Grundlagen der Theoretischen Informatik

Sebastian Iwanowski
FH Wedel

Kap. 2: Logik, Teil 2.1: Aussagenlogik

Aussagenlogik

Wdh.: Operatoren zwischen Aussagen

Durch Operatoren werden aus alten Aussagen neue Aussagen geschaffen:

Einstelliger Operator:

- **Negation (¬)**

Zweistellige Operatoren:

- **Konjunktion (∧)**
- **Disjunktion (∨)**
- **Implikation (→)**
- **Äquivalenz (↔)**

Wahrheitswerte für die neuen Aussagen:

p	q	¬p	p ∧ q	p ∨ q	p → q	p ↔ q
w	w	f	w	w	w	w
w	f	f	f	w	f	f
f	w	w	f	w	w	f
f	f	w	f	f	w	w

Aussagenlogik

Achtung: Warum einfach, wenn es auch kompliziert geht ?

Folgende Notationen sind in der Literatur ebenfalls gebräuchlich:

für logische Schlussregeln: $\dfrac{p\ ,\ q}{r}$ *ist dasselbe wie* $p \wedge q \Rightarrow r$

In der Logik benutzt man unterschiedliche Symbole für Schlussfolgerungen und Äquivalenzen je nach Anwendungsgebiet:

$\vdash$, $\models$, $\prec$, $\rightarrow$ *entsprechen der logischen Implikation:* $\Rightarrow$

$\dashv\vdash$, = , $\equiv$, $\prec\succ$, $\leftrightarrow$ *entsprechen der logischen Äquivalenz:* $\Leftrightarrow$

Aussagenlogische Formeln

- **Da Aussagen wegen der Unteilbarkeit sinnvollerweise mit Buchstaben abgekürzt werden, nennt man sie auch *Literale***

- **Eine aussagenlogische Formel ist eine Verknüpfung von endlich vielen Literalen mit logischen Operatoren.**

  - Die Literale einer Formel entsprechen Variablen, die mit w und f belegt werden können.
  - Formeln können auch Konstante enthalten:

    Die Konstante $\top$ ist immer wahr und das neutrale Element bezüglich der Konjunktion.

    Die Konstante $\bot$ ist immer falsch und das neutrale Element bezüglich der Disjunktion.

- **Eine Belegung einer Formel ist eine Zuweisung von Wahrheitswerten an die Literale derart, dass dieselben Literale immer denselben Wahrheitswert erhalten.**

  Die Formel als ganze bekommt durch die Belegung ebenfalls einen Wahrheitswert.

Aussagenlogische Formeln

- **Eine Formel heißt erfüllbar, wenn es eine Belegung gibt derart, dass die Formel den Wahrheitswert w hat.**

  Eine Formel, in der jeder Literal höchstens einmal vorkommt, ist immer erfüllbar!

  Eine Formel, in der keine Negation vorkommt, ist immer erfüllbar!

  ⇒ **Nur Formeln, die einen Literal mehrfach und mindestens eine Negation enthalten, könnten unerfüllbar sein.**

- **Eine Formel heißt Tautologie oder gültig, wenn sie bei jeder Belegung den Wahrheitswert w hat.**

- **Eine Formel heißt widersprüchlich, wenn sie bei keiner Belegung den Wahrheitswert w hat.**

Aussagenlogische Formeln

Erfüllbarkeitsproblem (Satisfiability, SAT):

Wie bekommt man heraus, ob eine gegebene Formel erfüllbar ist ?

Normalformen

Definition:

Eine aussagenlogische Formel ist in **konjunktiver Normalform (KNF)**, wenn sie als Konjunktion von Disjunktionen aus Aussagen oder Negationen von Aussagen dargestellt ist.

Etwas langsamer zum Mitdenken:

Ein **Literal** ist eine Aussage oder die Negation einer Aussage.

Beispiele: p ¬q r ¬r

Eine **Klausel** ist eine Disjunktion aus Literalen.

