KNF-DNF-Konverter

Dokumentation

Paul Staroch

12. April 2007

Inhaltsverzeichnis

1 Einleitung

Das vorliegende Programm, der *KNF-DNF-Konverter*, ist ein interaktives, in *Java*[1] geschriebenes Programm, welches im Rahmen eines Projektpraktikums bei Ao. Univ.Prof. Dr. Christian Fermüller[2] an der *Technischen Universität Wien*[3] entstanden ist. Es soll als Lehrbehelf für die Lehrveranstaltung *Theoretische Informatik und Logik*[4] dienen. Die Funktion des Programms besteht in der Umformung einer beliebigen aussagenlogischen Formel in ihre äquivalente *konjunktive Normalform* (*KNF*) und *disjunktive Normalform* (*DNF*), wobei diese Umformung mit Hilfe der *syntaktischen Methode* erfolgen soll.

In dieser Dokumentation wird zunächst auf den theoretischen Hintergrund („Was ist eine aussagenlogische Formel?“, „Was versteht man unter der KNF/DNF?“) eingegangen. Im Anschluss daran wird das Programm beschrieben; einerseits dessen Bedienung und andererseits die innere Struktur - wie aussagenlogische Formeln in einer entsprechenden Datenstruktur abgelegt werden, wie die Umformung einer solchen Formel vor sich geht und wie Formeln vereinfacht werden.

[1] `http://java.sun.com/`
[2] `http://www.logic.at/staff/chrisf`
[3] `http://www.tuwien.ac.at/`
[4] `http://www.logic.at/lvas/til/`

2 Theoretischer Hintergrund

Dieses Kapitel geht auf den theoretischen Hintergrund des *KNF-DNF-Konverters* ein.

2.1 Aussagenlogik

Die klassische Aussagenlogik beschäftigt sich mit der Verknüpfung einfacher Ja-Nein-Aussagen durch Junktoren wie „Und“, „Oder“, „Nicht“etc.

Die Syntax aussagenlogischer Formeln ist durch die Menge AF gegeben, die wie folgt induktiv definiert ist:

- *Aussagenlogische Variablen*: $AV = \left\{\underline{A}, \underline{B}, \underline{C}, \ldots, \underline{Z}\right\} \subset AF$;
- *Aussagenlogische Konstanten*: $\left\{\underline{\mathbf{t}}, \underline{\mathbf{f}}\right\} \subset AF$;
- $\underline{\neg} F \in AF$, wenn $F \in AF$;
- $\left(\underline{F \circ' G}\right) \in AF$, wenn $F, G \in AF$ und $\circ' \in \left\{\underline{\vee}, \underline{\wedge}, \underline{\subseteq}, \underline{\supseteq}, \underline{\uparrow}, \underline{\downarrow}, \underline{\not\supset}, \underline{\not\subset}, \underline{\equiv}, \underline{\not\equiv}\right\}$.

Aussagenlogische Variablen und Konstanten werden auch als *atomare aussagenlogische Formeln* oder kurz *Atome* bezeichnet. Die Elemente der oben bereits erwähnten Menge $\left\{\underline{\vee}, \underline{\wedge}, \underline{\subseteq}, \underline{\supseteq}, \underline{\uparrow}, \underline{\downarrow}, \underline{\not\supset}, \underline{\not\subset}, \underline{\equiv}, \underline{\not\equiv}\right\}$ heißen *Junktoren*, *Konnektoren* oder *logische Operatoren*.

Eine Funktion vom Typ $AV \to \left\{\underline{\mathbf{t}}, \underline{\mathbf{f}}\right\}$ nennt man *(aussagenlogische) Interpretation*. Die Menge aller Interpretationen wird mit INT bezeichnet.

Diese Semantik aussagenlogischer Formeln wird durch eine Funktion $M_{AF} : INT \times AF \to \{\mathbf{t}, \mathbf{f}\}$ bestimmt. Diese M_{AF} ist wie folgt definiert:

- $M_{AF}(I, v) = I(v)$ für $v \in AF$;
- $M_{AF}\left(I, \underline{\mathbf{t}}\right) = \mathbf{t}$ und $M_{AF}\left(I, \underline{\mathbf{f}}\right) = \mathbf{f}$
- $M_{AF}(I, \underline{\neg} A) = \neg\left(M_{AF}(I, A)\right)$ mit $A \in AF$; für die Funktion $\neg : \{\mathbf{t}, \mathbf{f}\} \to \{\mathbf{t}, \mathbf{f}\}$ gilt: $\neg\mathbf{t} = \mathbf{f}, \neg\mathbf{f} = \mathbf{t}$
- $M_{AF}\left(I, \left(\underline{A_1 \quad \circ' \quad A_2}\right)\right) = M_{AF}(I, A_1) \circ M_{AF}(I, A_2)$, wobei die zum Junktor $\circ'$ gehörende Funktion $\circ : \{\mathbf{t}, \mathbf{f}\}^2 \to \{\mathbf{t}, \mathbf{f}\}$ wie in unten stehender Tabelle definiert ist.

Die folgende Tabelle gibt eine Übersicht über alle nicht-trivialen zweistelligen aussagenlogischen Funktionen:

		$\wedge$	$\vee$	$\supset$	$\subset$	$\uparrow$	$\downarrow$	$\not\supset$	$\not\subset$	$\equiv$	$\not\equiv$
f	**f**	**f**	**f**	**t**	**t**	**t**	**t**	**f**	**f**	**t**	**f**
f	**t**	**f**	**t**	**t**	**f**	**t**	**f**	**f**	**t**	**f**	**t**
t	**f**	**f**	**t**	**f**	**t**	**t**	**f**	**t**	**f**	**f**	**t**
t	**t**	**t**	**t**	**t**	**t**	**f**	**f**	**f**	**f**	**t**	**f**

Die Semantik aussagenlogischer Formeln wurde hier lediglich eingeführt, um nun den Begriff der *Äquivalenz* zweier aussagenlogischer Formeln formulieren. Dieser Begriff ist wesentlich für das vorliegende Programm, da die im nächsten Kapitel beschriebenen Normalformen stets äquivalent zur Ausgangsformel sein müssen. Zwei Formeln $F, G \in AF$ heißen *äquivalent*, wenn $M_{AF}(I, F) = M_{AF}(I, G)$ für alle $I \in INT$. Mit diesem Wissen ist es nun möglich, eine aussagenlogische Formel F in eine andere aussagenlogische Formel G umzuformen, wobei F und G äquivalent sind.

