Institut für theoretische Informatik

Leibniz Universität Hannover

Bachelorarbeit

Algorithmen für Horn-Formeln

von Thorben Lemke
3216140

19. Juni 2020

Erstprüfer: Prof. Dr. Heribert Vollmer
Zweitprüfer: Dr. Maurice Chandoo

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst habe und keine anderen Hilfsmittel und Quellen als angegeben verwendet habe. Die Arbeit hat in dieser oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Thorben Lemke, Hannover, den 19. Juni 2020

Inhaltsverzeichnis

Einleitung

Diese Arbeit beschäftigt sich mit S. 217-266 des Kapitels „Horn-Logik" aus dem Werk „Aussagenlogik: Deduktion und Algorithmen" von Hans Kleine Büning und Theodor Lettmann (Stuttgart, 1994). Ausgelassen wurde Unterkapitel 5.5 „Reduktion von Horn-Formeln" (S. 257-263). „Aussagenlogik: Deduktion und Algorithmen" ist ein Standardwerk aus der Reihe „Leitfäden und Monographien der Informatik", welches den neuesten Stand der Wissenschaft wiedergibt.

Dabei soll der Fokus dieser Arbeit darauf liegen, die bereits gewonnenen Erkenntnisse konzise zusammenzufassen, um fachlich Interessierten und Studierenden einen leichteren Zugang zu ermöglichen. Dafür sind viele der im Original mit mathematischen Symbolen abgekürzten Definitionen und Sätze in Fließtext übersetzt worden, außer wenn es sich aufgrund von möglichen Unschärfen bei der Präzision nicht angeboten hat. Außerdem wurden einige Beweise und Passagen verkürzt, entfernt oder angepasst und Beispiele hinzugefügt. Auch wurden an einigen Stellen Grafiken zusammengefasst oder neu ergänzt, um Sachverhalte besser darzustellen und Zusammenhänge zwischen einzelnen Erkenntnissen herzustellen.

Zuerst werden dabei grundlegende Aussagen über Horn-Formeln erbracht. In Kapitel 2 wird das Verfahren der Umbenennung vorgestellt werden, um z.B. Formeln in konjunktiver Normalform zu Horn-Formeln umzubenennen, wofür ein linearer Algorithmus vorgestellt wird. Anschließend werden Resolutionsverfahren vorgestellt, um zu einem effizienten Verfahren für das Suchen einer erfüllenden Belegung für Horn-Formeln zu gelangen. In Kapitel 3 wird das Problem der Unique-Satisfiability für Horn-Formeln vorgestellt und ein nahezu linearer Algorithmus konstruiert. Schließlich wird in Kapitel 5 mit Äquivalenzproblemen für Horn-Formeln abgeschlossen, für die ebenfalls ein Algorithmus vorgestellt wird.

Anwendungsgebiete der Horn-Logik finden sich beispielsweise in der Graphentheorie, wie auch im Vorgehen des Algorithmus aus Kapitel 4 ablesbar ist. In Kapitel 3 wird zudem herausgestellt, dass sich für Horn-Formeln erfüllende Belegungen effizienter finden lassen, als dies für Formeln in konjunktiver Normalform der Fall wäre.

1 Grundlagen

Nachfolgendes Kapitel richtet sich inhaltlich nach dem Kapitel „Horn-Logik" und „Grundlagen", außer es ist entsprechend markiert ([KL94, vgl. S. 217-220]).

Zuerst seien einige grundlegenden Definitionen zu Klassen aussagenlogischer Formeln sowie deren Klauseln genannt:

Definition 1.1 (Konjunktive Normalform) *Eine aussagenlogische Formel α in konjunktiver Normalform besteht aus Klauseln α_i für $1 \leq i \leq m$ in selbiger Form. Dabei weisen Klauseln die folgende Struktur für Literale X auf: $(X_1 \vee ... \vee X_n)$. Eine Formel setzt sich nun wiederum aus den einzelnen Klauseln folgendermaßen zusammen: $\alpha = (\alpha_1 \wedge ... \wedge \alpha_m)$. Nachfolgend wird die Klasse der Formeln in konjunktiver Normalform mit „KNF" abgekürzt. Außerdem können Formeln aus KNF der Einfachheit halber in Mengenschreibweise dargestellt werden, d.h. eine Formel $\alpha = ((X_1 \vee ... \vee X_n) \wedge ... \wedge (X_1 \vee ... \vee X_n))$ kann auch als $\alpha = \{\{X_1, ..., X_n\}, ..., \{X_1, ..., X_n\}\}$ beschrieben werden. ([KL94, vgl. S. 24])*

Definition 1.2 (k-Konjunktive Normalform) *Eine Einschränkung für Formeln aus KNF ist die Klasse der Formeln in k-konjunktiver Normalform: Hier bestehen die einzelnen Klauseln aus maximal k Literalen für ein festgelegtes $k \in \mathbb{N}_{>0}$. Die Klasse der Formeln in k-konjunktiver Normalform wird mit „k-KNF" abgekürzt. ([KL94, vgl. S. 24])*

Definition 1.3 (Unit-Klausel) *Eine Unit-Klausel k ist eine Klausel, die lediglich aus einem Literal besteht. Abgekürzt kann eine solche Klausel bzw. ihr Atom auch als Unit bezeichnet werden. ([KL94, vgl. S. 24])*

Dabei sind Horn-Formeln eine Teilmenge der aussagenlogischen Formeln aus KNF, die nach ihrem Begründer Alfred Horn benannt wurden:

Definition 1.4 (Horn-Formel) *Eine aussagenlogische Horn-Formel α besteht aus Horn-Klauseln α_i für $1 \leq i \leq m$. Dabei weisen Klauseln die selbe Struktur auf, wie Klauseln aus KNF, wobei sie pro Klausel maximal ein positives Literal enthalten. Die Zusammensetzung einer Horn-Formel entspricht der einer Formel aus KNF, wobei die einzelnen Klauseln natürlich Horn-Klauseln sein müssen. Nachfolgend wird die Klasse der Horn-Formeln mit „HORN" abgekürzt.*

Definition 1.5 (Definite Horn-Formel) *Definite Horn-Formeln entsprechen in ihrem Aufbau den Formeln aus HORN, wobei der Unterschied ist, dass sie pro Klausel genau ein positives Literal enthalten. Nachfolgend wird die Klasse der definiten Horn-Formeln mit „DHORN" abgekürzt.*

Definition 1.6 (Implikationsschreibweise) *Klauseln aus HORN und DHORN können auch in Implikationsschreibweise angegeben werden. Für eine Klausel* $\alpha_i = (X_1 \vee \neg X_2 \vee ... \vee \neg X_n)$ *aus DHORN wäre dies* $\alpha_i = (X_2 \wedge ... \wedge X_n \rightarrow X_1)$. *Klauseln aus HORN, welche kein positives Literal enthalten (hier* X_1*) haben die Form* $\alpha_i = (X_2 \wedge ... \wedge X_n \rightarrow 0)$. *Positive Unit-Klauseln hingegen können als* $\alpha_i = (1 \rightarrow X_1)$ *ausgedrückt werden.*

Zuerst eine wichtige Aussage zur Erfüllbarkeit für Formeln aus HORN sowie DHORN:

Lemma 1.7

1. *Alle Formeln aus DHORN sind erfüllbar.*
2. *Formeln aus HORN sind immer erfüllbar, solange sie keine positiven Unit-Klauseln beinhalten.*

Beweis.

1. Trivial, wenn man eine Belegung wählt, die allen Atomen den Wahrheistwert *wahr* zuordnet.
2. Betrachtet man Formeln aus HORN, die der Definition aus dem Lemma entsprechen, kommt in allen Klauseln mindestens ein negatives Literal vor. Solche Formeln werden dann von einer Belegung erfüllt, die allen Atomen den Wahrheitswert $falsch$ zuordnet.

□

Widerspruchsvolle Formeln aus HORN erhalten nun die Eigenschaft, dass es in ihren nur negativen Klauseln genau eine widerspruchsvolle Klausel geben muss, sodass die Klauseln aus DHORN mit dieser einen Klausel zusammen bereits widersprüchlich sind:

Lemma 1.8 *Sei eine widerspruchsvolle Formel* $\alpha = \beta \cup \delta$ *aus HORN gegeben, wobei* β *aus DHORN sei und* δ *nur aus negativen Klauseln bestehe. Es gilt:* α *ist widerspruchsvoll genau dann, wenn eine Klausel* δ_0 *aus* δ *existiert, für welche* $\{\delta_0\} \cup \beta$ *widerspruchsvoll ist.*

Beweis. Man nehme an, dass α widerspruchsvoll sei und β konjugiert mit δ_0 für alle δ_0 aus δ erfüllbar sei. Seien dann $A_1, ..., A_m$ alle aus β folgerbaren Atome, so sind die Atome von δ_0 keine Teilmenge von $A_1, ..., A_m$. Jetzt setze eine Belegung $\mathfrak{I}$ alle Atome $A_1, ..., A_m$ auf *wahr* und die übrigen Atome auf $falsch$, womit auch alle δ_0 den Wahrheitswert $falsch$ annehmen würden.

Andererseits würde unter dieser Belegung eine unerfüllbare Klausel aus β nur Prämissen aus $A_1, ..., A_m$ enthalten. Somit wäre die Konklusion – im Widerspruch zur Unerfüllbarkeit der Klausel – ebenfalls ein folgerbares Atom. Daraus folgt $\mathfrak{I}(\beta) = 1$ und somit auch im Widerspruch zu der Grundvoraussetzung $\mathfrak{I}(\alpha) = 1$ □

Für den Beweis des übernächsten Lemmas braucht man zusätzlich noch folgendes Lemma (siehe Definition 3.1 für die Erklärung der Resolution sowie des Resolutionsbaumes, [KL94, vgl. S. 147] für das Lemma sowie seinen Beweis und [KL94, vgl. S. 142f.] für das Vertauschungslemma und den dazugehörigen Beweis):

Lemma 1.9 *Sei α aus KNF und k eine Klausel, dann gilt: k folgt semantisch aus α ($\alpha \models k$) genau dann, wenn eine Teilklausel k^* von k existiert mit: k^* lässt sich mittels der Resolution aus α herleiten ($\alpha \vdash_{RES} k^*$)*

Beweis. Es ist bekannt, dass die Resolution korrekt ist, daher genügt es für den Beweis, die Folgerbarkeit des zweiten Teils der Aussage aus dem ersten zu zeigen.

Für eine Formel α und eine Klausel $k = (X_1 \vee ... \vee X_i)$ gelte $\alpha \models k$ (erster Teil der Aussage). Mittels der Resolution (aufgrund ihrer Widerlegungsvollständigkeit) lässt sich dann aus $\alpha \cup \{(\neg X_1), ..., (\neg X_i)\}$ die leere Klausel herleiten, da die Formel offenbar widersprüchlich ist.

Durch das Vertauschen von Resolutionsschritten kann man nun in dem Resolutionsbaum die Unit-Resolutionsschritte ans Ende verschieben, insbesondere solche, die $\{(\neg X_1), ..., (\neg X_i)\}$ als Elternklauseln enthalten.

Dies ist möglich durch das sogenannte „Vertauschungslemma". Dabei werden drei Klauseln a, b und c mit denen in ihnen in folgender Form enthaltenen Literale X_1 und X_2 betrachtet:
$\{X_2\} \in a$
$\{X_1, \neg X_2\} \in b$
$\{\neg X_1\} \in c$

Für eine Resolutionsherleitung, welche als erstes b und c über X_1 resolviert und die so entstehende Resolvente schließlich mit a über X_2 resolviert, kann die Herleitung nun folgendermaßen vertauscht werden:

Zuerst werden nur Resolutionen über X_2 vorgenommen, und anschließend die Resolution über X_1 durchgeführt. Die daraus resultierende Resolvente ist dann entweder genau die gleiche Resolvente, wie die letzte der ursprünglichen Herleitung, oder eine Teilklausel derer.

Aufgrund der Teilklauseleigenschaft des Vertauschungslemmas kann der so veränderte Resolutionsbaum der ursprünglich betrachten Formel kürzer ausfallen, jedoch bleibt die Herleitung der leeren Klausel erhalten.

Somit kann man insgesamt davon ausgehen, dass man in dieser Herleitung eine Klausel k^* erhält, welche jetzt nur noch mittels Resolutionsschritten über $\{\neg X_1, ..., \neg X_i\}$ in die leere Klausel überführt werden kann. Wichtig ist dabei, dass in sämtlichen vorigen Resolutionsschritten zur Herleitung von k^* eben $\{\neg X_1, ..., \neg X_i\}$ nicht vorkamen.

Daraus folgt, dass $k^* = (X_{l_1} \vee ... \vee X_{l_r})$ sein muss, mit $1 \leq l_j \leq n$. k^* kann dann auch die leere Klausel sein, aber insbesondere ist k^* eine Teilklausel von k und $\alpha \vdash_{RES} k^*$ gilt. □

Lemma 1.10 *Die Klassen HORN, DHORN und 2-KNF sind abgeschlossen gegen Deduktion. Also: Wenn eine nicht-tautologische Klausel β semantisch aus α folgt (α aus HORN, DHORN oder 2-KNF), dann gibt es eine Teilklausel δ aus β, welche semantisch aus α folgt und eine Klausel aus HORN, DHORN oder 2-KNF ist (je nachdem, was auf α zutrifft).*

Beweis. Durch das vorige Lemma folgt sofort die Behauptung. Erklärung: Für jede Klausel β, die semantisch aus α folgerbar ist, gibt es eine Teilklausel δ aus β, welche mittels

der Resolution aus α erzeugt werden kann. Außerdem liefert die Resolution, angewandt auf Klauseln aus HORN, DHORN oder 2-KNF, wieder eine Klausel aus selbiger Klasse (für HORN und 2-KNF: die leere Klausel muss ebenfalls als aus der entsprechenden Klasse aufgefasst werden). □

Das letzte Lemma dieses Kapitels stellt einen wichtigen Zusammenhang für den Durchschnitt zweier erfüllenden Belegungen für Formeln aus HORN her:

Lemma 1.11 *Für eine Formel α aus HORN sind zwei erfüllende Belegungen $\mathfrak{I}_1$ und $\mathfrak{I}_2$ im Durchschnitt wieder eine erfüllende Belegung.*

Beweis. Sei α eine erfüllbare Formel aus HORN mit den Atomen $A_1, ..., A_n$ und zwei erfüllende Belegungen $\mathfrak{I}_1$ und $\mathfrak{I}_2$ gegeben. Es folgt eine Auflistung aller möglichen Typen für eine Klausel γ aus α:

1. $\gamma = (X)$, also eine Unit-Klausel: $1 = \mathfrak{I}_1(X) = \mathfrak{I}_2(X) = (\mathfrak{I}_1 \cap \mathfrak{I}_2)(X)$.

2. $\gamma = (A_0 \vee \neg A_1 \vee ... \vee \neg A_k)$ aus DHORN: $\mathfrak{I}_1(A_0) = \mathfrak{I}_2(A_0) = 1$ und somit folgt natürlich $(\mathfrak{I}_1 \cap \mathfrak{I}_2)(\gamma) = 1$.

   Sollte aber $\mathfrak{I}_1(A_0) = \mathfrak{I}_2(A_0) = 0$ sein, gibt es ein r und s mit $\mathfrak{I}_1(A_r) = \mathfrak{I}_2(A_s) = 0$. Daraus folgt, dass $(\mathfrak{I}_1 \cap \mathfrak{I}_2)(A_r) = (\mathfrak{I}_1 \cap \mathfrak{I}_2)(A_s) = 0$ und somit auch $(\mathfrak{I}_1 \cap \mathfrak{I}_2)(\gamma) = 1$.

   Der letzte Fall für γ wäre, dass $\mathfrak{I}_1(A_0) = 1$ und $\mathfrak{I}_2(A_0) = 0$ ist, wobei dann ein r existieren muss, mit $\mathfrak{I}_2(A_r) = 0$. Da nun $(\mathfrak{I}_1 \cap \mathfrak{I}_2)(A_r) = 0$ gilt, stimmt wieder $(\mathfrak{I}_1 \cap \mathfrak{I}_2)(\gamma) = 1$.

3. Negative Klausel $\gamma = (\neg A_0 \vee ... \vee \neg A_k)$: Es existiert ein r und s mit $\mathfrak{I}_1(\neg A_r) = \mathfrak{I}_2(\neg A_s) = 1$. Es folgt direkt, dass $(\mathfrak{I}_1 \cap \mathfrak{I}_2)(A_r) = (\mathfrak{I}_1 \cap \mathfrak{I}_2)(A_s) = 0$ und somit auch $(\mathfrak{I}_1 \cap \mathfrak{I}_2)(\gamma) = 1$.

