

Logik

Teil 1: Aussagenlogik



Aussagenlogik: Überblick

- Aussagenlogik behandelt die logische Verknüpfung von Aussagen mittels Junktoren wie **und**, **oder**, **nicht**, gdw.
- Jeder Aussage ist ein **Wahrheitswert** (wahr/falsch) zugeordnet.
- Man interessiert sich insbesondere für den Wahrheitswert zusammengesetzter Aussagen, z. B.:

„**A oder B**“ wahr gdw. **A wahr oder B wahr**

A oder B könnten z. B. stehen für „Die Erde ist ein Planet“ oder „Leipzig liegt am Ganges“. Davon wird abstrahiert.

- Die **Ausdrucksstärke** von Aussagenlogik ist **sehr begrenzt**.
- Es ergeben sich jedoch **fundamentale algorithmische Probleme** (z. B. das **Erfüllbarkeitsproblem**).



Next



- 1.1 Syntax und Semantik**
- 1.2 Auswertung und Äquivalenz
- 1.3 Normalformen und funktionale Vollständigkeit
- 1.4 Erfüllbarkeit, Gültigkeit, Folgerbarkeit
- 1.5 Tseitin-Transformation
- 1.6 Horn Logik
- 1.7 Resolution
- 1.8 Kompaktheit

Syntax und Semantik

Jede Logik ist definiert durch **Syntax und Semantik**

Syntax: Welche Gestalt haben die Formeln?

„Was darf ich hinschreiben?“

Semantik: Was bedeuten die Formeln?

„Was soll das heißen?“



Syntax

Wir fixieren eine abzählbar unendliche Menge $\text{VAR} = \{x_1, x_2, x_3, \dots\}$ von **Aussagenvariablen**.

Intuitiv kann jedes x_i die Wahrheitswerte **wahr** oder **falsch** annehmen und **repräsentiert eine Aussage** wie „Leipzig liegt am Ganges“

Definition 1.1 (Aussagenlogik, Syntax)

Die Menge der **aussagenlogischen Formeln** ist wie folgt definiert:

1. Jede Aussagenvariable ist eine aussagenlogische Formel.
2. Wenn φ und ψ aussagenlogische Formeln sind,
dann auch $\neg\varphi$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$.
3. Es gibt keine weiteren aussagenlogischen Formeln.

Es handelt sich um eine **induktive Definition**:

Von atomaren Formeln (1.) zu immer komplexeren zusammengesetzten (2.)

Beispiele: $\neg x_1$, $\neg\neg x_3$, $(x_1 \wedge \neg x_4)$, $(\neg(x_1 \vee x_2) \wedge \neg(\neg x_1 \vee \neg x_2))$



Griechische Buchstaben

Formeln werden traditionell (meistens) mit griechischen Buchstaben bezeichnet.

Hier die in dieser VL am meisten verwendeten Buchstaben:

Buchstabe	φ oder ϕ	ψ	ϑ oder θ	τ	σ	π	ρ
Name	phi	psi	theta	tau	sigma	pi	rho

Buchstabe	α	β	γ	δ	Σ	Γ	Δ
Name	alpha	beta	gamma	delta	Sigma	Gamma	Delta

Am Ende der Tafelanschreibe
findet sich eine vollständige Liste



Sprechweisen und Konventionen

- $\neg\varphi$ sprechen wir „nicht φ “, der Junktor „ \neg “ heißt **Negation**
- $(\varphi \vee \psi)$ sprechen wir „ φ oder ψ “
Formel der Form $(\varphi \vee \psi)$ heißt **Disjunktion**, φ, ψ sind die **Disjunkte**
- $(\varphi \wedge \psi)$ sprechen wir „ φ und ψ “
Formel der Form $(\varphi \wedge \psi)$ heißt **Konjunktion**, φ, ψ sind die **Konjunkte**
- Die **atomaren Formeln** sind die Formeln in VAR
- Alle anderen Formeln sind **zusammengesetzt**

