keine Weitergabe von Einerklauseln möglich, so bestimmen wir erneut die obigen Anzahlen in Bezug auf die Klauselmenge $\Delta|B_i$. Entsprechend dieser neu bestimmten Anzahlen wählen wir den nächsten Kandidaten für eine Verzweigung aus. Die Neuberechnung der Anzahlen hat nur einen linearen Aufwand und stellt sicher, dass zu jeder Zeit eine Verzweigung bezüglich der Atome gemacht werden, die es am meisten zu verschulden haben, dass die Klauselmenge noch nicht in DNNF vorliegt. Im ersten Schritt wenden wir also die Verzweigungsregel mit dem c Atom an und erhalten:



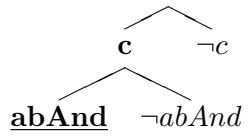
Der aktive Zweig B_1 ist wie gewohnt durch Unterstreichen gekennzeichnet. Wir bilden

$$\begin{aligned}\Delta|B_1 = \Delta|\{c\} = & abInv \vee \neg a,\\ & abAnd \vee \neg b \vee d,\\ & abAnd \vee b \vee \neg d\end{aligned}$$

Diese Klauselmenge ist nicht in DNNF und sie enthält auch keine Einerklausel. Daher bestimmen wir die obige Tabelle für $\Delta|B_1$ und erhalten:

abInv	*a*	*abAnd*	*b*	*d*
1	*1*	*2*	*2*	*2*

Und wählen abAnd für die nächste Anwendung der Verzweigungsregel aus und erhalten:



Um festzustellen, ob wir den aktiven Zweig weiterbearbeiten müssen, bilden wir

$$\Delta|B_1 = \Delta|\{c, abAnd\} = abInv \vee \neg a$$

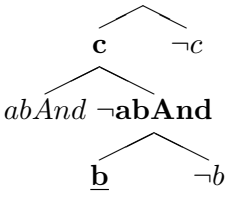
Da diese Klauselmenge in DNNF ist, arbeiten wir am nächsten Zweig weiter. Nun ist der Zweig B_2 mit der Beschriftung $c, \neg abAnd$ aktiv. Wir bilden $\Delta|B_2$

$$\begin{aligned}\Delta|B_2 = \Delta|\{c, \neg abAnd\} = & abInv \vee \neg a,\\ & \neg b \vee d,\\ & b \vee \neg d\end{aligned}$$

Diese Klauselmenge liegt noch nicht in DNNF vor. Daher zählen wir erneut das Vorkommen der einzelnen Atome und erhalten:

abInv	*a*	*b*	*d*
1	*1*	*2*	*2*

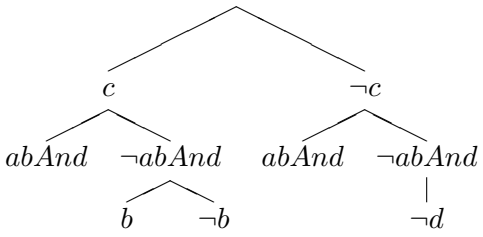
Dieser Tabelle entsprechend wählen wir das Atom b für die nächste Verzweigungsregel aus und bilden:



Erneut überprüfen wir, ob der aktive Zweig weiterbearbeit werden muss:

$$\begin{aligned}\Delta|B_2 = \Delta|\{c, \neg abAnd, b\} = & abInv \vee \neg a,\\ & d\end{aligned}$$

Diesmal liegt $\Delta|B_2$ tatsächlich in DNNF vor. Wir können also zum nächsten Zweig übergehen. Nach diesem Vorgehen erstellen wir einen kompletten Baum. Das Ergebnis sieht wie folgt aus.



Die Verzweigung nach dem Atom, dass am häufigsten in der Klauselmenge enthalten ist, führt zu einer starken Vereinfachung der Klauselmenge. Daher entsteht bei der Verwendung dieser Heuristik stets ein recht kleines Tableau. Ein kleines Tableau hat den Vorteil, dass die daraus abgelesene DNNF auch eher kurz ist. Bezogen auf die Diagnose läßt sich allerdings bei einem so erstellten Tableau keine Aussage darüber machen, aus welchen Zweigen wir später minimale Erklärungen ablesen können.

Anwendung der Jeroslow-Wang Heuristik

Wenden wir nun die einseitige Jeroslow-Wang Heuristik auf den DPLL-FTab Kalkül an. Bei dieser Heuristik werden nicht nur Atome vorgezogen, die häufig in der betrachteten Klauselmenge auftreten, sondern auch Atome, die häufig in kurzen Klauseln vorkommen.

Beispiel 4.4.2 *Sei wieder die folgende Klauselmenge gegeben.*

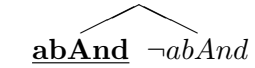
$$\begin{aligned}\Delta = & abInv \vee \neg a \vee \neg c,\\ & abInv \vee a \vee c,\\ & abAnd \vee \neg c \vee \neg b \vee d,\\ & abAnd \vee c \vee \neg d,\\ & abAnd \vee b \vee \neg d\end{aligned}$$

Im nächsten Schritt müssen wir für jedes in der Klauselmenge Δ vorkommende Literal l den Wert $J(l)$ nach Formel 3.1 berechnen. Die berechneten Werte sehen wie folgt aus:

Literal	**abInv**	**abAnd**	**a**	**¬a**	**b**
J(Literal)	0,25	0,3125	0,125	0,125	0,125

	¬b	**c**	**¬c**	**d**	**¬d**
	0,0625	0,25	0,1875	0,0625	0,25

Nun wird das Literal mit dem höchsten J-Wert für die erste Verzweigung ausgewählt. In diesem Fall ist es abAnd. Wir erhalten das folgende Tableau:



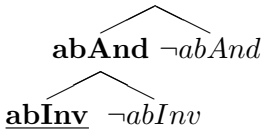
Wir bilden die Klauselmenge $\Delta|abAnd$ *und erhalten:*

$$\Delta|abAnd = abInv \vee \neg a \vee \neg c,$$
$$abInv \vee a \vee c$$

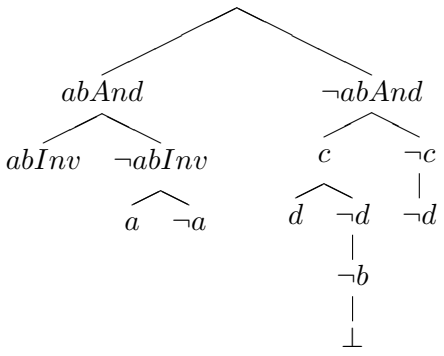
Da diese Klauselmenge noch nicht in DNNF vorliegt, arbeiten wir am aktiven Zweig weiter. Wir berechnen die J-Werte für die in $\Delta|abAnd$ *enthaltenen Literale:*

Literal	**abInv**	**a**	**¬a**	**c**	**¬c**
J(Literal)	0,25	0,125	0,125	0,125	0,125

Da das Literal $abInv$ *den höchsten J-Wert hat, verzweigen wir als nächstes nach* $abInv$*:*



Benutzen wir weiter die Jeroslow-Wang Heuristik für die Auswahl des nächsten Verzweigungsliterals, so erhalten wir das folgende Tableau.



Auch das nach der Jeroslow-Wang Heuristik erstellte Tableau ist relativ klein. Da die Jeroslow-Wang Atome vorzieht, die in kurzen Klauseln vorkommen, führen die Verzweigungen mit diesen Atomen dazu, dass kurze Klauseln noch kürzer werden. Dadurch

kommt es häufig zur Weitergabe von Einerklauseln. Da diese Heuristiken nur auf der Syntax der betrachteten Klauselmengen arbeiten, können sie für Beispiele aus allen Domänen verwendet werden. In manchen Domänen sind jedoch zusätzliche Informationen zur betrachteten Klauselmenge verfügbar. Dabei kann es sich bei der Diagnose beispielsweise um ein Modell des funktionierenden Schaltkreises oder auch um Ausfallwahrscheinlichkeiten der einzelnen Schaltelemente handeln. Es bietet sich an, diese Informationen bei der Konstruktion des Tableaus zu benutzen. In Kapitel 6 werden wir genauer auf diese Idee eingehen.

5 Partielle Wissenskompilation

Der oben beschriebene DPLL-FTab Kalkül wandelt eine gegebene Menge von Klauseln in DNNF um. Die von diesem Kalkül berechnete DNNF hat einen ganz speziellen Aufbau.

Satz 5.0.3 ***(von DPLL-FTab Kalkül bestimmte DNNF, DNNF-Teilstück)*** *Sei Δ eine Klauselmenge und T ein mit dem DPLL-FTab erstelltes D-Tableau zu Δ. T bestehe aus den Zweigen $B_1, \ldots, B_n$. Die vom DPLL-FTab Kalkül berechnete DNNF zu einer gegebenen Klauselmenge Δ hat die Form: $\mathsf{dnnf}_{FTab}(\Delta) = \bigvee_{i\in\{1,\ldots,n\}}(B_i \wedge \Delta|B_i)$. Ein Disjunktionsglied $B_i \wedge \Delta|B_i$ dieser DNNF nennen wir genau dann DNNF-Teilstück, wenn der Zweig B_i nicht geschlossen ist.*

Man erhält ein DNNF-Teilstück durch Ablesen aus dem von DPLL-FTab erzeugten Tableau. Es ist klar, dass solche DNNF-Teilstücke bereits vor der kompletten Fertigstellung des Tableaus abgelesen werden können. Das erste DNNF-Teilstück lässt sich direkt nach dem Fertigstellen des ersten Zweiges aus dem Tableau ablesen. Die gesamte DNNF, die aus einem fertigen DPLL-FTab Tableau abgelesen werden kann, ist eine Disjunktion von DNNF-Teilstücken. Daher ist jedes Modell eines DNNF-Teilstückes auch ein Modell der kompletten DNNF und damit auch der Ausgangsklauselmenge.

5.1 Modellberechnung auf DNNF-Teilstücken

Wie bereits angedeutet, ist es möglich aus einem DNNF-Teilstück ein Modell der zugehörigen Klauselmenge abzulesen. Außerdem haben wir gesehen, dass nicht die komplette DNNF berechnet werden muss, um ein DNNF-Teilstück zu erhalten. Dadurch ist es möglich, schon aus partiell präkompilierten Klauselmengen, Modelle abzulesen. Jedes DNNF-Teilstück liegt selbst auch in DNNF vor. Daher können alle Operationen, die wir in Abschnitt 2.2 kennen gelernt haben, auch auf DNNF-Teilstücken durchgeführt werden. Insbesondere können wir Modelle aus DNNF-Teilstücken mit einem zeitlichen Aufwand ablesen, der quadratisch zur Anzahl der Modelle des DNNF-Teilstücks ist[1] [Dar98]. Da die komplette DNNF eine disjunktive Verknüpfung aller DNNF-Teilstücke ist, muss es sich bei einem aus einem DNNF-Teilstück abgelesenen Modell zwar auch um ein Modell der betrachteten Klauselmenge handeln, jedoch nicht um ein minimales Modell. Bei manchen Anwendungen, wie z.B. der Diagnose, interessieren wir uns jedoch besonders für minimale Modelle. Daher macht es Sinn, sich mit der Frage zu beschäftigen, wie man Modelle auf Minimalität überprüfen kann.

