

Eine für mehrere Algorithmen bedeutsame Beobachtung ist, dass α genau dann erfüllbar ist, wenn wenigstens eine der beiden Formeln erfüllbar ist, die sich aus α ergeben, wenn ein fester Wahrheitswert für eines der Atome x unterstellt wird. Für KNF-Formeln kann die Vorgabe von Wahrheitswerten für Aussagensymbole sehr leicht durch folgende syntaktische Transformationen “simuliert” werden.

Da das Streichen tautologischer Klauseln, also von Klauseln der Form $\dots \vee \neg x \vee x \vee \dots$, die zueinander komplementäre Literale enthalten, keinen Einfluss auf die Wahrheitswerte einer KNF-Formel hat, können wir im Folgenden o.E. voraussetzen, dass die gegebene KNF-Formel α keine tautologischen Klauseln enthält. Wegen der Idempotenzgesetze kann man weiter annehmen, dass die Literale in den Klauseln paarweise verschieden sind.

Definition 4.24 (Ersetzung von Atomen durch Wahrheitswerte). Sei α eine KNF-Formel und x ein Atom. Die Formel $\alpha[x/0]$ entsteht aus α , indem folgende Transformationen durchgeführt werden:

- (1) In jeder Klausel von α , welche das positive Literal x enthält, wird das Literal x gestrichen:
- (2) Jede der verbliebenen Klausel, welche das negative Literal $\neg x$ enthält, wird (ersatzlos) gestrichen.

Abbildung 33 veranschaulicht die Definition. In Analogie hierzu sind die Formeln $\alpha[x/1]$ definiert. Im ersten Schritt ist jede Klausel, die das negative Literal $\neg x$ enthält, um $\neg x$ zu verkürzen. Der zweite Schritt eliminiert alle Klauseln, welche das positive Literal x enthalten. ■

$$\begin{array}{c}
 \kappa_1 \wedge \dots \wedge \kappa_{i-1} \wedge (L_1 \vee \dots \vee L_{j-1} \vee x \vee L_{j+1} \vee \dots \vee L_k) \wedge \kappa_{i+1} \wedge \dots \wedge \kappa_m \\
 \quad \quad \quad (1) \quad \Downarrow \text{wird ersetzt durch} \\
 \kappa_1 \wedge \dots \wedge \kappa_{i-1} \wedge (L_1 \vee \dots \vee L_{j-1} \vee L_{j+1} \vee \dots \vee L_k) \wedge \kappa_{i+1} \wedge \dots \wedge \kappa_m \\
 \\
 \kappa_1 \wedge \dots \wedge \kappa_{i-1} \wedge (L_1 \vee \dots \vee L_{j-1} \vee \neg x \vee L_{j+1} \vee \dots \vee L_k) \wedge \kappa_{i+1} \wedge \dots \wedge \kappa_m \\
 \quad \quad \quad (2) \quad \Downarrow \text{wird ersetzt durch} \\
 \kappa_1 \wedge \dots \wedge \kappa_{i-1} \wedge \kappa_{i+1} \wedge \dots \wedge \kappa_m
 \end{array}$$

Abbildung 33: Ersetzen von x durch Wahrheitswert 0 in α

Die Rechtfertigung für die beiden Schritte der Transformation ist wie folgt. Zunächst stellen wir fest, dass die Vorgabe des Wahrheitswerts 0 für x einer syntaktischen Ersetzung von x durch *false* entspricht. Klauseln, welche das positive Literal x enthalten, etwa

$$\kappa = x \vee L_2 \vee \dots \vee L_k,$$

werden durch die Ersetzung von x durch *false* zu:

$$\text{false} \vee L_2 \vee \dots \vee L_k \equiv \underbrace{L_2 \vee \dots \vee L_k}_{\kappa \text{ nach Streichen des positiven Literals } x}$$

Dies erklärt den ersten Schritt der Transformation, in dem alle Klauseln um das positive Literal x verkürzt werden. Schritt 2 der Transformation beruht auf der Beobachtung, dass alle Klauseln von α , die das negative Literal $\neg x$ enthalten, durch die Ersetzung von x durch *false* zu Tautologien werden und daher gestrichen werden können. Beachte:

$$\underbrace{(\dots \vee \neg \text{false} \vee \dots)}_{\equiv \text{true}} \wedge \kappa_2 \wedge \dots \wedge \kappa_m \equiv \text{true} \wedge \kappa_2 \wedge \dots \wedge \kappa_m \\ \equiv \kappa_2 \wedge \dots \wedge \kappa_m$$

Die Formeln $\alpha[x/0]$ und $\alpha[x/\text{false}]$ sind also äquivalent, wobei $\alpha[x/\text{false}]$ durch die syntaktische Ersetzung aller Vorkommen des Atoms x in α durch die Formel *false* entsteht. Ebenso sind $\alpha[x/1]$ und $\alpha[x/\text{true}]$ äquivalent. Hieraus ergibt sich, dass die Formeln $\alpha[x/b]$ folgende Eigenschaft haben, von der wir im Folgenden sehr häufig Gebrauch machen werden:

Lemma 4.25. *Sei α eine KNF-Formel über AP und $x \in AP$. Dann gilt für jede Belegung I:*

$$\alpha^I = \begin{cases} \alpha[x/0]^I & : \text{ falls } x^I = 0 \\ \alpha[x/1]^I & : \text{ falls } x^I = 1 \end{cases}$$

Beispiel 4.26 (Ersetzung von Atomen durch Wahrheitswerte in KNF-Formeln). Wir betrachten folgende Formel

$$\alpha = (x \vee \neg y \vee \neg z) \wedge (y \vee z \vee \neg w \vee \neg v) \wedge (\neg x \vee w) \wedge (\neg x \vee \neg z).$$

Zur Bildung von $\alpha[x/0]$ werden x aus der ersten Klausel entfernt und die dritte und vierte Klausel ersatzlos gestrichen. Die zweite Klausel wird unverändert übernommen, da sie weder x noch $\neg x$ als Literal enthält. Man erhält also:

$$\alpha[x/0] = (\neg y \vee \neg z) \wedge (y \vee z \vee \neg w \vee \neg v)$$

Zur Bildung von $\alpha[x/1]$ wird die erste Klausel gestrichen (da sie das positive Literal x enthält) und die zweite Klausel bleibt unverändert. Aus der dritten und vierten Klausel wird jeweils $\neg x$ gestrichen, also:

$$\alpha[x/1] = (y \vee z \vee \neg w \vee \neg v) \wedge w \wedge \neg z.$$

Durch das Streichen von Literalen aus Einheitsklauseln (im ersten Schritt der Transformation $\alpha \rightsquigarrow \alpha[x/\dots]$) entsteht stets die leere Klausel. Ist z.B.