Andere Namen für Aussagenlogik: **Bool'sche Logik, Propositionale Logik**



Sprechweisen und Konventionen

- Statt x_1, x_2, \dots verwenden wir manchmal auch andere Symbole für Variablen, insbesondere x, y, z
- Klammern werden weggelassen, wenn das Resultat eindeutig ist, wobei \neg stärker bindet als \wedge und \vee

Beispiele:

- $\neg x \wedge y$ steht für $(\neg x \wedge y)$, nicht für $\neg(x \wedge y)$
- $x \wedge y \vee x' \wedge y'$ ist nicht eindeutig, darum nicht erlaubt
- Auch bei geschachteltem \wedge und \vee lassen wir die Klammern weg, z.B. $x_1 \wedge x_2 \wedge x_3$ statt $((x_1 \wedge x_2) \wedge x_3)$

Wir werden später sehen, dass dadurch bzgl. der Bedeutung (Semantik) keine Mehrdeutigkeiten entstehen (Assoziativität von \wedge und \vee)



Semantik

Definition 1.2 (Aussagenlogik, Semantik)

Eine Belegung ist eine Abbildung

$$V : \text{VAR} \rightarrow \{0, 1\},$$

ordnet also jeder Variablen einen der Wahrheitswerte 0 oder 1 zu.

Man erweitert die Abbildung V wie folgt induktiv von Variablen auf zusammengesetzte Formeln:

- $V(\neg\varphi) = 1 - V(\varphi)$
- $V(\varphi \wedge \psi) = \begin{cases} 1 & \text{falls } V(\varphi) = 1 \text{ und } V(\psi) = 1 \\ 0 & \text{sonst} \end{cases}$
- $V(\varphi \vee \psi) = \begin{cases} 1 & \text{falls } V(\varphi) = 1 \text{ oder } V(\psi) = 1 \\ 0 & \text{sonst} \end{cases}$

Wenn $V(\varphi) = 1$, dann sagen wir, dass φ von V erfüllt wird.

Wir schreiben dann auch $V \models \varphi$ und nennen V ein Modell von φ .



Beispiel:

Belegung V mit $V(x_1) = 0$ und $V(x_2) = 1, V(x_3) = 1, \dots$

Dann z.B.

$$V(\neg x_1) = 1$$

$$V(\neg x_1 \wedge x_2) = 1$$

$$V(\neg(\neg x_1 \wedge x_2)) = 0$$

$$V(\neg(\neg x_1 \wedge x_2) \vee x_3) = 1$$

Es gilt also $V \models \neg(\neg x_1 \wedge x_2) \vee x_3$

und V ist ein Modell von $\neg(\neg x_1 \wedge x_2) \vee x_3$.

Wahrheitstafeln

Die **Semantik der Junktoren** können übersichtlich als **Wahrheitstafeln** dargestellt werden:

$V(\varphi)$	$V(\neg\varphi)$	$V(\varphi)$	$V(\psi)$	$V(\varphi \wedge \psi)$	$V(\varphi)$	$V(\psi)$	$V(\varphi \vee \psi)$
0	1	0	0	0	0	0	0
1	0	0	1	0	0	1	1
		1	0	0	1	0	1
		1	1	1	1	1	1

Auch die **Semantik von Formeln** lässt sich so darstellen, z.B.:

$V(x_1)$	$V(x_2)$	$V(\neg(\neg x_1 \wedge x_2))$
0	0	1
0	1	0
1	0	1
1	1	1

In der obersten Zeile lassen wir das V meist weg.



Wahrheitstafeln

Es kann sinnvoll sein, auch Spalten für Teilformeln einzuführen:

x_1	x_2	$\neg x_1$	$\neg x_1 \wedge x_2$	$\neg(\neg x_1 \wedge x_2)$
0	0	1	0	1
0	1	1	1	0
1	0	0	0	1
1	1	0	0	1

Es ist üblich, die Belegungen so anzurichten, dass sie **als binäre Zahlen betrachtet** in aufsteigender Reihenfolge erscheinen.