[1] Siehe dazu auch Definition 2.2.17 auf Seite 14.

5.2 Minimalitätstests

Wie bereits im obigen Abschnitt erwähnt, ist es bei der Modellbestimmung auf DNNF-Teilstücken möglich, dass nichtminimale Modelle abgelesen werden. Interessieren wir uns jedoch nur für minimale Modelle, so müssen wir nach dem Ablesen des Modells überprüfen, ob es sich dabei um ein minimales Modell handelt. Natürlich wäre es möglich, alle Modelle der betrachteten Klauselmenge zu bilden und danach die Modelle auszuschließen, die eine Obermenge eines anderen Modells bilden. Diese Möglichkeit scheidet für uns jedoch als Minimalitätstest aus, da wir uns dafür nicht nur jedes gefundene Modell merken müssten, sondern auch alle Modelle bestimmen müssten. Um jedoch alle Modelle bestimmen zu können, müssten uns alle DNNF-Teilstücke vorliegen. Unser Ziel ist es hier aber, bereits auf einem oder mehreren DNNF-Teilstücken minimale Modelle zu bestimmen, ohne die komplette DNNF berechnen zu müssen. Wir benötigen also einen anderen Minimalitätstest.

Ilkka Niemelä stellt in [Nie96a] einen Minimalitätstest vor, der einen polynomiellen Speicherbedarf hat. Dabei benutzt er die übliche Definition für ein minimales Modell, die hier unter Definition 2.2.11 auf Seite 12 zu finden ist. Ilkka Niemelä charakterisiert minimale Modelle durch die Menge der Atome, die von dem betrachteten Modell nicht mit wahr belegt werden.

Satz 5.2.1 *Sei Σ eine Klauselmenge und M eine Interpretation. Dann ist M genau dann ein minimales Modell für Σ, wenn für jedes Atom a aus M gilt: $\Sigma \cup N_\Sigma(M) \models a$ mit $N_\Sigma(M) = \{\neg b \mid b \text{ kommt positiv in einer Klausel vor und } M \not\models b\}$.*

Anders ausgedrückt ist M genau dann ein minimale Modell der Klauselmenge Σ, wenn $\Sigma \cup N_\Sigma(M) \cup \{\overline{M}\}$ unerfüllbar ist. Wobei $\{\overline{M}\}$ die Klausel ist, die man durch Negation des Modells M erhält. Die Menge $N_\Sigma(M)$ erhält alle Atome, die in der Klauselmenge Σ positiv vorkommen, in M jedoch nicht enthalten sind. Eine Menge, die die in Satz 5.2.1 geforderte Eigenschaft erfüllt, nennen wir grundiert.

Unser Ziel ist es, diesen Grundierungstest einzusetzen, um herauszufinden, ob ein gefundenes Modell minimal ist. Bezogen auf unsere Anwendung Diagnose müssen wir uns jedoch zuerst überlegen, welche Klauselmenge der Menge Σ aus Satz 5.2.1 entspricht. Sei Δ die zum funktionierenden Schaltkreis gehörige Klauselmenge und Γ ein durch den oben beschriebenen Kalkül erstelltes DNNF-Teilstück. Es gilt:

$$\Delta \equiv \mathsf{dnnf}_{FTab}(\Delta) \equiv \Gamma_1 \vee \Gamma_2 \vee \ldots \vee \Gamma_n$$

wobei das erstellte D-Tableau aus den Zweigen $B_1, \ldots, B_n$ besteht und die Γ_i für $i \in \{1, \ldots, n\}$ die DNNF-Teilstücke sind, die bei der Umwandlung in DNNF entstehen. Daher ist jedes Modell M von Γ auch ein Modell für die komplette DNNF von Δ und daher auch ein Modell für die Ausgangsklauselmenge Δ. Betrachten wir nun die Auswirkungen der Operationen, die bei der Diagnose nach Darwiche durchgeführt werden.

Zur Erinnerung kann sich der Leser nochmals 2.5 ansehen. Sei $\Gamma' = \mathsf{Project}(\Gamma, A \cup O)$ die Projektion des DNNF-Teilstücks Γ auf die Menge der Schaltelemente und der Ein- und Ausgabevariablen des Schaltkreises. Klar ist, dass jedes Modell für Γ' auch ein Modell für $\mathsf{Project}(\Delta, A \cup O)$ ist. Sei nun Γ'' gegeben als $\Gamma'' = \mathsf{smooth}(\Gamma')$. Da die Umwandlung einer DNNF in eine gleichmäßige DNNF äquivalenzerhaltend ist, ist jedes Modell für Γ'' ein Modell für $\mathsf{Project}(\Delta, A \cup O)$. Sei nun $\Gamma''' = \Gamma''|\alpha$. Die Konditionierung einer Klauselmenge durch eine Menge von Literalen entspricht dem Hinzufügen von entsprechenden Einerklauseln zur betrachteten Klauselmenge, dem anschließenden Vereinfachen durch Weitergabe der Einerklauseln und späteres Entfernen der hinzugefügten Einerklauseln. Da außerdem $(\mathsf{Project}(\Delta, A \cup O))|\alpha = \mathsf{Project}(\Delta|\alpha, A \cup O)$ gilt, ist jedes Modell für Γ''' auch ein Modell für $\mathsf{Project}(\Delta|\alpha, A \cup O)$.

Um die Grundmenge für den Grundierungstest zu bestimmen, müssen wir also die Menge $\mathsf{Project}(\Delta|\alpha, A \cup O)$ bilden. Um diese Menge bilden zu können, müssen wir uns zuerst mit der Projektion einer Klauselmenge auf eine Menge von Atomen beschäftigen. Unter [Wer06b] findet man eine Definition für Projektion und Forgetting einer Klauselmenge auf eine Menge von Atomen.

Definition 5.2.2 *(Projektion, Vergessen) F' ist eine Projektion von F auf die Menge von Atomen S, wenn für alle Interpretationen I gilt:*

- *$I \models F'$ genau dann, wenn es eine Interpretation I' gibt, mit $I' \models F$ und $I' \cap S = I \cap S$.*

In der Aussagenlogik gibt es für jede Formel F und jede Menge von Atomen S eine eindeutig bestimmte Klasse von äquivalenten Formeln, bezeichnet als $\mathsf{Project}(F, S)$, die die Projektion von F auf S ist. Das Vergessen einer Menge von Atomen über einer Formel ist dual zur Projektion. D.h. $forget(F, S) \equiv \mathsf{Project}(F, \overline{S})$.

Es führt also zum selben Ergebnis, eine Formel auf eine Menge von Atomen zu projizieren, oder die komplementäre Menge von Atomen auf dieser Formel zu vergessen. Betrachten wir nun, wie man eine Klauselmenge auf eine Menge von Atomen projiziert. Unter [Wer06a] findet man einen Kalkül, mit dem man diese Projektion vornehmen kann. Er entspricht dem Kalkül zur Umwandlung einer Klauselmenge in DNNF. Die einzigen Unterschiede betrachten wir im Folgenden, wobei Δ die Klauselmenge bezeichnet und S die Menge der Atome, auf die Δ projiziert werden soll.

1. Wir verzweigen nur über die Menge der Atome, die wir vergessen wollen.
2. Bei der Wechselregel wird die Voraussetzung, dass sich $\Delta|B$ in DNNF befindet durch die Voraussetzung, dass $\Delta|B$ keine Atome aus S enthält, ersetzt.

Aus dem so erstellten Tableau liest man das Ergebnis der Projektion wie folgt ab:

$$\mathsf{Project}(\Delta, S) = \bigvee_i B_i \setminus S \wedge \Delta|B_i$$

Betrachten wir hierzu ein Beispiel:

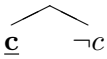
Beispiel 5.2.3 *Sei der Schaltkreis aus Beispiel 2.3.3 gegeben, der durch die folgende Klauselmenge beschrieben wird:*

$$\begin{aligned}\Delta = \{&abInv \vee \neg a \vee \neg c,\\ &abInv \vee a \vee c,\\ &abAnd \vee \neg c \vee \neg b \vee d,\\ &abAnd \vee c \vee \neg d,\\ &abAnd \vee b \vee \neg d\}\end{aligned}$$

Die Menge der Schaltelemente, Ein- und Ausgabevariablen ist $A \cup O = \{abInv, abAnd, a, b, d\}$. *Außerdem sei die Beobachtung* $\alpha = \neg a \wedge b \wedge \neg d$ *gegeben. Damit ergibt sich* $\Delta|\alpha$ *als:*

$$\begin{aligned}\Delta|\alpha = \{&abInv \vee c,\\ &abAnd \vee \neg c\}\end{aligned}$$

Bilden wir nun die Projektion $\mathsf{Project}(\Delta|\alpha, A \cup O) = Forget(\Delta|\alpha, \{c\})$. *Da wir nur c vergessen wollen, gibt es nur eine Möglichkeit für die Verzweigungsregel. Das Ergebnis ist das folgende Tableau, in dem der linke Zweig aktiv ist:*



Die Klauselmenge $(\Delta|\alpha)|c = abAnd$ *enthält das zu vergessende Atom nicht mehr. Also arbeiten wir am nächsten Zweig weiter. Auch die Klauselmenge* $(\Delta|\alpha)|\neg c = abInv$ *enthält c nicht mehr. Damit können wir das Ergebnis der Projektion angeben:*

$$\mathsf{Project}(\Delta|\alpha, A \cup O) = abAnd \vee abInv$$

Nun kennen wir die Grundmenge, auf der wir den Grundierungstest durchführen können. Angenommen, wir hätten aus einem DNNF-Teilstück das Modell $M = \{abInv, abAnd\}$ *abgelesen. Wir wollen nun überprüfen, ob es sich bei dieser Erklärung um eine minimale Erklärung handelt. Bilden wir nun die Menge* $N_{\mathsf{Project}(\Delta|\alpha, A\cup O)}(M) = \{\}$. *Außerdem bestimmen wir* $\overline{M} = \neg abInv \vee \neg abAnd$. *Nun müssen wir überprüfen, ob* $\mathsf{Project}(\Delta|\alpha, A \cup O) \cup N_{\mathsf{Project}(\Delta|\alpha, A\cup O)}(M) \cup \{\overline{M}\}$ *erfüllbar ist.*

$$\begin{aligned}&Project(\Delta|\alpha, A \cup O) \cup N_{\mathsf{Project}(\Delta|\alpha, A\cup O)}(M) \cup \{\overline{M}\}\\ =\ &\{abAnd \vee abInv,\\ &\ \neg abAnd \vee \neg abInv\}\end{aligned}$$