Beispiele: p ∨ ¬q ∨ r ¬p ∨ ¬q ∨ ¬r p ∨ q

Eine **Formel in KNF** ist eine Konjunktion von Klauseln.

Beispiel: (p ∨ ¬q ∨ r) ∧ (¬p ∨ ¬q ∨ ¬r) ∧ (p ∨ q)

Normalformen

Definition:

Eine aussagenlogische Formel ist in **konjunktiver Normalform (KNF)**, wenn sie als Konjunktion von Disjunktionen aus Aussagen oder Negationen von Aussagen dargestellt ist.

Spezialfälle:

Eine **Klausel** darf auch nur aus einem Literal bestehen.

Beispiel: $(p \vee \neg q \vee r) \wedge \neg p \wedge (p \vee q)$ ist auch in KNF

Eine **Formel** darf auch nur aus einer Klausel bestehen.

Beispiel: $p \vee \neg q \vee r$ ist auch in KNF

Eine **leere Klausel** entspricht dem neutralen Element der Disjunktion: $\bot$

Beispiel: $(p \vee \neg q \vee r) \wedge \bot \wedge (p \vee q)$ ist auch in KNF

Eine **leere Formel** entspricht dem neutralen Element der Konjunktion: $\top$

Beispiel: $\top$ ist auch in KNF

Normalformen

Welche Formeln sind eigentlich nicht in KNF ?

- Formeln, die andere als die 3 Booleschen Operatoren enthalten

  Beispiel: $((p \vee \neg q \vee r) \rightarrow (\neg p \vee \neg q \vee \neg r)) \leftrightarrow (p \vee q)$

- Formeln, in denen andere Terme als atomare Aussagen negiert werden

  Beispiel: $\neg(p \vee \neg q \vee r) \wedge (\neg p \vee \neg q \vee \neg r) \wedge \neg(p \vee q)$

- Formeln, in denen Konjunktionen und Disjunktionen wild durcheinander sind

  Beispiel: $(p \wedge (\neg q \vee r)) \wedge (\neg p \vee \neg q \vee \neg r) \vee p \wedge q$

  Beispiel: $p \wedge (\neg q \vee r \wedge \neg p) \vee (\neg q \wedge \neg r) \vee p \wedge q$

- Formeln, in denen eine tiefere Klammerschachtelungstiefe vorliegt

  Beispiel: $p \wedge (\neg q \vee (r \wedge q)) \wedge (\neg p \vee \neg q \vee \neg r) \wedge p \vee q$

Satz: **Jede aussagenlogische Formel lässt sich durch endlich viele äquivalente Umformungen in KNF bringen.**

Normalformen

Warum wollen wir Formeln in KNF bringen ?

- **Übersichtlichere Auswertung beim Erfüllbarkeitstest:**

  Eine Formel in KNF ist erfüllbar.
  ⇔ Eine Belegung enthält für jede Klausel wenigstens einen wahren Literal.

  Mögliche Belegungsstrategie:

  - Gehe die Klauseln nacheinander durch:
  - Belege genau einen noch nicht festgelegten Literal mit **`t`**

    (dadurch werden gleiche Literale oder deren Negationen in anderen Klauseln festgelegt)
  - Wenn es keine Möglichkeit mehr gibt, springe zurück zur vorigen Klausel und nimm eine andere Belegung

Vorsicht vor Illusionen:

Im schlechtesten Fall bringt das keinen Zeitgewinn verglichen mit purem Ausprobieren !

Normalformen

Warum wollen wir Formeln in KNF bringen ?