$$\gamma = (\neg y \vee \neg z) \wedge x,$$

so ist

$$\gamma[x/0] = (\neg y \vee \neg z) \wedge \perp.$$

Für die resultierende Formel $\alpha[x/0]$ kann das Erfüllbarkeitsproblem daher sofort mit “nein” beantwortet werden. (Siehe zweiter Sonderfall “leere Klausel”.) Für KNF-Formeln β , in der jede Klausel das negative Literal $\neg x$ enthält, entsteht beim Übergang zu $\beta[x/0]$ die triviale KNF-Formel *true* (die keine Klauseln enthält). Z.B. werden bei der Erstellung von $\beta[x/0]$ für

$$\beta = (\neg x \vee y \vee z) \wedge (\neg x \vee \neg z)$$

beide Klauseln gestrichen. Also ist $\beta[x/0] = \text{true}$. ■

Offenbar kommt das Atom x in $\alpha[x/b]$ nicht mehr vor. Weiter ist klar, dass die Formeln $\alpha[x/0]$ und $\alpha[x/1]$ höchstens so viele Klauseln wie α enthalten und in Zeit $\mathcal{O}(|\alpha|)$ erstellt werden können. Ferner enthält jede der Klauseln von $\alpha[x/0]$ und $\alpha[x/1]$ höchstens so viele Literale wie die entsprechende Klausel von α .

Lemma 4.27 (Splitting-Regel). *Sei α eine KNF-Formel, x ein Atom. Dann gilt: α ist genau dann erfüllbar, wenn wenigstens eine der beiden Formeln $\alpha[x/0]$ oder $\alpha[x/1]$ erfüllbar ist.*

Beweis. “ \implies ”: Sei α erfüllbar und I eine erfüllende Belegung für α . Aus Lemma 4.25 folgt:

- Falls $x^I = 1$, so ist I ein Modell für $\alpha[x/1]$. Also ist $\alpha[x/1]$ erfüllend.
- Falls $x^I = 0$, so ist I ein Modell für $\alpha[x/0]$. Also ist $\alpha[x/0]$ erfüllend.

“ \impliedby ”: Ist etwa $\alpha[x/0]$ erfüllbar und I ein Modell für $\alpha[x/0]$, so betrachten wir die Belegung $I[x := 0]$, welche auf allen Atomen $y \neq x$ mit I übereinstimmt und x den Wahrheitswert 0 zuordnet. Also:

$$y^{I[x:=0]} \stackrel{\text{def}}{=} \begin{cases} y^I & : \text{ falls } y \in AP \setminus \{x\} \\ 0 & : \text{ falls } y = x \end{cases}$$

Wegen $x^{I[x:=0]} = 0$ und da x nicht in $\alpha[x/0]$ vorkommt, gilt:

$$\alpha^{I[x:=0]} = \alpha[x/0]^{I[x:=0]} = \alpha[x/0]^I = 1,$$

wobei in der ersten Gleichheit Lemma 4.25 für die Belegung $I[x := 0]$ benutzt wurde. □

Beruhend auf der Splitting-Regel kann das naive Backtracking für beliebige aussagenlogische Formeln (Algorithmus 6) für KNF-Formeln umformuliert werden. Anstelle einer expliziten Darstellung der Belegungen (mit Hilfe der booleschen Variablen b_1, \dots, b_n) verkürzen wir hier die auf Erfüllbarkeit zu testende KNF-Formel α , indem wir zu $\alpha[x/1]$ bzw. $\alpha[x/0]$ übergehen. Dies ist in Algorithmus 7 formuliert.

Algorithmus 7 Naiver KNF-SAT-Beweiser: rekursiver Algorithmus $\text{SAT}(\alpha)$

```

(*)      Eingabe:      KNF-Formel  $\alpha$   *)
(* Rückgabe:  true, falls  $\alpha$  erfüllbar. Andernfalls false.  *)

```

```

IF  $\alpha$  enthält die leere Klausel THEN return false  FI
IF  $\alpha = true$  THEN return true  FI
wähle ein Atom  $x$ , das in  $\alpha$  vorkommt  (* Splitting-Regel *)
IF  $\text{SAT}(\alpha[x/0])$ 
    THEN return true
    ELSE return  $\text{SAT}(\alpha[x/1])$ 
FI

```

Die Rekursionsstruktur liefert im Wesentlichen den Entscheidungsbaum, jedoch besteht die Chance, dass $\alpha[x/\dots]$ deutlich kürzer als α ist und neben x auch andere Atome, die in α vorkommen, keine Vorkommen in $\alpha[x/\dots]$ haben. Da hier nur nach der Erfüllbarkeit gefragt ist,

kann das Verfahren abgebrochen werden, sobald ein mit *true* beschriftetes Blatt erreicht wird, da dann $\alpha[\dots] = \text{true}$ und somit auf die Erfüllbarkeit von α geschlossen werden kann.

Der Davis-Putnam-Algorithmus

Der Davis-Putnam-Algorithmus ist ein KNF-SAT-Beweiser, der sich in der Praxis sehr gut bewährt hat und auf dem naiven Backtracking-Algorithmus beruht. Die Idee besteht darin, durch den Einsatz gewisser Regeln das binäre Verzweigen gemäß der Splitting-Regel zu verhindern.

Im Folgenden verwenden wir häufig die Abkürzung “DP-Algorithmus”. In der Literatur wird dieser auch häufig DPLL-Algorithmus genannt, wobei die vier Buchstaben DPLL für die Namen der vier Wissenschaftler Davis, Putnam, Logeman und Loveland stehen, auf die die wesentlichen Konzepte des Algorithmus zurückgehen.