Diese Klauselmenge ist erfüllbar, denn z.B. ist $M' = \{abAnd\}$ *ein Modell. Daher handelt es sich bei M nicht um eine minimale Erklärung.*

5.2.1 Aufwandsbetrachtung

Es stellt sich die Frage, wie aufwändig es ist, ein aus einem DNNF-Teilstück zur Klauselmenge Σ_0 abgelesenes Modell M, mit dem oben beschriebenen Grundierungstest auf Minimalität zu überprüfen. Zuerst müssen wir dafür die Menge $N_{\Sigma_0}(M)$ bilden. Diese Menge enthält alle Literale $\neg a$, wobei a in Σ_0 positiv vorkommt und nicht in M enthalten ist. Der zeitliche Aufwand zur Bestimmung der Menge $N_{\Sigma_0}(M)$ ist linear zur Größe von Σ_0. Im nächsten Schritt wird die Menge $\Sigma_0 \cup N_{\Sigma_0}(M)$ gebildet. Da $N_{\Sigma_0}(M)$ eine Menge von Literalen ist, kann $\Sigma_0 \cup N_{\Sigma_0}(M)$ durch Konditionierung vereinfacht werden. Besteht $N_{\Sigma_0}(M)$ aus l Literalen und enthält Σ_0 n verschiedene Atome, so enthält $\Sigma_0|N_{\Sigma_0}(M)$ nur noch $n-l$ verschiedene Atome. Diese Konditionierung hat einen linearen Aufwand. Im letzten Schritt muss die Klauselmenge $\Sigma_0 \cup N_{\Sigma_0}(M) \cup \{\overline{M}\} = \Sigma_0|N_{\Sigma_0}(M) \cup \{\overline{M}\}$ auf Erfüllbarkeit getestet werden. Dies hat im schlechtesten Fall einen Aufwand von $2^{n-l+|M|}$. Insgesamt ergibt sich daraus der Aufwand für die einmalige Durchführung des Grundierungstests:

$$c \cdot |\Sigma_0| + 2^{n-l+|M|} \tag{5.1}$$

Der kritische Teil in Formel 5.1 ist der Erfüllbarkeitstest mit dem Aufwand von $2^{n-l+|M|}$. Es ist klar, dass der Grundierungstest besonders aufwändig ist, wenn $l = |N_{\Sigma_0}(M)|$ sehr viel kleiner als n (der Anzahl der in der Ausgangsklauselmenge auftretenden Atome) ist. Ist hingegen l annähernd so groß wie n, wird der Grundierungstest wenig aufwändig. Da l gerade $|N_{\Sigma_0}(M)|$ entspricht, sollte die Menge $N_{\Sigma_0}(M)$ also groß sein. Dies ist genau dann der Fall, wenn es viele positive Literale in der Ausgangsklauselmenge gibt und das zu untersuchende Modell nur aus wenigen Literalen besteht.

Der durch Formel 5.1 ausgedrückte Aufwand wird durch jeden Grundierungstest verursacht. Daher fällt dieser Aufwand bei mehrfacher Anwendung des Grundierungstests auch entsprechend öfters an. Im ungünstigsten Fall wird für jedes DNNF-Teilstück mindestens ein Modell berechnet und auf Minimalität überprüft. Daher wäre der oben beschriebene ungünstigste Fall also aufwändiger als die Bestimmung der kompletten DNNF, auf der man dann in quadratischer Zeit die minimalen Modelle ablesen könnte.

Beim Beispiel der Diagnose fällt noch ein zusätzlicher Aufwand an. Denn wir müssen einmalig die Grundmenge Σ_0 für den Grundierungstest bilden. Dafür müssen wir $\Delta|\alpha$ bilden, wobei Δ die Klauselmenge ist, die den Schaltkreis beschreibt und α die Beschreibung des fehlerhaften Verhaltens ist. Der Aufwand für diese Konditionierung ist linear zur Größe der Klauselmenge Δ. Enthält Δ genau d und α genau a Atome, so verbleiben in $\Delta|\alpha$ noch $d-a$ Atome. Danach wird die Projektion $\mathsf{Project}(\Delta|\alpha, A \cup O)$ gebildet. Werden dabei k Atome wegprojiziert, so hat dies im schlechtesten Fall einen zeitlichen Aufwand von 2^k. In der bei der Projektion entstehenden Klauselmenge sind noch $n-a-k$ verschiedene Atome enthalten. Die Menge $\mathsf{Project}(\Delta|\alpha, A \cup O)$ muss jedoch nur einmal gebildet und in CNF umgewandelt werden, wobei die Umwandlung in eine äquivalenzerhaltende CNF auch exponentiell ist. Insgesamt beträgt der Aufwand für die

einmalige Durchführung des Grundierungstests für das Beispiel der Diagnose (wobei l in der folgenden Formel die gleiche Bedeutung wie in Formel 5.1 hat):

$$c' \cdot |\Delta| + 2^k + 2^{n-a-k} + c \cdot |cnf(\mathsf{Project}(\Delta|\alpha, A \cup O)| + 2^{n-a-k-l+|M|}$$

Bei dem oben betrachteten Grundierungstest muss für den Schaltkreis einmal die Menge $\mathsf{Project}(\Delta|\alpha, A \cup O)$ gebildet werden. Die Projektion kann man sich jedoch sparen, wenn man einen etwas abgewandelten Grundierungstest verwendet.

Unter [Nie96b] findet man einen Grundierungstest, der Zirkumskription (engl. Circumscription) unterstützt. Bevor wir jedoch diesen Grundierungstest genauer betrachten, beschäftigen wir uns mit Zirkumskription [McC80]. Wir stellen Modelle wie üblich als eine Menge von Variablen dar, die mit wahr belegt werden. Dann ist die Zirkumskription in der Aussagenlogik wie folgt definiert:

Definition 5.2.4 ***(Zirkumskription)*** *Sei F eine aussagenlogische Formel. Dann ist $Circ(F)$ definiert als:*

$$Circ(F) = \{M \mid M \models F \text{ und } \nexists N \text{ mit } N \models F \text{ und } N \subseteq M\}$$

Mit anderen Worten ist die Zirkumskription einer aussagenlogischen Formel F gerade die Menge der minimalen Modelle von F.

Vladimir Lifschitz hat die Zirkumskription um die Idee erweitert, dass nicht alle Variablen minimiert werden müssen. Er teilt die in der Formel auftretenden aussagenlogischen Variablen in drei Mengen auf:

- **Veränderlich:** Menge der Variablen, die bei der Minimalisierung nicht betrachtet werden.
- **Fixiert:** Menge der Variablen, die bei der Minimalisierung mit einem festen Wert versehen werden. Es werden also nur die Modelle bei der Bildung der Zirkumskription betrachtet, bei denen diese Variablen den ihnen fest zugewiesenen Wert aufweisen.
- **Minimiert:** Enthält alle nicht veränderlichen oder fixierten Variablen. Bezüglich dieser Variablen soll minimiert werden.

Die Erweiterung der Zirkumskription um diese Menge ist wie folgt definiert:

Definition 5.2.5 ***(Zirkumskription mit veränderlichen und fixierten Variablen)*** *Sei Σ eine aussagenlogische Formel, F die Menge der fixierten Variablen und P die Menge der Variablen, bezüglich derer Σ minimiert werden soll. Die Menge der veränderlichen Variablen wird durch die Variablen gebildet, die nicht in $P \cup F$ liegen. Dann ist die Zirkumskription der Formel Σ mit den Mengen P und F definiert als:*

$$\begin{aligned} Circ(\Sigma; P, F) = \{&M \mid M \models \Sigma \text{ und } \nexists N \text{ mit } N \models \Sigma,\ N \cap P \subset M \cap P \\ &\text{und } N \cap F = M \cap F\} \end{aligned}$$

Bei der erweiterten Zirkumskription wird nur noch in Bezug auf die Variablen in P minimiert. Darüber hinaus werden nur die Modelle miteinander verglichen, die die Variablen aus F mit den geforderten Werten belegen. Alle anderen Variablen werden bei der Bildung der Zirkumskription nicht beachtet.

Nachdem wir den Begriff der Zirkumskription erklärt haben, können wir uns nun mit dem Grundierungstest für Zirkumskription beschäftigen. Betrachten wir zunächst die Frage, wann ein Modell in Bezug auf die zu minimierende Variablenmenge P und die Menge der fixierten Variablen F minimal ist. In der folgenden Definition beschreibt die Menge $sat(I, P) = \{p \in I \mid p \in P\}$ die Menge der Variablen aus P, die durch die Interpretation I wahr werden.

Definition 5.2.6 *Seien I_1 und I_2 zwei Interpretationen und P und F zwei disjunkte Mengen von Variablen mit der oben beschriebenen Semantik.*

1. *Wir sagen, dass $I_1 \leq_{\langle P,F\rangle} I_2$ genau dann gilt, wenn $sat(I_1, P) \subseteq sat(I_2, P)$ und $sat(I_1, F) = sat(I_2, F)$. Außerdem gilt $I_1 <_{\langle P,F\rangle} I_2$ genau dann, wenn zwar $I_1 \leq_{\langle P,F\rangle} I_2$ gilt, aber $I_2 \leq_{\langle P,F\rangle} I_1$ nicht.*

2. *Eine Interpretation I ist genau dann ein $\langle P, F\rangle$ minimales Modell einer Klauselmenge Σ, wenn I ein Modell für Σ ist und es kein anderes Modell I' von Σ gibt, mit $I' <_{\langle P,F\rangle} I$.*

Die nächste Definition benötigen wir als Kriterium für den Grundierungstest.

Definition 5.2.7 *Sei Σ eine aussagenlogische Formel. Eine Menge von aussagenlogischen Atomen M heißt genau dann grundiert in $\langle \Sigma, P, F\rangle$, wenn für alle $a \in M$, gilt: wenn $a \in P$ ist, dann gilt $\Sigma \cup N^{\langle P,F\rangle}(M) \models a$ mit $N^{\langle P,F\rangle}(M) = \{\neg q \mid q \in P \cup F,\ q \notin M\} \cup \{q \mid q \in F,\ q \in M\}$.*

Mit Hilfe dieser Definition können wir nun ein Kriterium dafür angeben, wann ein Modell minimal in Bezug auf die Mengen P und F ist.