- **Kompakte Darstellbarkeit im Computer:**

  Stelle Klauseln als Mengen von Literalen dar:

  {p,¬q, r} entspricht (p v ¬q v r) **als Klausel**

  Stelle Formeln als Mengen von Klauseln dar:

  {{p,¬q, r} , {¬p,¬q,¬r} , {p, q}} entspricht (p v ¬q v r) ∧ (¬p v ¬q v ¬r) ∧ (p v q)

  Das funktioniert sogar für die Spezialfälle:

  {{p,¬q, r}} entspricht (p v ¬q v r) **als Formel**

  {{p},{¬q},{r}} entspricht (p ∧ ¬q ∧ r) **als Formel**

  {} entspricht ⊤ **als Formel**

  {{}} entspricht ⊥ **als Formel**

Warnung:

Was dem Computer glasklar ist, kann für den Menschen höchst verwirrend sein:

- Das Trennzeichen (,) in **inneren** Klammern (Klauseln) entspricht einer Disjunktion (v)
- Das Trennzeichen (,) in **äußeren** Klammern (Formeln) entspricht einer Konjunktion (∧)

Normalformen

Algorithmus zum Umwandeln einer Formel in die KNF:

1. Eliminiere alle Operatoren der Form ↔ und → mit den Ersetzungsregeln durch ∧ und ∨ !

2. Ziehe alle Negationszeichen vor Klammern in die Klammern hinein mit den deMorganschen Regeln !

3. Wende die Distributivgesetze so lange an, bis auf oberster Ebene nur noch Konjunktionen und darunter Disjunktionen sind !

Das funktioniert immer !

Andere Normalformen

Definition:

Eine aussagenlogische Formel ist in **disjunktiver Normalform (DNF)**, wenn sie als Disjunktion von Konjunktionen aus Aussagen oder Negationen von Aussagen dargestellt ist.

Jetzt fällt das Mitdenken schon leichter:

Was ist in DNF ein **Literal** ?

Was ist in DNF eine **Klausel** ?

Was ist in DNF eine **Formel** ?

Beispiele ?

Warum wollen wir Formeln in DNF bringen ?

wollen wir nicht: Eine Normalform reicht uns aus !

Zusammenfassung: Aussagenlogische Formeln

Formeln haben logische Eigenschaften (Semantik):

- **Erfüllbarkeit**
- **Widerspruch** äquivalent zu $\bot$ } komplementär zueinander
- **Tautologie** äquivalent zu $\top$ } komplementär zueinander
- ***Falsifizierbarkeit***

Formeln haben Darstellungseigenschaften (Syntax):

- **Normalformen (KNF oder DNF)**
- **Mengendarstellung der KNF**

Die eben genannten Darstellungseigenschaften sind für jede Formel durch Äquivalenz erzielbar.

Es gibt auch Darstellungseigenschaften, die nur für bestimmte Formeln möglich sind.

Bsp.: KNF mit Klauseln aus genau drei Literalen

Anwendung: Logische Programmierung

Konstruktionsaufgabe: ***nicht berechenbar im Allgemeinen !***

Gegeben eine Menge $\mathcal{F}$ von logischen Formeln. Bestimme alle Formeln F, die aus $\mathcal{F}$ folgen.

Verifikationsaufgabe: ***auch nicht berechenbar im Allgemeinen!***

Gegeben eine Menge $\mathcal{F}$ von logischen Formeln und eine (neue) logische Formel F.
Berechne, ob F aus $\mathcal{F}$ folgt.

Äquivalente Formulierungen zur Verifikationsaufgabe:

1) Gegeben eine Menge $\mathcal{F}$ von logischen Formeln und eine (neue) logische Formel F. Berechne, ob die Formelmenge $\{\neg F\} \cup \mathcal{F}$ widersprüchlich ist.
2) Gegeben eine Menge $\mathcal{F}$ von logischen Formeln. Berechne, ob sie widersprüchlich ist.

Weitere Möglichkeit der Vereinfachung des Problems:

Schränke die zulässigen Formeln ein!

Konkret: Widerspruchsfindung mittels Resolution

Aufgabe:

Gegeben eine Menge $\mathcal{F}$ von logischen Formeln. Berechne, ob sie widersprüchlich ist.