Dies bitte auch in den Übungen stets so handhaben!



Iterierte Konjunktion / Disjunktion

Notation:

- Wir schreiben

$$\bigwedge_{i=1..n} \varphi_i \text{ für } \varphi_1 \wedge \dots \wedge \varphi_n \text{ (iterierte Konjunktion)}$$

$$\bigvee_{i=1..n} \varphi_i \text{ für } \varphi_1 \vee \dots \vee \varphi_n \text{ (iterierte Disjunktion)}$$

- Für den Spezialfall $n = 0$ ergibt sich:

$$\bigvee_{i=1..n} \varphi_i := 0 \quad (\text{leere Disjunktion})$$

(denn eine Disjunktion ist wahr, wenn mind. ein Disjunkt wahr ist)

$$\bigwedge_{i=1..n} \varphi_i := 1 \quad (\text{leere Konjunktion})$$

(denn eine Konjunktion ist wahr wenn alle Konjunkte wahr sind)



Implikation

Weitere interessante Junktoren sind als Abkürzung definierbar, z. B.:

Implikation $\varphi \rightarrow \psi$ steht für $\neg\varphi \vee \psi$

Biimplikation $\varphi \leftrightarrow \psi$ steht für $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$

Es ergeben sich folgende Wahrheitstafeln:

φ	ψ	$\varphi \rightarrow \psi$
0	0	1
0	1	1
1	0	0
1	1	1

φ	ψ	$\varphi \leftrightarrow \psi$
0	0	1
0	1	0
1	0	0
1	1	1

Wir vereinbaren, dass \neg, \wedge, \vee stärker binden als \rightarrow und \leftrightarrow ,

$x \wedge y \rightarrow z$ steht also für $(x \wedge y) \rightarrow z$



Weitere Junktoren

Ein weiteres Beispiel: exklusives oder (XOR)

φ	ψ	$\varphi \oplus \psi$
0	0	0
0	1	1
1	0	1
1	1	0

Definierbar als:

$$\neg(\varphi \leftrightarrow \psi)$$

Wir definieren auch **0-stellige** Junktoren 0 und 1, die **Wahrheitskonstanten**:

0 steht für $x \wedge \neg x$ 1 steht für $x \vee \neg x$

wobei x eine beliebige (aber feste) Variable ist

Es gilt also: $V(0) = 0$ und $V(1) = 1$ für alle Belegungen V

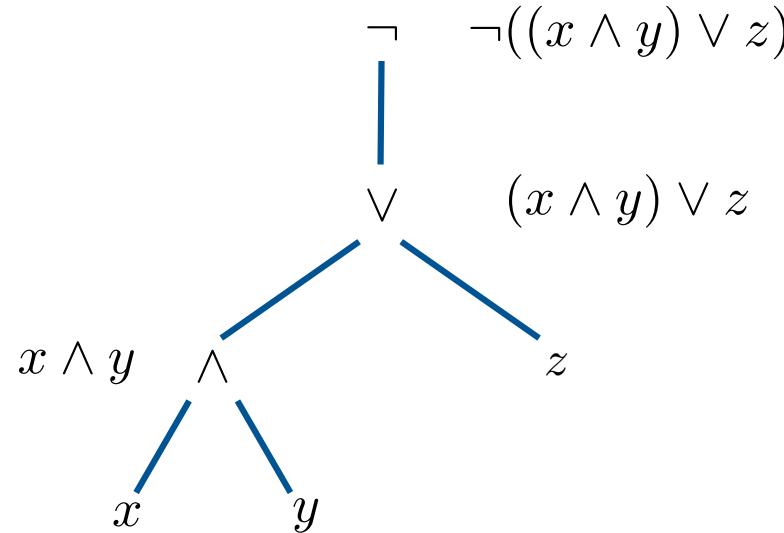
(Achtung: selbes Symbol für Konstante und Wahrheitswert)



Syntaxbäume

Es ist häufig bequem, Formeln als **Syntaxbäume** darzustellen

Beispiel $\neg((x \wedge y) \vee z)$:



Jeder Knoten im Baum entspricht einer **Teilformel**

Der Syntaxbaum für eine Formel φ ist nicht größer als φ , insbesondere gilt:

- Anzahl der inneren Knoten = Anzahl Junktorenvorkommen in φ und
- Anzahl der Blätter = Anzahl Variablenvorkommen in φ .