Satz 5.2.8 ***($\langle$P, F$\rangle$ minimales Modell)*** *Eine Interpretation I ist genau dann ein $\langle P, F\rangle$ minimales Modell einer Klauselmenge Σ, wenn I ein Modell für Σ ist und I grundiert in $\langle \Sigma, P, F\rangle$ ist.*

Beim Beispiel der Diagnose entsprechen die Schaltkomponenten und die Ein- und Ausgabevariablen der oben benutzten Menge P. Die obige Menge F ist leer, da bei der Diagnose keine Variablen auf fixe Werte gesetzt werden und daher läßt sich Definition 5.2.7 vereinfachen. Damit ist eine Menge von Atomen M bei der Anwendung der Diagnose genau dann grundiert in $\langle \Sigma, P\rangle$, wenn für alle $a \in M$ gilt: wenn $a \in P$ ist, dann gilt $\Sigma \cup N^P(M) \models a$ mit $N^P(M) = \{\neg q \mid q \in P,\ q \notin M\}$.

Satz 5.2.9 ***(Negation eines Modells)*** *Sei $M = \{a_1, \ldots, a_n\}$ ein Modell einer Klauselmenge. Dann bezeichnen wir $\overline{M} = \neg a_1 \vee \ldots \vee \neg a_n$ als Negation des Modells M.*

Um zu beweisen, dass wir für Diagnosebeispiele statt dem oben genannten Grundierungstest auch den Grundierungstest für Zirkumskription benutzen können, müssen wir den folgenden Zusammenhang zeigen.

Satz 5.2.10 *Sei M ein Modell für $\mathsf{Project}(\Sigma, P)$. Dann gilt: M ist genau dann grundiert in $\langle \Sigma, P \rangle$, wenn M in Bezug auf das Ergebnis der Projektion $\mathsf{Project}(\Sigma, P)$ grundiert ist.*

Im folgenden Widerspruchsbeweis bezeichnet $\overline{M}$ stets die Negation des Modells M.

Beweis 5.2.11 ($\Leftarrow$) *Sei nun also M in Bezug auf das Ergebnis der Projektion $\mathsf{Project}(\Sigma, P)$ grundiert. Das bedeutet, dass $\{\overline{M}\} \cup \mathsf{Project}(\Sigma, P) \cup N_{\mathsf{Project}(\Sigma,P)}(M)$ unerfüllbar ist.*
Wir zeigen nun, dass dann auch $\{\overline{M}\} \cup \Sigma \cup N^P(M)$ unerfüllbar ist und damit M in $\langle \Sigma, P \rangle$ grundiert ist.

- *Annahme: Sei also $\{\overline{M}\} \cup \mathsf{Project}(\Sigma, P) \cup N_{\mathsf{Project}(\Sigma,P)}(M)$ unerfüllbar und $\{\overline{M}\} \cup \Sigma \cup N^P(M)$ erfüllbar. M_1 sei ein Modell für $\{\overline{M}\} \cup \Sigma \cup N^P(M)$.*
- *Da $M_1 \models \Sigma$ gilt, gilt auch $M_1 \models \mathsf{Project}(\Sigma, P)$.*
- *Aus $N_{\mathsf{Project}(\Sigma,P)}(M) \subseteq N^P(M)$ und $M_1 \models N^P$ folgt $M_1 \models N_{\mathsf{Project}(\Sigma,P)}(M)$.*
- *Damit gilt $M_1 \models \{\overline{M}\}$, $M_1 \models N_{\mathsf{Project}(\Sigma,P)}(M)$ und $M_1 \models \mathsf{Project}(\Sigma, P)$ woraus $M_1 \models \{\overline{M}\} \cup \mathsf{Project}(\Sigma, P) \cup N_{\mathsf{Project}(\Sigma,P)}(M)$*
- *Diese Klauselmenge ist aber laut Voraussetzung unerfüllbar. Also muss auch $\{\overline{M}\} \cup \Sigma \cup N^P(M)$ unerfüllbar sein.*

($\Rightarrow$)*Sei nun $\{\overline{M}\} \cup \Sigma \cup N^P$ unerfüllbar. Wir zeigen, dass dann auch $\{\overline{M}\} \cup \mathsf{Project}(\Sigma, P) \cup N_{\mathsf{Project}(\Sigma,P)}(M)$ unerfüllbar sein muss.*

- *Annahme: Sei also $\{\overline{M}\} \cup \Sigma \cup N^P$ unerfüllbar und $\{\overline{M}\} \cup \mathsf{Project}(\Sigma, P) \cup N_{\mathsf{Project}(\Sigma,P)}(M)$ erfüllbar. Sei M_2 ein Modell für $\{\overline{M}\} \cup \mathsf{Project}(\Sigma, P) \cup N_{\mathsf{Project}(\Sigma,P)}(M)$.*
- *Wie bereits oben erwähnt, gilt $N_{\mathsf{Project}(\Sigma,P)}(M) \subseteq N^P(M)$. In $N^P(M) \setminus N_{\mathsf{Project}(\Sigma,P)}(M)$ sind gerade die Literale $\neg a$ enthalten, für die gilt: $a \notin M$ und $a \in P$ und $a \notin \mathsf{Project}(\Sigma, P)$. Das bedeutet, dass a in $\mathsf{Project}(\Sigma, P)$ nur negativ vorkommt.*
- *Unser Modell M_2 erfüllt laut Voraussetzung $N_{\mathsf{Project}(\Sigma,P)}$.*

- *Da alle Atome a mit* $\neg a \in N^P(M) \setminus N_{\mathsf{Project}(\Sigma,P)}(M)$ *in* $\mathsf{Project}(\Sigma, P)$ *nur negativ vorkommen, ist* M_2 *auch ein Modell für* $N^P(M)$*.*

- *Da* M_2 *ein Modell für* $\mathsf{Project}(\Sigma, P)$ *ist, kann gibt es ein Modell* M_2'*, das ein Modell für* Σ *ist. Diese Erweiterung fügt nur Belegungen für die wegprojizierten Atome hinzu. Also ist* M_2' *immer noch ein Modell für* $\overline{M}$ *und* $N^P(M)$*.*

- *Aus* $M_2' \models \{\overline{M}\}$*,* $M_2' \models \Sigma$ *und* $M_2' \models N^P(M)$ *folgt* $M_2' \models \cup\Sigma \cup N^P(M)$*.*

- *Diese Klauselmenge ist jedoch laut Voraussetzung unerfüllbar. Daher muss auch* $\{\overline{M}\} \cup \mathsf{Project}(\Sigma, P) \cup N_{\mathsf{Project}(\Sigma,P)}(M)$ *unerfüllbar sein.*

Im folgenden Beispiel wollen wir den oben in Beispiel 5.2.3 durchgeführten Minimalitätstest erneut vornehmen. Diesmal benutzen wir dabei jedoch den Grundierungstest für Zirkumskription.

Beispiel 5.2.12 *Sei wieder das Modell* $M = abInv \wedge abAnd$ *gegeben. Wir wollen es nun mit dem Grundierungstest für Zirkumskription auf Minimalität überprüfen. Die Grundmenge für unseren Grundierungstest ist* $\Delta|\alpha$*, wobei* Δ *die Klauselmenge ist, die den funktionierenden Schaltkreis beschreibt und* $\alpha = \neg a \wedge b \wedge \neg d$ *die Beobachtung ist.* $\Delta|\alpha$ *ist also gegeben als:*

$$abInv \vee c,$$
$$abAnd \vee \neg c$$

Die Menge P*, auf die wir den Schaltkreis ursprünglich projizieren wollten, ist gegeben als* $P = \{abInv, abAnd, a, b, d\}$*. Damit können wir nun die Menge* $N^P(M)$ *bestimmen:* $N^P(M) = \{\neg a, \neg b, \neg d\}$*. Die Negation des Modells* M *ergibt sich als* $\overline{M} = \neg abInv \vee \neg abAnd$*. Bilden wir nun die Menge* $\Delta|\alpha \cup N^P(M) \cup \{\overline{M}\}$ *und überprüfen, ob sie erfüllbar ist.*

$$\begin{aligned}\Delta|\alpha \cup N^P(M) \cup \{\overline{M}\} = & abInv \vee c,\\ & abAnd \vee \neg c,\\ & \neg a,\\ & \neg b,\\ & \neg d,\\ & \neg abInv \vee \neg abAnd\end{aligned}$$

Diese Klauselmenge ist erfüllbar, denn z.B. $M' = \{abAnd, c\}$ *ist ein Modell. Daher ist unser untersuchtes Modell* M *kein minimales Modell.*

Bei der Diagnose bietet es sich an, den Grundierungstest für Zirkumskription zu verwenden, da man sich dadurch die Bildung der Projektion auf der Ausgangsklauselmenge sparen kann. Es sei an dieser Stelle jedoch auf einen Unterschied zur Minimierung nach Darwiche [Dar01] hingewiesen. Darwiche minimiert die berechnete DNNF mit Hilfe der oben beschriebenen Minimize Operation. Das Ergebnis dieser Operation ist eine DNNF, die von genau den Modellen mit minimaler Kardinalität der ursprünglichen DNNF erfüllt werden. Bei der Modellberechnung auf DNNF-Teilstücken hingegen wird die Minimalität des bestimmten Modells mit Hilfe des Grundierungstests überprüft. Bei diesem Grundierungstest wird die Minimalität von Modellen wie üblich über die Mengeninklusion definiert. Daher handelt es sich also um zwei unterschiedliche Minimalitätsbegriffe. Es kann daher vorkommen, dass auf den DNNF-Teilstücken Modelle berechnet werden, die bei der Methode nach Darwiche nicht berechnet werden.