2.2 Normalformen

Für eine Reihe von Anwendungen ist es sinnvoll und notwendig, an Stelle einer aussagenlogischen Formel eine äquivalente aussagenlogische Formel zu verwenden, die eine standardisierte, vereinfachte Form aufweist.

Ein *Literal* ist eine aussagenlogische Variable $A \in AB$ oder das Negat einer solchen. Sind $L_{i,j}$ Literale für $i = 1, \ldots, m$, $j = 1, \ldots, n$, so ist eine Formel in *disjunktiver Normalform* (*DNF*), wenn sie die Form

$$(L_{1,1} \wedge \cdots \wedge L_{1,n_1}) \vee \cdots \vee (L_{m,1} \wedge \cdots \wedge L_{m,n_m})$$

besitzt, oder in *konjunktiver Normalform* (*KNF*), wenn sie die Form

$$(L_{1,1} \vee \cdots \vee L_{1,n_1}) \wedge \cdots \wedge (L_{m,1} \vee \cdots \vee L_{m,n_m})$$

besitzt. Eine Formel in disjunktiver Normalform ist also eine Disjunktion von Konjunktionen von Literalen, eine Formel in konjunktiver Normalform eine Konjunktion von Disjunktionen von Literalen. Die Disjunktionen einer KNF werden auch *Klauseln* genannt, die KNF entsprechend *Klauselnormalform*. Spezialfälle für konjunktive beziehungsweise disjunktive Normalformen sind **t** beziehungsweise **f**.

Konjunktion und Disjunktion sind

- assoziativ: $(A \wedge B) \wedge C = A \wedge (B \wedge C)$ beziehungsweise $(A \vee B) \vee C = A \vee (B \vee C)$ für alle $A, B, C \in AF$,
- kommutativ: $A \wedge B = B \wedge A$ sowie $A \vee B = B \vee A$ für alle $A, B \in AF$ und
- idempotent: $A \wedge A = A$ beziehungsweise $A \vee A = A$ für alle $A \in AF$.

Daher ist es möglich, für Formeln in KNF beziehungsweise DNF die folgende Mengenschreibweise zu verwenden:

$$\{\{L_{1,1} \ldots L_{1,n_1}\}, \ldots, \{L_{m,1} \ldots L_{m,n_m}\}\}$$

Eine solche Literalmenge kann jedoch sowohl als DNF als auch als KNF interpretiert werden. Um Unklarheiten zu vermeiden, wollen wir diese Schreibweise ausschließlich für KNFs verwenden.

Im Folgenden wird nun die *syntaktische (algebraische) Methode* beschrieben, um zu einer beliebigen aussagenlogischen Formel die äquivalente KNF beziehungsweise DNF zu ermitteln. Eine andere Möglichkeit stellt die *semantische Methode* dar, die

eine KNF beziehungsweise DNF an Hand der Wahrheitstafel einer Formel konstruiert. Die semantische Methode ist an dieser Stelle jedoch irrelevant, da der *KNF-DNF-Konverter* ausschließlich die syntaktische Methode verwendet.

Im Folgenden stehen A, B und C für beliebige Formeln $A, B, C \in AF$.

Schritt 1: Ersetze alle Operatoren durch $\wedge$, $\vee$ sowie $\neg$:

$$\begin{array}{rcl@{\qquad}rcl} A \supset B & = & \neg A \vee B & A \not\supset B & = & A \wedge \neg B \\ A \subset B & = & A \vee \neg B & A \not\subset B & = & \neg A \wedge B \\ A \uparrow B & = & \neg A \vee \neg B & A \downarrow B & = & \neg A \wedge \neg B \\ (A \equiv B) & = & (A \wedge B) \vee (\neg A \wedge \neg B) & (A \not\equiv B) & = & (\neg A \wedge B) \vee (A \wedge \neg B) \end{array}$$

Schritt 2: Schiebe alle Negationen direkt vor die Variablen und Konstanten (die ersten beiden Umformungen sind die *De Morgan'schen Gesetze*):

$$\neg (A \wedge B) = \neg A \vee \neg B \quad \neg (A \vee B) = \neg A \wedge \neg B \quad \neg\neg A = A$$

Schritt 3: Eliminiere die Konstanten **t** und **f**; dabei darf auf die Kommutativität von Konjunktion und Disjunktion (siehe oben) nicht vergessen werden:

$$A \wedge \mathbf{t} = A \quad A \wedge \mathbf{f} = \mathbf{f} \quad A \vee \mathbf{t} = \mathbf{t} \quad A \vee \mathbf{f} = A \quad \neg\mathbf{t} = \mathbf{f} \quad \neg\mathbf{f} = \mathbf{t}$$

Schritt 4: Wende die Distributivgesetze an. Dabei führt die linke Äquivalenz zu einer DNF, die rechte zu einer KNF; wieder ist die Kommutativität von Konjunktion und Disjunktion zu beachten.

$$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C) \quad A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$$

Dieses Verfahren basiert auf drei Tatsachen:

- Alle angegebenen Gleichungen sind gültige Äquivalenzen. Das bedeutet, dass jede Formel, auf welche das Verfahren angewandt wird, äquivalent zur ursprünglichen Formel ist.
- Auf jede Formel, die noch nicht in KNF beziehungsweise DNF vorliegt, kann eine der Äquivalenzen angewandt werden.
- Es entstehen keine unendlich langen Umformungsketten, wenn die Äquivalenzen von links nach rechts angewandt werden [5].