Da so alle möglichen Klauselformen abgedeckt sind und erfüllt werden, wird auch α durch den Durchschnitt von $\mathfrak{I}_1$ und $\mathfrak{I}_2$ erfüllt und somit ist die Aussage bewiesen. □

2 Umbenennung

Nachfolgendes Kapitel richtet sich inhaltlich nach dem Kapitel „Umbenennung", außer es ist entsprechend markiert ([KL94, vgl. S. 221-231]).

Das Verfahren der Umbenennung kann genutzt werden, um z.B. Formeln aus KNF in Formeln aus HORN zu überführen. Die Umbenennung sei wie folgt definiert:

Definition 2.1 (Umbenennung) *Für eine Formel α aus KNF heißt die Funktion $f : literals(\alpha) \to literals(\alpha)$ Umbenennung genau dann, wenn für alle Literale X aus α ein $f(X)$ existiert, das jeweils X entweder positiv oder negativ abbildet und die Umbenennungsfunktion $f(\neg X) \approx \neg f(X)$ gilt. Alle Umbenennungen von α werden mit $U(\alpha)$ bezeichnet.*

Ein Beispiel einer Umbenennung f für eine Formel α aus KNF nach HORN wäre:
$\alpha = \{\{X_1, X_2, \neg X_3\}, \{X_1, X_3\}, \{\neg X_2, \neg X_3, X_4, X_5\}\}$
$f(X_i) = \begin{cases} \neg X_i & \text{falls } i = 1 \text{ oder } i = 4 \\ X_i & \text{sonst.} \end{cases}$ für $1 \leq i \leq 5$.
$f(\alpha) = \{\{\neg X_1, X_2, \neg X_3\}, \{\neg X_1, X_3\}, \{\neg X_2, \neg X_3, \neg X_4, X_5\}\}$

Die so resultierende Formel $f(\alpha)$ liegt nun in HORN, und kann somit z.B. von dem im Kapitel 3 vorgestellten Algorithmus Unit-Res (bzw. Sat-HORN) auf Erfüllbarkeit geprüft werden. Für die Ursprungsformel aus KNF hingegen könnte mit diesem nur überprüft werden, ob die Formel widerspruchsvoll ist.

Natürlich existiert aber nicht zu jeder Formel aus KNF eine Umbenennung in eine Formel aus HORN, was sich schon an dem Größenverhältnis der beiden Klassen zueinander ablesen lässt:

Betrachtet man beispielsweise Formeln aus 3-KNF mit n Klauseln, so gibt es ingesamt $A(3n) \cdot 2^{3n}$ Formeln ($A(3n)$ sind die möglichen Verteilungen von bis zu $3n$ Atomen auf $3n$ Stellen, 2^{3n} beschreibt die mögliche Verteilung von Negationen auf diese $3n$ Stellen). Hingegen belaufen sich die Formeln aus HORN mit n Klauseln der Länge 3 auf $A(3n) \cdot 2^{2n}$ (2^{2n}, da die Anzahl der möglichen Negationen auf n Klauseln bei 4^n liegt). Im Verhältnis ergibt sich insgesamt $1 : 2^n$. [KL94, vgl. S. 57]

Deshalb ist es wichtig in Linearzeit entscheiden zu können, ob zu einer Formel aus KNF eine Umbenennung nach HORN möglich ist (der vorgestellte Algorithmus beruht laut [KL94, vgl. S.221] auf den Arbeiten von [Asp80] und [Lew78]):

Satz 2.2 *Die Klasse der Umbenennungen die für eine Formel α aus KNF eine Formel $f(\alpha)$ aus HORN erzeugen ist in der Länge n der Ausgangsformel in $\mathcal{O}(n)$ entscheidbar.*

Beweis. Der Beweis der Aussage beruht auf einer linearen Tranformation T, die zu einer Formel α aus KNF eine Formel $T(\alpha)$ aus 2-KNF zuordnet. Die Bedingung dabei ist,

dass $T(\alpha)$ erfüllbar sei genau dann, wenn eine Umbenennung f aus $U(\alpha)$ mit $f(\alpha)$ aus HORN existiert. Somit würde jede Belegung, die $T(\alpha)$ erfüllt, eine Umbenennung liefern. Außerdem können Unit-Klauseln vernachlässigt werden, da sie – unabhängig von einer etwaigen Umbenennung – Klauseln aus HORN bleiben.

Man betrachte für den Beweis also eine Formel $\alpha = \{\alpha_1, ..., \alpha_m\}$ ohne Unit-Klauseln mit $\alpha_i = (X_{i,1} \vee ... \vee X_{i,k_i})$ für $1 \leq i \leq m$. Es sei nun ein S für α mit $S(\alpha) := \{(X_{i,j} \vee X_{i,p} | 1 \leq i \leq m, 1 \leq j \neq p \leq k_i)\}$ definiert. Zuerst der Beweis, dass dieses S erfüllbar ist, wenn eine Umbenennung $f(\alpha)$ aus HORN existiert:

Sei also eine Umbenennung $f(\alpha)$ aus HORN gegeben. Man führe eine partielle erfüllende Belegung $\mathfrak{I}$ ein, mit $\mathfrak{I}(X_{i,j}) = 1$ für negative $f(X_{i,j})$ und $\mathfrak{I}(X_{i,j}) = 0$ sonst. Damit gilt, dass für je zwei Literale $f(X_{i,j})$ und $f(X_{i,p})$ mit $1 \leq j \leq p \leq k_i$ mindestens eines negativ ist und insgesamt maximal ein positives Literal $f(X_{i,t})$ mit $1 \leq t \leq k_i$ existiert. Aus dieser Eigenschaft folgt auch, dass die Belegung $\mathfrak{I}$ für ein paar $(X_{i,j} \vee X_{i,p})$ mit $1 \leq j \leq p \leq k_i$ den Wahrheitswert 1 annimmt. Unter dieser Belegung könnte ein solches Paar nur dann den Wahrheitswert 0 annehmen, wenn beide Literale positiv wären. Das wäre allerdings ein Widerspruch zu der Vorraussetzung $f(\alpha_i) \in HORN$.

Jetzt der umgekehrte Fall, also wie aus einer erfüllenden Belegung von S eine Umbenennung $f(\alpha)$ aus HORN erzeugt wird:

Sei $S(\alpha)$ also erfüllbar. Dann folgt für eine erfüllende Belegung $\mathfrak{I}$ und eine Klausel α_i: $\mathfrak{I}(\{(X_{i,j} \vee X_{i,p} | 1 \leq j \neq p \leq k_i)\}) = 1$. Somit gilt wieder $\mathfrak{I}(X_{i,t}) = 0$ für maximal ein t $(1 \leq t \leq k_i)$.

Jetzt definiert man – basierend auf dieser Eigenschaft der erfüllenden Belegungen – für jedes Atom A aus α die Funktion f mit $f(A) := \neg A$ für $\mathfrak{I}(A) = 1$ und $f(A) := A$ sonst.

Diese Umbenennung impliziert, dass die einzelnen Klauseln aus α umbenannt in HORN liegen, denn sonst hätte man ein ähnliches Problem wie vorhin: zwei positive Literale $f(X_{i,j})$ und $f(X_{i,p})$ mit $1 \leq j \leq p \leq k_i$ aus $f(\alpha_i)$ würden bedeuten, dass die Klausel $(X_{i,j} \vee X_{i,p})$ aus $S(\alpha)$ unter der Belegung $\mathfrak{I}$ nicht erfüllbar wäre.

Jetzt hat man zwar ein $S(\alpha)$ aus 2-KNF erzeugt, jedoch liegt der Aufwand der Konstruktion momentan quadratisch in der Länge der Eingabe, da jedes mögliche Literalpaar gebildet werden müsste. Wenn man nun allerdings eine Hilfsfunktion $T(\alpha)$ hinzunimmt, die dazu dient, mit neuen Variablen $S(\alpha)$ zu implizieren, kann man den Aufwand wieder linear beschränken. Dafür muss $T(\alpha)$ in seiner Länge linear zu der Länge von α sein.

Führe man nun also für jede Klausel $\alpha_i = (X_{i,1} \vee X_{i,k_i})$ neue Atome $Y_{i,1}, ..., Y_{i,k_i-1}$ ein und definiere das $T(\alpha) := \bigcup_{1 \leq i \leq m} T(\alpha_i)$ mit $T(\alpha_i) := \{(X_{i,1} \vee Y_{i,1})\} \cup$
$\bigcup_{1 \leq j \leq k_i} \{(\neg Y_{i,j-1} \vee X_{i,j}), (\neg Y_{i,j-1} \vee Y_{i,j}), (X_{i,j} \vee Y_{i,j})\} \cup \{(\neg Y_{i,k_i-1} \vee X_{i,k_i})\}$.
Der zugehörige Implikationsgraph zu einer Klausel $T(\alpha_i)$:

$$\begin{array}{ccccc}
\neg X_{i,2} & \neg X_{i,3} & \cdots & \neg X_{i,k_i} \\
\downarrow & \downarrow & \cdots & \downarrow \\
\neg Y_{i,1} \leftarrow & \neg Y_{i,2} \longleftarrow & \cdots & \leftarrow \neg Y_{i,k_{i-1}} \\
\downarrow & \downarrow & \cdots & \downarrow \\
X_{i,1} & X_{i,2} & \cdots & X_{i,k_{i-1}}
\end{array}$$

Nun gilt offenbar $T(\alpha) \models X_{i,j} \vee X_{i,p}$ für $1 \leq i \leq m$ und $1 \leq j \neq p \leq k_i$ (oder auch:

je zwei verschiedene Literale einer Klausel aus α folgen semantisch aus $T(\alpha)$). Es folgt direkt, dass $T(\alpha) \models S(\alpha)$ gilt.

Der letzte Schritt in dem Beweis ist nun zu zeigen, dass wenn $S(\alpha)$ von einer Belegung $\mathfrak{I}$ erfüllt wird, auch die Erfüllbarkeit von $T(\alpha)$ folgt. Dafür ist eine Fallunterscheidung abhängig von der Belegung der Atome unter $\mathfrak{I}$ genügend (verfolgbar am Implikationsgraphen von $T(\alpha_i)$:

Betrachte man also nacheinander den Fall, dass für alle $1 \leq p \leq k_i$ zwar $\mathfrak{I}(X_{i,p}) = 0$, jedoch für alle $1 \leq j \leq p$ $\mathfrak{I}(X_{i,j}) = 1$ ist.

Man weiß, dass $\mathfrak{I}(S(\alpha)) = 1$ ist und es entsprechende 2-Klauseln in $S(\alpha)$ gibt. Somit folgt $\mathfrak{I}(X_{i,q}) = 1$ für $p \leq q \leq k_i$. Jetzt kann man die Belegungen für Y festlegen mit $\mathfrak{I}(Y_{i,j}) := 0$ für $1 \leq j \leq p$ und $\mathfrak{I}(Y_{i,q}) := 1$ für $p \leq q \leq k_i$ und erhält so $\mathfrak{I}(T(\alpha_i)) = 1$. Da alle Atome Y pro Klausel jeweils neu gewählt wurden und damit für jeweils zwei Klauseln disjunkt sind, hat man eine erfüllende Erweiterung von $\mathfrak{I}$ für $T(\alpha)$ definiert.

Nun noch einmal zur Laufzeitbetrachtung des Vorgehens: Die Konstruktion von $T(\alpha)$ ist linear abhängig von der Länge der Eingabeformel α. $T(\alpha)$ kann also in linearer Zeit generiert werden. Außerdem handelt es sich bei $T(\alpha)$ um eine Formel aus 2-KNF, weshalb man auf $T(\alpha)$ den Linearzeit-Algorithmus SAT-2-KNF(2-KNF α) (zu finden unter Algorithmus 6.12, mit leichten Abänderungen übernommen, [KL94, vgl. S. 86]) anwenden kann und erhält somit, falls $T(\alpha)$ erfüllbar ist, eine erfüllende Belegung und damit eine Umbenennung $f(\alpha)$ aus HORN. Dieses Vorgehen bezeichnet man nun als UMBENENNUNG(KNF α), zu finden als Algorithmus 2.1. □

Algorithmus 2.1 UMBENENNUNG

1: **function** UMBENENNUNG(KNF α) ▷ Input: Formel α aus KNF ▷ Output: *true* und eine Funktion $f(\alpha)$ aus allen Umbenennungen von α, mit der Bedingung, dass $f(\alpha)$ aus HORN ist, sollte eine solche Umbenennung existieren. Andernfalls: $false$.
2: {Sei $\alpha = \{\alpha_1, ..., \alpha_m, \alpha_{m+1}, ..., \alpha_r\}$ mit Unit-Klauseln $\{\alpha_{m+1}, ..., \alpha_r\}$.}
3: $\sigma := \{\alpha_1, ..., \alpha_m\}$
4: $\beta := \{\alpha_{m+1}, ..., \alpha_r\}$
5: $T(\alpha) := \emptyset$
6: **for** $i = 1$ to m **do**
7: {Sei $\alpha_i = (X_{i,1} \vee ... \vee X_{i,k_i})$}
8: **for** $j = 1$ to $k_i - 1$ **do**
9: $Y_{i,j}$:= NEWATOM
10: $T_i := \{(X_{i,1} \vee Y_{i,1})\} \cup \{(\neg Y_{i,k_i-1} \vee X_{i,k_i})\}$
11: $T_i := T_i \cup \bigcup_{1<j<k_i} \{(\neg Y_{i,j-1} \vee X_{i,j}), (\neg Y_{i,j-1} \vee Y_{i,j}), (X_{i,j} \vee Y_{i,j})\}$
12: $T(\alpha) := T(\alpha) \cup T_i$
13: **if** SAT-2-KNF($T(\alpha)$) $= false$ **then**
14: **return** $false$
15: **else**
16: {Sei $\mathfrak{I}$ die berechnete erfüllende Belegung.}
17: **for** atom $A : \sigma$ **do**
18: **if** $\mathfrak{I}(A) = 1$ **then**
19: $f(A) := \neg A$
20: **else**
21: $f(A) := A$
22: **for** literal $X : \beta$ **do**
23: **if** ATOM(X) $\notin$ ATOMS(σ) **then**
24: $f(X) := X$
25: **return** *true*

Es folgt die Definition der Erfüllbarkeitsäquivalenz zweier Formeln, welche für das nächste Lemma wichtig ist:

Definition 2.3 (Erfüllbarkeitsquivalenz) *Zwei Formeln α und β werden als zueinander erfüllbarkeitsäquivalent bezeichnet, wenn gilt: α ist erfüllbar genau dann, wenn β erfüllbar ist. Abgekürzt kann dies auch mit $\alpha \approx^{sat} \beta$ ausgedrückt werden. ([KL94, vgl. S. 13])*

Wie schnell zu erkennen ist, ist die Erfüllbarkeitsäquivalenz ein relativ schwaches Kriterium, da sie für alle erfüllbaren (bzw. unerfüllbaren) Formeln untereinander gilt – unabhängig von der jeweils betrachtenen Belegung. Ein stärkeres Kriterium hingegen ist die Äquivalenz, welche im späteren Verlauf unter Definition 5.1 vorgestellt wird.

Nun aber zur Erfüllbarkeitsäquivalenz: Der Algorithmus TRANS-K-KNF(KNF α, integer k) (zu finden unter Algorithmus 6.13, mit leichten Abänderungen übernommen [KL94, vgl. S. 40]) erzeugt zu beliebigen Formeln aus KNF erfüllbarkeitsäquivalente

Formeln aus k-KNF ($k > 2$). Er ist hier deshalb interessant, da nach seiner Anwendung die Umbenennungsmöglichkeit einer Formel α aus KNF nach HORN erhalten bleibt. Genauer (der Beweis des Lemmas wurde selbst erbracht):

Lemma 2.4 *Sei α eine Formel aus KNF und β = TRANS-K-KNF(α, 3). Es existiert eine Umbenennung $f_0(\alpha)$ aus HORN genau dann, wenn eine Umbenennung $f_1(\beta)$ aus HORN existiert.*

Beweis. Sei α eine beliebige Formel aus KNF und β = TRANS-K-KNF(α, 3).

Zuerst die Richtung von links nach rechts: Es existiere eine Umbenennungsfunktion $f_0(\alpha)$ aus HORN. Der Beweis erfolgt nun über eine Transformation von f_0 zu f_1, sodass $f_1(\beta)$ ebenfalls in HORN liegt. Das Hinzufügen der neuen Atome aus β an f_0 wird hier der Einfachheit halber vernachlässigt, solange diese nur positv von f_1 abgebildet werden sollen.