6 Informationsgesteuerte Bildung von DNNF Teilstücken

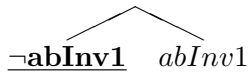
Es stellt sich nun die Frage, ob es sich lohnt, Modelle bereits auf DNNF-Teilstücken zu bestimmen. Wenn wir nur an minimalen Modellen interessiert sind, dann hängt die Effektivität der Bestimmung von Modellen auf DNNF-Teilstücken davon ab, wo die minimalen Modelle im Tableau erscheinen. Der schlechteste Fall sieht wie folgt aus: das einzige minimale Modell erscheint im letzten Zweig und alle anderen Zweige liefern nichtminimale Modelle. Dann haben wir zur Bestimmung des minimalen Modells die komplette DNNF erstellt und für jeden Zweig mindestens einmal den Grundierungstest durchgeführt. Finden wir dagegen im linken Zweig direkt ein minimales Modell, so haben wir uns dadurch die Berechnung der kompletten DNNF gespart. Daher ist es durchaus sinnvoll, sich über den Aufbau der Tableaux Gedanken zu machen. Es ist klar, dass der Aufbau eines Tableaus stark von der Wahl der Atome abhängt, mit denen die Verzweigungsregel durchgeführt wird. Betrachten wir wieder die Diagnose als Anwendung. Verzweigt man bei einem Diagnosebeispiel über die am häufigsten vorkommenden Atome, so läßt sich im allgemeinen nichts darüber aussagen, in welchem Zweig das erste minimale Modell erscheint. Wenn man jedoch Wissen über den Schaltkreis bei der Auswahl der Atome für die Verzweigungen benutzt, so kann man durchaus Aussagen über die Position von minimalen Modellen im Tableau machen. In der Regel kann man bei der Diagnose davon ausgehen, dass die meisten Komponenten im Schaltkreis korrekt funktionieren. Meist sind nur ein paar wenige Komponenten defekt. Daher ist es sinnvoll, dieses Wissen bei der Konstruktion des Tableaus auszunutzen. Stellen wir uns nun vor, dass wir eine initiale Interpretation benutzen. Eine initiale Interpretation ist eine Menge von Literalen, von denen wir annehmen, dass sie wahr sind, oder die zumindest mit hoher Wahrscheinlichkeit wahr sind. Diese initiale Interpretation könnte z.B. der Annahme entsprechen, dass alle Komponenten des Schaltkreises korrekt funktionieren. Eine solche initiale Interpretation würde wie folgt aussehen: $I = \{\neg abComp \mid abComp \textit{ ist ein Schaltelement im betrachteten Schaltkreis}\}$. Diese initiale Interpretation könnte man nun für die Wahl der Atome für die Verzweigungsregel verwenden. Wählt man für die Verzweigungsregel bevorzugt die Literale aus der initialen Interpretation, so erhält man ein Tableau, in dem der linke Zweig der initialen Interpretation entspricht. Im Fall der oben beschriebenen initialen Interpretation würde der linke Zweig der Annahme entsprechen, dass keine Komponente defekt ist. Bewegt man sich nun weiter nach rechts im Tableau, so werden nach und nach alle Kombinatio-

nen von defekten Schaltelementen aufgeführt. Tauchen die defekten Schaltelemente weit unten im Tableau auf, so ist damit zu rechnen, dass die Zweige im linken Bereich des Tableaus mit hoher Wahrscheinlichkeit minimale Modelle liefern. Betrachten wir hierzu ein Beispiel.

Beispiel 6.0.13 *Sei der Schaltkreis aus 1.1 gegeben. Die dazugehörige Klauselmenge* Δ *ist:*

$$abInv1 \vee \neg a \vee \neg b,$$
$$abInv1 \vee a \vee b,$$
$$abInv2 \vee \neg b \vee \neg c,$$
$$abInv2 \vee b \vee c$$

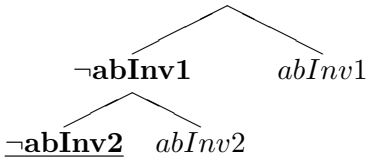
Außerdem sei die Beobachtung $\alpha = \neg a \wedge c$ *gegeben. Diese Beobachtung entspricht nicht dem erwarteten Verhalten des Schaltkreises. Eigentlich sollte bei Eingabe von* $\neg a$ *die Ausgabe* $\neg c$ *erscheinen. Im Folgenden suchen wir eine minimale Erklärung für das fehlerhafte Verhalten des Schaltkreises. Dazu bilden wir schrittweise die einzelnen DNNF-Teilstücke und untersuchen auf jedem dieser DNNF-Teilstücke, ob es eine minimale Erklärung für das beobachtete Verhalten des Schaltkreises liefert. Die schrittweise Umwandlung der Klauselmenge* Δ *in DNNF steuern wir wie oben beschrieben durch die initiale Interpretation* $I = \{\neg abInv1, \neg abInv2\}$*. Wir verzweigen also zuerst nach abInv1 und erhalten das folgende Tableau, in dem der linke Zweig aktiv ist:*



Wir bilden $\Delta|\neg abInv1$ *und erhalten:*

$$\neg a \vee \neg b,$$
$$a \vee b,$$
$$abInv2 \vee \neg b \vee \neg c,$$
$$abInv2 \vee b \vee c$$

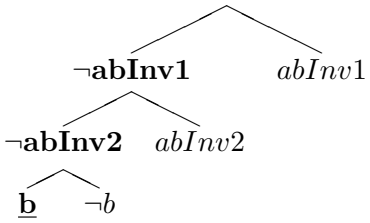
Diese Klauselmenge ist noch nicht in DNNF. Daher arbeiten wir am aktiven Zweig weiter und benutzen das nächste in der initialen Interpretation vorkommende Literal für die Verzweigungsregel.



Wir bilden $\Delta|\{\neg abInv1, \neg abInv2\}$ *und erhalten:*

$$\neg a \vee \neg b,$$
$$a \vee b,$$
$$\neg b \vee \neg c,$$
$$b \vee c$$

Auch diese Klauselmenge ist noch nicht in DNNF. Da wir bereits alle Literale aus der initialen Interpretation verwendet haben, sind wir bei der Wahl des Atoms für die nächste Verzweigung frei. Wir wählen hier b, da b das am häufigsten vorkommende Atom ist, das in mehreren Klauseln vorkommt:



Die Klauselmenge $\Delta|\{\neg abInv1, \neg abInv2, b\} = \{\neg a, \neg c\}$ *ist in DNNF. Der linke Zweig ist also fertig und wir können daraus die Teil-DNNF ablesen und erhalten:*

$$\Gamma_1 = \neg abInv1 \wedge \neg abInv2 \wedge b \wedge \neg a \wedge \neg c$$

Untersuchen wir nun, ob dieses DNNF-Teilstück eine Erklärung für $\alpha = \neg a \wedge c$ *liefert. Dafür bilden wir zunächst:*

$$\Gamma_1|\alpha = false$$

Da $\Gamma_1|\alpha = false$ *ist, liefert das erste DNNF-Teilstück leider keine Erklärung für das fehlerhafte Verhalten des Schaltkreises. Wir müssen also weitere DNNF-Teilstücke berechnen. Dafür wenden wir die Wechselregel an und arbeiten am nächsten Zweig weiter. Nun ist der Zweig mit der Beschriftung* $\neg abInv1, \neg abInv2, \neg b$ *aktiv. Wir bilden* $\Delta|\{\neg abInv1, \neg abInv2, \neg b\} = \{a, c\}$ *und stellen fest, dass wir auch an diesem Zweig bereits die DNNF erreicht haben. Wir lesen aus dem aktiven Zweig das DNNF-Teilstück ab:*

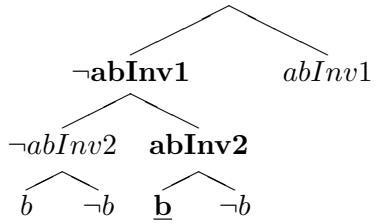
$$\Gamma_2 = \neg abInv1 \wedge \neg abInv2 \wedge \neg b \wedge a \wedge c$$

Auch dieses DNNF-Teilstück liefert keine minimale Erklärung für das beobachtete Verhalten des Schaltkreises, da $\Gamma_2|\alpha = false$ *ist. Wir wenden erneut die Wechselregel an*

und erhalten als aktiven Zweig den Zweig mit der Beschriftung $\neg abInv1, abInv2$. *Wir bilden* $\Delta|\{\neg abInv1, abInv2\}$*:*

$$\neg a \vee \neg b,$$
$$a \vee b$$

Bei dieser Klauselmenge ist die DNNF noch nicht erreicht. Wir müssen also erneut verzweigen. Auch in diesem Zweig haben wir bereits alle Literale aus der initialen Interpretation verwendet und wählen daher eins der geteilt vorkommenden Atome aus. Wir entscheiden uns für b *und erhalten das folgende Tableau in dem der Zweig mit der Beschriftung* $\neg abInv1, abInv2, b$ *aktiv ist:*



Da $\Delta|\{\neg abInv1, abInv2, b\} = \{\neg a\}$ *bereits in DNNF ist, ist der aktive Zweig fertig. Wir lesen aus ihm das DNNF-Teilstück ab:*

$$\Gamma_3 = \neg abInv1 \wedge abInv2 \wedge b \wedge \neg a$$

Untersuchen wir nun, ob das DNNF-Teilstück Γ_3 *eine minimale Erklärung für* $\alpha = \neg a \wedge c$ *liefert. Dafür bilden wir zunächst:*

$$\Gamma_3|\alpha = \neg abInv1 \wedge abInv2 \wedge b$$

Im nächsten Schritt projizieren wir $\Gamma_3|\alpha$ *auf die Menge der Ein- und Ausgabevariablen und die Schaltelemente und erhalten:*

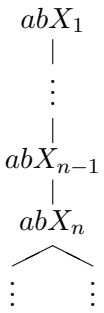
$$\mathit{Project}(\Gamma_3|\alpha, A \cup O) = \neg abInv1 \wedge abInv2$$

Hieraus läßt sich die Erklärung $M = abInv2$ *ablesen. Der Grundierungstest liefert die Bestätigung dafür, dass es sich bei* M *um eine minimale Erklärung für das beobachtete fehlerhafte Verhalten des Schaltkreises handelt. Schon der dritte Zweig unseres Tableaus liefert eine minimale Erklärung für das beobachtete Verhalten. In unserem Beispiel mussten wir nur einmal den Grundierungstest durchführen. Ist man nur an einer minimalen Erklärung interessiert, so ist man nach der Berechnung dieser drei Zweige fertig und muss die restliche DNNF nicht mehr bilden. Interessiert man sich hingegen für alle minimalen Erklärungen, so muss man auch die noch fehlenden DNNF-Teilstücke berechnen.*

Das obige Beispiel zeigt, dass eine gut gewählte initiale Interpretation die Umwandlung der Beschreibung des betrachteten Schaltkreises so steuern kann, dass bereits aus den ersten Zweigen eine minimale Erklärung abgelesen werden kann.

Durch zusätzliche Informationen kann die Umwandlung in DNNF optimiert werden. Ist z.B. bekannt, welche Schaltelemente eine hohe Ausfallwahrscheinlichkeit haben, so kann die initiale Interpretation entsprechend sortiert werden. Dabei werden die Schaltelemente der initialen Interpretation aufsteigend nach der Ausfallwahrscheinlichkeit sortiert. Dadurch erscheint das Schaltelement mit der höchsten Ausfallwahrscheinlichkeit am Ende der initialen Interpretation und tritt so auch im Tableau weit unten auf. Je weiter unten im Tableau ein Schaltelement erscheint, desto früher wird es durch Backtracking als defekt angenommen.