Methode:

Äquivalente Formelumformungen: Ziel ist es, die Konstante $\perp$ herzuleiten.

Wir arbeiten so lange mit allgemeinen Formeln, bis es nicht mehr weitergeht!

Resolutionsprinzip:

Generierung einer neuen Formel als Folgerung aus 2 gegebenen Formeln

Prinzip: Finde Literal c, der in den Formeln a ∨ c und b ∨ ¬c vorkommt.

Dann kann c **eleminiert** werden: $(a \vee c) \wedge (b \vee \neg c) \rightarrow (a \vee b)$

Die neue Formel heißt **Resolvente** der alten Formeln.

Durch eine solche Eliminierung können einzelne Literale isoliert werden:

Bsp.: $(a \vee c) \wedge \neg c \rightarrow a$ Interpretation: a muss in der Formelsammlung gelten.

Wenn auf diese Weise auch die Negation isoliert wird, ergibt sich ein Widerspruch: ***Widerspruch!***

Bsp.: $(\neg a \vee d) \wedge \neg d \rightarrow \neg a$ Interpretation: ¬a muss in der Formelsammlung gelten.

Logische Programmierung

Das Prinzip der Programmiersprache PROLOG:

PROLOG versucht, mittels wiederholter und verschachtelter Anwendung
von **Resolution** und anderer Techniken
zu einer gegebenen Formelmenge einen Widerspruch zu finden.

Satz (Widerspruchsvollständigkeit):

Falls die Formelmenge widersprüchlich ist, kann man den Widerspruch immer finden.

Was fehlt ?

Satz (*Folgerung* der Folgerbarkeit aus der Widerspruchsaufdeckung):

Wer zu jeder Formelmenge jeden Widerspruch aufdecken kann,
kann zu jeder Formelmenge und zu jeder neuen ***daraus folgenden*** Formel beweisen,
dass die neue Formel aus der alten Formelmenge folgt.

Was fehlt hier ?

Logische Programmierung

Wie macht man aus PROLOG eine <u>vollständige</u> Programmiersprache ?

Durch Beschränkung der Eingabe !

PROLOG akzeptiert nur Mengen von Formeln der Form:

p ∧ q ∧ . . . ∧ r → x

Regeln (Hornklauseln)

In der Voraussetzung darf nur eine Konjunktion von positiven Literalen stehen.

<u>Satz (Vollständigkeit der Resolution auf Hornklauseln):</u>

Für jede Menge von Hornklauseln und eine neue Hornklausel kann Prolog nach endlicher Zeit entscheiden, ob die neue Hornklausel aus der alten Menge folgt ***oder nicht***

☹ ***Anmerkung: „Endliche Zeit“ kann „sehr lange“ heißen !***

Logische Programmierung

Wie erkennt man, ob eine Formelmenge nur aus Formeln besteht, die äquivalent zu Hornklauseln sind ?

Die Formelmenge sei in KNF gegeben, d.h. als eine Konjunktion von Disjunktionen.
Die einzelnen Klauseln (Disjunktionen) seien die einzelnen Formeln.
Eine einzelne Formel ist nach Definition eine Hornklausel, wenn sie äquivalent zu einer Regel ist, deren Voraussetzungen alle durch einen positiven Literal repräsentiert sind.

Eine Hornklausel sieht also immer so aus:

$$\neg p \vee \neg q \vee \ldots \vee \neg r \vee x$$

Maximal ein Literal ist positiv.

Anmerkung:
Natürlich kann man eine beliebige Aussage auch durch einen negativen Literal repräsentieren.
Um die Hornklauseleigenschaft zu erhalten, muss dieser Literal dann aber in allen Voraussetzungen von Regeln, in denen er vorkommt, in negativer Form auftauchen.
Daher können wir uns ohne Beschränkung der Allgemeinheit auf positive Literale beschränken.