Dafür wird die Umbenennungsfunktion auf die Formel β angewandt und geprüft, ob $f_0(\beta)$ in HORN liegt. Sollte dies der Fall sein, müssen keine Veränderungen vorgenommen werden und $f_1 = f_0$. Andernfalls werden die Klauseln in $f_0(\beta)$ gesucht, die nicht in HORN liegen. Diese können nur aufgetrennte Formeln sein, da sonst die Klauseln direkt aus α in β übernommen wurden und somit diese mit der Umbenennungsfunktion f_0 immer noch in HORN liegen. Nachfolgend sind alle möglichen Klauseltypen von $f_0(\alpha)$ für eine Klausellänge $n > 3$ sowie die zugehörigen Klauseln aus $f_0(\beta)$ aufgelistet:

1. $f_0(\alpha) = (X_1 \vee \neg X_2 \vee ... \vee \neg X_n)$

   $f_0(\beta) = (X_1 \vee \neg X_2 \vee \neg A_1) \wedge ... \wedge (A_{n-2} \vee \neg X_n)$

   bzw.

   $f_0(\alpha) = (\neg X_1 \vee X_2 \vee ... \vee \neg X_n)$

   $f_0(\beta) = (\neg X_1 \vee X_2 \vee \neg A_1) \wedge ... \wedge (A_{n-2} \vee \neg X_n)$

   Das positive Literal ist also in der ersten Klausel und somit liegen die gesamten Klauseln in HORN. Mögliche Zwischenklauseln haben wegen der Grundbedingung, dass $f_0(\alpha)$ in HORN liegt, die Form $(A_{i-2} \vee \neg X_i \vee \neg A_{i-1})$ und erzeugen somit keinen Konflikt.

2. $f_0(\alpha) = (\neg X_1 \vee \neg X_2 \vee ... \vee X_n)$

   $f_0(\beta) = (\neg X_1 \vee \neg X_2 \vee \neg A_1) \wedge ... \wedge (A_{n-2} \vee X_n)$

   Das positive Literal befindet sich in der letzten Klausel. Also wird A_{n-2} negativ zur Umbenennung hinzugefügt. Da so ein neuer Konflikt in einer möglichen vorigen Zwischenklausel $(A_{i-2} \vee \neg X_i \vee A_{i-1})$ entsteht, wird dieser Vorgang so lange wiederholt, bis schließlich A_1 ebenfalls negativ zur Umbenennung hinzugefügt wurde. Dann liegen die betrachteten Klauseln mit der so erzeugten Umbenennung $f_1(\beta)$ ebenfalls in HORN.

3. $f_0(\alpha) = (\neg X_1 \vee \neg X_2 \vee ... \vee X_i \vee ... \vee \neg X_n)$

   $f_0(\beta) = (\neg X_1 \vee \neg X_2 \vee \neg A_1) \wedge ... \wedge (A_{i-2} \vee X_i \vee \neg A_{i-1}) \wedge ... \wedge (A_{n-2} \vee \neg X_n)$

Das positive Literal ist in einer Zwischenklausel. Hier ist das Vorgehen wieder analog zum 2. Fall: A_{i-2} wird der Umbenennung negativ hinzugefügt, wodurch erneut ein Konflikt in der vorigen Klausel entsteht. Also wird der Vorgang so lange wiederholt, bis schließlich A_1 wieder negativ zur Umbenennung hinzugefügt wurde und die Klauseln unter der neuen Umbenennung f_0 wieder in HORN liegen.

4. $f_0(\alpha) = (\neg X_1 \vee \neg X_2 \vee ... \vee \neg X_n)$

   $f_0(\beta) = (\neg X_1 \vee \neg X_2 \vee \neg A_1) \wedge ... \wedge (A_{n-2} \vee \neg X_n)$

   Da es sich hier um eine nur negative Klausel aus α handelt, müssen keine Veränderungen der Umbenennungsfunktion durchgeführt werden, da alle Klauseln maximal ein positives neues Atom enthalten.

Über die vorgestellten Fälle kann jede beliebige Umbenennungsfunktion $f_0(\alpha)$ aus HORN in eine Umbenennungsfunktion $f_1(\beta)$ aus HORN überführt werden.

Von rechts nach links: Die Transformation einer Umbenennung $f_1(\beta)$ aus HORN zu einer Funktion $f_0(\alpha)$ aus HORN ist wesentlich leichter: Es müssen lediglich die neu hinzugefügten Atome entfernt werden, und das so entstehende f_0 angewandt auf α liefert eine Formel aus HORN. Dies liegt darin begründet, dass β zu einer ursprünglichen Klausel aus α unter einer Umbenennung f_1 maximal ein positives Literal X_i enthalten kann. Angenommen dies wäre nicht der Fall und eine in β aufgeteilte Klausel aus α enthielte zwei positive Literal X_i und X_k, würde dies zu einem Konflikt führen. Beispiel: $(\neg X_1 \vee \neg X_2 \vee A_1) \wedge ... (\neg A_{i-2} \vee X_i \vee \neg A_{i-1}) \wedge ... \wedge (\neg A_{k-2} \vee X_k \vee \neg A_{k-1}) ... \wedge (A_{n-2} \vee \neg X_n)$. Zwar könnte von A_{i-2} ausgehend jedes vorige Atom in einer Umbenennung als negatives vorhanden sein, und somit wäre noch kein Widerspruch erzeugt. Bei dem Hinzufügen vom negativen A_{k-2} sowie den vorigen Atomen bis einschließlich A_{i-1} entsteht jedoch der Konflikt: A_{i-1} müsste negativ in der Klausel von X_i vorkommen, wäre mit einer notwendigen Umbenennung allerdings positiv, und somit kann eine Umbenennung $f_1(\beta)$ aus HORN pro ursprünglicher Klausel aus α lediglich ein positives Literal enthalten, das nicht neu hinzugefügt wurde.

Somit ist auch hier eine Transformation von $f_1(\beta)$ nach $f_0(\alpha)$ gegeben, und die Aussage ist bewiesen. □

Der nachfolgende Satz stellt einen Zusammenhang zwischen der Möglichkeit einer Umbenennung nach HORN und der Erfüllbarkeit für Formeln aus 2-KNF her:

Satz 2.5 *Sei α eine Formel aus 2-KNF.*

1. *Wenn α keine Unit-Klauseln enthält und eine Umbenennung $f(\alpha)$ existiert, sodass $f(\alpha)$ in HORN liegt, ist α erfüllbar.*

2. *Andersherum kann man daraus schließen, dass, wenn α erfüllbar ist, eine Umbenennung $f(\alpha)$ existiert, sodass diese in HORN liegt.*

Beweis.

1. Wenn eine Formel aus HORN keine Unit-Klauseln enthält, ist sie erfüllbar und da eine Umbenennung diese Erfüllbarkeit von α erhält, folgt die Aussage direkt.

2. Zuerst wendet man SAT-2-KNF auf α an und erhält damit eine erfüllende Belegung $\mathfrak{I}$. Eine sinnvolle Umbenennung wäre dann $f(A) = \neg A$, falls $\mathfrak{I}(A) = 1$ und $f(A) = A$ sonst. Betrachte man nun eine Klausel $(X_1 \vee X_2)$ aus α, für diese gilt dann $\mathfrak{I}(X_1) = 1$ oder $\mathfrak{I}(X_2) = 1$. O.B.d.A. gelte $\mathfrak{I}(X_1) = 1$. Wenn nun $X_1 = A$ ist, gilt $f(A) = \neg A$. Für $X_1 = \neg A$ und aufgrund von $\mathfrak{I}(\neg A) = 1$ folgt direkt $f(\neg A) = \neg A$ und somit ist $f(X_1)$ ein negiertes Atom. Mit der Grundbedingung, dass α aus 2-KNF ist, ist jede Klausel unter der Umbenennung f aus HORN und somit auch die gesamte Umbenennung $f(\alpha)$.

□

Bei einer weiteren Einschränkung der Klasse der Zielformeln erhält man jedoch ein deutlich komplexeres Problem. Eine hierfür bedeutsame Zielklasse wären Formeln aus DHORN:

Satz 2.6 *Das folgende Problem ist NP-vollständig: Sei α aus HORN. Es existiert eine Umbenennungsfunktion $f(\alpha)$, sodass diese in DHORN liegt.*

Beweis. Das Problem kann in polynomieller Zeit mit einem nicht deterministischen Algorithmus entschieden werden. Dafür wird eine Umbenennung geraten und anschließend überprüft, ob diese den Anforderungen entspricht. Hier wird für das Beweisen der Vollständigkeit das 1-3-SAT-Problem (im Anhang zu finden unter Problem 4) reduziert, jedoch nimmt man hierbei statt positiven Klauseln nur negative.

Sei also $\alpha = \{\alpha_1, ..., \alpha_m\}$ aus HORN mit $\alpha_i = (\neg A_{i,1} \vee \neg A_{i,2} \vee \neg A_{i,3})$ und Atomen $A_{i,j}$ gegeben. Dann gilt, dass $\alpha \in$ 1-3-SAT ist genau dann, wenn eine Umbenennungsfunktion $f(\alpha)$ existiert, sodass diese in DHORN liegt.

Wenn $\alpha \in$ 1-3-SAT nun gilt, dann existiert eine Belegung $\mathfrak{I}$, sodass aus jeder Klausel von α genau ein negatives Atom auf *wahr* gesetzt wird ($\mathfrak{I}(\neg A_{i,j}) = 1$ für alle $1 \leq i \leq m$ und jeweils ein $1 \leq j \leq 3$). Für diese auf *wahr* gesetzten negativen Atome A ist die Umbenennung f dann definiert als $f(A) = \neg A$ und $f(A) = A$ für die restlichen Atome. Da so in jeder Klausel exakt ein negiertes Atom durch ein nicht-negiertes Atom ersetzt wird, ist $f(\alpha)$ aus DHORN.

Jetzt zum umgekehrten Fall, also sei eine Umbenennung $f(\alpha)$ aus DHORN gegeben, dann existiert in jeder Klausel α_i genau ein negiertes Atom $\neg A_{i,j}$, welches durch $A_{i,j}$ ersetzt wird. Die restlichen negierten Atome bleiben erhalten und man erhält eine erfüllende Belegung die den Anforderungen entspricht, wenn man $\mathfrak{I}(\neg A) = 1$ setzt, falls $f(A) = \neg A$ und $\mathfrak{I}(A) = 0$ sonst. Somit liegt α in 1-3-SAT und die Aussage ist bewiesen. □

Durch den vorigen Satz ergibt sich folgendes Korollar:

Korollar 2.7 *Das folgende Problem ist NP-vollständig: Sei α aus KNF. Es existiert eine Umbenennungsfunktion $f(\alpha)$, sodass diese in DHORN liegt.*

Wenn man nun hingegen die Klasse 2-KNF betrachtet – anstelle von HORN bzw. KNF – ist schneller feststellbar, ob eine Umbenennung nach DHORN existiert:

Lemma 2.8 *Das folgende Problem ist in linearer Zeit entscheidbar: Sei α aus 2-KNF. Es existiert eine Umbenennungsfunktion $f(\alpha)$, sodass diese in DHORN liegt.*

Beweis. Wie in dem Beweis zu Satz 2.2 kann man auch hier wieder den Linearzeit-Algorithmus SAT-2-KNF für den Beweis der Aussage nutzen. Dafür konstruiert man für jede Formel $\alpha = \{\alpha_1, ..., \alpha_m\}$ aus 2-KNF eine Formel β – ebenfalls aus 2-KNF –, die erfüllbar ist genau dann, wenn eine Umbenennung $f(\alpha)$ aus DHORN existiert.

Fallunterscheidung für die Konstruktion von β:

$$\beta_i = \begin{cases} (\neg A \leftrightarrow B) & \text{falls } \alpha_i = (\neg A \vee \neg B) \text{ oder } \alpha_i = (A \vee B) \\ (A \leftrightarrow B) & \text{falls } \alpha_i = (\neg A \vee B) \text{ oder } \alpha_i = (A \vee \neg B) \\ (A) & \text{falls } \alpha_i = (A) \\ (\neg A) & \text{sonst.} \end{cases}$$

Nehme man nun an, dass β mit der Belegung $\mathfrak{I}$ erfüllbar ist, dann folgt hierfür die Umbenennung $f(A) = A$ für $\mathfrak{I}(A) = 1$ und $f(\neg A) = \neg A$ für $\mathfrak{I}(A) = 0$ sonst. Mit dieser Umbenennung wird α in eine Formel aus DHORN transformiert.

Betrachtet man den umgekehrten Fall, dass eine Umbenennung $f(\alpha)$ aus DHORN gegeben ist, so erhält man eine erfüllende Belegung $\mathfrak{I}$ für β mit $\mathfrak{I}(A) = 1$ für $f(A) = A$ und $\mathfrak{I}(A) = 0$ sonst.

Alles in allem bleibt der Aufwand hierfür linear, da die Konstruktion von β in Linearzeit erfolgen kann und mittels des Algorithmus SAT-2-KNF eine Formel aus 2-KNF auf ihre Erfüllbarkeit überprüft werden kann.

Das so aus diesem Beweis entstehende Verfahren wird als UMBENENNUNG-2-KNF-DHORN(2-KNF α) bezeichnet und ist unter Algorithmus 2.2 zu finden, wobei die Umbenennung als Markierung der Atom-Knoten in der Datenstruktur ausgegeben wird. □

Algorithmus 2.2 UMBENENNUNG-2-KNF-DHORN

1: **function** UMBENENNUNG-2-KNF-DHORN(2-KNF α) ▷ Input: Formel α aus 2-KNF ▷ Output: *true* und eine Funktion $f(\alpha)$ aus allen Umbenennungen von α als Markierung in den Atom-Knoten, mit der Bedingung, dass $f(\alpha)$ aus DHORN ist, sollte eine solche Umbenennung existieren. Andernfalls: $false$.
2: **for** $i = 1; i \leq m; i++$ **do**
3: **if** $\alpha_i = (\neg A \vee \neg B)$ or $\alpha_i = (\neg A \vee \neg B)$ **then**
4: $\beta_i := (A \vee B) \wedge (\neg A \vee \neg B)$
5: **if** $\alpha_i = (\neg A \vee B)$ or $\alpha_i = (\neg A \vee \neg B)$ **then**
6: $\beta_i := (\neg A \vee B) \wedge (A \vee \neg B)$
7: **if** $\alpha_i = (A)$ **then**
8: $\beta_i := A$
9: **if** $\alpha_i = (\neg A)$ **then**
10: $\beta_i := \neg A$
11: $\beta_i = \{\beta_1, ..., \beta_m\}$
12: **if** SAT-2-KNF$(\beta) = false$ **then**
13: **return** $false$
14: **else**
15: $\mathfrak{I}(\beta) = 1$
16: **for** atom $A : \alpha$ **do**
17: **if** $\mathfrak{I}(A) = 1$ **then**
18: $f(A) := A$
19: **else**
20: $f(A) := \neg A$
21: **return** $true$

Zwar gibt es nicht für jede Formel aus KNF eine Umbenennung in eine Formel aus HORN, aber trotzdem kann es sinnvoll sein, zumindest festzustellen, ob nicht mindestens eine Anzahl an Klauseln m in Klauseln aus HORN transformiert werden können. Dieses Problem wird mit „Partial Renaming HORN" bezeichnet:

Problem 1 (Partial Renaming HORN)
Eingabe: Formel $\alpha = \{\alpha_1, ..., \alpha_n\}$ aus KNF, $m \leq n$ gegeben.
Frage: Existiert eine Umbenennungsfunktion f aus allen möglichen Umbenennungen von α, sodass nach der Umbenennung wenigstens m Klauseln aus HORN sind.

Lemma 2.9 *Das Problem „Partial Renaming HORN" ist NP-vollständig.*

Beweis. Für den Beweis der NP-Vollständigkeit wird hier wieder eine Reduktion des 1-3-SAT-Problems genutzt. Sei $\alpha = \{\{A_{1,1}, A_{1,2}, A_{1,3}\}, ..., \{A_{n,1}, A_{n,2}, A_{n,3}\}\}$ mit nur positiven Literalen gegeben und β werde wie folgt aus α konstruiert: Zu allen Klauseln $(A_{i,1} \vee A_{i,2} \vee A_{i,3})$ sei $\beta_i = \{(A_{i,1} \vee A_{i,2} \vee A_{i,3}), (\neg A_{i,1} \vee \neg A_{i,2}), (\neg A_{i,2} \vee \neg A_{i,3}), (\neg A_{i,1} \vee \neg A_{i,3}), (\neg A_{i,1} \vee A_{i,2} \vee A_{i,3}), (A_{i,1} \vee \neg A_{i,2} \vee A_{i,3}), (A_{i,1} \vee A_{i,2} \vee \neg A_{i,3})\}$ und $\beta = \beta_1 \cup ... \cup \beta_n$.