Zunächst stellen wir fest, dass die Reihenfolge, in welcher die Atome bei dem naiven Backtracking-Verfahren betrachtet werden, entscheidenden Einfluss auf die Anzahl an Rekursionsaufrufe haben kann. Liegt z.B. die unerfüllbare KNF-Formel

$$\alpha = (x \vee y \vee z) \wedge (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee \neg y \vee z) \wedge \neg z$$

vor, so generiert der naive Backtracking-Algorithmus für die Reihenfolge x, y, z den vollständigen Entscheidungsbaum für α . Wesentlich günstiger ist die Reihenfolge z, y, x , da die Verzweigung $z/1$ sofort zu einer leeren Klausel führt und damit den rechten Teilbaum des Entscheidungsbaums für α nicht generiert. Diese Beobachtung lässt sich für beliebige KNF-Formel verallgemeinern, in denen eine Einheitsklausel vorkommt.

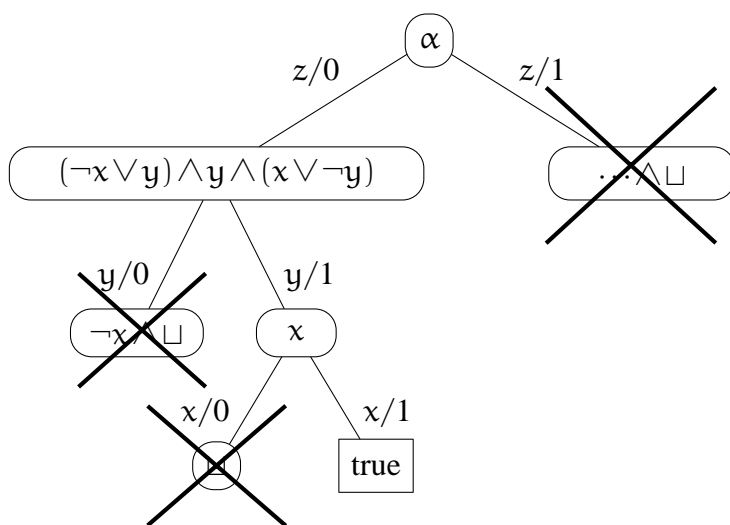


Abbildung 34: Beispiel zur Unit-Regel

Abbildung 34 illustriert diese Beobachtung am Beispiel der KNF-Formel

$$\alpha = (\neg x \vee y) \wedge (\neg z \vee y) \wedge (\neg z \vee x \vee \neg y) \wedge \neg z,$$

in der am Ende die Einheitsklausel $\neg z$ vorkommt. Die Formel

$$\alpha[z/0] = (\neg x \vee y) \wedge y \wedge (x \vee \neg y)$$

enthält die Einheitsklausel y . Daher ist nur $y/1$ relevant, während die Verzweigung $y/0$ eingespart werden kann, da $\alpha[z/0, y/0]$ die leere Klausel enthält.

Nehmen wir nun an, dass α eine KNF-Formel, in der eine Einheitsklausel vorkommt. Etwa $\alpha = \dots \wedge x \wedge \dots$. Ferner sei I eine Interpretation. Offenbar kann I höchstens dann für α erfüllend sein, wenn $x^I = 1$. In diesem Fall gilt $\alpha^I = \alpha[x/1]^I$ (siehe Lemma 4.25). Ist I also ein Modell für α , so ist I zugleich ein Modell für $\alpha[x/1]$. Ferner kann für jede erfüllende Belegung I für $\alpha[x/1]$ angenommen werden, dass x mit 1 belegt ist, da x keine Vorkommen in $\alpha[x/1]$ hat. Damit ergibt sich Teil (a) von Lemma 4.28. Die Aussage in Teil (b) ist symmetrisch und betrifft negative Einheitsklauseln.

Lemma 4.28 (Unit-Regel). *Sei α eine KNF-Formel und x ein Atom. Dann gilt:*

- (a) *Enthält α die Einheitsklausel x , so ist α genau dann erfüllbar, wenn $\alpha[x/1]$ erfüllbar ist.*
- (b) *Enthält α die negative Einheitsklausel $\neg x$, so ist α genau dann erfüllbar, wenn $\alpha[x/0]$ erfüllbar ist.*

Die Unit-Regel erlaubt es also, auf das binäre Verzweigen mit der Splitting-Regel zu verzichten und stattdessen nur eine der Formeln $\alpha[x/1]$ oder $\alpha[x/0]$ auf Erfüllbarkeit zu untersuchen (je nachdem ob die negative Einheitsklausel $\neg x$ oder die positive Einheitsklausel x in α vorkommt). Dies führt zur ersten Verfeinerung des naiven Backtracking-Algorithmus mit Hilfe der Unit-Regel, welche eingesetzt wird, wann immer sich eine KNF-Formel mit Einheitsklauseln ergibt.

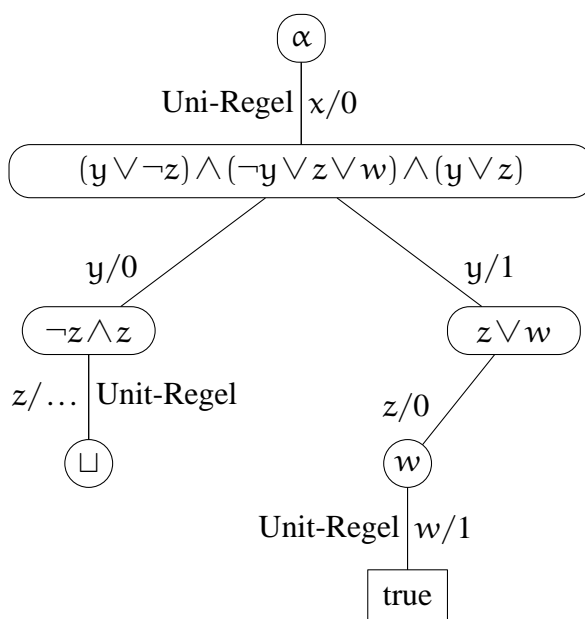


Abbildung 35: Beispiel zum Einsatz der Unit-Regel im DP-Algorithmus

Abbildung 35 zeigt den Rekursionsbaum, der sich für die erfüllbare Formel

$$\alpha = (x \vee y \vee \neg z) \wedge (x \vee \neg y \vee z \vee w) \wedge (x \vee y \vee z) \wedge \neg x$$