Dadurch terminiert das Verfahren, und das Ergebnis ist eine Formel in DNF beziehungsweise KNF, die zur ursprünglichen Formel äquivalent ist.

[5] Im Programm werden die Umformungen von außen nach innen angewandt, siehe unten.

3 Das Programm

Im Folgenden soll, aufbauend auf der Beschreibung des theoretischen Hintergrundes im vorhergehenden Kapitel, das Programm beschrieben werden. Neben einer grundsätzlichen Beschreibung soll diese auch an einem Beispiel, das zuerst „am Papier“und dann mit Hilfe des Programms durchgearbeitet wird, erfolgen. Zum Abschluss wird auf einige Besonderheiten der internen Struktur des Programms eingegangen.

3.1 Bedienung

Dieses Kapitel beschäftigt sich mit der Beschreibung der Bedienung des Programms.

3.1.1 Starten

Das Programm wurde in *Java*[6] entwickelt. Zur Ausführung ist daher notwendig, dass ein *Java Runtime Environment* (Version 5.0 oder höher)[7] installiert ist. Ist diese Voraussetzung erfüllt, gibt es zwei Möglichkeiten:

- Zum einen kann das Programm als *Applet* verwendet werden. Ist im verwendeten Browser das Java-Plugin installiert, ist nichts weiter zu tun als die URL `http://stud4.tuwien.ac.at/~e0425426/praktikum/applet.html` einzugeben.
- Zum anderen kann das Programm auch als eigenständiges Programm gestartet werden. Dazu lädt man die Datei `http://stud4.tuwien.ac.at/~e0425426/praktikum/cnf-dnf.jar` herunter. Anschließend kann das Programm unter Windows durch Doppelklick auf die heruntergeladene Datei im Explorer gestartet werden. In anderen Systemen erfolgt der Aufruf üblicherweise durch Eingabe von

  ```
  java -jar cnf-dnf.jar
  ```

  in einem Terminal.

3.1.2 Umformung

Nach dem Start präsentiert sich das Programm wie in Abbildung 1 zu sehen. In das große Textfeld kann eine beliebige aussagenlogische Formel eingegeben werden. Die elf Schaltflächen unterhalb des Eingabefeldes können verwendet werden, um Funktionssymbole einzugeben, die nicht über die Tastatur eingegeben werden können.

Die Syntax für eine solche Formel orientiert sich an der in Kapitel 2.1 beschriebenen Syntax, mit folgenden Unterschieden:

[6] `http://java.sun.com/`

[7] Das JRE kann von `http://java.sun.com/javase/downloads/index.jsp` bezogen werden

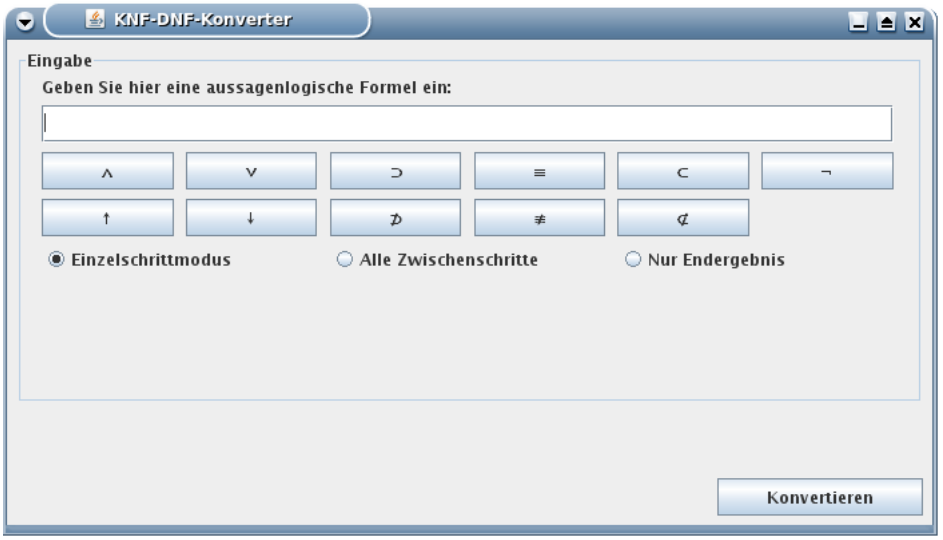

Abbildung 1: Programm nach dem Start

- Unter Berücksichtigung der Assoziativität von Konjunktion, Disjunktion, Antivalenz und Äquivalenz dürfen Klammern weggelassen werden. So kann zum Beispiel statt $(A \vee (B \vee C))$ auch $(A \vee B \vee C)$ geschrieben werden; derartige Formeln werden automatisch rechtsassoziativ geklammert, damit das Programm damit arbeiten kann.

- Die Klammern, welche die gesamte Formel umschließen, dürfen weggelassen werden ($(A \wedge B)$ wird zu $A \wedge B$). Diese Klammern werden automatisch ergänzt.

Die Grammatik, die vom Parser verwendet wird, wird in Kapitel 3.3 exakt beschrieben.

Als Variablen sind Zeichenketten erlaubt, die

- aus einem einzigen Groß-/Kleinbuchstaben bestehen, wobei `t` und `f` nicht erlaubt sind (diese Symbole sind für die Konstanten für die beiden Wahrheitswerte reserviert) oder

- aus mehreren Groß-/Kleinbuchstaben und Ziffern bestehen, wobei das erste Zeichen keine Ziffer sein darf.

Sonderzeichen, deutsche Umlaute und dergleichen sind nicht erlaubt.

Zulässige Variablen sind also beispielsweise `A`, `x`, `variable` oder `Irgendwas`. Nicht erlaubt sind hingegen zum Beispiel `(abc)`, `t` oder `a_b`.