Teilformeln

Wir definieren die Menge der Teilformeln auch nochmal formal:

Definition 1.4 (Teilformeln)

Sei φ eine Formel. Die Menge $\text{TF}(\varphi)$ der **Teilformeln** von φ ist induktiv definiert wie folgt:

- $\text{TF}(x) = \{x\}$
- $\text{TF}(\neg\varphi) = \{\neg\varphi\} \cup \text{TF}(\varphi)$
- $\text{TF}(\varphi \wedge \psi) = \{\varphi \wedge \psi\} \cup \text{TF}(\varphi) \cup \text{TF}(\psi)$
- $\text{TF}(\varphi \vee \psi) = \{\varphi \vee \psi\} \cup \text{TF}(\varphi) \cup \text{TF}(\psi)$

Zum Beispiel:

$$\text{TF}(\neg((x \wedge y) \vee z)) = \{x, y, z, x \wedge y, (x \wedge y) \vee z, \neg((x \wedge y) \vee z)\}$$

Anzahl der **Teilformeln** ist “klein”:

beschränkt durch Anzahl der Junktoren- und Variabenvorkommen



Koinzidenzlemma

Für den Wahrheitswert einer Formel φ ist nur die Belegung derjenigen Variablen von Bedeutung, die in φ vorkommen.

Formal wird das von folgendem Lemma ausgedrückt:

Lemma 1.3 (Koinzidenzlemma)

Sei φ eine Formel und V_1, V_2 Belegungen so dass $V_1(x) = V_2(x)$ für alle Variablen x in φ . Dann ist $V_1(\varphi) = V_2(\varphi)$.

Den (einfachen) Beweis wollen wir an dieser Stelle nicht führen.

Wenn wir mit einer Formel φ arbeiten, so erlaubt uns das Koinzidenzlemma, nur die Variablen in φ zu betrachten (z.B. in Belegungen).

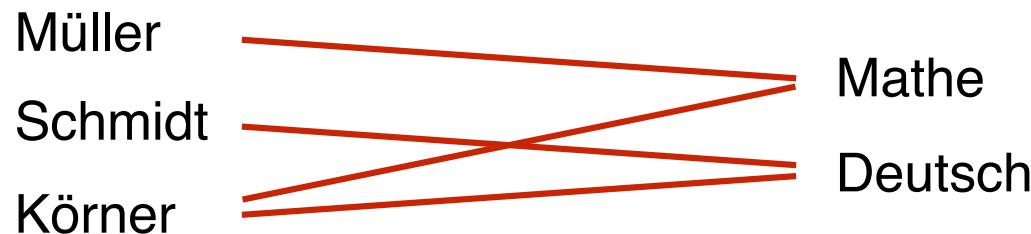
Dies werden wir ab jetzt häufig so handhaben.



Beispiel Repräsentation

Modellierung eines Zeitplanungs-Problems (Scheduling) in Aussagenlogik

An einer Schule gibt es drei Lehrer mit folgenden Fächerkombinationen:



Es soll folgender Lehrplan erfüllt werden:

	Klasse a)	Klasse b)
Stunde I	Mathe	Deutsch
Stunde II	Deutsch	Deutsch
Stunde III	Mathe	Mathe

Dabei soll jeder Lehrer mindestens 2 Stunden unterrichten.