ergibt, wenn die Splitting-Regel nur dann eingesetzt wird, wenn die Unit-Regel nicht anwendbar ist, also eine KNF-Formel ohne Einheitsklauseln vorliegt. Durch den Einsatz der Unit-Regel

finden also nur sieben Rekursionsaufrufe statt bis das “linkeste” mit *true* beschriftete Blatt gefunden wird. Für die Formel

$$\begin{aligned}\alpha &= (x \vee y \vee \neg z) \wedge (y \vee z) \wedge \beta, & \text{wobei} \\ \beta &= (z \vee w) \wedge (z \vee \neg w) \wedge (\neg z \vee w) \wedge (\neg z \vee \neg w)\end{aligned}$$

greift die Unit-Regel erst auf den unteren Ebenen, sofern zuerst die Variablen x und y betrachtet werden. Es gilt jedoch $\alpha[x/0] = \alpha[x/1] \wedge (y \vee \neg z)$. Aus diesem Grund wäre es ausreichend, die kürzere Formel $\alpha[x/1]$ auf Erfüllbarkeit zu testen, was mit der nun folgenden Pure-Literal-Regel erreicht wird.

Wir sagen, dass das Atom x in α *nur positiv* vorkommt, wenn keine der Klauseln von α das negative Literal $\neg x$ enthält und das positive Literal x wenigstens in einer Klausel von α enthalten ist. Entsprechend bedeutet die Formulierung, “ x kommt in α *nur negativ* vor”, dass keine der Klauseln von α das positive Literal x enthält und das negative Literal $\neg x$ wenigstens in einer Klausel von α enthalten ist.

Lemma 4.29 (Die Pure-Literal-Regel). *Sei α eine KNF-Formel und x ein Atom, das in α vorkommt. Dann gilt:*

- (a) *Kommt x nur positiv in α vor, so ist α genau dann erfüllbar, wenn $\alpha[x/1]$ erfüllbar ist.*
- (b) *Kommt x nur negativ in α vor, so ist α genau dann erfüllbar, wenn $\alpha[x/0]$ erfüllbar ist.*

Beweis. Wir weisen nur Teil (a) nach. Aussage (b) folgt mit analogen Argumenten. Aufgrund der Splitting-Regel impliziert die Erfüllbarkeit von $\alpha[x/1]$ die Erfüllbarkeit von α . Wir nehmen nun an, dass α erfüllbar ist und dass x nur positiv in α vorkommt. Zu zeigen ist die Erfüllbarkeit von $\alpha[x/1]$. Sei I ein Modell für α .

- Ist $x^I = 1$, so ist $\alpha[x/1]^I = \alpha^I = 1$ und somit I ein Modell für $\alpha[x/1]$.
- Sei $x^I = 0$ und κ eine Klausel von $\alpha[x/1]$. Dann ist κ entweder eine Klausel von α oder $\kappa \vee x$ ist eine Klausel von α . Da I eine erfüllende Belegung für α ist, gilt $\kappa^I = 1$ im ersten Fall. Im zweiten Fall gilt $(\kappa \vee x)^I = 1$. Wegen $x^I = 0$ folgt ebenfalls $\kappa^I = 1$.

In beiden Fällen ist I ein Modell für $\alpha[x/1]$.

Eine alternative Argumentation für den Beweis von Aussage (a) in Lemma 4.29 benutzt die Beobachtung, dass wenn x nur positiv in α vorkommt, so ergibt sich $\alpha[x/0]$ durch Streichen von x aus allen Klauseln, welche das positive Literal x enthalten, während $\alpha[x/1]$ aus α durch Streichen dieser Klauseln entsteht. Die KNF-Formel $\alpha[x/1]$ besteht also genau aus denjenigen Klauseln von α , in denen x nicht vorkommt. Somit gilt:

$$\alpha[x/0] \equiv \alpha[x/1] \wedge \gamma,$$

wobei γ für diejenige KNF-Formel steht, die man erhält, indem man die Konjunktion aller Klauseln von α bildet, welche x enthalten und anschließend aus jeder dieser Klauseln x streicht. Die KNF-Formel $\alpha[x/0]$ ist daher höchstens dann erfüllbar, wenn $\alpha[x/1]$ erfüllbar ist. Aus der Splitting-Regel folgt daher, dass α genau dann erfüllbar, wenn $\alpha[x/1]$ erfüllbar ist. \square

Erweitert man Algorithmus 7 um die Pure-Literal-Regel, so erhält man für die Eingabeformel

$$\alpha = (x \vee y \vee \neg z) \wedge (y \vee z) \wedge (\neg z \vee w)$$

die erfüllende Belegung $[x = 1, y = 1, z = 0, w = 1]$ auf direktem Weg, da x und y in α nur positiv vorkommen und z in

$$\alpha[x/1, y/1] = \neg z \vee w$$

nur negativ vorkommt. Siehe Abbildung 36 auf Seite 154. Eine weitere Verbesserung des naiven Backtracking Verfahrens beruht auf folgender Subsumierungsregel. Diese befasst sich mit dem Fall, dass eine Klausel κ' eine andere Klausel κ um zusätzliche Literale erweitert.

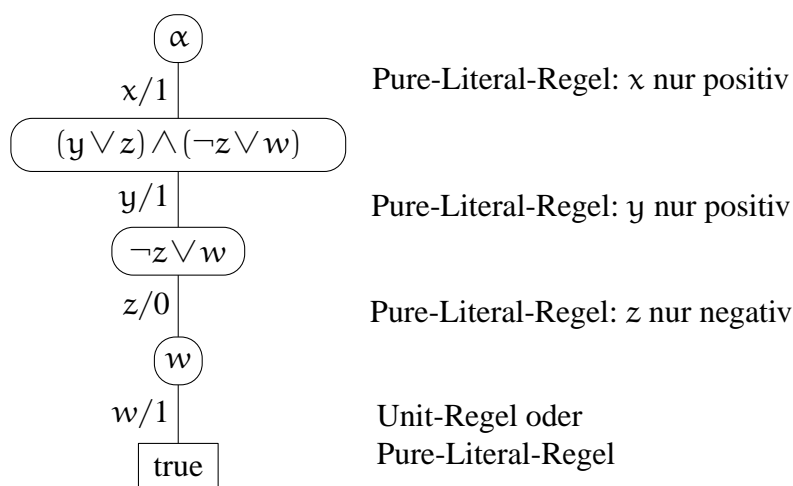


Abbildung 36: Beispiel zur Pure-Literal-Regel

Bezeichnung 4.30 (Teilklausel). Seien $\kappa = L_1 \vee \dots \vee L_k$ und $\lambda = L'_1 \vee \dots \vee L'_m$ zwei Klauseln. Wir sagen, dass κ Teilklausel von λ ist, i.Z. $\kappa \subseteq \lambda$, falls jedes Literal von κ in λ enthalten ist:

$$\kappa \subseteq \lambda \quad \text{gdw} \quad \underbrace{\{L_1, \dots, L_k\}}_{\text{Menge der Literale von } \kappa} \subseteq \underbrace{\{L'_1, \dots, L'_m\}}_{\text{Menge der Literale von } \lambda}$$