Bei Eingaben, die nicht als zulässige aussagenlogische Formeln erkannt werden, wird der Hintergrund des Eingabefeldes rötlich eingefärbt. Klickt man auf die Schaltfläche „Konvertieren“, wird im unteren Bereich des Fensters eine Meldung angezeigt, die darauf hinweist, was der Eingabe fehlt, um vom Programm als aussagenlogische Formel gewertet zu werden.

Wenn die Einfügemarke hinter einer öffnenden oder schließenden Klammer positioniert wird, werden im Eingabefeld werden zusammengehörende Klammern markiert.

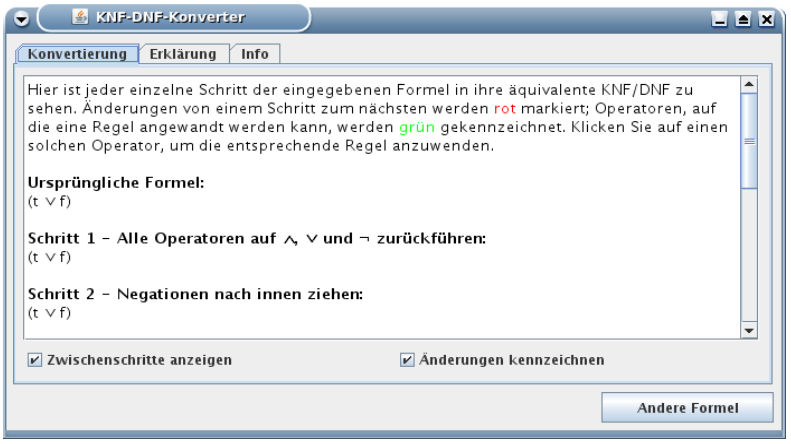

Abbildung 2: Ergebnisansicht

Unterhalb der Schaltflächen für die Junktoren kann man einen von drei Modi auswählen, den das Programm zur Umformung der eingegebenen Formel verwenden soll (die Bezeichnungen sollten ohnehin selbsterklärend sein):

- *Einzelschrittmodus*: Der Benutzer muss jeden einzelnen Schritt, der durchgeführt werden soll, selbst auswählen (siehe unten).
- *Alle Zwischenschritte*: Die gesamte Umformung wird automatisch vom Programm durchgeführt. Dabei werden die Regeln von außen nach innen angewandt, das heißt, im Ausdruck $(A\ f\ B)$ mit $f \in \{\wedge, \vee, \supset, \subset, \uparrow, \downarrow, \not\supset, \not\subset, \equiv, \not\equiv\}$ wird zuerst auf den gesamten Ausdruck eine Regel angewandt (beispielsweise die Ersetzung von $(A \uparrow B)$ durch $(\neg A \vee \neg B)$); erst dann werden die beiden durch die Funktion verknüpften Formeln auf mögliche Regelanwendungen untersucht, wobei von links nach rechts vorgegangen wird.
- *Nur Endergebnis*: Für jeden der in Kapitel 2.2 beschriebenen Schritte wird nur jeweils das Endergebnis angegeben, die Zwischenschritte nicht.

Durch Auswahl der Schaltfläche „Konvertieren“ wird die Ansicht gewechselt (siehe Abbildung 2). Nun sind drei Tabs zu sehen:

- *Konvertierung*: Hier wird die Umformung der eingegebenen Formel in die äquivalente KNF beziehungsweise DNF durchgeführt. Darüber hinaus werden die so erhaltenen Formeln nach Möglichkeit (des Programms) vereinfacht (siehe Kapitel 3.3 für Details).
- *Erklärung*: In diesem Tab werden die Schritte, welche im Tab *Konvertierung* zu sehen sind, im Detail erklärt. Werden im Tab *Konvertierung* die Zwischenschritte nicht angezeigt, sondern nur die Endergebnisse, so werden hier keine Erklärungen angezeigt.
- *Info*: In diesem Tab ist eine kurze Information zum Programm zu finden.

Werden alle Zwischenschritte angezeigt, so werden die Funktionssymbole an Stellen, an denen eine Regelanwendung möglich ist, in grüner Farbe angezeigt. Klicken

Sie darauf, um die entsprechende Regel anzuwenden. Alle bereits vorhandenen Umformungsschritte ab jener Formel, die angeklickt wird, werden dabei verworfen. Junktoren an Stellen, an denen eine Regel angewandt wurde, um zum nächsten Umformungsschritt zu kommen, werden grau unterlegt.

Im Einzelschrittmodus werden Links, die dem Beginnen eines Schritts, der Anwendung eines Vereinfachungsschrittes beziehungsweise der Transformation der KNF in Klauselform und wieder zurück dienen, in blauer Farbe dargestellt.

Zusammengehörende Klammern werden auch hier markiert, wenn die Einfügemarke hinter einer öffnenden oder schließenden Klammer positioniert wird.

Mit Hilfe des Kontrollkästchens „Zwischenschritte anzeigen“kann festgelegt werden, ob alle Zwischenschritte der Umformung dargestellt werden oder nicht. Diese Option kann nicht verändert werden, wenn der Einzelschrittmodus aktiv ist.

Die Änderungen von einem Einzelschritt zum nächsten werden angezeigt, wenn das Kontrollkästchen „Änderungen kennzeichnen“ aktiviert ist. Werden keine Einzelschritte angezeigt, ist diese Option nicht verfügbar.

Möchte man eine andere Formel eingeben (oder die derzeit verwendete Formel verändern), klickt man einfach auf die Schaltfläche „Andere Formel“ und kommt zurück zur Eingabemaske.

3.2 Ein Beispiel

Im Folgenden sollen nun zur Formel

$$F = ((\mathbf{t} \uparrow \neg\neg C) \vee (A \not\supset B)) \supset \neg(\neg\neg B \downarrow \neg A)$$

die äquivalente konjunktive beziehungsweise disjunktive Normalform gefunden werden. Das Ergebnis soll nach Möglichkeit vereinfacht werden.