Außerdem ist es denkbar, dass wir wissen, dass bestimmte Teile des Schaltkreises korrekt funktionieren. Auch diese Informationen können wir bei der Bildung der DNNF Teilstücke ausnutzen. Nehmen wir dazu an, dass wir über die Information verfügen, dass die Bauteile $K = \{abX_1, \ldots, abX_n\}$ korrekt funktionieren. Dann muss keine Fallunterscheidung bezüglich dieser Bauteile vorgenommen werden. Wir können die gegebene Information als eine Menge von Einerklauseln ansehen, die zur Beschreibung des Schaltkreises hinzukommen. Sei nun Δ die Beschreibung des Schaltkreises. Dann können wir direkt am Anfang der Umwandlung die entsprechenden Weitergabe von Einerklauseln vornehmen und so den Suchraum erheblich verkleinern. Im Tableau sieht das wie folgt aus:



7 Implementierung

Im Rahmen dieser Diplomarbeit wurde ein Prologprogramm entwickelt, dass Klauselmengen in DNNF umwandelt. Dabei wird die DNNF schrittweise durch einen DPLL-basierten Algorithmus berechnet, so dass bereits auf DNNF-Teilstücken Modelle berechnet werden können. Außerdem besteht die Möglichkeit, verschiedene Heuristiken für die Auswahl der Verzweigungsatome zu verwenden.

1. **Häufigkeitsheuristik (Häufig-Heur.):** Das Atom, das am häufigsten in der Klauselmenge auftritt, wird für die nächste Verzweigung gewählt. Die zuerst untersuchte Polarität bestimmen wir zufällig.

2. **Polaritätsheuristik (Pol-Heur.):** Bei dieser Heuristik wird eine initiale Interpretation übergeben. Für die nächste Verzweigung wird wieder das am häufigsten vorkommende Atom ausgewählt. Diesmal wird die zuerst untersuchte Polarität jedoch aus der initialen Interpretation abgelesen. Ist das ausgewählte Atom nicht in der initialen Interpretation enthalten, so wird die Polarität wie bei der Häufigkeitsheuristik bestimmt.

3. **Verzweigung entsprechend der initialen Interpretation (Init-Heur.):** Auch hier wird eine initiale Interpretation übergeben. Die Literale aus der initialen Interpretation werden nun zuerst für Verweigungen ausgwählt. Sind alle diese Atome bereits im Zweig enthalten, so wird wie bei der Häufigkeitsheuristik ausgewählt.

4. **Kombination aus Häufigkeitsheuristik und initialer Interpretation (HInit-Heur.):** Hier wird eine initiale Interpretation übergeben. Bei der Bestimmung des Atoms für die nächste Verzweigung wird nun die Menge der Atome gebildet, die am häufigsten vorkommen. Das nächste Verzweigungsatom wir nun aus dem Schnitt diese Menge mit der initialen Interpretation ausgewählt. Ist dieser Schnitt leer, so wird wie das Atom wie bei der Häufigkeitsheuristik ausgewählt.

Der Hauptteil der Umwandlung wird von dem Prädikat `Convert` übernommen. Dieses Prädikat setzt den oben beschriebenen Kalkül und einige der Erweiterungen um. Sehen wir uns den Pseudocode zu `Convert` an:

```
convert(Clauses,Branch,_,_,(Branch & Clauses)):-
        isDNNF(Clauses),
```

```
        !.


convert(Clauses,Branch,Heuristic,Forgetatoms,DNNF):-
        unit(Clauses,Unit),
        \+ Unit=[],
        append(Unit,Branch,Newbranch),
        conditioning(Clauses,Unit,Condresult),
        \+ member([],Condresult),
        convert(Condresult,Newbranch,Heuristic,Forgetatoms,DNNF).


convert(Clauses,Branch,Heuristic,Forgetatoms,DNNF):-
        forget(Clauses,Forgetatoms,Stillforget,Newclauses),
        convert(Newclauses,Branch,Heuristic,Stillforget,DNNF).


convert(Clauses,Branch,Heuristic,ForgetAtoms,DNNF):-
        choose(Clauses,Split,Branch,Heuristic),
        (
           condition(Clauses,Split,Condres1),
           \+ member([],Condres1),
           convert(Condres1,[Split|Branch],Heuristic,Forgetatoms,DNNF)
        ;
           negation(Split,Negsplit),
           condition(Clauses,Negsplit,Condres2),
           \+ member([],Condres2),
           convert(Condres2,[Negsplit|Branch],Heuristic,Forgetatoms,DNNF)
        ).
```

`Convert` wird durch vier Regeln beschrieben. Bei der ersten Regel handelt es sich um den Abbruchfall. Hier wird überprüft, ob die betrachtete Klauselmenge bereits in DNNF vorliegt. Ist dies der Fall, so wird die Klauselmenge konjunktiv mit dem aktuellen Zweig verknüpft und zurückgegeben. Bei der zweiten Regel wird die Menge der Einerklauseln bestimmt und Weitergabe von Einerklauseln angewandt. Dabei werden alle gefundenen Einerklauseln zum aktuellen Zweig hinzugefügt und die betrachtete Klauselmenge wird durch Konditionierung vereinfacht. Die dritte Regel projiziert wie oben beschrieben auf das Komplement der Menge der zu vergessenden Atome. Das Prädikat `forget` wendet die oben beschriebene Isol- und Isol*-Regel an, um zu vergessende Atome aus der betrachteten Klauselmenge zu entfernen. Außerdem übernimmt `forget` die Beseitigung von pur vorkommenden Atomen, die vergessen werden sollen. Die vierte Regel

bestimmt das Atom, das für die nächste Verzweigung verwendet werden soll. Bei der Auswahl dieses Atoms fließt die ausgewählte Heuristik ein. Als nächstes wir die Verzweigung durchgeführt. Da die Regeln in dieser Reihenfolge im Programm erscheinen, ist sichergestellt, dass die Weitergabe von Einerklauseln und die Projektion der Verzweigung vorgezogen werden. Da die Diagnose von Schaltkreisen häufig als Anwendung für Wissenskompilation verwendet wird, wurde auch die Diagnose nach Darwiche implementiert. So können nun bereits auf DNNF-Teilstücken Erklärungen für das beobachtete Verhalten eines Schaltkreises berechnet werden. Um überprüfen zu können, ob die berechnete Erklärung minimal ist, wurde auch der Grundierungstest implementiert.

7.1 Test der einzelnen Heuristiken

Die Modellberechnung auf DNNF-Teilstücken sowie die Benutzung von verschiedenen initialen Interpretationen wurde mit Hilfe des oben erwähnten Programms auf einige der ISCAS89 Benchmarks [ISC89] angewandt. Bei den ISCAS89 Benchmarks handelt es sich um eine Sammlung von 31 digitalen Schaltwerken. Eine solche Schaltkreisbeschreibung kann nun mit einem beobachteten Verhalten des entsprechenden Schaltkreises kombiniert werden. Danach können Erklärungen für das beobachtete Verhalten berechnet werden.

7.1.1 Benchmarks

Einer der getesteten Schaltkreise ist s27. Dieser Schaltkreis hat vier Eingabe- und eine Ausgabevariable. Die Schaltung besteht aus drei D-FlipFolps, zwei Invertern, einem AND-, einem NAND-, zwei OR- und vier NOR-Gatter. Belegt man die vier Eingabevariablen mit true, so sollte die Ausgabevariable ebenfalls true liefern. Fügt man nun die Beobachtung hinzu, dass bei Belegung aller Eingabevariablen mit true, die Ausgabevariable false liefert, so gibt es für dieses Verhalten zwei minimale Erklärungen. Entweder ist der zweite Inverter defekt oder das zweite NOR-Gatter. Tabelle 7.1 zeigt, wo diese minimalen Erklärungen im Tableau auftauchen. In der ersten Spalte steht

Heuristik	i.I.	Zweig: $abInv2$	Zweig: $abNor2$	Zweige im Tableau
Häufig-Heur.	-	10,83	10,71	20
Pol-Heur.	$\neg abX$	10,64	11,216	20
Pol-Heur.	α	10,64	11,216	20
Init-Heur.	$\neg abX$	5	37	211
Init-Heur.	α	3,9	3,97	190
HInit-Heur.	$\neg abX$	10,65	11,68	20
HInit-Heur.	α	10,0	9,94	20

Tabelle 7.1: Ergebnisse von s27

die verwendete Heuristik. Hierbei handelt es sich um die auf Seite 67 aufgezählten Heuristiken. Die zweite Spalte gibt die verwendete initiale Interpretation an. Steht hier ein '-', so bedeutet dies, dass keine initiale Interpretation benutzt wurde. Die Angabe $\neg abX$ bedeutet, dass die initiale Interpretation für alle im Schaltkreis enthaltenen Komponenten X das Literal $\neg abX$ enthält. Das entspricht der Annahme, dass alle Komponenten korrekt funktionieren. Die Angabe α hingegen bedeutet, dass das beobachtete Verhalten des Schaltkreises als initiale Interpretation dient. In diesem Fall wird die Umwandlung der Klauselmenge in DNNF also durch die beobachtete Belegung der Ein- und Ausgabevariablen gesteuert. Die dritte Spalte bezieht sich auf die oben erwähnten minimalen Erklärung, dass der zweite Inverter defekt ist. Hier wird angegeben, in welchem Zweig im Durchschnitt diese minimale Erklärung gefunden wurde. Wobei die Zweige im Tableau von links nach rechts fortlaufend nummeriert werden. Die vierte Spalte ist analog zur dritten Spalte zu lesen. In der letzten Spalte wird die gesamte Anzahl der Zweige im Tableau angegeben.
Man erkennt, dass beide minimalen Modelle bei Verwendung der Häufigkeitsheuristik im Durchschnitt in der Mitte des Tableaus erscheinen. Bei Verwendung der Polaritätsheuristik ist dies, unanbhängig von der Wahl der initialen Interpretation, auch der Fall. Allerdings haben diese beiden Heuristiken den Vorteil, dass das berechnete Tableau insgesamt aus nur 20 Zweigen besteht. Bei Benutzung der Init-Heuristik hingegen werden eher große Tableaux erzeugt. Allerdings erscheinen die minimalen Modelle, bei geschickter Wahl der initialen Interpretation, schon wesentlich früher im Tableau. Interessiert man sich nur für ein einziges minimales Modell, so liefert diese Heuristik schon nach der Berechnung weniger DNNF-Teilstücke eine Antwort.