Nun gilt es zu zeigen, dass α in 1-3-SAT liegt genau dann, wenn es ein δ gibt, das eine Teilmenge von β ist, sodass gilt: Es gibt mindestens $5n$ Klauseln in δ und eine Umbenennungsfunktion f aus allen möglichen Umbenennungen von β existiert mit $f(\delta)$ ist aus HORN.

Nehme man nun an, dass α aus 1-3-SAT ist, so existiert eine Belegung $\mathfrak{I}$, mit der in jeder Klausel genau ein Atom $A_{i,j}$ wahr wird. O.B.d.A. sei dies für alle i das erste Atom, also $A_{i,1}$. Dann definiert man $f(A) = \neg A$ für $\mathfrak{I}(A) = 1$ und $f(A) = A$ sonst und erhält so eine Umbenennung, die fünf Klauseln aus β_i in Klauseln aus HORN transformiert (genauer: die zweite, dritte, vierte, sechste und siebte Klausel).

Jetzt noch die umgekehrte Richtung:

Existiere eine Umbenennung f, so existiert für jedes einzelne i ein j mit $f(A_{i,j}) = \neg A_{i,j}$ und $f(A_{i,t}) = A_{i,t}$ für $t \neq j$. Durch diese Umbenennung wird eine Belegung $\mathfrak{I}$ induziert, die $\mathfrak{I}(A_{i,j}) = 1$ für $f(A_{i,j}) = \neg A_{i,j}$ und $\mathfrak{I}(A_{i,j}) = 0$ für $f(A_{i,j}) = A_{i,j}$ lautet. Mit dieser Belegung $\mathfrak{I}$ wird ein Atom pro Klausel aus α wahr und somit liegt α in 1-3-SAT, womit unsere Annahme bewiesen ist. □

Stelle man sich nun praktisch vor, dass man eine Wissensbasis hat, bei der ein Vorrat an Fakten für jede Eingabe festgelegt ist. Ein Nutzer kann nun einzelne Fakten (positive Unit-Klauseln) hinzufügen. Gibt es nun eine Eingabe, die zu einem Widerspruch führen könnte? Formal ausgedrückt:

Problem 2 (Auswahl HORN)

Eingabe: Menge von Atomen $\{A_1, B_1\}, ..., \{A_n, B_n\}$*; Formel* α *aus HORN.*

Frage: Existiert aus jedem Paar $\{A_m, B_m\}$ *ein* Z_m *für* $1 \leq m \leq n$*, sodass gilt:* $\alpha \cup \bigcup_{1 \leq i \leq n} \{(Z_i)\}$ *ist widerspruchsvoll.*

Lemma 2.10 *Das Problem „Auswahl HORN" ist NP-vollständig.*

Beweis. Vorgehen: Man rate eine Folge $(Z_1, ..., Z_n)$ und überprüfe, ob $\alpha \cup \{(Z_1, ..., Z_n)\}$ widerspruchsvoll ist, was in Linearzeit möglich ist. Für den Beleg der Vollständigkeit kann man schließlich eine Reduktion des 3-SAT-Problems durchführen (zu finden unter Problem 5).

Sei nun also $\alpha = \bigcup_{1 \leq i \leq n} \{(X_{i,1} \vee X_{i,2} \vee X_{i,3})\}$ gegeben. α setzt sich zusammen aus den Literalen $X_{i,j}$ und den Atomen $A_1, ..., A_m$.

Nun konstruiert man eine Formel β aus α. Zuerst seien dafür neue Atome A_i^+ und A_i^- sowie neue Literale $X_{i,j}^*$ defniert über die Mengen

$M_i := \{A_i^+, A_i^-\}$

$$X_{i,j}^* := \begin{cases} A_k^+ & \text{falls } X_{i,j} = A_k \\ A_k^- & \text{sonst.} \end{cases}$$

für $1 \leq k \leq m$, $1 \leq i \leq n$ und $1 \leq j \leq 3$. D.h. die neuen Literale $X_{i,j}^*$, welche für jedes ursprüngliche Literal der Formel erstellt werden, sind im Falle eines ursprünglich positiven Literals mit A_k^+ definiert und im Falle eines ursprünglich negativen mit A_k^-.

β sei nun folgendermaßen definiert, mit weiteren neuen Atomen $Y_1, ..., Y_n$, Y^+ und Y^-:

$\beta := \{X^*_{i,j} \to Y_i \mid 1 \leq j \leq 3, 1 \leq i \leq n\}$
$\cup\{\neg A_i^+ \vee \neg A_i^- \mid 1 \leq i \leq n\}$
$\cup\{(Y_1, ..., Y_n \to Y^+), (Y_1, ..., Y_n \to Y^-), (\neg Y^+ \vee \neg Y^-)\}$.

Das so konstruierte β ist nun eine Formel aus HORN. Außerdem ist α erfüllbar genau dann, wenn es eine Auswahl $Z_1 \in M_1, ..., Z_m \in M_m$ gibt, mit der $\beta \cup \{(Z_1, ..., Z_m)\}$ eine widerspruchsvolle Formel ist.

Sei nun α aus 3-SAT, dann existiert eine Belegung $\mathfrak{I}$, mit der für alle $1 \leq i \leq n$ jeweils mindestens ein $j \in \{1, 2, 3\}$ existiert mit $\mathfrak{I}(X_{i,j}) = 1$. Aufgrund dessen sei Z_k für $1 \leq k \leq m$ mit

$$Z_k = \begin{cases} A_k^+ & \text{falls } \mathfrak{I}(A_k) = 1 \\ A_k^- & \text{sonst.} \end{cases}$$

definiert.

Damit gibt es für alle i nun ein j und ein k mit $X^*_{i,j} = Z_k$. Dadurch folgt: $Z_1 \wedge ... \wedge Z_m \wedge \beta \models Y_1 \wedge ... \wedge Y_n \models Y^+ \wedge Y^-$. Zusammen mit $(\neg Y^+ \vee \neg Y^-)$ entsteht so ein Widerspruch.

Schließlich noch der umgekehrte Fall:

Gebe es ein $Z_k \in M_k$ mit $1 \leq k \leq m$, für welches $Z_1 \wedge ... \wedge Z_m \wedge \beta$ widerspruchsvoll sei. Ohne $(\neg Y^+ \vee \neg Y^-)$ wäre die Formel erfüllbar. Y^+ und Y^- können nur folgen, wenn auch $Y_1, ..., Y_n$ folgen (also: für jedes i mit $1 \leq i \leq n$ folgt ein $X^*_{i,j}$). Als Konsequenz daraus existiert für jedes k mit $1 \leq k \leq m$ ein Z_k mit $Z_k = X^*_{i,j}$. Dabei ist Z_k entweder A_k^+ oder A_k^-.

Wählt man nun eine Belegung $\mathfrak{I}$ mit $\mathfrak{I}(A_k) = 1$, falls $Z_k = A_k^+$ ist, und $\mathfrak{I}(A_k) = 0$ sonst, erhält man eine erfüllende Belegung. Dies liegt daran, dass dann für jede Klausel $(X_{i,1} \vee X_{i,2} \vee X_{i,3})$ ein Literal $X_{i,j}$ mit *wahr* bewertet wird. Somit ist die Aussage bewiesen. $\square$

3 Resolution

Nachfolgendes Kapitel richtet sich inhaltlich nach dem Kapitel „Unit-Resolution", außer es ist entsprechend markiert ([KL94, vgl. S. 231-243]).

Die Resolution ist ein Verfahren der Logik, mit dem Widerlegungen logischer Formeln aus KNF bewiesen werden können. Dabei wird auf je zwei Klauseln einer Formel die Resolutionsregel angewandt, die schließlich die Resolvente liefert. Ziel ist es dabei, die leere Klausel zu erzeugen, um die betrachtete Formel zu widerlegen. Es folgt die genaue Definition:

Definition 3.1 (Resolutionsregel) *Seien zwei Klauseln α und β gegeben. α enthält das Literal X und β $\neg X$. Dann ist die Resolution auf die beiden Klauseln anwendbar und liefert eine neue Klausel $(\alpha \setminus X) \cup (\beta \setminus \neg X))$, die als Resolvente bezeichnet wird. α und β werden als die Elternklausel der Resolvente bezeichnet und man kann sagen, dass α und β über X bzw. $\neg X$ resolviert werden. Das Schema*

$$\frac{\alpha \qquad \beta}{(\alpha \setminus X) \cup (\beta \setminus \neg X))}$$

wird als Resolutionsregel bezeichnet. ([KL94, vgl. S. 137])

Ein Beispiel einer Resolutionswiderlegung über einen Resolutionsbaum für die Formel $\alpha = (\neg X_1) \wedge (X_1 \vee X_2) \wedge (X_1 \vee \neg X_2) \wedge (X_1 \vee \neg X_2 \vee X_3)$ ist nachfolgend abgebildet:

$$\dfrac{\dfrac{\dfrac{(\neg X_1) \qquad (X_1 \vee \neg X_3)}{(\neg X_3)} \qquad \dfrac{\dfrac{(\neg X_1) \qquad (X_1 \vee X_2)}{(X_2)} \qquad (X_1 \vee \neg X_2 \vee X_3)}{(X_1 \vee X_3)}}{(X_1)} \qquad (\neg X_1)}{\sqcup}$$

Außerdem wird im nachfolgenden Verlauf auch auf Primimplikanten eingegangen werden, welche für die Vereinfachung von Formeln eine elementare Bedeutung haben.

Definition 3.2 (Primimplikant) *Betrachte man eine erfüllbare Formel α, dann heißt eine Klausel k Primimplikant von α genau dann, wenn k semantisch aus α folgt und keine echte Teilklausel k^* von k mit $\alpha \models k^*$ existiert.*

Für eine widerspruchsvolle Formel α ist die leere Klausel ($\sqcup$) der einzige Primimplikant von α. ([KL94, vgl. S.148])

Der Primimplikant des vorigen Beispiels wäre also die leere Klausel.

Außerdem ist für ein besseres Verständnis der Beziehungen der nachfolgend vorgestellten Resolutionsverfahren eine Visualisierung unter Abbildung 3.1 dargestellt. Dabei beinhalten die weiter oben angeordneten Verfahren die mit ihnen direkt oder indirekt verbundenen weiter unten stehenden Verfahren in einer Art Teilmengenbeziehung.

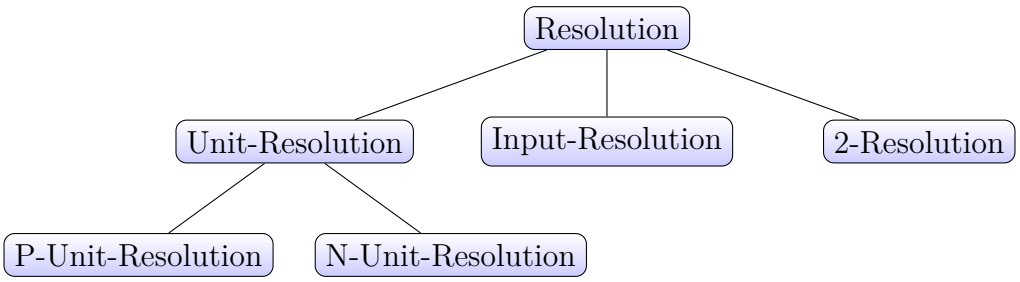


Abbildung 3.1: Übersicht der nachfolgend vorgestellten Resolutionsverfahren und deren Teilmengenbeziehung zueinander.

3.1 Unit-Resolution

Beginne man also mit der Unit-Resolution, die als Einschränkung der Resolution wie folgt definiert sei:

Definition 3.3 (Unit-Resolution) *Die Unit-Resolution ist eine Einschränkung des Resolutionsverfahrens, bei dem in jedem Resolutionsschritt eine der Elternklauseln aus genau einem Literal bestehen muss.*

Der erste Satz hierzu ist von grundlegender Bedeutung, um zuverlässig mit der Unit-Resolution arbeiten zu können:

Satz 3.4 *Die Unit-Resolution ist für HORN widerlegungsvollständig und korrekt. D.h. eine Formel α aus HORN ist widerspruchsvoll genau dann, wenn sich mittels der Unit-Resolution in endlich vielen Schritten aus α die leere Klausel erzeugen lässt.*

Beweis. Darauf zu schließen, dass eine Formel α widerspruchsvoll ist, wenn mittels der Unit-Resolution aus α die leere Klausel erzeugt werden kann, folgt aus der Korrektheit der Resolution. In die andere Richtung kann der Beweis hingegen über vollständige Induktion über die Anzahl an Atomen aller widerspruchsvollen Formeln aus HORN verlaufen. Als Induktionsanfang $n = 1$ gibt es nur einen Typ an Formeln: $\alpha = \{A, \neg A\}$. Die Behauptung folgt sofort.

Für $n > 1$ und eine widerspruchsvolle Formel aus HORN mit n Atomen muss mindestens eine positive Klausel in α existieren, und da α in HORN liegt, muss es sich hierbei um eine Unit-Klausel A handeln. Gibt es eine Klausel $\neg A$ in α, wird mit einem Unit-Resolutionsschritt A mit $\neg A$ resolviert.

Andernfalls resolviert man alle Klauseln die $\neg A$ als Literal enthalten mit A. Somit erhält man $\alpha[A/1]$: Eine wiederum widerspruchsvolle Formel aus HORN, die sich aus den vorigen Resolventen sowie aller Klauseln von α ohne Vorkommen von $\neg A$ zusammensetzt. Mit der Induktionsvorraussetzung gilt nun $\alpha[A/1] \vdash_{U-RES} \sqcup$, und da $\alpha \vdash_{U-RES} \alpha[A/1]$ gilt, erhält man eine Unit-Resolutionswiderlegung von α. □

Sollte also die betrachtete Formel aus HORN einen Widerspruch enthalten, würde dieser in endlich vielen Resolutionsschritten entdeckt werden. Für KNF gilt dies allerdings nicht, wie an folgendem Gegenbeispiel abzulesen ist:

$(X \vee Y) \wedge (\neg X \vee Y) \wedge (X \vee \neg Y) \wedge (\neg X \vee \neg Y)$

Die Formel ist offenbar widerspruchsvoll, jedoch lässt sich die Unit-Resolution nicht anwenden. Außerdem lässt sich aus dem Beweis auch die Eigenschaft ablesen, dass man für eine Unit-Resolutionswiderlegung keine zusätzlichen Klauseln speichern muss, da stets eine der Elternklauseln durch eine Resolvente ersetzt werden kann. Dies liegt daran, dass die Elternklausel von der Resolvente subsummiert wird.

Anmerkung: Betrachtet man eine Formel α aus HORN mit den Klauseln α_1 bis α_n und ein Literal X. Es gelte $X, \alpha_j \vdash^1_{U-RES} \beta$ (also: aus X und α_j lässt sich in einem Schritt mittels der Unit-Resolution β resolvieren). Dann gilt: α ist widerspruchsvoll genau dann, wenn $(\alpha \setminus \{\alpha_\mathrm{j}\}) \cup \{\beta\}$ widerspruchsvoll ist.

Satz 3.5 *Betrachte man nun eine Formel α aus KNF. Es gilt, dass sich mit der Unit-Resolution aus α die leere Klausel erzeugen lässt genau dann, wenn ein $\alpha' \subseteq \alpha$ existiert, für das es eine Umbenennung $f \in U(\alpha')$ gibt, mit der Bedingung, dass $f(\alpha')$ aus HORN und widerspruchsvoll ist.*

Beweis. Von rechts nach links: Dies liegt offenbar daran, dass eine Unit-Widerlegung auch dann gilt, wenn Umbenennungen einer Formel rückgängig gemacht werden.

Von links nach rechts: Der Beweis erfolgt über vollständige Induktion über die Anzahl der Atome in α': Es gelte $\alpha \vdash_{U-RES} \sqcup$ und α' sei eine minimale Teilmenge der Klauseln aus α für die $\alpha' \vdash_{U-RES} \sqcup$ gelte. Für den Induktionsanfang $n = 1$ besitzt α' nur ein Atom und ist somit $\alpha' = \{X, \neg X\}$ und damit eine widerspruchsvolle Formel aus HORN.