3.2.1 „Am Papier“

Wie in Kapitel 2.2 beschrieben, gehen wir in vier Schritten vor.

Schritt 1: Alle Operatoren auf $\wedge$, $\vee$ und $\neg$ zurückführen:

$$\begin{aligned}
F &= ((\mathbf{t} \uparrow \neg\neg C) \vee (A \not\supset B)) \supset \neg(\neg\neg B \downarrow \neg A) \\
&= \neg((\mathbf{t} \uparrow \neg\neg C) \vee (A \not\supset B)) \vee \neg(\neg\neg B \downarrow \neg A) \\
&= \neg((\neg\mathbf{t} \vee \neg\neg\neg C) \vee (A \not\supset B)) \vee \neg(\neg\neg B \downarrow \neg A) \\
&= \neg((\neg\mathbf{t} \vee \neg\neg\neg C) \vee (A \wedge \neg B)) \vee \neg(\neg\neg B \downarrow \neg A) \\
&= \neg((\neg\mathbf{t} \vee \neg\neg\neg C) \vee (A \wedge \neg B)) \vee \neg(\neg\neg\neg B \wedge \neg\neg A)
\end{aligned}$$

Schritt 2: Negationen nach innen ziehen:

$$
\begin{aligned}
&= (\neg(\neg\mathbf{t} \vee \neg\neg\neg C) \wedge \neg(A \wedge \neg B)) \vee \neg(\neg\neg\neg B \wedge \neg\neg A)\\
&= ((\neg\neg\mathbf{t} \wedge \neg\neg\neg\neg C) \wedge \neg(A \wedge \neg B)) \vee \neg(\neg\neg\neg B \wedge \neg\neg A)\\
&= ((\mathbf{t} \wedge \neg\neg\neg\neg C) \wedge \neg(A \wedge \neg B)) \vee \neg(\neg\neg\neg B \wedge \neg\neg A)\\
&= ((\mathbf{t} \wedge \neg\neg C) \wedge \neg(A \wedge \neg B)) \vee \neg(\neg\neg\neg B \wedge \neg\neg A)\\
&= ((\mathbf{t} \wedge C) \wedge \neg(A \wedge \neg B)) \vee \neg(\neg\neg\neg B \wedge \neg\neg A)\\
&= ((\mathbf{t} \wedge C) \wedge (\neg A \vee \neg\neg B)) \vee \neg(\neg\neg\neg B \wedge \neg\neg A)\\
&= ((\mathbf{t} \wedge C) \wedge (\neg A \vee B)) \vee \neg(\neg\neg\neg B \wedge \neg\neg A)\\
&= ((\mathbf{t} \wedge C) \wedge (\neg A \vee B)) \vee (\neg\neg\neg\neg B \vee \neg\neg\neg A)\\
&= ((\mathbf{t} \wedge C) \wedge (\neg A \vee B)) \vee (\neg\neg B \vee \neg\neg\neg A)\\
&= ((\mathbf{t} \wedge C) \wedge (\neg A \vee B)) \vee (B \vee \neg\neg\neg A)\\
&= ((\mathbf{t} \wedge C) \wedge (\neg A \vee B)) \vee (B \vee \neg A)
\end{aligned}
$$

Schritt 3: Konstanten eliminieren:

$$= (C \wedge (\neg A \vee B)) \vee (B \wedge \vee A)$$

Schritt 4 (DNF): Ausdistributieren

$$= ((C \wedge \neg A) \vee (C \wedge B)) \vee (B \vee \neg A)$$

Dies ist nun eine Formel in disjunktiver Normalform. Unter Zuhilfenahme einiger aussagenlogischer Rechenregeln lässt sich diese Formel in die äquivalente Form $\neg A \wedge B$ umformen.

Schritt 4 (KNF): Ausdistributieren

$$= (C \vee (B \vee \neg A)) \wedge ((\neg A \vee B) \vee (B \vee \neg A))$$

Dies ist nun eine Formel in konjunktiver Normalform. Durch eine sture Konvertierung in (Multi-)Mengennotation erhalten wir

$$\{\{C, B, \neg A\}, \{\neg A, B, B, \neg A\}\}$$

Nach Duplikatelimination erhalten wir „echte“ Mengen:

$$\{\{C, B, \neg A\}, \{\neg A, B\}\}$$

Tautologien (Klauseln, die sowohl eine Variable als auch ihre Negation enhalten) gibt es nicht, dadurch ändert sich durch einen entsprechenden Vereinfachungsschritt nichts. Jedoch ist $\{\neg A, B\}$ eine Untermenge von $\{C, B, \neg A\}$, was bedeutet, dass $\{C, B, \neg A\}$ von $\{\neg A, B\}$ subsumiert wird. Übrig bleibt die Menge

$$\{\{\neg A, B\}\}$$

Sortieren (siehe Kapitel 3.3) ändert auch nichts. Nun haben wir eine möglichst einfache Klauselform, die wir wieder als Formel anschreiben wollen. Wir erhalten schließlich die Formel:

$$\neg A \vee B$$

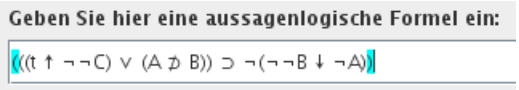

Abbildung 3: Eingabe der Formel

3.2.2 Mit Hilfe des Programms

Und nun wollen wir zum Vergleich diese Umformung mit Hilfe des *CNF-DNF-Konverters* durchführen. Wir geben die gegebene Formel in das Textfeld ein (siehe Abbildung 3) ein und klicken auf die Schaltfläche „Konvertieren“. In den Abbildungen 4 bis 9 sind die Ausgaben des Programms zu sehen. Wir erkennen: Das Programm kommt zu den selben Ergebnissen wie sie in Kapitel 3.2 zu finden sind.