Bei diesem Programmläufen wurde die Projektion in die Berechnung der DNNF-Teilstücke mit einbezogen. Um den Vorteil davon zu unterstreichen, führen wir nun die gleichen Berechnungen für die Häufigkeitsheuristik erneut durch, ohne die Projektion in die Berechnung der DNNF-Teilstücke zu integrieren. Das Ergebnis zeigt Tabelle 7.2. Man

Heuristik	$abInv2$	$abNor2$	Zweige im Tableau
Häufig-Heur.	165,43	169,4	336

Tabelle 7.2: Ergebnisse von s27 ohne Projektion

sieht deutlich, dass die Berechnung ohne die Integration der Projektion aufwändiger ist. Das berechnete Tableau enthält wesentlich mehr Zweige. Dadurch, dass die Modelle bei der Häufigkeitsheuristik im Durchschnitt in der Mitte des Tableaus erscheinen, dauert es in diesem Fall wesentlich länger, bis das erste minimale Modell gefunden wird. Es ist also durchaus sinnvoll, die Projektion in die Berechnung der DNNF zu integrieren.

Betrachten wir nun einige größere Benchmarks. Da diese Benchmarks wesentlich größer als das oben betrachtete Beispiel s27 sind und es für ein beobachtetes Verhalten

eine wesentlich höhere Anzahl an minimalen Erklärungen gibt, interessieren wir uns im Folgenden stets dafür, in welchem Zweig die erste minimale Erklärung erscheint. Das nächste Beispiel ist s298. Es hat drei Eingabe- und 6 Ausgabevariablen. Es ist aus 44 Invertern, 14 D-Flipflops und 75 anderen Gattern aufgebaut. Werden die drei Eingabevariablen mit 1 belegt, so sollten die Ausgabevariablen $p_4 - p_8$ mit 1 und p_9 mit 0 belegt sein. Hier gehen wir jedoch davon aus, dass alle Ausgabevariablen eine 1 liefern und berechnen minimale Erkärungen für dieses Verhalten. Die Ergebnisse sind in Tabelle 7.3 aufgeführt. Die erste Spalte zeigt die verwendete Heuristik. Hierbei

Heuristik	i.I.	erstes minimales Modell
Häufig-Heur.	-	30,52
Pol-Heur.	$\neg abX$	30,05
Pol-Heur.	α	30,44
Init-Heur.	$\neg abX$	timeout!
Init-Heur.	α	1
HInit-Heur.	$\neg abX$	31,52
HInit-Heur.	α	31,35

Tabelle 7.3: Ergebnisse von s298

handelt es sich um die auf Seite 67 aufgezählten Heuristiken. In der zweiten Spalte wird angegeben, welche initiale Interpretation verwendet wurde. Hier bedeutet $\neg abX$, dass die initiale Interpretation für alle im Schaltkreis vorkommenden Komponenten X das Literal $\neg abX$ enthält. Steht α in dieser Spalte, so bedeutet das, dass das beobachtete Verhalten des Schaltkreises als initiale Interpretation dient. In der dritten Spalte wird angegeben, im welchem Zweig das erste minimale Modell gefunden wurde. Dabei handelt es sich um Durchschnittswerte, die aus 250 Programmläufen berechnet wurden. Die Zweige im Tableau wurden hier wieder von links nach rechts fortlaufend nummeriert.

Bei diesem Beispiel wird deutlich, welchen Nutzen eine gut gewählte initiale Interpretation bei der Init-Heuristik haben kann. Wählt man bei dieser Heuristik die Beobachtung α als initiale Interpratation, so wird bereits im ersten Zweig ein minimales Modell gefunden. Warum dies so ist, sieht man leicht, wenn man sich überlegt, dass nur in den Zweigen des Tableaus Erklärungen für die Beobachtung α gefunden werden können, die konsistent zu α sind. Durch die Steuerung der Umwandlung durch die Beobachtung erhalten wir ein zielgesteuertes Verhalten der Modellsuche.

Das nächste Beispiel ist s208. Es hat 10 Eingabe- und eine Ausgabevariable. Es ist aus 38 Invertern, 8 D-Flipflops und 66 Gattern aufgebaut. Belegt man in diesem Schaltkreis alle Eingabevariablen mit 1, so sollte die Ausgabevariable eine 1 liefern. Wir gehen hier davon aus, dass sie jedoch eine 0 liefert und berechnen minimale Erklärungen

für dieses Verhalten. Tabelle 7.4 zeigt die Ergebnisse. In der ersten Spalte wird die

Heuristik	i.I.	erstes minimales Modell	Lösung gefunden
Häufig-Heur.	-	1799	6 %
Pol-Heur.	$\neg abX$	5239	2 %
Pol-Heur.	α	72	12 %
Init-Heur.	$\neg abX$	-	0 %
Init-Heur.	α	3,13	16 %
HInit-Heur.	$\neg abX$	1302,25	8 %
HInit-Heur.	α	191,67	6 %

Tabelle 7.4: Ergebnisse von s208

verwendete Heuristik angegeben. Hierbei handelt es sich wieder um die auf Seite 67 aufgezählten Heuristiken. Die zweite Spalte gibt die verwendete initiale Interpretation an. Die Angabe $\neg abX$ bedeutet, dass die initiale Interpretation für alle im Schaltkreis vorkommenden Komponenten X das Literal $\neg abX$ enthält. Die Angabe α hingegen bedeutet, dass das beobachtete Verhalten des Schaltkreises als initiale Interpretation dient. Die dritten Spalte gibt die Nummer des Zweiges an, in dem das erste minimale Modell gefunden wurde. Dabei handelt es sich um Durchschnittswerte, die aus 250 Programmläufen berechnet wurden. Die Zweige im Tableau wurden hier wieder von links nach rechts fortlaufend nummeriert. Da dieses Beispiel ein sehr großes Tableau liefert, haben wir die Berechnung begrenzt. Wird bei einem Programmlauf innerhalb von 20 Sekunden kein minimales Modell gefunden, so wird die Berechnung abgebrochen. Steht bei einer Heuristik in der dritten Spalte ein '-', so bedeutet dies, dass bei keinem der Programmläufe in den ersten 20 Sekunden ein minimales Modell gefunden wurde. Die letzte Spalte gibt an, bei wieviel Prozent der Programmläufe in den ersten 20 Sekunden eine Lösung gefunden wurde.

Es ist auffällig, dass die Verwendung der Beobachtung α als initiale Interpretation bei der Init-Heuristik dazu führt, dass schon in den ersten Zweigen minimale Modelle gefunden werden.

Betrachten wir nun das Beispiel s349. Dieser Schlaltkreis besteht aus 9 Eingabe- und 11 Ausgabevariablen, 15 D-Flipflops, 57 Invertern und 104 anderen Schaltelementen. Werden alle Eingabevariablen mit 1 belegt, so sollten die Ausgabevariablen $p_{11} - p_{20}$ eine 0 und p_{10} eine 1 liefern. Wir gehen hier davon aus, dass jedoch alle Ausgabevariablen eine 0 liefern und suchen minimale Erklärungen für dieses Verhalten. Tabelle 7.5 zeigt die Ergebnisse. In der ersten Spalte wird die verwendete Heuristik angegeben. Hierbei handelt es sich wieder um die auf Seite 67 aufgezählten Heuristiken. In der zweiten Spalte wird die verwendete initiale Interpretation angegeben. Die Angabe $\neg abX$ bedeutet, dass

Heuristik	i.I.	erstes minimales Modell	Lösung gefunden
Häufig-Heur.	-	-	0 %
Pol-Heur.	$\neg abX$	61	2 %
Pol-Heur.	α	-	0 %
Init-Heur.	$\neg abX$	-	0 %
Init-Heur.	α	6,62	100 %
HInit-Heur.	$\neg abX$	169,5	4 %
HInit-Heur.	α	1631,32	38 %

Tabelle 7.5: Ergebnisse von s349

die initiale Interpretation für alle im Schaltkreis vorkommenden Komponenten X das Literal $\neg abX$ enthält. Das entspricht der Annahme, dass alle Komponenten korrekt funktionieren. Die Angabe α bedeutet, dass das beobachtete Verhalten des Schaltkreises als initiale Interpretation dient. In der dritten Spalte wird angegeben, in welchem Zweig das erste minimale Modell gefunden wurde. Da auch dieses Beispiel ein sehr großes Tableau erzeugt, beschränken wir wieder die maximale Zeit eines Programmlaufs. Wurde nach 20 Sekunden kein minimales Modell gefunden, so wurde der Programmlauf abgebrochen. In der letzten Spalte wird angegeben, bei viel Prozent der durchgeführten Programmläufe innerhalb der ersten 20 Sekunden eine minimale Erklärung gefunden wurde.

Auswertung: Zusammenfassend kann man sagen, dass das ungesteuerte Verzweigen der Häufigkeitsheuristik entsprechend, bei großen Beispielen zu langsam ist. Da bei der Häufigkeitsheuristik die minimalen Modelle im Durchschnitt in der Mitte des Tableaus erscheinen, ist diese Heuristik bei großen Beispielen nicht effizient. Bei geschickter Benutzung einer initialen Interpretation bei der dritten Heuristik hingegen, kann auch bei großen Schaltkreisen sehr schnell ein minimales Modell bestimmt werden.

8 Verwandte Arbeiten

Wie bereits erwähnt, gibt es viele verschiedene Ansätze zum Thema Wissenskompilation in der Aussagenlogik. Einen Ansatz, der dem Ansatz der DNNF sehr ähnelt, stellt die Dissolution dar. Unter [MR03] findet man eine ausführliche Einführung zum Thema Dissolution und den Zusammenhang zur Kompilation in DNNF. Mit Hilfe von Dissolution kann eine in NNF vorliegende Formel in einen vollständigen Dissolventen umgewandelt werden. Vollständige Dissolventen sind Formeln, die in NNF vorliegen und verbindungslos (linkless) sind.

Definition 8.0.1 ***(verbindungslos)*** *Eine Formel F heißt verbindungslos, wenn für jede in F enthaltene Konjunktion $\alpha = \alpha_1 \wedge \ldots \wedge \alpha_n$ gilt: es gibt kein Atom a mit $a \in atome(\alpha_1) \cap \ldots \cap atome(\alpha_n)$ und $\neg a \in atome(\alpha_1) \cap \ldots \cap atome(\alpha_n)$.*

Damit ist klar, dass jede Formel, die in DNNF vorliegt automatisch auch ein vollständiger Dissolvent ist. Zusätzlich gilt für Formeln in DNNF jedoch noch die Zerlegbarkeitseigenschaft. Dies ist eine strengere Eigenschaft als die Eigenschaft verbindungslos. Daher stellt sich die Frage, welche Vorteile die Zerlegbarkeitseigenschaft für die spätere Verarbeitung der Formel bietet. Betrachten wir nun kurz die Eigenschaften von vollständigen Dissolventen und vergleichen sie mit den Eigenschaften einer in DNNF vorliegenden Formel.