Enthalte α' nun $n + 1$ Atome. Da es sich um eine unit-widerlegbare Formel handelt, ist eine Unit-Klausel X in α' vorhanden. Da es sich außerdem um eine minimale Formel handelt, kann man davon ausgehen, dass α' keine weitere Klausel hat, die X enthält. Streicht man nun diese Unit-Klausel sowie alle weiteren Vorkommen von $\neg X$ aus den übrigen Klauseln, erhält man die Formel $\alpha'[X/1]$. Aus dieser lässt sich mit der Unit-Resolution wieder die leere Klausel herleiten, außerdem ist sie auch wieder minimal. Durch die Induktionsvoraussetzung ist bekannt, dass eine Umbenennung $f(\alpha') \in HORN$ existiert, die man zu einer Umbenennung von α' erweitern kann: Ist X positiv, dann sei $f(X) = X$ und $f(X) = \neg X$ sonst. Mit der entstehenden Umbenennung f gilt schließlich, dass $f(\alpha')$ aus HORN ist. □

Eine weitere Einteilung, die sich aufgrund des Beweises der Widerlegungsvollständigkeit und Korrektheit der Unit-Resolution aus Satz 3.4 anbietet, ist die in positive und negative Unit-Resolution (P-/N-Unit-Resolution):

Definition 3.6 (P-/N-Unit-Resolution) *Bei der P-/N-Unit-Resolution muss in jedem Resolutionsschritt – analog zur Unit-Resolution – eine der Elternklauseln aus genau einem Literal bestehen, wobei dieses bei P-Unit-Resolutionsschritten positiv und bei N-Unit-Resolutionsschritten negativ sein muss.*

Aus dem Satz 3.4 ergibt sich:

Korollar 3.7 *Die P-Unit-Resolution ist für HORN widerlegungsvollständig und korrekt. D.h. eine Formel α aus HORN ist widerspruchsvoll genau dann, wenn sich mittels der P-Unit-Resolution in endlich vielen Schritten aus α die leere Klausel erzeugen lässt.*

Dies gilt jedoch nicht für die N-Unit-Resolution, was sich an folgendem Gegenbeispiel ablesen lassen kann: $(X) \wedge (Y) \wedge (\neg X \vee \neg Y)$. Zwar ist die Formel widerspruchsvoll, jedoch ist die N-Unit-Resolution nicht anwendbar.

3.2 2-Resolution

Außerdem sei kurz die 2-Resolution vorgestellt:

Definition 3.8 (2-Resolution) *Als 2-Resolutionsschritt wird ein Resolutionsschritt bezeichnet, der die Einschränkung erfüllt, dass mindestens eine der Elternklauseln aus zwei Literalen bestehen muss. Eine Reihe von solchen 2-Resolutionsschritten wird als 2-Resolution bezeichnet. ([KL94, vgl. S. 202])*

Mit dieser lassen sich alle Units einer erfüllbaren Formel aus HORN herleiten:

Lemma 3.9 *Sei α eine erfüllbare Formel aus HORN. Für alle Literale X gilt dann: X folgt semantisch aus α genau dann, wenn aus α mittels der 2-Resolution X hergeleitet werden kann.*

Beweis. Aufgrund der Ableitbarkeit mittels der P-Unit-Resolution aller Atome A, die aus einer erfüllbaren Formel α aus HORN semantisch folgerbar sind ([KL94, vgl. S. 233]), genügt es einen Beweis für negative Literale X zu erbringen. Außerdem ist ebenfalls bereits bekannt, dass die Resolution und damit auch die 2-Resolution korrekt sind. Infolgedessen ist es also ausreichend hier aufzuzeigen, dass für eine erfüllbare Formel α aus HORN die semantische Folgerbarkeit einer negativen Unit aus α ($\alpha \models \neg A$) für ein Atom A eine Ableitbarkeit mit Hilfe der 2-Resolution impliziert ($\alpha \vdash_{2-Res} \neg A$).

Nehme man nun an, dass $\alpha \models \neg A$ gilt. Dann ist $\alpha \wedge A$ eine widerspruchsvolle Formel aus HORN, für die eine Unit-Resolutionswiderlegung $\alpha \wedge A \vdash_{U-Res} \sqcup$ existiert. Das selbe gilt für $\alpha[A/1]$ ($\alpha[A/1] \vdash_{U-Res} \sqcup$). Wenn man nun allerdings in $\alpha[A/1]$ die eliminierten Vorkommen von $\neg A$ wieder einsetzt, so gilt: $\alpha[A/1](\neg A) \vdash_{2-Res} \neg A$. Da bekannt ist, dass α erfüllbar ist, müssen in α Klauseln mit dem Literal $\neg A$ existieren. Ein Teil dieser Klauseln muss wiederum für die Widerlegung $\alpha \wedge A \vdash_{U-Res} \sqcup$ notwendig sein. $\alpha[A/1](\neg A)$ ist eine Teilformel von α und somit gilt $\alpha \vdash_{2-Res} \neg A$. □

3.3 Input-Resolution

Ein weiteres Resolutionsverfahren – nämlich die Input-Resolution – kann hier ebenfalls von Nutzen sein:

Definition 3.10 (Input-Resolution) *Als Input-Resolution wird eine Reihe von Resolutionsschritten benannt, bei der jeweils eine der Ausgangsklauseln eine Klausel der ursprünglichen Formel, und die andere eine Resolvente aus dem vorigen Resolutionsschritt sein muss. ([KL94, vgl. S. 188])*

Eine wichtige Eigenschaft der Input-Resolution:

Lemma 3.11 *Mit der Input-Resolution lassen sich alle Primimplikanten für erfüllbare Formeln aus HORN herleiten.*

Beweis. Gegeben sei eine erfüllbare Formel α aus HORN, sowie ein Primimplikant $\beta = (X_1 \vee ... \vee X_n)$ von α. $\alpha \wedge \neg X_1 \wedge ... \wedge \neg X_n$ wäre also widerspruchsvoll. Sei $\alpha_R = \alpha[X_1/0, ..., X_n/0]$, also die Formel, die beim Streichen aller X_i sowie aller Klauseln, welche $\neg X_i$ beinhalten resultiert. α_R ist dann auch widerspruchsvoll.

Da die Input-Resolution für Formeln aus HORN sowohl korrekt als auch widerlegungsvollständig ist ([KL94, vgl. S. 194]), existiert nun also eine Widerlegung mit der Input-Resolution, die aus α_R die leere Klausel erzeugt. Fügt man nun alle eben entfernten Vorkommen von X_i wieder zu α hinzu, führt die Input-Resolution nicht mehr zur leeren Klausel, sondern zu $\beta' \subseteq \beta$. Da β ein Primimplikant von α ist, folgt $\beta' = \beta$ und somit ist die Aussage bewiesen. □

Das Verfahren von Quine/McCluskey dient dazu, logische Funktionen zu minimieren. Dabei ist ein schwieriger Schritt in diesem Verfahren die Auswahl von geeigneten Primimplikanten, in diesem Fall für Formeln in disjunktiver Normalform. Für HORN besteht ein ähnliches Problem, da die Anzahl der Primimplikanten möglicherweise exponentiell in der Länge der Ausgangsformel steigen kann:

Lemma 3.12 *Für alle $k \geq 2$ gibt es eine Formel α_k aus HORN, die die Länge $6k$ hat, und mindestens 2^k Primimplikanten besitzt.*

Beweis. Sei α_k für $k \geq 2$ wie folgt definiert: $\alpha_\mathrm{k} := \beta_0 \wedge ... \wedge \beta_\mathrm{k}$. Dabei sei $\beta_0 := (\neg X_1 \vee ... \vee \neg X_{2\mathrm{k}})$ und $\beta_\mathrm{i} := (\neg X_\mathrm{i} \vee X_{\mathrm{k}+1}) \wedge (X_\mathrm{i} \vee \neg X_{\mathrm{k}+1})$ für $1 \leq i \leq k$. Dann hat jedes α_k die Klauseln $\gamma_{\epsilon_1,...,\epsilon_k} = (\neg X_{1+\gamma_1 \cdot k} \vee ... \vee \neg X_{k+\gamma_k \cdot k})$ als Primimplikanten mit $(\gamma_1, ..., \gamma_\mathrm{k}) \in \{0,1\}^k$. Also hat α_k mindestens 2^k Primimplikanten und eine Länge von $6k$. □

3.4 Algorithmen

Mit diesen Methoden und Erkenntnissen kann nun effizienter entschieden werden, ob eine Formel aus HORN widerspruchsvoll ist. Das gängige Verfahren hierfür wäre, die Unit-Resolution so lange anzuwenden, bis die leere Klausel erzeugt worden ist bzw. keine weiteren Unit-Resolutionsschritte mehr möglich sind. Der so funktionierende Algorithmus erhält natürlich die Vollständigkeit und Korrektheit der Unit-Resolution.

Dieses Verfahren ist implementiert im Algorithmus 3.3: Einfach-Unit-Res(KNF α), der eigentlich für Formeln aus KNF konzipiert ist, aber auch für Formeln aus HORN funktioniert. Wenn er auf Formeln aus HORN angewandt wird, wird er der Übersichtlichkeit halber als Einfach-Sat-HORN(HORN α) bezeichnet. Er liefert das Ergebnis $false$, wenn die Formel mit der Unit-Resolution widerlegbar ist. Eine Bedingung für die Eingabe ist hier noch, dass Klauseln keine Mehrfachvorkommen von Literalen aufweisen. Da hierfür allerdings in jeder Schleife die ganze Formel sowie alle neuen Klauseln

durchlaufen werden muss, befindet sich die Laufzeit nicht mehr im linearen bzw. logarithmischen Bereich.

Algorithmus 3.3 EINFACH-UNIT-RES

```
1:  function EINFACH-UNIT-RES(KNF α)        ▷ Input: Formel α aus KNF ohne
    Mehrfachvorkommen von Literalen innerhalb einer Klausel. ▷ Output: false, wenn
    α unit-widerlegbar ist. Andernfalls: true.
2:      U := ∅
3:      for clause k : α do
4:          if IsUnit(k) then
5:              U := U ∪ FirstActiveLiteral(k)
6:      while U ≠ ∅ do
7:          X := SelectElement(U)
8:          U := U \ X
9:          for clause k : α do
10:             if X ∈ k then
11:                 DeleteClause(k, α)
12:         for clause k : α do
13:             if ¬X ∈ k then
14:                 DeleteLiteral(¬X, k)
15:                 if k = ⊔ then
16:                     return false
17:                 if IsUnit(k) then
18:                     U := U ∪ x
19:     return true
```

Um diesen Algorithmus nun zu optimieren, wird eine angepasste Datenstruktur konzipiert, um Suchprozesse zu beschleunigen und in einem Schritt auszuführen (zusammengetragen, [KL94, vgl. S. 36ff. sowie S. 237f.]). Das Verfahren richtet sich laut [KL94, vgl. S. 217 und S. 237] nach [DG84] sowie [Min88].

Die Datenstruktur muss dabei folgendermaßen zusammengesetzt sein: Die „oberste Ebene" wird als knf-form-entry benannt und enthält zentrale Elemente der Formel. Diese sind:

name: Ein Name für die gesamte Formel.

nr-of-clauses: Die Anzahl an Klauseln.

max-clauses-nr: Die größte vergebene Klauselnummer.

max-del-mark: Die bisher größte Löschmarke, um virtuell und nachvollziehbar Klauseln und Literale entfernen zu können.

atom-array: Ein Array aus Pointern auf die enthaltenen Atome.

clause-list: Eine Liste aus Pointern auf die Klauseln.

Die Liste der Atome ist hier als Array von Records realisiert (aufsteigend sortiert), welches wiederum aus Einträgen zusammengesetzt ist, die hier mit atom-entry bezeichnet werden. Bei den einzelnen Einträgen handelt es sich um:

name: Ein Bezeichner.

value: Der Wahrheitswert des Atoms.

bound: Eine Markierung, ob der Wahrheitswert gesetzt wurde.

status: Ein Zustandsfeld. Dieses kann abhängig von der entsprechenden Unit-Ableitbarkeit des Atoms den Wert *positiv*, *negativ* oder *nounit* annehmen, letzerer bedeutet, dass das Atom entweder nicht als unit ableitbar ist, oder die Ableitbarkeit noch unbekannt ist.

neg-oc-list: Ein Liste aus Pointern auf die negativen Vorkommen des Atoms.

pos-oc-list: Ein Liste aus Pointern auf die positiven Vorkommen des Atoms.

neg-oc: Ein Zähler der negativen Vorkommen des Atoms.

pos-oc: Ein Zähler der positiven Vorkommen des Atoms.

Die Einträge der Klauselliste (hier als clause-entry bezeichnet) sind:

clause-nr: Die Nummer der Klausel.

del-mark: Eine Löschmarke, die diesmal für das virtuelle Löschen ganzer Klauseln benutzt werden kann.

literal-list: Eine Liste aus Pointern auf die enthaltenen Literale.

lit-oc: Ein Zähler für die Anzahl an enthaltenen Literalen.

formula: Ein Point auf die zugehörige Formel.

Außerdem gibt es noch eine Datenstruktur für die einzelnen Literale, die als literal-entry bezeichnet werden:

sign: Das Vorzeichen des Literals (0 für *negativ*, 1 für *positiv*).

del-mark: Eine Löschmarke, für das virtuelle Löschen einzelner Literale.

atom: Ein Zeiger auf das zugehörige Atom.

clause: Ein Zeiger als Rückverweis auf die zugehörige Klausel.

Die entsprechenden Datenstrukturen sind der Verständlichkeit halber noch einmal als Tabelle 3.1 abgebildet. Außerdem ist unter dem selbst erbrachten Beispiel 3.2 ausschnittweise eine Formel in der Datenstruktur dargestellt.

Tabelle 3.1: Datenstrukturen für eine Formel aus KNF (zusammengetragen, [KL94, vgl. S. 36ff. sowie S. 237f.]).
Dabei wird zuerst die zugehörige Datenstruktur angegeben, und nachfolgend die jeweiligen Elemente, wobei der Datentyp links steht. Mit ↑ werden Pointer bezeichnet, wobei ihr Datentyp immer direkt nachfolgend angegeben ist.

knf-form-entry:	
string	name
integer	nr-of-clauses
integer	max-clauses-nr
integer	max-del-mark
↑records-array	atom-array
↑dotted-pair	clause-list
atom-entry:	
string	name
boolean	value
boolean	bound
integer	status
↑dotted-pair	neg-oc-list
↑dotted-pair	pos-oc-list
integer	neg-oc
integer	pos-oc
clause-entry:	
integer	clause-nr
integer	del-mark
↑dotted-pair	literal-list
integer	lit-oc
↑knf-form-entry	formula
literal-entry:	
integer	sign
integer	del-mark
↑atom-entry	atom
↑clause-entry	clause

Tabelle 3.2: Beispiel für eine Formel aus HORN, selbst erbracht.
Zuerst ist jeweils die Bezeichnung der aktuell betrachteten Datenstruktur angegeben, mit dem zugehörigen Element in Klammern. In der jeweiligen Datenstruktur selbst steht links der Bezeichner und rechts der Inhalt. Pointer werden mit $\rightarrow$ angegeben.

Beispiel: $(X_1 \vee \neg X_2) \wedge (X_3)$	
knf-form-entry(Beispiel):	
name	Beispiel
nr-of-clauses	2
max-clauses-nr	2
max-del-mark	0
atom-array	$\rightarrow [A_1, A_2, A_3]$
clause-list	$\rightarrow [k_1, k_2]$
atom-entry(A_1):	
name	A_1
value	NULL
bound	false
status	nounit
neg-oc-list	$\rightarrow$ NULL
pos-oc-list	$\rightarrow [k_1]$
neg-oc	0
pos-oc	1
clause-entry(k_1):	
clause-nr	1
del-mark	0
literal-list	$\rightarrow [X_1, X_2]$
lit-oc	2
formula	$\rightarrow [Beispiel]$
literal-entry(X_1):	
sign	1
del-mark	0
atom	$\rightarrow A_1$
clause	$\rightarrow k_1$

Nun wird der Algorithmus an die eben entworfene Datenstruktur angepasst und als UNIT-RES(KNF α) bezeichnet (siehe Algorithmus 3.4) bzw. SAT-HORN(HORN α) für Formeln aus HORN:

Wie der vorige Algorithmus liefert auch dieser wieder das Ergebnis $false$, wenn die Formel unit-widerlegbar ist. Allerdings ist bei dem Ergebnis $true$ nicht unbedingt sicher, ob die Formel erfüllbar ist. Für Formeln aus KNF bedeutet es nur, dass sie nicht unit-widerlegbar ist. Nur für Formeln aus HORN kann dann auf die Erfüllbarkeit geschlossen werden (siehe Satz 3.4). Es sei vorausgesetzt, dass die zu bearbeitenden Formeln zum einen keine Mehrfachvorkommen von Literalen beinhalten, und zum anderen auch keine Tautologien sind (beide Bedingungen können über ein Markierungsverfahren in linearer Zeit sichergestellt werden).

Das Vorgehen ist dabei wie folgt:

Erst werden die vorhandenen Unit-Klauseln verarbeitet, um bei den Atomen die Einträge entsprechend ihres in der Unit-Klausel vorkommenden Vorzeichens zu markieren. Dazu werden die Unit-Klauseln auf einem Stapel gesammelt und dabei bearbeitet: Das zugehörige Atom einer Unit-Klausel wird mit dem Vorzeichen aus der Unit-Klausel gekennzeichnet. Sollte es bereits markiert gewesen sein, wurde es bereits hergeleitet. Beim Auftauchen eines komplementären Vorzeichens hingegen muss die Formel zwei zueinander komplementäre Units des gleichen Atoms enthalten und der Algorithmus stoppt und liefert $false$.