Der Begriff “Teilklausel” bezieht sich also auf die Sichtweise von Klauseln als Mengen von Literalen. Z.B. ist $\kappa = y \vee z$ Teilklausel von $\lambda = x \vee y \vee z$. ■

Ist κ Teilklausel von λ , so ist jede erfüllende Belegung von κ zugleich eine erfüllende Belegung von λ . Es gilt also $\kappa \models \lambda \equiv \kappa \vee \tau$ für eine geeignete Klausel τ . Die Folgerungsrelation \models bezogen auf Klauseln wird auch *Subsumierungsrelation* genannt. Kommen beide Klauseln κ und λ in einer KNF-Formel α vor, so ist Klausel λ redundant, da der der “Effekt” von λ durch κ subsumiert wird:

$$\text{wegen } \kappa \models \lambda \text{ gilt: } \kappa \wedge \lambda \equiv \kappa$$

Die gegebene KNF-Formel α ist also zu derjenigen KNF-Formel äquivalent, die aus α entsteht, indem die längere Klausel λ aus α gestrichen wird. Diese KNF-Formel bezeichnen wir mit

$\alpha \setminus \{\lambda\}$. Hat etwa α die Form $\kappa \wedge \lambda \wedge \beta$, so gilt:

$$\underbrace{\kappa \wedge \lambda \wedge \beta}_{\alpha} \equiv \underbrace{\kappa \wedge \beta}_{\alpha \setminus \{\lambda\}}$$

Für den Spezialfall, dass α zwei oder mehr Vorkommen der Klausel κ enthält (d.h., $\kappa = \lambda$), so ist die Schreibweise $\alpha \setminus \{\kappa\}$ so zu lesen, dass alle bis auf *eines* der Vorkommen von κ gestrichen werden. Beispielsweise gilt:

$$\underbrace{(x \vee \neg y)}_{=\kappa} \wedge \underbrace{(x \vee \neg y \vee z)}_{=\kappa'} \wedge (\neg x \vee \neg y \vee w) \equiv (x \vee \neg y) \wedge (\neg x \vee \neg y \vee w)$$

Lemma 4.31 (Subsumierungsregel). *Sei α eine KNF-Formel und κ und λ zwei Klauseln von α , so dass κ Teilklausel von κ' ist. Dann sind α und $\alpha \setminus \{\lambda\}$ äquivalent.*

Im Kontext des DP-Algorithmus ist nur die Erfüllbarkeitsäquivalenz von α und $\alpha \setminus \{\lambda\}$ relevant. Auch wenn initial eine KNF-Formel vorliegt, die keine Klauseln enthält, die Teilklausel anderer Klauseln sind, so können durch den rekursiven Abstieg des Backtracking-Algorithmus subsumierte Klauseln entstehen. Ist z.B.

$$\alpha = (\neg x \vee y \vee z) \wedge (w \vee u \vee y \vee z) \wedge \dots,$$

so enthält $\alpha[x/1]$ u.a. die beiden Klauseln $y \vee z$ und $w \vee u \vee y \vee z$.

Der Pseudo-Code des Davis-Putnam-Algorithmus mit Unit-, Pure-Literal- und Subsumierungsregel ist in Algorithmus 8 auf Seite 156 angegeben. Die Wahl zuerst die Pure-Literal-Regel, dann die Unit-Regel auf Anwendbarkeit zu prüfen, ist willkürlich. Beide Regeln haben zum Ziel die Splitting-Regel zu verzögern, um so den Rekursionsbaum klein zu halten. Da die Vereinfachung einer KNF-Formel durch die Subsumierungsregel im allgemeinen sehr aufwendig ist, kann es empfehlenswert sein, auf die Subsumierungsregel zu verzichten, und stattdessen nur die Pure-Literal-, Unit- und Splitting-Regel anzuwenden. Statt im zweiten Schritt nur den Trivialfall der KNF-Formel *true* zu behandeln, können auch weitere syntaktische Kriterien für die Erfüllbarkeit eingesetzt werden. So kann etwa die Aussage von Lemma 4.23 auf Seite 147 benutzt werden, um einen weiteren Terminalfall zu integrieren.

Beispiel 4.32 (Worst-case für den Davis-Putnam-Algorithmus). Trotz der eingesetzten Regeln, die das naive Backtracking verfeinern und oftmals das Erfüllbarkeitsproblem sehr schnell lösen können, ist die worst-case Anzahl an Rekursionsaufrufen des Davis-Putnam-Algorithmus (Algorithmus 8 auf Seite 156) exponentiell in der Anzahl an Aussagensymbolen der vorliegenden Formel. Ein Beispiel für KNF-Formeln, für welche der Davis-Putnam-Algorithmus auch unter Anwendung aller Regeln exponentielle Kosten verursacht, sind die wie folgt rekursiv definierten Formeln α_n :

$$\alpha_0 \stackrel{\text{def}}{=} (x_0 \vee y_0) \wedge (\neg x_0 \vee y_0) \wedge (x_0 \vee \neg y_0) \wedge (\neg x_0 \vee \neg y_0)$$

und

$$\alpha_n \stackrel{\text{def}}{=} \alpha_{n-1} \wedge (x_n \vee y_n) \wedge (\neg x_n \vee \neg y_n)$$

für $n = 1, 2, 3, \dots$. Da bereits α_0 unerfüllbar ist, ist jede der Formeln α_n unerfüllbar. Wir diskutieren nun die Kosten von Algorithmus 8 für die Eingabeformel α_n , wobei wir die Reihenfolge