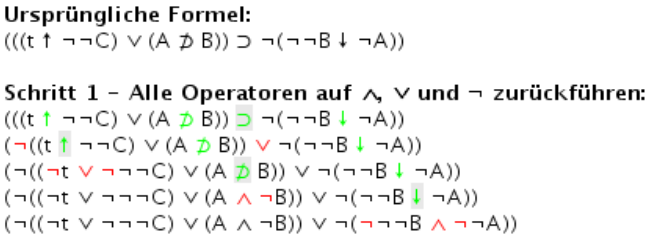

Abbildung 4: Schritt 1

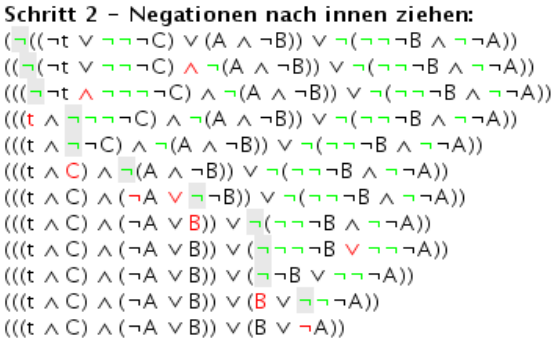

Abbildung 5: Schritt 2

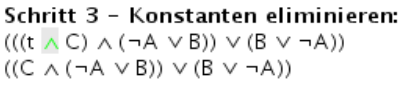

Abbildung 6: Schritt 3

3.3 Interne Funktionsweise

Dieses Kapitel befasst sich ein wenig mit der internen Struktur des *CNF-DNF-Konverters* und damit, wie dieser mit aussagenlogischen Formeln arbeitet.

Schritt 4 (DNF) - Ausdistributieren:
((C ∧ (¬A ∨ B)) ∨ (B ∨ ¬A))
(((C ∧ ¬A) ∨ (C ∧ B)) ∨ (B ∨ ¬A))

Vereinfachte Form: ¬A ∨ B

Abbildung 7: Schritt 4 (DNF)

Schritt 4 (KNF) - Ausdistributieren:
((C ∧ (¬A ∨ B)) ∨ (B ∨ ¬A))
((C ∨ (B ∨ ¬A)) ∧ ((¬A ∨ B) ∨ (B ∨ ¬A)))

Abbildung 8: Schritt 4 (KNF)

Klauselform: {{C, B, ¬A}, {¬A, B, B, ¬A}}
Klauselform ohne Duplikate: {{C, B, ¬A}, {¬A, B}}
Klauselform ohne Tautologien: {{C, B, ¬A}, {¬A, B}}
Klauselform nach Entfernung subsumierter Klauseln: {{¬A, B}}
Sortierte Klauselform: {{¬A, B}}
Vereinfachte Formel in KNF: ¬A ∨ B

Abbildung 9: Klauselform und Vereinfachungen

Abbildung 10 zeigt die Klassenstruktur, die zur Speicherung aussagenlogischer Formeln in einer leicht verarbeitbaren Form verwendet wird. Jede aussagenlogische Formel wird als Instanz der Klasse `Formula` gespeichert. `Formula` bietet unter anderem Funktionen, um

- eine Formel schrittweise in ihre äquivalente KNF und DNF umzuformen,
- zu prüfen, ob noch ein bestimmter Umformungsschritt möglich ist oder
- aussagenlogische Formeln unter einer gegebenen Interpretation auszuwerten.

Um aus einer Texteingabe eine Instanz von `Formula` zu erhalten, verwendet der KNF-DNF-Konverter eine Kombination von Scanner (JFlex (JFlex [8])
sowie Parser (CUP[9]).

Die Grammatik, auf welche bereits in den vorhergehenden Kapiteln Bezug genommen wurde, ist die Grammatik $G =< V, T, P, S >$, die in Abbildung 11 beschrieben wird.

Neben der Datenstruktur, die auf `Formula` aufbaut, gibt es weiters Datenstrukturen, die eine Formel in KNF beziehungsweise DNF in Mengennotation repräsentieren. Instanzen von Subklassen von `Formula`, die das Interface `ClauseConvertible` oder `DNFClauseConvertible` implementieren, können, sofern es sich bei dieser Instanz um die Repräsentation einer Formel in KNF oder DNF handelt, in Mengennotation übertragen werden. Eine Formel in Mengennotation kann auch wieder in eine aussagenlogische Formel in „herkömmlicher Darstellung“ konvertiert werden.

Das Besondere an dieser Mengennotation im *CNF-DNF-Konverter* ist einerseits, dass sie nicht nur, wie im Skriptum zur Lehrveranstaltung *Theoretische Informatik*