Anschließend wird der Stapel abgearbeitet:

Literale werden aus einer Klausel gestrichen, indem das Literal entsprechend gekennzeichnet wird und der Literalzähler der Klausel um eins verringert wird. Sollte der Literalzähler einer Klausel den Wert eins erreichen, wurde ein bis dahin noch unmarkiertes Literal gefunden, das nun eine Unit-Klausel ist und wie vorhin beschrieben markiert und auf den Stapel gelegt wird. Sobald der Stapel leer ist und kein Widerspruch entdeckt wurde, lässt sich die betrachtete Formel nicht mit der Unit-Resolution widerlegen.

Es folgt noch eine Beschreibung der Hilfsfunktionen:

FIRSTACTIVELITERAL(clause k): Durchsucht die Liste an Literalen einer Klausel k und gibt das erste als nicht gelöscht markierte Literal zurück.

DELETELITERAL(literal X): Löschmarke vom Literal X wird gesetzt, außerdem wird der Literalzähler im Knoten der Klausel um eins verringert.

ISUNIT(clause k): Überprüft, ob der Literalzähler von k den Wert eins aufweist.

Wenn der Algorithmus nun mit dem Wert $false$ terminiert, sind ein Teil der ableitbaren Units bei den Atomen markiert. D.h. die betrachtete Formel ist widerspruchsvoll und daher lassen sich alle Units aus LITERALS(ATOMS(α)) folgern. Beim Wert $true$ hingegen sind sämtliche Units, die mit der Unit-Resolution folgerbar sind, entsprechend bei den Atomen markiert.

Algorithmus 3.4 UNIT-RES

```
1: function UNIT-RES(KNF α)   ▷ Input: Formel α aus KNF ohne Mehrfachvorkommen von Literalen innerhalb einer Klausel sowie ohne tautologische Klauseln.   ▷ Output: false, wenn α unit-widerlegbar ist. Andernfalls: true.
2:     {Zuerst sammelt man die Units der Eingabeformel ein und testet auf das Vorkommen komplementärer Units:}
3:     for clause k : α do
4:         if IsUnit(k) then
5:             literal X := First-Active-Literal(k)
6:             if atom(X) = nounit then
7:                 status(atom(X)) = sign(X)
8:                 Push(atom(X), stack)
9:             else if status(atom(X))! = sign(X) then
10:                return false
11:    {Abarbeiten des Stapels}
12:    while !Empty(stack) do
13:        A := Top(stack)
14:        Pop(stack)
15:        if status(A) = pos then
16:            delete-literal-list = neg-oc-list(A)
17:        else
18:            delete-literal-list = pos-oc-list(A)
19:        for literal X : delete − literal − list do
20:            k := clause(X)
21:            DeleteLiteral(X)
22:            if IsUnit(k) then
23:                X1 := FirstActiveLiteral(k)
24:                if status(atom(X1)) = nounit then
25:                    status(atom(X1)) := Sign(X1)
26:                    Push(atom(X1), Stack)
27:                else if status(atom(X1)) ≠ Sign(X1) then
28:                    return false
29:    return true
```

Nun noch zur Laufzeitbetrachtung:

Sei m die Anzahl an Atomen und n die Länge der Formel. Das Aufbauen der Datenstruktur ist in $\mathcal{O}(n)$ möglich. Dies liegt daran, dass aus den Atomnamen A_i direkt die Position bestimmt werden kann (da sie aufsteigend sortiert sind). Hätte man diese Bedingung nicht durch das Array sichergestellt, müsste beim Einlesen der Formel parallel eine Symboltabelle angelegt werden, deren Konstruktion $\mathcal{O}(n \cdot log \cdot n)$ Schritte benötigen würde.

Auch die Ergebnisformel kann in $\mathcal{O}(n)$ ausgegeben werden.

Die Atome werden jeweils maximal ein Mal auf dem Stapel abgelegt und können nach der Bearbeitung an ihrer Markierung erkannt werden. Das anschließende Abarbeiten des Stapels betrifft jedes Atom auch nur ein Mal, was über die Vorkommensliste sichergestellt wird. FirstActiveLiteral wird pro Klausel ein Mal aufgerufen. Das Sammeln der Units für den Stapel wird durch einmaliges Durchsuchen der Formel erledigt. Also folgt die Gesamtlaufzeit für Unit-Res: $\mathcal{O}(n)$ und damit gilt:

Satz 3.13 *Für Formeln aus KNF entscheidet der Algorithmus Unit-Res(KNF α) in Linearzeit, ob diese mit der Unit-Resolution widerlegt werden können.*

Für Sat-HORN folgt direkt aus der vorigen Aussage sowie Satz 3.4:

Korollar 3.14 *Das Erfüllbarkeitsproblem für Formeln aus HORN ist in linearer Zeit mit dem vorgestellten Algorithmus Sat-HORN(HORN α) entscheidbar.*

Für erfüllbare Formeln aus KNF liefert der eben vorgestellt Algorithmus alle mit der Unit-Resolution herleitbaren Unit-Klauseln. Im Fall von Formeln aus HORN sogar alle folgerbaren positiven Unit-Klauseln.

Außerdem ist die bearbeitete Formel bereits teilweise reduziert. Im Detail bedeutet das, dass keine Klausel zu den Units komplementäre Literale mehr enthält. Um diese Reduktion abzuschließen, also auch alle von herleitbaren Unit-Klauseln subsummierte Klauseln zu entfernen, muss lediglich noch der Algorithmus Unit-Reduce(KNF α) (siehe Algorithmus 3.5) auf die Formel angewandt werden, welcher in $\mathcal{O}(n)$ terminiert (n: Länge der Formel).

Die Linearzeit des Algorithmus lässt sich leicht an den Schleifen ablesen: Jede Klausel wird nur einmal betrachtet, so wie auch in den Klauseln jedes Literal nur einmal betrachtet wird. Außerdem kann eventuell eine Klausel gelöscht werden, was jedoch auch wieder in der Länge der jeweiligen Klausel beschränkt wäre.

Sollte gewünscht sein, dass die Formel nur nach den positiven Units reduziert werden soll, müsste man die Literallöschungen der negativen Units in Unit-Res rückgängig machen und im Verfahren Unit-Reduce dürften nur Klauseln für positive Units entfernt werden. Dieses Verfahren kann als Pos-Unit-Reduce(KNF α) bezeichnet werden und ist in seiner Laufzeit wieder linear. Für erfüllbare Formeln aus HORN können außerdem alle positiven Units in linearer Zeit ermittelt werden, indem nach der Anwendung von SAT-HORN alle Atome aufgesammelt werden, für die das Feld status den Wert *positiv* hat. Dieses Vorgehen wird als Pos-Unit-HORN(HORN α) bezeichnet.

Das letzte in diesem Kapitel vorgestellte Verfahren soll UNIT-HORN(HORN α) sein (siehe Algorithmus 3.6). Dies berechnet die Menge aller ableitbaren Units für eine erfüllbare Formel aus HORN mit einer Laufzeit von $\mathcal{O}(n \cdot m)$, wobei m hier die Anzahl der Atome beschreibt.

Somit folgt schließlich der letzte Satz dieses Kapitels:

Satz 3.15 *Die Menge aller folgerbaren positiven Units einer erfüllbaren Formel aus HORN kann in Linearzeit mit dem Algorithmus POS-UNIT-HORN(HORN α) ermittelt werden, die Menge aller folgerbaren Units hingegegen mit dem Algorithmus UNIT-HORN(HORN α), wobei die Zeit hier $\mathcal{O}(n \cdot m)$ beträgt, mit der Anzahl an Atomen m und der Länge der Formel n.*

Algorithmus 3.5 UNIT-REDUCE

```
1:  function UNIT-REDUCE(KNF α)                                          ▷
    Input: Formel α aus KNF ohne Mehrfachvorkommen von Literalen innerhalb einer
    Klausel sowie ohne tautologische Klauseln.        ▷ Output: Reduzierte Formel zu α.
2:      if UNIT-RES(α) then
3:          for clause k : α do
4:              for literal X : α do
5:                  if DEL-MARK(X) = active then
6:                      if STATUS(ATOM(X)) = SIGN(X) then
7:                          DELETECLAUSE(k)
8:      else
9:          for clause k : α do
10:             DELETECLAUSE(k)
11:             ADDEMPTYCLAUSE(α)
12:     return α
```

Algorithmus 3.6 UNIT-HORN

```
1:  function UNIT-HORN(HORN α)    ▷ Input: Erfüllbare Formel α aus HORN.    ▷
    Output: Menge aller ableitbaren Units von α.
2:      U := POS-UNIT-HORN(α)
3:      α' := POS-UNIT-REDUCE(α)
4:      for atom A : ATOMS(α') do
5:          if SAT-HORN(α' ∧ A) = false then
6:              U := U ∪ ¬A
7:      return U
```

4 Unique-Satisfiability für Formeln aus HORN

Nachfolgendes Kapitel richtet sich inhaltlich nach dem Kapitel „Unique-Satisfiability für Horn-Formeln", außer es ist entsprechend markiert ([KL94, vgl. S. 244-257]).

Unique-Satisfiability bezeichnet das Problem, ob es für eine gegebene aussagenlogische Formel genau eine erfüllende Belegung gibt. Für Formeln aus Horn wäre das also:

Problem 3 (Unique-Satisfiability für Formeln aus HORN)
Eingabe: Eine aussagenlogische Formel α aus HORN.
Frage: Gibt es für α genau eine erfüllende Belegung $\mathfrak{I}$.

Definition 4.1 (UNI-SAT) *Eine Formel α aus HORN wird der Einfachheit halber als aus der Klasse UNI-SAT bezeichnet, wenn sie das Problem der Unique-Satisfiability für Formeln aus HORN erfüllt.*

Das Problem besitzt für Formeln aus HORN eine deutlich geringere Komplexität als für Formeln aus KNF, für die es $coNP$-hart ist und in D^P liegt. Formal ausgedrückt kann man auch sagen, dass eine Formel α mit Atomen $A_1, ..., A_m$ genau dann in UNI-SAT liegt, wenn $\alpha \approx X_1 \wedge ... \wedge X_m$ gilt, und für $i = 1...m$ gilt, dass $X_i = A_i$ oder $X_i = \neg A_i$ ist (die Erklärung zur Äquivalenz ist zu finden unter Definition 5.1). Für ein α aus HORN muss also lediglich noch der Linearzeit-Algorithmus SAT-HORN für die Überprüfung der Erfüllbarkeit angewandt werden und schließlich geprüft werden, ob für jedes Atom aus α das positive oder negative Literal folgt, um die Zugehörigkeit in quadratischer Zeit zu entscheiden.

Nachfolgend soll nun gezeigt werden, dass für die Klasse HORN der Zeitaufwand sogar linear beschränkt werden kann. Dafür bietet es sich an, einen Algorithmus einzuführen, der die betrachtete Formel vorweg vereinfacht. Er wird mit PREPARE-UNIQUE-HORN(HORN α) bezeichnet und nutzt zuerst den Algorithmus POS-UNIT-REDUCE (Abwandlung von UNIT-REDUCE, Algorithmus 3.5), um eine Formel α aus HORN um die folgerbaren positiven Unit-Klauseln zu reduzieren (in Linearzeit). Sollte α nach diesem Schritt die leere Klausel enthalten, kann abgebrochen werden, da α dann widersprüchlich ist. Sonst sind diese Unit-Klauseln entfernt und die entsprechenden Atome markiert in α^*. Allerdings muss dann noch getestet werden, ob alle folgerbaren positiven Units aus α vereinigt mit den Atomen aus α^* die Menge der Atome aus α sind, da sonst für die übrigen Atome aus α beliebige Bewertungen möglich wären – dies wäre ein Widerspruch zu der Definition der Unique-Satisfiability.

Aus diesen Schritten folgt der Linearzeit-Algorithmus 4.7: PREPARE-UNIQUE-HORN.

Algorithmus 4.7 PREPARE-UNIQUE-HORN

1: **function** PREPARE-UNIQUE-HORN(HORN α) ▷ Input: Formel α aus HORN. ▷ Output: Vereinfachte Formel α^*.
Sollte α nicht erfüllbar sein oder die positiven Unit-Klauseln von α zusammen mit den Atomen von α^* nicht den Atomen von α entsprechen: $NULL$.
2: $\alpha^* :=$ POS-UNIT-REDUCE(α)
3: **if** ISEMPTYCLAUSECONTAINED(α^*) **then**
4: **return** $NULL$
5: **for** atom A : ATOM-LIST(α^*) **do**
6: **if** (STATUS(A) $\neq$ *pos*) and (POS-OC(A) + NEG-OC(A) = 0) **then**
7: **return** $NULL$
8: **return** α^*

Wenn zwar für die Formel α noch nicht festgestellt wurde, dass diese die Unique-Satisfiabilty erfüllt, kann man dennoch einige Aussagen über die Formel (bzw. die entstehende Formel) treffen:

1. α^* ist erfüllbar.
2. Aus α^* folgen keine weiteren positiven Unit-Klauseln.
3. α ist aus UNI-SAT genau dann, wenn α^* aus UNI-SAT ist genau dann, wenn aus α^* semantisch alle negativen Atome A aus α^* folgen.

Die Ausgabe des Algorithmus ist entweder bei erfolgreicher Verarbeitung ein Pointer auf α^* oder aber $NULL$, sonst.

Mit nachfolgendem Lemma erhält man für Formeln aus HORN eine leicht überprüfbare notwendige Bedingung für die Unique-Satisfiability:

Lemma 4.2 *Sei eine erfüllbare Formel α aus HORN gegeben und bereits mit PREPARE-UNIQUE-HORN vereinfacht worden. Ist α in UNI-SAT, dann gibt es für alle Atome A_i aus α eine Klausel $\neg A_i$ oder $\neg A_i \vee A_j$ in α mit $i \neq j$.*

Beweis. Mit der Grundvoraussetzung, dass die Formel α aus HORN ist, und mit PREPARE-UNIQUE-HORN bereits vereinfacht wurde und erfüllbar ist, folgt, dass $\mathfrak{I}$ mit $\mathfrak{I}(A) = 0$ für alle Atome A aus α eine erfüllende Belegung ist.

Nehme man nun an, dass für ein Atom A_k aus α weder eine Klausel $\neg A_k$ noch $\neg A_k \vee A_j$ mit $k \neq j$ in α enthalten sei, außerdem kann in α auch nicht die positive Unit-Klausel A_k vorkommen, da α bereits mit PREPARE-UNIQUE-HORN bearbeitet wurde, und somit die positiven Unit-Klauseln entfernt wurden.

Deshalb ist in jeder Klausel, welche A_k als Literal in positiver oder negativer Form enthält, ein vom Atom A_k verschiedenes weiteres negatives Literal beinhaltet. Daraus würde folgen, dass eine Belegung $\mathfrak{I}$ mit $\mathfrak{I}(A) = 0$ für alle Atome außer A_k aus α und $\mathfrak{I}(A_k) = 1$ sonst, α erfüllen würde. Damit entstünde aber ein Widerspruch zur Voraussetzung, dass α in UNI-SAT liegt und die Aussage ist bewiesen. □

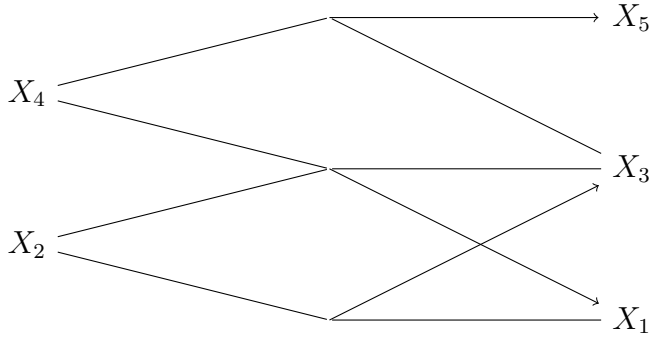
Für die Überprüfung der Unique-Satisfiability von Formeln aus HORN bietet es sich an, Graphen einzusetzen. Da Klauseln allerdings mehr als zwei Literale enthalten können, wäre eine Darstellung in Form eines normalen Graphen mit den Klauseln als Kanten und Atomen als Knoten schwierig, weshalb hier die Darstellung als Hypergraph eingeführt wird. Ein Hypergraph sei wie folgt definiert:

Definition 4.3 (gerichteter Hypergraph) *Sei V eine Menge an Knoten. Eine Hyperkante e ist ein Paar $(T(e), H(e)) \in 2^V \times 2^V$. Dabei bezeichnet $T(e)$ die Menge der Anfangsknoten (Tail-Menge) und $H(e)$ die dazu disjunkte Menge der Zielknoten (Head-Menge). Der gerichtete Hypergraph $H = (V, HE)$ besteht dann aus einer Knotenmenge V und den Hyperkanten HE.*

Betrachtet man nun die Konstruktion eines Hypergraphen für eine Formel aus DHORN, ergibt sich folgende Definition, die also quasi der Implikationsschreibweise entspricht:

Definition 4.4 (assoziierter Hypergraph) *Sei eine Formel $\alpha = \{\alpha_1, ..., \alpha_n\}$ aus DHORN gegeben. Dann ordnet man jeder Klausel α_i mit dem positiven Atom A_{i_0} und den Restatomen $\neg A_{i_1}, ..., \neg A_{i_{r_i}}$ für $1 \leq i \leq n$ eine Hyperkante $K_i = (\{A_{i_1}, ..., A_{i_{r_i}}\}, \{A_{i_0}\})$ zu. Der dann entstehende Hypergraph $H(\alpha)$ mit der Knotenmenge der Atome aus α und der Kantenmenge $\{K_1, ..., K_n\}$ heißt der zu α assoziierte Hypergraph.*

Ein Beispiel (selbst erbracht) eines assoziierten Hypergraphen zu einer Formel $\alpha = (X_1 \vee \neg X_2 \vee \neg X_3 \vee \neg X_4) \wedge (\neg X_1 \vee \neg X_2 \vee X_3) \wedge (\neg X_3 \vee \neg X_4 \vee X_5)$ aus DHORN wäre:



Für Formeln aus HORN kann eine ähnliche Konstruktion vollzogen werden, wenn ein neues Atom $falsum$ eingeführt wird, welches an jede nur negative Atome enthaltende Klausel $(\neg A_1 \vee ... \vee \neg A_m)$ folgendermaßen angehängt wird: $(falsum \vee \neg A_1 \vee ... \vee \neg A_m)$. Dieses Atom wird immer mit $falsch$ interpretiert und stellt im entstehenden Hypergraphen eine Senke dar.