Algorithmus 8 Davis-Putnam-Algorithmus für KNF-Formeln: rekursiver Algorithmus SAT(α)

(* Eingabe: KNF-Formel α *)

(* Rückgabe: *true*, falls α erfüllbar. Andernfalls *false*. *)

(* Terminalfälle *)

IF α enthält die leere Klausel \perp

THEN return *false* **FI**

IF $\alpha = true$

THEN return *true* **FI**

(* Pure-Literal-Regel *)

IF es gibt ein Atom x , das in α nur positiv vorkommt

THEN return SAT($\alpha[x/1]$) **FI**

IF es gibt ein Atom x , das in α nur negativ vorkommt

THEN return SAT($\alpha[x/0]$) **FI**

(* Unit-Regel *)

IF α enthält eine Einheitsklausel **THEN**

wähle eine Einheitsklausel κ

IF $\kappa = x$ ist eine positive Einheitsklausel

THEN return SAT($\alpha[x/1]$) **FI**

IF $\kappa = \neg x$ ist eine negative Einheitsklausel

THEN return SAT($\alpha[x/0]$) **FI**

FI

(* Subsumierungsregel *)

WHILE α enthält zwei Klauseln κ und λ mit $\kappa \subseteq \lambda$ **DO**

wähle solche Klauseln κ und λ und streiche λ

OD

(* Splitting-Regel *)

wähle ein Atom x , das in α vorkommt

IF SAT($\alpha[x/0]$)

THEN return *true*

ELSE return SAT($\alpha[x/1]$)

FI

$x_n, y_n, x_{n-1}, y_{n-1}, \dots, x_0, y_0$ annehmen, in der die Aussagensymbole als Kandidatinnen für die Pure-Literal-, Unit- oder Splitting-Regel herangezogen werden. Die folgenden Betrachtungen gelten aus Symmetriegründen auch für jede andere Reihenfolge, in der x_0 und y_0 am Ende stehen. Der durch den Aufruf von $\text{SAT}(\alpha_n)$ erzeugte Baum hat die in Abbildung 37 angegebene Gestalt.

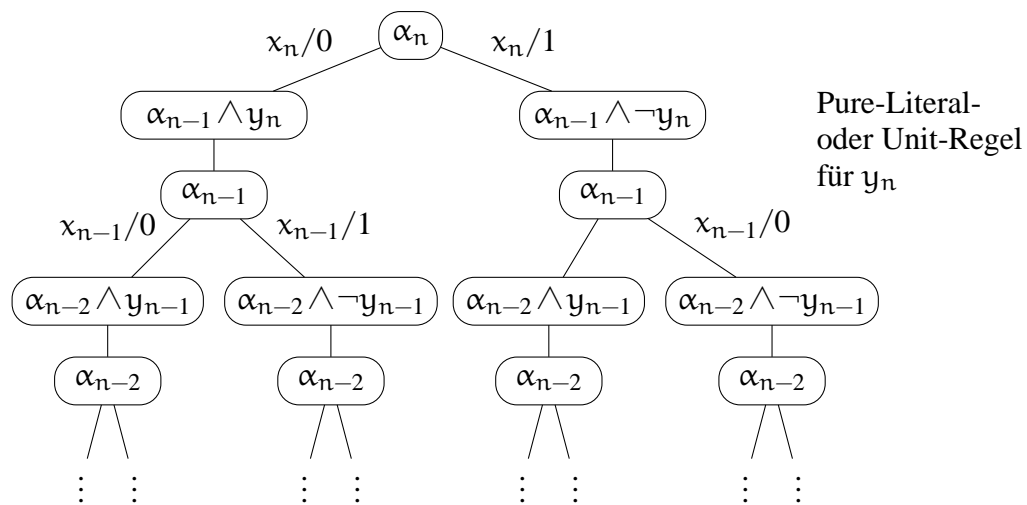


Abbildung 37: Beispiel zum worst-case für den Davis-Putnam-Algorithmus

Zunächst stellen wir fest, dass in den α_i -Ebenen die Splitting-Regel angewandt wird. Diese liefert Formeln der Gestalt $\alpha_{i-1} \wedge y_i$ und $\alpha_{i-1} \wedge \neg y_i$, für welche die Pure-Literal-Regel (oder auch Unit-Regel) greift. Die Subsumierungsregel kommt überhaupt nicht zum Zuge. Der Rekursionsbaum enthält 2^i Knoten für die Formel α_{n-i} . Daher erhält man insgesamt $\Theta(2^n)$ Rekursionsaufrufe. Die worst-case-Laufzeit ist also bereits ohne Berücksichtigung der Kosten für die Regelanwendungen exponentiell. ■

Die exponentielle worst-case Laufzeit bezieht sich auf "extreme" KNF-Formeln. Für Formeln mit zahlreichen erfüllenden Belegungen ist eine weitaus geringere Laufzeit zu erwarten. Z.B. finden für gültige Eingabeformeln mit n Atomen höchstens $n + 1$ Rekursionsaufrufe statt, da dann ein mit *true* beschriftetes Blatt erreicht wird. (Dies gilt bereits auch für das naive Backtracking.) Aber auch für unerfüllbare Eingabeformeln zeigen Erfahrungswerte, dass der Davis-Putnam-Algorithmus oftmals sehr viel bessere (als exponentielle) Laufzeit hat; insbesondere dann, wenn geeignete Heuristiken eingesetzt werden, mit denen ein Aussagensymbol für die Anwendung der Splitting-Regel ausgewählt wird, z.B.:

- Wähle ein Atom, das am häufigsten in α vorkommt.
- Wähle ein Atom x , für welches die Summe der Längen aller Klauseln, welche x oder $\neg x$ als Literal enthalten, maximal ist.
- Wähle ein Atom, das in den kürzesten Klauseln am häufigsten vorkommt.
- Wähle ein Atom x , für welches die Differenz der positiven und negativen Vorkommen von x in den kürzesten Klauseln (betragsmäßig) maximal ist.

⋮

Die ersten beiden Heuristiken haben zum Ziel, möglichst kurze Formeln $\alpha[x/\dots]$ zu generieren. Die Motivation für die dritte Heuristik besteht darin, möglichst schnell die Unit-Regel anwenden zu können. Entsprechend zielt die vierte Heuristik auf die Pure-Literal-Regel ab.