[8] `http://www.jflex.de/`
[9] `http://www2.cs.tum.edu/projects/cup/`

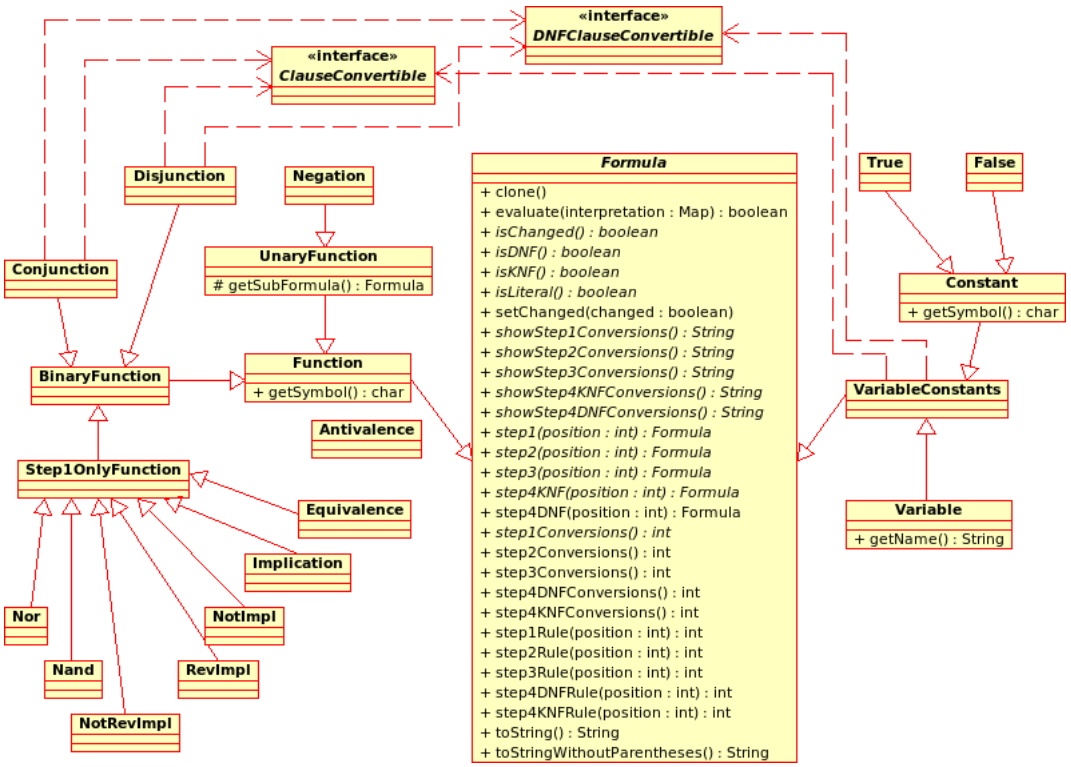

Abbildung 10: Klassendiagramm der Datenstruktur für aussagenlogische Formeln

und Logik, für Formeln in KNF, sondern auch für Formeln in DNF Verwendung findet. Der Sinn dahinter ist jener, dass in Mengennotation Vereinfachungen der Formel leichter möglich sind. Dazu wird die Formel zunächst in Mengennotation übertragen, dann werden die unten beschriebenen Vereinfachungen durchgeführt und die Formel wird wieder in „normale Darstellung„ übertragen.

Folgende Vereinfachungsschritte werden dabei vom Programm durchgeführt[10]:

- Eliminieren von Duplikaten (aus Multimengen werden „echte“ Mengen): Aus $\{A, A, \neg A, B\}$ wird $\{A, \neg A, B\}$, aus $\{\{A\}, \{A\}, \{B\}\}$ wird $\{\{A\}, \{B\}\}$.

- Eliminieren von Tautologien (KNF): Unter der Annahme, dass es sich bei der Klauselmenge $\{\{A, \neg A\}, \{B\}\}$ um die Repräsentation einer Formel in KNF handelt, entspricht diese Menge der aussagenlogischen Formel $(A \vee \neg A) \wedge B$. Die Teilformel $A \vee \neg A$ wertet unter jeder Interpretation zum Wahrheitswert $\mathbf{t}$ aus; eine solche Formel nennt man *Tautologie*. Da weiters $\mathbf{t} \wedge A = A$ gilt für jede aussagenlogische Formel A, ist $(A \vee \neg A) \wedge B$ äquivalent zu B. Jede Klausel, die sowohl eine Variable als auch eine Negation enthält, repräsentiert eine Tautologie und kann daher weggestrichen werden.

- Eliminieren von Kontradiktionen (DNF): Repräsentiert eine Menge von Literalmengen eine aussagenlogische Formel in DNF, so kann man die Menge

[10]Die Erklärungen, die im Folgenden als Begründungen für die Korrektheit der einzelnen Umformungsschritte gegeben werden, sind rein informativer Natur und nicht als formale Beweise anzusehen.

$$
\begin{array}{lcll}
V & = & \{formel, funktion, subformel, undkette, oderkette, xorkette, \\
 & & nexorkette, variable, buchstabe\} \\
T & = & \{(,), \wedge, \vee, \supset, \subset, \uparrow, \downarrow, \not\supset, \not\subset, \equiv, \not\equiv, A, \ldots, Z, a \ldots, z, 0, \ldots, 9\} \\
P & = & \{formel & \Rightarrow & funktion \\
 & & & | & subformel, \\
 & & funktion & \Rightarrow & subformel \supset subformel \\
 & & & | & subformel \subset subformel \\
 & & & | & subformel \not\supset subformel \\
 & & & | & subformel \not\subset subformel \\
 & & & | & subformel \uparrow subformel \\
 & & & | & subformel \downarrow subformel \\
 & & & | & subformel \wedge undkette \\
 & & & | & subformel \vee oderkette \\
 & & & | & subformel \equiv nexorkette \\
 & & & | & subformel \not\equiv xorkette \\
 & & subformel & \Rightarrow & variable \\
 & & & | & \mathbf{t} \\
 & & & | & \mathbf{f} \\
 & & & | & (subformel) \\
 & & & | & \neg subformel \\
 & & & | & (funktion) \\
 & & variable & \Rightarrow & \texttt{([A-Za-eg-su-z]|[A-Za-z][A-Za-z0-9]+)} \\
 & & undkette & \Rightarrow & subformel \wedge undkette \\
 & & & | & variable \\
 & & oderkette & \Rightarrow & subformel \vee oderkette \\
 & & & | & variable \\
 & & xorkette & \Rightarrow & subformel \not\equiv xorkette \\
 & & & | & variable \\
 & & nexorkette & \Rightarrow & subformel \equiv nexorkette \\
 & & & | & variable\} \\
S & = & formel
\end{array}
$$

Abbildung 11: Vom Parser verwendete Grammatik $G =< V, T, P, S >$

$\{\{A, \neg A\}, \{B\}\}$ als $(A \wedge \neg A) \vee B$ anschreiben. Hier wertet $A \wedge \neg A$ immer zum Wahrheitswert $\mathbf{f}$ aus; eine solche Formel wird als *Kontradiktion* bezeichnet. Außerdem gilt $\mathbf{f} \vee A = A$ für jede aussagenlogische Formel A, daher ist $(A \wedge \neg A) \vee B$ äquivalent zur Formel B. Wie bei der KNF kann daher jede Literalmenge, die sowohl ein Literal sowie dessen Negation enthält, weggestrichen werden, da diese Menge eine Kontradiktion darstellt.