Beispiel Repräsentation

		Klasse a)	Klasse b)
Müller	Mathe	Mathe	Deutsch
Schmidt	Deutsch	Deutsch	Deutsch
Körner		Mathe	Mathe
Jeder Lehrer mindestens 2 Stunden			

Wir verwenden Variablen der Form

$$x_{sk}^L \text{ mit } L \in \{\text{M, S, K}\}, s \in \{\text{I, II, III}\}, k \in \{a, b\}$$

Zum Beispiel repräsentiert die Variable $x_{\text{II}a}^{\text{M}}$ die Aussage

„Müller unterrichtet Klasse a) in Stunde II“

Wir finden aussagenlogische Formel φ deren erfüllende Belegungen genau den Lösungen des Zeitplanungsproblems entsprechen

T1.1



Beispiel Repräsentation

		Klasse a)	Klasse b)
Müller	Mathe	M Mathe	S Deutsch
Schmidt	Deutsch	K Deutsch	S Deutsch
Körner		K Mathe	M Mathe

Jeder Lehrer mindestens 2 Stunden

Beispiel für eine erfüllende Belegung:

$V(x) = 1$ für folgende Variablen x :

$$x_{Ia}^M, x_{Ib}^S, x_{IIa}^K, x_{IIb}^S, x_{IIIa}^K, x_{IIIb}^M$$

$V(x) = 0$ für alle anderen Variablen.

Gibt es noch andere erfüllende Belegungen?

Wie formalisiert man zusätzlich, dass jeder Lehrer beide Klassen unterrichten soll?

Gibt es dann immernoch eine erfüllende Belegung?



Next



- 1.1 Syntax und Semantik
- 1.2 Auswertung und Äquivalenz**
- 1.3 Normalformen und funktionale Vollständigkeit
- 1.4 Erfüllbarkeit, Gültigkeit, Folgerbarkeit
- 1.5 Tseitin-Transformation
- 1.6 Horn Logik
- 1.7 Resolution
- 1.8 Kompaktheit

Algorithmische Probleme

Wir werden in dieser Vorlesung verschiedene **algorithmische Probleme** kennenlernen, die im Zusammenhang mit Logik stehen:

Auswertungsproblem, Erfüllbarkeitsproblem, usw.

Soll eine Formel als **Eingabe für einen Algorithmus** verwendet werden, so muss sie **mittels eines endlichen Alphabets repräsentiert** werden.

Wir verwenden hier die Alphabetssymbole $\neg, \wedge, \vee, (,), 0, 1$ wobei 0, 1 der Repräsentation der **Variablennamen** dienen

Zum Beispiel wird $\varphi = \neg x \vee (y \wedge z)$ repräsentiert als $\neg 00 \vee (01 \wedge 10)$

Größe von φ , bezeichnet mit $|\varphi|$, ist Länge dieser Repräsentation

Also z.B. $|\neg x \vee (y \wedge z)| = 11$

Meist abstrahieren wir jedoch von den Details dieser Repräsentation



Auswertung

Definition 1.5 (Auswertungsproblem)

Das Auswertungsproblem der Aussagenlogik ist:

Gegeben: Aussagenlogische Formel φ , Belegung V für φ

Frage: Gilt $V \models \varphi$?

Theorem 1.6 (Komplexität Auswertungsproblem)

Das Auswertungsproblem der Aussagenlogik ist in Linearzeit lösbar.

Linearzeit heißt Zeit $O(n)$ wobei n die Größe der Eingabe

Anmerkung:

Das Auswertungsproblems hat zwei Eingaben, aber V kann nicht größer sein als φ ; wir können daher als Eingabegröße $n = |\varphi|$ verwenden.