Der entstehende Hypergraph enthält außerdem einen Teilgraphen, welcher ein gerichteter Graph ist und als Graph-Teil des Hypergraphen bezeichnet wird:

Definition 4.5 (Graph-Teil) *Der Graph-Teil eines Hypergraphen bezeichnet eine Teilmenge eines Hypergraphen. Dabei beinhaltet er alle Knoten des Graphen sowie solche Hyperkanten, deren Head- und Tailmenge genau eins ist.*

Im weiteren Verlauf werden auch Klausel und Kante bzw. Atom und Knoten synonym verwendet.

4.1 Konstruktion von UNIQUE-SAT-HORN

Begonnen wird damit, dass die Formel α mit PREPARE-UNIQUE-HORN vereinfacht wird. Der dazu assoziierte Hypergraph erhält dann die Eigenschaft, dass er keine Hyperkanten mit leerer Tailmenge enthält, da aus α durch die Vorbereitung keine positiven Literale folgen. Außerdem folgt aus dem Lemma 4.2, dass es im Graph-Teil keine isolierten Knoten gibt, wenn α aus UNI-SAT ist. Dies folgt daraus, dass zu jedem Atom A_i aus α entweder eine auf $falsum$ zeigende Kante, oder eine auf A_j zeigende Kante mit $j \neq i$, enthalten sein muss.

Das Vorgehen besteht nun darin, den Graph-Teil mit Hilfe einer Tiefensuche zu durchlaufen und dabei Senken zu suchen, die nicht $falsum$ sind. Dabei werden die benötigten Schritte jeweils direkt durchgeführt, was in einem linearen Zeitaufwand resultiert, wenn der benötigte Zeitaufwand für das Zusammenfassen von Knoten herausgenommen wird. Grob dargestellt ist der Ablauf also folgendermaßen (dabei folgt das Verfahren laut [KL94, vgl. S.251] der Darstellung in [BFS92]):

1. Der Graph-Teil wird mit Tiefensuche nach einer Senke durchsucht (nach noch nicht besuchten Knoten). Sollte eine Senke ungleich $falsum$ entdeckt werden, kann α nicht aus UNI-SAT sein (siehe Lemma 4.2).

2. Falls ein Kreis entdeckt wird, werden alle in ihm enthaltenen Knoten zusammengefasst (es wird immer die aktuellste zusammengefasste Version des Graphen betrachtet).

3. Jede Kante aus dem Graph-Teil wird bei der Suche verwendet, um längere Klauseln abzukürzen. Sollte dabei eine Klausel mit zwei Literalen entstehen, wird diese mit in den Graph-Teil aufgenommen.

4. Sollte bei den vorigen Schritten keine Senke gefunden worden sein – also nachdem alle Knoten verarbeitet wurden – die ungleich $falsum$ ist, ist α aus UNI-SAT.

Vorweg noch zur Datenstruktur: Hier kann eine ähnliche benutzt werden, wie sie unter Tabelle 3.1 zu finden ist. Jedoch wird statt des Arrays an Records eine Liste der Atome verwendet (atom-list). Das Statusfeld status zeigt hier an, ob ein Atom bereits verarbeitet wurde ($visited = 1$) oder noch betrachtet werden muss ($unvisited = 0$). Für die auf dem Graph-Teil stattfindende Tiefensuche bietet es sich außerdem an, längere Klauseln von Klauseln mit zwei Literalen zu trennen, was durch eine bei den Atomen ergänzte Adjazenzliste (adj-list) ermöglicht wird. In ihr werden die Atome gespeichert, die über eine direkte Kante vom betrachteten Atom erreichbar sind, also mit ihr zusammen eine Klausel der Länge zwei bilden. Hier muss noch beachtet werden, dass das Zusammenfassen von Knoten auch deren Adjazenzlisten beinhaltet, weshalb auch ein Verweis auf das Ende dieser Liste enthalten sein sollte.

Der Eintrag bei clause-entry names lit-oc muss zusätzlich in neg-lit-oc und pos-lit-oc aufgesplittet werden, sodass diese respektive die Anzahl der negativen und positiven

Literale beinhalten. Außerdem müssen noch folgende Felder in clause-entry ergänzt werden: head, verweist auf das positive Literal der Klausel; first-visited, zu Beginn leer, weist sonst auf ein Atom; und visit-count, der zu Beginn auf null steht.

Nun zur genaueren Betrachtung des Ablaufes: Für das Zusammenfassen von Kreisen wie in Schritt zwei, deren Atome alle zu einer Äquivalenzklasse gehören, muss man die Repräsentanten der entsprechenden Klasse bestimmen und auch mehrere Klassen zusammenfassen können. Dafür dienen die Funktionen des Algorithmus:

MakeSet(atom A), zum Bilden einer neuen einelementigen Äquivalenzklasse, mit dem Repräsentanten und Element A.

Find(atom A), zum Finden des Repräsentanten, der Klasse, die das Element A beinhaltet.

Unite(atom A, atom B), zum Vereinen von zwei Äquivalenzklassen die das Element A (Klasse 1) und B (Klasse 2) enthalten.

Diese Abläufe spiegeln eine effiziente Lösung des Disjoint Set Union Problems wieder, das allerdings nicht in dieser Arbeit vorgestellt wird und laut [KL94, vgl. S.252] zu finden ist in [Tar75] und [GT85]. Natürlich sind hier die einzelnen Elemente einer Äquivalenzklasse Atome. Für das Entwerfen des Algorithmus muss außerdem noch beim Verfahren Unite darauf geachtet werden, dass der Repräsentant der entstehenden Klasse die vereinigte Adjazenzliste enthält.

Im Algorithmus selbst muss zusätzlich beachtet werden, dass man statt mit dem Atom A immer mit dem Repräsentanten Find arbeitet. Aus dem Graph-Teil werden auch nur Kanten betrachtet, für die Anfangs- und Endknoten (noch) nicht in der gleichen Klasse anzufinden sind, was sich im Ignorieren dieser Klassen bzw. Kanten in der Funktion EmptyAdjList(atom A) wiederspiegelt.

EmptyAdjList(atom A) liefert *true*, falls in der Adjazenzliste eines Atoms A kein Atom aus einer anderen Klasse als der von A mehr enthalten ist und $false$ sonst.

NextElemInAdjList(atom A) liefert hingegen das nächste Atom aus der Adjazenzliste vom Atom A, das nicht in der Klasse von A enthalten ist. Dabei wird, um sicherzustellen, dass eine Kante nicht mehrfach betrachtet wird, das ausgegebene Atom aus der Adjazenzliste gelöscht.

Bei der Tiefensuche kann ein Kreis über einen bereits besuchten Knoten aus der gleichen Äquivalenzklasse ermittelt werden. Sollte dies der Fall sein, werden die Klassen aller seitdem besuchten Knoten mit der aktuellen zusammengefasst. Für das Zusammenfassen der besuchten Knoten, müssen diese zwischengespeichert werden, was über die Pfadliste P geschieht. Initialisiert wird sie mit $falsum$ und die besuchten Atome werden ihr in umgedrehter Reihenfolge hinzugefügt, sodass das erste Listenelement dem zuletzt besuchten Atom entspricht. Beim Hinzufügen werden die Knoten mit *visited* markiert.

Betrachte man nun die Hilfsfunktion SimplifyClauses(HORN α, atom A): Diese dient dazu, negative Literale in längeren Klauseln zusammenzufassen.

Die Prozedur betrachtet dabei alle Klauseln k, welche das zuletzt in der Pfadliste aufgenommene Atom A als negatives Literal enthalten. Sollte der Wert von visit-count noch null sein, wird in first-visited das Atom A hinterlegt (also das vorher in die Pfadliste aufgenommene und an die Funktion übergebene Atom A). Anschließend wird der Ein-

trag visit-count um eins erhöht. Sobald visit-count den Wert von neg-lit-oc (Anzahl an negativen Literalen der Klausel k) erreicht, wurden alle negativen Literale der aktuell betrachteten Klausel k bearbeitet. Somit wurde aus der langen Klausel (Hyperkante) eine Klausel welche H genannt wird, bestehend aus lediglich zwei Literalen (das erste ist das first-visited und das zweite der head, also die Implikation), die schließlich in die Adjazenzliste adj-list übernommen wird. Dieser letzte Schritt wird durch den Code aus Zeile 8-12 realisiert.

Algorithmus 4.8 SIMPLIFY-CLAUSES

```
1:  function SIMPLIFY-CLAUSES(HORN α, Atom A)   ▷ Input: Formel α aus HORN, welche bereits mit PREPARE-UNIQUE-HORN bearbeitet wurde. Außerdem das letzte in die Pfadliste aufgenommene Atom A. ▷ Output: Formel α aus HORN, wobei Klauseln die A als negatives Literal enthalten wie oben beschrieben enstprechend vereinfacht wurden.
2:      {Vereinfachung der Klausel:}
3:      for literal X : NEG-OC-LIST(A) do
4:          k := CLAUSE(X)
5:          if VISIT-COUNT(k) = 0 then
6:              FIRST-VISITED(k) := A
7:          VISIT-COUNT(k) := VISIT-COUNT(k) + 1
8:          {Übernahme in Adjazenzliste:}
9:          if VISIT-COUNT(k) = NEG-LIT-OC(k) then
10:             H := FIND(FIRST-VISITED(k))
11:             ADJ-LIST(H) := CONS(HEAD(k), ADJ-LIST(H))
12:             DELETE-CLAUSE(k)
```

Algorithmus 4.9 UNIQUE-SAT-HORN

```
1: function UNIQUE-SAT-HORN(HORN α)   ▷ Input: Formel α aus HORN.   ▷ Output: true, wenn α aus UNI-SAT ist. Ansonsten: false.
2:     α* := PREPARE-UNIQUE-HORN(α)
3:     if α* = NULL then
4:         return false
5:     {Hinzufügen von falsum:}
6:     α := ∅
7:     for clause k : α* do
8:         if k = (¬A_{i_1} ∨ ... ∨ ¬A_{i_k}) then
9:             α := α ∪ (¬A_{i_1} ∨ ... ∨ ¬A_{i_k} → falsum)
10:        else
11:            α := α ∪ k
12:    {Initilialisierung:}
13:    for atom A : α do
14:        STATUS(A) = unvisited
15:        MAKESET(A)
16:    for clause k : α do
17:        {Klauseln mit zwei Literalen in Adjanzenzliste übernehmen:}
18:        if POS-OC-LIST(k) + NEG-OC-LIST(k) = 2 then
19:            H := ATOM(FIRST-NEG-LIT(k))
20:            ADJ-LIST(H) := CONS(ATOM(FIRST-POS-LITERAL(k)), ADJ-LIST(H))
21:            DELETE-CLAUSE(k)
22:        else
23:            VISIT-COUNT(k) := 0
24:            FIRST-VISITED(k) := NULL
25:            HEAD(k) := ATOM(FIRST-POS-LIT(k))
26:    return VERARBEITUNG-DER-PFADLISTE(α)
```

Algorithmus 4.10 VERARBEITUNG-DER-PFADLISTE

```
1: function VERARBEITUNG-DER-PFADLISTE(α)   ▷ Input: Formel α aus HORN, welche bereits mit UNIQUE-SAT-HORN bearbeitet wurde. ▷ Output: true, wenn α aus UNI-SAT ist. Ansonsten: falsum.
2:     {Initialisierung der Pfadliste:}
3:     P := CONS(falsum, NULL)
4:     STATUS(falsum) := visited
5:     while true do
6:         if FIND(FIRST(P)) = FIND(falsum) then
7:             {Existiert noch ein Knoten der eine Senke sein könnte?}
8:             if EXISTS-UNIVSITED-ATOM-IN(α) then
9:                 A := NEXT-UNVISITED-ATOM-IN(α)
10:                P := CONS(A, P)
11:                STATUS(A) := visited
12:                SIMPLIFY-CLAUSES(A, α)
13:            else
14:                {falsum ist die einzige Senke.}
15:                return true
16:        else
17:            {Die Pfadliste enthält mehrere Knoten.}
18:            A := FIND(FIRST(P))
19:            {Existiert die Klausel (A → B)? Sonst ist A eine Senke.}
20:            if EMPTY-ADJ-LIST(A) then
21:                return false
22:            B := FIND(NEXT-ELEM-IN-ADJ-LIST(A))
23:            {Es ist FIND(B) ≠ A und B aus ADJ-LIST(A) entfernt.}
24:            {Verschmelzung der auf dem Zyklus liegenden Knoten:}
25:            if STATUS(B) = visited then
26:                while FIND(FIRST(P)) ≠ FIND(B) do
27:                    H := FIRST(P)
28:                    P := REST(P)
29:                    UNITE(H, FIRST(P))
30:                {Kante folgen:}
31:            else
32:                P := CONS(B, P)
33:                STATUS(B) := visited
34:                SIMPLIFY-CLAUSES(B, α)
```

4.2 Laufzeitbetrachtung

Alle Vorbereitungen bis zur while-Schleife in VERARBEITUNG-DER-PFADLISTE(α) sind in linearer Zeit zu erledigen. Für die Funktion UNITE ist die Anzahl an Atomen die Obergrenze an Aufrufen. In der Schleife wird bei der Tiefensuche jede Klausel maximal ein Mal besucht, zudem wird auch jedes negative Literal maximal ein Mal aus jeder Klausel entfernt. Damit beträgt die obere Schranke des Zeitaufwandes $\mathcal{O}(|\alpha|)$, wenn von dem benötigten Zeitaufwand von find abgesehen wird. Die Prozedur find wird auch in konstanter Anzahl pro Schleifendurchlauf aufgerufen, wenn hierfür von der Schleife für das Verschmelzen von Knoten und dem Aufruf in SIMPLIFYCLAUSES abgesehen wird. Die While-Schleife ist in der Anzahl der Atome beschränkt, d.h. sie bilden ihre Obergrenze. In SIMPLIFYCLAUSES ist eine Obergrenze die Anzahl der Klauseln für das Übernehmen der verkürzten Klauseln in die Adjazenzliste und somit auch für die Aufrufe von FIND(i)n der damit verbundenen Schleife. Nun ist durch die Anzahl an Atomen ja bereits die Obergrenze der Aufrufe von UNITE festgestellt, weshalb diese auch für die Durchgänge der Repeat-Schleife gilt, und dadurch auch für die Aufrufe von FIND, weshalb der Aufwand nun $\mathcal{O}(n) \cdot \text{FIND} + \mathcal{O}(m) \cdot \text{UNITE}$ beträgt, wobei n die Länge der Formel ist und m die Anzahl an Atomen.