- Subsumtion: Seien A, B und C beliebige aussagenlogische Formeln. Wertet unter einer beliebigen Interpretation $A \vee B$ zu $\mathbf{t}$ aus, so ist auch $A \vee B \vee C = \mathbf{t}$ (wegen $\mathbf{t} \vee C = \mathbf{t}$). Das bedeutet, dass $(A \vee B) \wedge (A \vee B \vee C)$ zu $(A \vee B)$ vereinfacht werden kann. Selbiges gilt für die Konjunktion: Wertet $A \wedge B$ zu $\mathbf{f}$ aus, so

auch $A \wedge B \wedge C$ (wegen $\mathbf{f} \wedge C = \mathbf{f}$). Daher kann $(A \wedge B) \vee (A \wedge B \wedge C)$ zu $(A \wedge B)$ vereinfacht werden.
In Mengenschreibweise bedeutet das (unabhängig davon, ob diese Menge von Literalmengen eine Formel in KNF oder eine Formel in DNF repräsentiert), dass $\{\{A, B\}, \{A, B, C\}\}$ zu $\{\{A, B\}\}$ vereinfacht werden kann. Allgemeiner ausgedrückt: Eine Literalmenge D subsumiert eine andere Literalmenge E, falls D eine Teilmenge von E ist. Die Menge E kann aus dem Ausdruck gestrichen werden, ohne den Wert des Ausdrucks unter irgendeiner Interpretation zu verändern.

- Sortieren: Dieser Schritt vereinfacht die Mengen nicht, sondern sorgt lediglich für eine einheitliche Darstellung. Sortiert werden zunächst die einzelnen Literale innerhalb der Literalmengen und anschließend die Literalmengen selbst. Die Literale werden aufsteigend nach dem vorkommenden Variablennamen sortiert; kommen sowohl eine Variable als auch deren Negation in dem Ausdruck vor (was nach Entfernung von Tautologien beziehungsweise Kontradiktionen ohnehin nicht der Fall ist), so kommt das Literal, das keine Negation enthält, vor dem Literal, welches die Negation enthält.
  Die Literalmengen werden aufsteigend nach dem Namen der im ersten Literal vorkommenden Variable sortiert. Ist dieser Name gleich, so kommt die Menge, in welcher das erste Literal eine nicht-negierte Variable ist, vor der Menge, in welcher das erste Literal eine negierte Variable ist. Ist dadurch keine eindeutige Sortierreihenfolge gegeben, werden die zweiten Literale der beiden Mengen verglichen. Ist eine Menge Untermenge einer anderen Menge (sollte nach Entfernung subsumierter Formeln im vorhergehenden Schritt ohnehin nicht mehr vorkommen), so kommt zuerst die Untermenge und dann die Obermenge.

Im Programm werden die Ergebnisse dieser Umformungsschritte nur für Formeln in KNF im Detail beschrieben, für Formeln in DNF bekommt der Benutzer lediglich die vereinfachte Form in „herkömmlicher Darstellung“ zu sehen. Der Grund dafür ist jener, dass die Mengennotation in der Lehrveranstaltung *Theoretische Informatik und Logik* nur für aussagenlogische Formeln in KNF verwendet wird und daher die Teilnehmer dieser Lehrveranstaltung nicht durch die Verwendung der Mengennotation für Formeln in DNF im Programm verwirrt werden sollen.

A Verwendete Software

Im Folgenden ist eine Auflistung der Software zu finden, die bei der Entwicklung verwendet wurde. Da diese Software (bis auf das Java Runtime Environment) für die Ausführung des *KNF-DNF-Konverters* nicht notwendig ist und daher nicht weitergegeben werden muss, erfolgt lediglich ein Hinweis darauf.

- Java (`http://java.sun.com/`) - Java ist ein Warenzeichen von Sun Microsystems, Inc.
- CUP Parser Generator for Java (`http://www2.cs.tum.edu/projects/cup/`).
- JFlex - The Fast Scanner Generator for Java (`http://jflex.de/`) - lizenziert unter der GPL. (http://www.gnu.org/licenses/gpl.txt).
- JUnit, Testing Resources for Extreme Programming (`http://www.junit.org/`) - lizenziert unter der Common Public Licence (`http://junit.sourceforge.net/cpl-v10.html`).
- ant_latex (`http://antlatex.berlios.de/`) - Lizenz: `http://antlatex.berlios.de/index.html#license`.
- Checkstyle (`http://checkstyle.sourceforge.net/`) - lizenziert unter der LGPL (`http://www.gnu.org/licenses/lgpl.html`).
- Cobertura (`http://cobertura.sourceforge.net/`) - lizenziert teilweise unter der GPL (`http://www.gnu.org/licenses/gpl.txt`), teilweise unter der Apache Software Licence (`http://www.apache.org/licenses/`).
- PMD (`http://pmd.sourceforge.net/`) - lizeniziert unter einer „BSD-style licence“(`http://pmd.sourceforge.net/license.html`)
- log4j (`http://logging.apache.org/log4j/docs/`) - lizenziert unter der Apache Software Licence (`http://www.apache.org/licenses/`).
- JDepend (`http://clarkware.com/software/JDepend.html`) - lizenziert unter der BSD-Lizenz (`http://www.clarkware.com/software/license.txt`).
- ANTLR (`http://www.antlr.org/`) - lizenziert unter der BSD-Lizenz (`http://www.antlr.org/license.html`).
- Jakarta Commons (`http://jakarta.apache.org/commons/`) - lizenziert unter der Apache Software Licence (`http://www.apache.org/licenses/`).