Für eine Implementierung des Davis-Putnam-Algorithmus empfiehlt sich die Verwendung von geeigneten Datenstrukturen, welche die Generierung der Formeln $\alpha[x/\dots]$ unterstützen (z.B. Listen mit Verweisen auf alle Klauseln, die x bzw. $\neg x$ als Literal enthalten) sowie Datenstrukturen, die alle Atome, auf welche die Pure-Literal- oder Unit-Regel anwendbar ist, geeignet verwalten.

Abschliessend bemerken wir, dass der Platzbedarf des Davis-Putnam-Algorithmus nur linear in der Formellänge ist, da der zusätzliche Platz durch den Rekursionsstack dominiert wird und die Rekursionstiefe durch $\mathcal{O}(n)$ beschränkt ist. Dasselbe gilt auch für das naive Backtracking.

Die Annahme, dass der Davis-Putnam Algorithmus mit einer KNF-Formel gestartet wird, ist zwar keine Einschränkung, da jede aussagenlogische Formel in eine äquivalente KNF-Formel transformiert werden kann, jedoch kann die konstruierte KNF-Formel exponentiell länger als die ursprüngliche Formel sein. In Satz 4.14 auf Seite 136 hatten wir nachgewiesen, dass dieses Phänomen unabhängig von dem gewählten Verfahren zur Konstruktion einer äquivalenten KNF-Formel ist.

Von SAT zu 3SAT

Das etwaige exponentielle Blowup durch den Übergang von einer aussagenlogischen Formel α zu einer äquivalenten KNF-Formel kann im Kontext des Erfüllbarkeitsproblems verhindert werden, indem auf die Erstellung einer äquivalenten KNF-Formel verzichtet und stattdessen eine *erfüllbarkeitsäquivalente* KNF-Formel derselben asymptotischen Länge konstruiert wird. Tatsächlich ist eine solche algorithmische Transformation sogar dann möglich, wenn höchstens drei Literale pro Klausel erlaubt sind. Damit kann SAT auf 3SAT (das Erfüllbarkeitsproblem für KNF-Formeln mit höchstens drei Literalen pro Klausel) zurückgeführt werden, ohne ein exponentielles Blowup in Kauf nehmen zu müssen.

Definition 4.33 (Erfüllbarkeitsäquivalenz). *Zwei aussagenlogische Formeln α_1, α_2 heißen erfüllbarkeitsäquivalent, i.Z.*

$$\alpha_1 \equiv^{\text{SAT}} \alpha_2,$$

wenn sie entweder beide erfüllbar oder beide unerfüllbar sind. ■

Erfüllbarkeitsäquivalente Formeln können völlig verschiedene erfüllende Belegungen haben und sind im allgemeinen nicht semantisch äquivalent. Beispielsweise sind die atomaren Formeln x , wobei $x \in AP$, paarweise *nicht* semantisch äquivalent, jedoch sind sie erfüllbarkeitsäquivalent. Die Literale x und $\neg x$ bilden ein Beispiel für erfüllbarkeitsäquivalente Formeln, die *keine* gemeinsame erfüllende Belegung haben. Man beachte, dass es lediglich zwei Äquivalenzklassen hinsichtlich Erfüllbarkeitsäquivalenz gibt: die Klasse aller erfüllbaren Formeln und die Klasse aller unerfüllbaren Formeln.

Bezeichnung 4.34 (3KNF). *Eine aussagenlogische Formel α ist in 3KNF, falls sie in KNF ist und jede Klausel aus höchstens drei Literalen besteht. Analoge Bedeutung hat die Bezeichnung 2KNF oder k -KNF für ganze Zahlen $k \geq 1$.* ■

Z.B. ist $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3 \vee \neg x_3)$ eine 3KNF-Formel. Durch Wiederholen von Literalen kann man zu jeder 3KNF-Formel eine äquivalente 3KNF-Formel mit genau drei Literalen pro Klausel erstellen. Z.B. ist $(x_1 \vee x_1 \vee x_1) \wedge (\neg x_2 \vee \neg x_2 \vee x_3)$ eine zu $x_1 \wedge (\neg x_2 \vee x_3)$ äquivalente 3KNF-Formel mit genau drei Literalen pro Klausel. Die Existenz von äquivalenten 3KNF-Formeln mit genau drei Literalen pro Klausel gilt auch für 3KNF-Formeln mit der leeren Klausel. Hier kann die leere Klausel durch $(x \vee x \vee x) \wedge (\neg x \vee \neg x \vee \neg x)$ für ein beliebiges Atom x ersetzt werden.

Ist α eine k -KNF-Formel, so entsteht durch den Übergang von α zu $\alpha[x/1]$ oder $\alpha[x/0]$ wieder eine KNF-Formel mit höchstens k Literalen pro Klausel. Dies ist klar, da $\alpha[x/1]$ und $\alpha[x/0]$ durch Streichen von Klauseln und Streichen von Literalen aus Klauseln aus α hervorgehen. Mit α sind also auch $\alpha[x/1]$ oder $\alpha[x/0]$ k -KNF-Formeln.

Satz 4.35 (Linearzeit-Transformation: Formel \rightsquigarrow erfüllbarkeitsäquivalente 3KNF). *Zu jeder aussagenlogischen Formel α kann eine erfüllbarkeitsäquivalente 3KNF-Formel (d.h. eine KNF-Formel mit höchstens 3 Literalen pro Klausel) in linearer Zeit $\mathcal{O}(|\alpha|)$ konstruiert werden.*

Da in linearer Zeit nur polynomiell viele Daten verarbeitet werden können, ist auch die Länge der konstruierten 3KNF-Formel durch $\mathcal{O}(|\alpha|)$ beschränkt.

Beweis. Die wesentliche Idee der Transformation ist, die Formel in PNF zu überführen und dann den inneren Knoten des Syntaxbaums 3KNF-Formeln konstanter Länge zuzuordnen. Im Folgenden sei α die Eingabeformel mit den Atomen x_1, \dots, x_n . Der erste Schritt der Transformation führt α in eine äquivalente Formel α_{PNF} in PNF derselben asymptotischen Länge über.