Eine sehr häufige Anwendung ist die folgende: eine Wissensbasis K und eine Klausel C sind gegeben. Es soll nun überprüft werden, ob die Klausel C eine Folge aus der Wissensbasis K ist. Die Frage, ob $K \models C$ gilt, ist äquivalent zur Frage ob $(K \wedge \neg C)$ unerfüllbar ist. Wir haben bereits erläutert, dass auf Formeln in DNNF ein Erfüllbarkeitstest in linearer Zeit möglich ist. Aber auch auf vollständigen Dissolventen ist es möglich, in linearer Zeit zu überprüfen, ob $K \models C$ gilt. Eine ausführliche Begründung dafür findet sich unter [MR03]. Bei der Projektion eines vollständigen Dissolventen auf eine Menge von Atomen geht man wie bei der Projektion einer DNNF vor. Daher ist auch hier der Aufwand linear.

Allerdings kann man vollständige Dissolventen nicht mit den oben beschriebenen Mitteln[1] minimieren. Das Problem ist die Berechnung des MCard Wertes einer Formel F. $\mathsf{MCard}(F)$ gibt die minimale Kardinalität aller Modelle der Formel F an. Mit anderen Worten gibt $\mathsf{MCard}(F)$ die minimale Anzahl von Atomen an, die mit wahr belegt werden müssen, damit die Formel F wahr wird.

[1] Siehe dazu Definition 2.2.15 auf Seite 13.

Beispiel 8.0.2 *Die Formel* $F = (d \vee a) \wedge (a \vee b)$ *ist ein vollständiger Dissolvent, liegt aber nicht in DNNF vor. Berechnen wir nun* $\mathsf{MCard}(F)$*:*

$$\begin{aligned} & \mathsf{MCard}((d \vee a) \wedge (a \vee b)) \\ = \ & \mathsf{MCard}(d \vee a) + \mathsf{MCard}(a \vee b) \\ = \ & min(\mathsf{MCard}(d), \mathsf{MCard}(a)) + min(\mathsf{MCard}(a), \mathsf{MCard}(b)) \\ = \ & min(1,1) + min(1,1) \\ = \ & 2 \end{aligned}$$

Aber bereits $M = \{a\}$ *ist ein Modell für* F *und* $|M| = 1$*.*

Obwohl die Minimierung vollständiger Dissolventen nicht so einfach wie bei Formeln in DNNF funktioniert, bieten sich auch verbindungslose Formeln für die Wissenskompilation an.

9 Fazit und Ausblick

Wir haben in dieser Arbeit die Möglichkeit kennengelernt, bereits auf teilweise präkompilierten Wissensbasen Modelle zu berechnen. Auch das Problem der eventuellen Nichtminimalität dieser Modelle wurde gelöst. Außerdem wurde eine Möglichkeit vorgestellt, um die Umwandlung von Klauselmengen in DNNF durch zusätzlich vorhandene Informationen zu steuern. In Kapitel 7 haben wir gesehen, dass bei geschickter Wahl einer initialen Interpretation auch bei den größeren ISCAS89 Benchmarks bereits in den ersten Zweigen ein minimales Modell gefunden werden kann.

Es stellt sich die Frage, ob die vorgeschlagene Modellberechnung auf DNNF-Teilstücken auch für verbindungslose Formeln funktioniert. Die Eigenschaften, die eine Formel haben muss, um verbindungslos zu sein, stellen weniger starke Forderungen dar, als die bei der DNNF geforderten Eigenschaften. Daher ist die Umwandlung von Klauselmengen in DNNF aufwändiger als die Umwandlung in eine verbindungslose Formel. Der oben beschriebene DPLL-FTab Kalkül kann leicht so abgeändert werden, dass er Klauselmengen in eine verbindungslose Formel umwandelt. Wir haben gesehen, dass verbindungslose Formeln auch sehr einfach projiziert werden können. Auch die Modellbestimmung ist wenig aufwändig. Daher ist es denkbar, analog zu DNNF-Teilstücken, verbindungslose-Teilstücke zu berechnen und darauf Modelle zu bestimmen. Allerdings haben wir oben gesehen, dass die Minimierung verbindungsloser Formeln nicht so einfach wie bei Formeln in DNNF funktioniert. Dies stellt für unsere Anwendung jedoch keine Einschränkung dar, da der in dieser Arbeit beschriebene Grundierungstest auch in diesem Fall als Minimalitätstest für die berechneten Modelle dienen kann. Besonders für die Diagnose von Schaltkreisen wäre dieser Ansatz interessant. Denn bei in der Diagnose sind die betrachteten Klauselmengen bereits in Bezug auf alle Atome in $\{abX \mid X \textit{ ist Schaltelement im betrachteten Schaltkreis }\}$ verbindungslos. Daher ist zu erwarten, dass die Umwandlung der Schaltkreisbeschreibung in eine verbindungslose Formel wesentlich weniger aufwändig ist, als die Umwandlung in DNNF. Auch die Anwendung einzelner Heuristiken, insbesondere die Verwendung von initialen Interpretationen, wäre eine interessante Weiterführung dieser Diplomarbeit.

10 Glossar

Einerklausel Unit Clause

gleichmäßige DNNF smooth DNNF

grundiert grounded

Grundierungstest Groundednestest

Konditionierung Conditioning

verbindungslos linkless

Weitergabe von Einerklauseln Unit Propagation

Wissenskompilation Knowledge Compilation

Zerlegbare Negationsnormalform Decomposable Negation Normalform

Zerlegbarkeitseigenschaft Decomposability Property

Zirkumskription Circumscription

Literaturverzeichnis

[BFFN97] Peter Baumgartner, Peter Fröhlich, Ulrich Furbach und Wolfgang Nejdl: *Semantically Guided Theorem Proving for Diagnosis Applications.* In: M. E. Pollack (Herausgeber): *15th International Joint Conference on Artificial Intelligence (IJCAI 97)*, Nagoya, 1997. Morgan Kaufmann.

[Bib92] Wolfgang Bibel: *Deduktion, Automatisierung der Logik.* R. Oldenbourg Verlag, Wien, 1992.

[CA93] James M. Crawford und Larry D. Auton: *Experimental Results on the Crossover Point in Satisfiability Problems.* In: *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI)*, Washington, 1993. The AAAI Press/The MIT Press.

[Dar98] Adnan Darwiche: *Model-Based Diagnosis using Structured System Descriptions.* Journal of Artificial Intelligence Research (JAIR), 8, 1998.

[Dar00] Adnan Darwiche: *On the Tractable Counting of Theory Models and its Application to Belief Revision and Truth Maintenance.* CoRR, cs.AI/0003044, 2000.

[Dar01] Adnan Darwiche: *Decomposable Negation Normal Form.* Journal of the ACM, 48(4), 2001.

[Dar04] Adnan Darwiche: *New Advances in Compiling CNF into Decomposable Negation Normal Form.* In: Ramon López de Mántaras und Lorenza Saitta (Herausgeber): *Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI-04)*, Valencia, 2004. IOS Press.

[DD01] Olivier Dubois und Gilles Dequen: *A Backbone Search Heuristic for Efficient Solving of Hard 3-SAT Formulae.* In: Bernhard Nebel (Herausgeber): *17th International Joint Conference on Artificial Intelligence (IJCAI 01)*, Seattle, 2001. Morgan Kaufmann.

[DGP04] Heidi E. Dixon, Matthew L. Ginsberg und Andrew J. Parkes: *Generalizing Boolean Satisfiability I: Background and Survey of Existing Work.* Journal of Artificial Intelligence Research (JAIR), 21, 2004.

[DH05] Adnan Darwiche und Jinbo Huang: *DPLL with a Trace: From SAT to Knowledge Compilation*. In: Leslie Pack Kaelbling und Alessandro Saffiotti (Herausgeber): *19th International Joint Conference on Artificial Intelligence (IJCAI 05)*, Nagoya, 2005.

[DLL62] Martin Davis, George Logemann und Donald Loveland: *A Machine Program for Theorem Proving*. Communications of the ACM, 5(7), Juli 1962.

[DP60] Martin Davis und Hilary Putman: *A Computing Procedure for Quantification Theory*. Journal of the ACM, 7(3), Juli 1960.

[ISC89] *The ISCAS89 Benchmarks*. http://www.cbl.ncsu.edu, 1989.

[McC80] John L. McCarthy: *Circumscription - A Form of Non-Monotonic Reasoning*. Artificial Intelligence, 13(1-2), 1980.

[MR03] Neil V. Murray und Erik Rosenthal: *Tableaux, Path Dissolution, and Decomposable Negation Normal Form for Knowledge Compilation*. In: Marta Cialdea Mayer und Fiora Pirri (Herausgeber): *Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2003)*, Band 2796 der Reihe *Lecture Notes in Computer Science*. Springer, 2003.

[Nie96a] Ilkka Niemelä: *A Tableau Calculus for Minimal Model Reasoning*. In: Pierangelo Miglioli, Ugo Moscato, Daniele Mundici und Mario Ornaghi (Herausgeber): *Theorem Proving with Analytic Tableaux and Related Methods (TABLEAUX 1996)*, Band 1071 der Reihe *Lecture Notes in Computer Science*. Springer, 1996.

[Nie96b] Ilkka Niemelä: *Implementing Circumscription Using a Tableau Method*. In: Wolfgang Wahlster (Herausgeber): *Proceedings of the 12th Eureopean Conference on Artificial Intelligence (ECAI-96)*, Budapest, 1996. John Wiley and Sons, Chichester.

[SS95] Wolfram Schiffmann und Robert Schmitz: *Technische Informatik 1, Grundlagen der digitalen Elektronik*. Springer-Verlag, 1995.

[Sta00] Michael Stanek: *Model-Aided Diagnoses for High Voltage Circuit Breakers*. Doktorarbeit, Vienna University of Technology, 2000. Im Web: http://e-collection.ethbib.ethz.ch/ecol-pool/diss/fulltext/eth13507.pdf.

[Wer06a] Christoph Wernhard: *A Formalized Variant of DPLL for Formula Transformation*. Technischer Bericht, Universität Koblenz-Landau, 2006. In Vorbereitung.

[Wer06b] Christoph Wernhard: *Tableaux Between Proving, Projection and Compilation*. Technischer Bericht, Universität Koblenz-Landau, 2006. In Vorbereitung.

[Zha] Lintao Zhang: *Efficient SAT-Solving: From Davis-Putman to Zchaff and Beyond*. http://research.microsoft.com/users/lintaoz/SATSolving/sat_course2.pdf.

[ZM02] Lintao Zhang und Sharad Malik: *The Quest for Efficient Boolean Satisfiability Solvers*. In: Andrei Voronkov (Herausgeber): *18th International Conference on Automated Deduction (CADE-02)*, Band 2392 der Reihe *Lecture Notes in Computer Science*, Kopenhagen, 2002. Springer.