Das Disjoint Set Union Verfahren benötigt laut [KL94, vgl. S. 257] in einer standardmäßigen Implementierung wie in [Tar75] beschrieben einen Zeitaufwand von $\mathcal{O}(n \cdot A^-(n))$ für $m \leq n$. A^- ist dabei die Inverse der Ackermannfunktion:

$$A^-(n) := \text{kleinstes } k \text{ mit } A(k,k) \geq n.$$

Die Ackermannfunktion $A : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ ist wie folgt definiert:

$$A(0,x) = x + 2$$

$$A(i,0) = 0 \text{ für } i \geq 1$$

$$A(i,1) = 2 \text{ für } i \geq 1$$

$$A(i,x) = A(i-1, A(i,x-1)) \text{ für } i,x \geq 2$$

$\mathcal{O}(n \cdot A^-(n))$ steigt zwar nicht linear, aber nahezu linear. Dies ist darin begründet, dass die Ackermannfunktion zwar schneller als alle primitiv rekursiven Funktionen wächst, jedoch immer noch rekursiv ist und somit A^- eine zwar nicht konstant, jedoch sehr langsam wachsende Funktion ist.

Für $k = 4$ ist sie beispielsweise $A(4,4) = 2^{\cdots^2}|$ *insgesamt* 65536 *mal*. Somit ist die Inverse $A^-(n) \leq 4$ für $n \leq A(4,4)$. Es lässt sich leicht feststellen, dass diese Grenze in der Praxis nicht überschritten werden sollte.

Eine schnellere Umsetzung des Disjoint Set Union Problems wird laut [KL94, vgl. S. 257] in [GT85] vorgestellt. Dieses ist auf eine Einschränkung des Problems, nämlich auf inkrementelle Bäume entwickelt worden (incremental tree set union). Die Änderungen beinhalten, dass UNITE in einem baumartigen Muster aufgerufen werden muss, und bei dem Erstellen neuer einelementiger Mengen mit MAKESET bekannt sein muss, an

welcher Stelle des Baumes diese eingefügt werden müssen. Somit würde sich der Aufwand auf $\mathcal{O}(m + n)$ belaufen, wobei m und n die Anzahl an Aufrufen von UNITE und von FIND sind.

Der Beweis, dass der Spezialfall für UNIQUE-SAT-HORN zutrifft, wird laut [KL94, vgl. S. 257] in [Pre92a] und [Pre92b] erbracht. Die Voraussetzungen an die Struktur werden dabei dadurch eingehalten, dass UNITE jeweils den vordersten Knoten mit dem nächsten Knoten der Pfadliste verbindet und somit zum einen die baumartige Struktur erfüllt wird, und zum anderen bekannt ist, wo sich die einelementige neue Menge in der Struktur befindet.

Insgesamt ist also die Linearität des Verfahrens bewiesen, es folgt:

Satz 4.6 *Das Problem der Unique-Satisfiability ist für Formeln aus HORN in Linearzeit entscheidbar.*

5 Äquivalenzprobleme für Formeln aus HORN

Nachfolgendes Kapitel richtet sich inhaltlich nach den ersten Seiten des Kapitels „Äquivalenzprobleme", außer es ist entsprechend markiert ([KL94, vgl. S. 263-266]).

Ein weiteres für die Praxis interessantes Problem ist das, ob zwei gegebene Formeln α und β äquivalent sind:

Definition 5.1 (Äquivalenz) *Zwei Formeln $\alpha = \alpha_1, ..., \alpha_m$ und $\beta = \beta_1, ..., \beta_n$ sind äquivalent, falls sowohl $\alpha \vDash \beta$ als auch $\beta \vDash \alpha$ gilt. Abgekürzt kann dies ausgedrückt werden mit $\alpha \approx \beta$.*

Dafür muss lediglich für alle Klauseln α_i aus α geprüft werden, ob diese von β erfüllt werden, also ob $\beta \vDash \alpha_i$ gilt. Der umgekehrte Fall muss natürlich auch überprüft werden ($\alpha \vDash \beta_i$ für alle β_i aus β). Diese Eigenschaft kann für Klauseln aus HORN in linearer Zeit überprüft werden, wobei sich ein für die Formel insgesamt quadratischer Zeitaufwand ergibt ($\mathcal{O}((n+m) \cdot |\alpha \wedge \beta|)$. Aus diesem Vorgehen ergibt sich der Algorithmus ÄQUIVALENZ-HORN(HORN α, HORN β) (mit Änderungen übernommen).

Algorithmus 5.11 ÄQUIVALENZ-HORN

```
1: function ÄQUIVALENZ-HORN(HORN α, HORN β)   ▷ Input: Formeln α und β aus HORN.   ▷ Output: true, wenn α äquivalent zu β ist. Andernfalls: false.
2:     for clause k : α do
3:         if SAT-HORN(α ∪ {UNIT-CLAUSE(¬X)|X ∈ k}) = true then
4:             return false
5:     for clause k : β do
6:         if SAT-HORN(β ∪ {UNIT-CLAUSE(¬X)|X ∈ k}) = true then
7:             return false
8:     return true
```

Somit folgt folgendes Lemma:

Lemma 5.2 *Für zwei Formeln α und β aus HORN ist das Äquivalenzproblem $\alpha \approx \beta$ in quadratischer Zeit lösbar.*

Wird hingegen eine Formel aus KNF auf Äquivalenz mit einer Formel aus HORN oder DHORN überprüft, erhält man ein coNP-vollständiges Problem:

Satz 5.3 *Für eine Formel α aus KNF und eine zweite Formel β aus HORN oder DHORN ist das Äquivalenzproblem $\alpha \approx \beta$ coNP-vollständig.*

Beweis. Betrachte man zuerst das Problem für eine Formel β aus HORN. Sei $\beta = A \wedge \neg A$, dann ist das Problem für α aus KNF gleichbedeutend mit der Frage, ob α widerspruchsvoll ist. Diese Frage ist coNP-vollständig.

Ist β allerdings aus DHORN, ist der Beweis etwas komplexer: Sei $\beta = \{A\}$ und $\alpha^* = \{\alpha_1, ..., \alpha_n\}$ aus KNF mit der Bedingung, dass α^* das Atom A nicht enthält. Für $\alpha = \{(\alpha_1 \vee A), ..., (\alpha_n \vee A)\}$ folgt, dass α und β äquivalent sind, gdw. α und A äquivalent sind, gdw. α^* nicht erfüllbar ist. Somit ist auch hier das Problem coNP-vollständig. □

Wird nun zu einer Formel aus KNF eine äquivalente Formel aus HORN oder DHORN gesucht, entsteht wieder ein coNP-vollständiges Problem:

Lemma 5.4 *Das folgende Problem ist coNP-vollständig: Gegeben sei eine Formel α aus KNF. Gesucht wird nun eine Formel β aus DHORN bzw. HORN, welche äquivalent zu α ist.*

Beweis. Sei T der betrachtete Formeltyp (also entweder HORN oder DHORN), α eine erfüllbare Formel aus KNF, mit den Klauseln $\alpha_1, ..., \alpha_n$ und es sei vorausgesetzt, dass ein β aus T existiere mit $\alpha \approx \beta$. Aufgrund der Abgeschlossenheit gegen Deduktion einer Formel aus HORN oder DHORN (siehe Lemma 1.10), folgt, dass es eine Teilklausel k_i aus α_i gibt, die aus α semantisch folgt und aus T ist. Sollte α_i bereits aus T sein, wird diese direkt als k_i übernommen. Schließlich erhält man $\alpha \approx k$ mit der Eigenschaft, dass $k = \{k_1, ..., k_n\}$ aus T ist, wobei deshalb nicht-deterministisch entschieden werden kann, ob α das Problem nicht erfüllt:

Für β aus HORN:

Zuerst testet man, ob α erfüllbar ist, da, wenn dies nicht der Fall wäre, jede auch nicht-erfüllbare Formel β äquivalent zu α wäre. Sollte α also nicht erfüllbar sein, wird $\beta = A \wedge \neg A$ gesetzt und wäre eine äquivalente Formel aus HORN.

Andernfalls wird für jede Klausel $\alpha_i = (A_1 \vee ... \vee A_r \vee \neg A_{r+1} \vee ... \vee \neg A_k)$ überprüft, ob die Teilklausel $(A_j \vee \neg A_{r+1} \vee ... \vee \neg A_k)$ für alle j mit $(1 \leq j \leq r)$ semantisch nicht aus α folgt. Sollte ein i mit $1 \leq i \leq n$ existieren, für das dieser Fall zutrifft, kann es für α keine äquivalente Formel aus HORN geben.

Für β aus DHORN:

Zuerst wird überprüft, ob es in α negative Klauseln gibt, in welchem Fall für α keine äquivalente Formel aus DHORN existieren kann. Ansonsten ist nun bekannt, dass α keine Klauseln ohne positives Literal besitzt, und daher auch erfüllbar ist.

Jetzt wird wieder derselbe Test wie eben ausgeführt, also ob eine Klausel α_i mit $1 \leq i \leq n$ existiert, für welche es ein j gibt mit $(1 \leq j \leq r)$, sodass $\alpha \not\models (A_j \vee \neg A_{r+1} \vee ... \vee \neg A_k)$ gilt. Sollte dies der Fall sein, kann es wieder für α keine äquivalente Formel aus DHORN geben

Zurück zur allgemeinen Betrachtung:

Das vorgestellte Problem ist offenbar für T gleich HORN oder DHORN aus coNP, wobei noch die Vollständigkeit zu beweisen bleibt.

Hierfür betrachte man die Menge $M := \{\alpha \in KNF \mid \alpha \textit{ enthält eine positive Klausel}\}$. Für $SAT \cap M$ ist das Problem NP-vollständig.

Seien nun ein neues Atom A für α und ein α aus M gegeben, außerdem wird α jetzt eine Formel α^* zugeordnet, welche die Form $\{(\alpha_1 \vee A), ..., (\alpha_n \vee A)\}$ aufweist. Jetzt zeigt man, dass α genau dann nicht erfüllbar ist, wenn ein β aus T existiert, für das $\alpha^* \approx \beta$ gilt.

Nehme man dafür nun an, dass α nicht erfüllbar wäre. Daraus würde folgen, dass $\alpha^* \approx A$ gilt.

Umgekehrt wird nun angenommen, dass für ein β aus T bereits ein äquivalentes α^* gegeben wäre. Dann gilt für alle i mit $1 \leq i \leq n$, dass $(\alpha_i \vee A)$ semantisch aus β folgen würde, also insbesondere auch für die vorausgesetzte positive Klausel $(\alpha_{i_0} \vee A)$.

Aufgrund der Abgeschlossenheit gegen Deduktion von T und der nur positiven Literale in α_{i_0}, müsste A semantisch aus β folgen oder A_{i_0} müsste semantisch aus β folgen, für ein A_{i_0} aus α_{i_0}. Würde letzeres gelten, müsste auch A_{i_0} semantisch aus α^* folgen. Dies würde jedoch einen Widerspruch darstellen, da eine erfüllende Belegung $\mathfrak{I}$ für α^* existiert mit: $\mathfrak{I}(A) = 1$ und $\mathfrak{I}(A_{i_0}) = 0$. Deshalb folgt, dass A semantisch aus β folgt, und damit auch aus α^*. Damit würde α wieder nicht erfüllbar sein. □

6 Anhang

Nachfolgend sind noch einige Algorithmen und Probleme abgedruckt, die zwar nicht explizit behandelt und erklärt werden, jedoch für einige Aussagen und Beweise von elementarer Bedeutung sind.

6.1 Algorithmen

SAT-2-KNF(2-KNF α) (mit leichten Abänderungen übernommen, [KL94, vgl. S. 86]) sucht eine erfüllende Belegung für eine Formel α aus 2-KNF und gibt *true* aus, sollte diese existieren. Die Belegung ist dann in den Atomfeldern von α markiert.

Algorithmus 6.12 SAT-2-KNF

```
1:  function SAT-2-KNF(2-KNF α)        ▷ Input: Formel α aus 2-KNF.        ▷
    Output: true, wenn α erfüllbar ist sowie eine erfüllende Belegung als Markierung in
    den Atomfeldern. Ansonsten: false.
2:      S := Liste der starken Zusammenhangskomponenten des assoziierten Graphen
3:      {S wird mit dem Algorithmus von Tarjan berechnet und ist eine Liste von Listen
    aus Paaren von Vorzeichen (0 = negiert, 1 = nicht negiert) sowie Atomen.}
4:      for s : S do
5:          if s enthält komplementäre Literale then
6:              return false
7:          {Im weiteren Verlauf ist bekannt, dass α erfüllbar ist.}
8:      for atom A : α do
9:          VALUE(A) := unbelegt
10:     for s : S do
11:         for PAIR(sign, A) : s do
12:             if VALUE(A) = unbelegt then
13:                 VALUE(A) = sign
14:     return true
```

TRANS-K-KNF(KNF α, integer k) (mit leichten Abänderungen übernommen [KL94, vgl. S. 40]) gibt eine zur Formel α erfüllbarkeitsäquivalente Formel in k-KNF aus.

Algorithmus 6.13 TRANS-k-KNF

1: **function** TRANS-K-KNF(KNF α, integer k) ▷ Input: Formel α aus KNF in der entsprechenden Datenstruktur, natürliche Zahl $k > 2$. ▷ Output: Erfüllbarkeitsäquivalente Formel zu α in k-KNF.
2: atom A
3: literal X
4: literal-list *lit-list*-1
5: literal-list *lit-list*-2
6: **for** clause $k : \alpha$ **do**
7: **if** NR-OF-LITERALS$(k) > k$ **then**
8: {Splitte *literal-list* von k nach $k-1$ Literalen in *lit-list*-1 und *lit-list*-2 auf:}
9: $A :=$ NEWATOM
10: $X :=$ NEWLITERAL$(\neg A)$
11: *lit-list*-1 := APPEND(*lit-list*-1, X)
12: $X :=$ NEWLITERAL(A)
13: *lit-list*-2 := APPEND(X, *lit-list*-2)
14: $k1 :=$ NEWCLAUSE(*lit-list*-2)
15: {Füge $k1$ nach k in *clause-list* von α ein.}

6.2 Probleme

Folgende Probleme seien als NP-vollständig bekannt:

Problem 4 (1-3-SAT, abgeändert übernommen [KL94, vgl. S. 60], NP-vollständig nach [Sch78])
Eingabe: Formel α aus 3-KNF mit nur positiven Literalen.
Frage: Existiert eine Belegung $\mathfrak{I}$, sodass pro Klausel genau ein Literal wahr wird?

Problem 5 (3-SAT, abgeändert übernommen [KL94, vgl. S. 57f.])
Eingabe: Formel α aus 3-KNF.
Frage: Ist α erfüllbar?

Literaturverzeichnis

[Asp80] Aspvall, Bengt: Recognizing Disguised NR(1) Instances of the Satisfiability Problem. In: *Journal of Algorithms* 1 (1980), S. 97–103

[BFS92] Berman, Kenneth A. ; Franco, John ; Schlipf, John S.: Unique Satisfiability for Horn Sets Can Be Solved in Nearly Linear Time / Computer Science Department, University of Cincinnati. 1992 (CS-TR-92-5). – Forschungsbericht

[DG84] Dowling, William F. ; Gallier, Jean H.: Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. In: *Journal of Logic Progamming* 1 (1984), S. 267–284

[GT85] Gabow, Harold N. ; Tarjan, Robert Endre: A Linear-Time Algorithm for a Special Case of Disjoint Set Union. In: *Journal of Computer and System Science* 30 (1985), S. 209–221

[KL94] Kleine Büning, Hans ; Lettmann, Theodor ; Appelrath, Hans-Jürgen (Hrsg.) ; Claus, Volker (Hrsg.) ; Hotz, Günter (Hrsg.) ; Waldschmidt, Klaus (Hrsg.): *Aussagenlogik: Deduktion und Algorithmen.* B.G. Teubner, 1994 (Leitfäden und Monographien der Informatik)

[Lew78] Lewis, H. R.: Renaming a Set of Clauses as a Horn Set. In: *Journal of the Association for Computing Machinery* 25 (1978), S. 134–135

[Min88] Minoux, Michel: LTUR: A Simplified Linear-Time Unit Resolution Algorithm for Horn Formulae and Computer Implementation. In: *Information Processing Letters* 29 (1988), S. 1–12

[Pre92a] Pretolani, Daniele: A Linear-Time Algorithm for the Unique Horn Satisfiability Problem / Dipartimento di Informatica, Università di Pisa. 1992 (TR 31/92). – Forschungsbericht

[Pre92b] Pretolani, Daniele: *Satisfiability and Hypergraphs.* 1992

[Sch78] Schaefer, Thomas J.: The Complexity of Satisfiability Problems. In: *Proceedings of the tenth annual ACM symposium on Theory of Computing*, 1978, S. 216–226

[Tar75] Tarjan, Robert Endre: Efficiency of a Good But Not Linear Set Union Algorithm. In: *Journal of the Association for Computing Machinery* 22 (1975), S. 215–225