- Durch Ersetzen jeder Teilformel der Form $\beta \wedge \text{true}$ oder $\text{true} \wedge \beta$ durch β und jeder Teilformel $\beta \vee \text{true}$ oder $\text{true} \vee \beta$ durch true , sowie analoge Transformationen für jedes Vorkommen von *false*, entsteht eine zu α und α_{PNF} äquivalente Formel α' , so dass entweder $\alpha' = \text{true}$ oder $\alpha' = \text{false}$ oder α' ist eine PNF-Formel, welche die Konstanten *true* und *false* nicht enthält.
- Falls $\alpha' = \text{true}$, so ist α' eine 3KNF-Formel mit 0 Klauseln. Falls $\alpha' = \text{false}$, so ist α' eine 3KNF-Formel bestehend aus der leeren Klauseln. In beiden Fällen ist die Transformation abgeschlossen.

Im Folgenden nehmen wir an, dass $\alpha_{\text{PNF}} = \alpha'$ eine PNF-Formel der Länge $|\alpha_{\text{PNF}}| = \mathcal{O}(|\alpha|)$ ist, welche zu α äquivalent ist und in welcher die Konstanten *true* oder *false* nicht vorkommen. Wir betrachten nun den *Syntaxbaum* T von α_{PNF} , wobei wir die in α vorkommenden Literale wie atomare Formeln behandeln. Die inneren Knoten des Syntaxbaums für α_{PNF} stehen für die Teilformeln von α_{PNF} , deren Top-Level-Operator eine Konjunktion oder Disjunktion ist. Die inneren Knoten werden entsprechend mit \wedge oder \vee beschriftet. Die Blätter des Syntaxbaums stehen für die Vorkommen der Literale in α_{PNF} . Wir schreiben β_v für die durch Knoten v des Syntaxbaums dargestellte Teilformel von α_{PNF} . Ist also v ein innerer mit *op* beschrifteter Knoten von T , wobei $op \in \{\wedge, \vee\}$, und sind w und u die beiden Söhne von v , so ist

$$\beta_v = \beta_w \text{ op } \beta_u.$$

Jedes Blatt v identifizieren wir mit dem durch v repräsentierten Literal, also

$$\beta_v \in \{x_1, \dots, x_n, \neg x_1, \dots, \neg x_n\}, \text{ falls } v \text{ ein Blatt ist.}$$

Für die inneren Knoten verwenden wir beliebige Knotennamen und setzen voraus, dass diese paarweise verschieden sind und nicht in $\{x_1, \dots, x_n, \neg x_1, \dots, \neg x_n\}$ vorkommen. Im Folgenden bezeichne v_0 den Wurzelknoten von T und $\text{In}(T)$ die Menge (der Knotennamen) aller inneren Knoten von T .

Die nun konstruierte 3KNF-Formel $\alpha_{3\text{KNF}}$ verwendet die in α vorkommenden Atome x_1, \dots, x_n und zusätzlich die Namen der inneren Knoten. Die zugrundeliegende Menge an Aussagensymbolen ist also

$$AP \stackrel{\text{def}}{=} \{x_1, \dots, x_n\} \cup \text{In}(T).$$

Die 3KNF-Formel $\alpha_{3\text{KNF}}$ entsteht nun durch die Konjunktion der Einheitsklausel v_0 und 3KNF-Formeln σ_v für jeden inneren Knoten v des Syntaxbaums. Die 3KNF-Formeln σ_v ergeben sich durch geeignete Umformungen aus Formeln der Gestalt

$$\begin{aligned} v &\leftrightarrow (w \vee u), && \text{falls } v \text{ ein innerer Knoten mit den Söhnen } w \text{ und } u \\ &&& \text{und mit } \vee \text{ beschriftet ist,} \\ v &\leftrightarrow (w \wedge u), && \text{falls } v \text{ ein innerer Knoten mit den Söhnen } w \text{ und } u \\ &&& \text{und mit } \wedge \text{ beschriftet ist.} \end{aligned}$$

Zur Erinnerung: die Söhne w und u von v sind entweder innere Knoten (also Atome in AP) oder Blätter, also Literale in $\{x_1, \dots, x_n, \neg x_1, \dots, \neg x_n\}$.

Die Formeln $v \leftrightarrow (w \text{ op } u)$ werden nun in 3KNF mit jeweils genau drei Klauseln gebracht. Wie zuvor ist $\text{op} \in \{\wedge, \vee\}$ die Beschriftung von Knoten v . Ist v ein mit \wedge beschrifteter Knoten und sind w und u die beiden Söhne von v , so definieren wir:

$$\sigma_v \stackrel{\text{def}}{=} (v \vee \neg w \vee \neg u) \wedge (\neg v \vee w) \wedge (\neg v \vee u)$$

Man überzeugt sich nun leicht davon, dass die 3KNF-Formel σ_v zur Formel $v \leftrightarrow (w \wedge u)$ äquivalent ist. Es gilt nämlich:

$$\begin{aligned} \sigma_v[v/0] &= \neg w \vee \neg u \equiv \neg(w \wedge u) \\ \sigma_v[v/1] &= w \wedge u \end{aligned}$$

und somit $\sigma_v \equiv v \leftrightarrow (w \wedge u)$.

Jedem AND-Knoten v im Syntaxbaum kann also eine 3KNF-Formel σ_v mit drei Klauseln zugeordnet werden. Analoges gilt für die OR-Knoten. Ist v ein mit \vee beschrifteter Knoten mit den Söhnen w und u , so definieren wir:

$$\sigma_v \stackrel{\text{def}}{=} (v \vee \neg w) \wedge (v \vee \neg u) \wedge (\neg v \vee w \vee u)$$

Offenbar sind auch in diesem Fall $v \leftrightarrow (w \vee u)$ und σ_v äquivalent, da

$$\begin{aligned} \sigma_v[v/0] &= \neg w \wedge \neg u \equiv \neg(w \vee u) \\ \sigma_v[v/1] &= w \vee u \end{aligned}$$

Die resultierende zu α und α_{PNF} äquivalente 3KNF-Formel ist nun wie folgt definiert:

$$\alpha_{3\text{KNF}} \stackrel{\text{def}}{=} v_0 \wedge \bigwedge_{v \in \text{In}(T)} \sigma_v,$$

wobei – wie zuvor – v_0 für den Wurzelknoten des Syntaxbaums T steht und $\text{In}(T)$ die Menge aller inneren Knoten von T bezeichnet.