Wir wollen hier nur Lösbarkeit in quadratischer Zeit zeigen



Auswertung

Wir verwenden einen einfachen rekursiven Algorithmus:

Function Ausw(φ, V):

```
if  $\varphi = x \in \text{VAR}$  then return  $V(x)$ 
else if  $\varphi = \neg\psi$  then return  $1 - \text{Ausw}(\psi, V)$ 
else if  $\varphi = \psi_1 \wedge \psi_2$  then return  $\min\{\text{Ausw}(\psi_i, V) \mid i \in \{1, 2\}\}$ 
else if  $\varphi = \psi_1 \vee \psi_2$  then return  $\max\{\text{Ausw}(\psi_i, V) \mid i \in \{1, 2\}\}$ 
```

T1.2

Der Algorithmus tut das Gewünschte:

Lemma 1.7

Für alle Formeln φ und Belegungen V gilt: $\text{Ausw}(\varphi, V) = 1$ gdw. $V \models \varphi$

Lemma 1.7 ist sehr leicht einzusehen; wir führen der Beweistechnik wegen trotzdem kurz einen formalen Beweis.



Induktion über den Formelaufbau

In der Logik werden Beweise oft per Induktion über den Aufbau von Formeln geführt. Dies gilt insbesondere für Behauptungen der Art

Für alle Formeln φ gilt: Aussage A über φ

Man nennt dies auch strukturelle Induktion:

Induktionsanfang:

Man zeigt, dass alle atomaren Formeln φ die Eigenschaft A haben.

In der Aussagenlogik sind die atomaren Formeln die Aussagenvariablen

Induktionsschritt:

Man zeigt, dass alle zusammengesetzten Formeln φ die Eigenschaft A haben unter der Annahme (Induktionsvoraussetzung, IV), dass alle Teilformeln A erfüllen

Zum Beispiel $\varphi = \psi_1 \wedge \psi_2$ erfüllt A wenn ψ_1 und ψ_2 jeweils A erfüllen

Üblicherweise Fallunterscheidung gemäß äußerstem Junktor in φ .



Auswertung

Wir verwenden einen einfachen rekursiven Algorithmus:

Function Ausw(φ, V):

```
if  $\varphi = x \in \text{VAR}$  then return  $V(x)$ 
else if  $\varphi = \neg\psi$  then return  $1 - \text{Ausw}(\psi, V)$ 
else if  $\varphi = \psi_1 \wedge \psi_2$  then return  $\min\{\text{Ausw}(\psi_i, V) \mid i \in \{1, 2\}\}$ 
else if  $\varphi = \psi_1 \vee \psi_2$  then return  $\max\{\text{Ausw}(\psi_i, V) \mid i \in \{1, 2\}\}$ 
```

Der Algorithmus tut das Gewünschte:

Lemma 1.7

Für alle Formeln φ und Belegungen V gilt: $\text{Ausw}(\varphi, V) = 1$ gdw. $V \models \varphi$

Wir führen den Beweis per Induktion über den Aufbau der Formel φ .

T1.3



Laufzeit

Wir verwenden einen einfachen rekursiven Algorithmus:

Function Ausw(φ, V):

```
if  $\varphi = x \in \text{VAR}$  then return  $V(x)$ 
else if  $\varphi = \neg\psi$  then return  $1 - \text{Ausw}(\psi, V)$ 
else if  $\varphi = \psi_1 \wedge \psi_2$  then return  $\min\{\text{Ausw}(\psi_i, V) \mid i \in \{1, 2\}\}$ 
else if  $\varphi = \psi_1 \vee \psi_2$  then return  $\max\{\text{Ausw}(\psi_i, V) \mid i \in \{1, 2\}\}$ 
```

Wir analysieren nun noch kurz die **Laufzeit** unseres Algorithmus

Offensichtlich ist der **Rekursionsbaum** von $\text{Ausw}(\varphi, V)$ genau
der **Syntaxbaum** von φ

Es folgt:

Anzahl rekursiver Aufrufe = Anzahl Knoten im Syntaxbaum, also $\leq n = |\varphi|$

Zeitaufwand einzelner Aufruf: $V(x)$ in V finden ist $O(n)$

Insgesamt **$O(n^2)$**



Äquivalenz

Syntaktisch verschiedene Formeln können dieselbe Bedeutung haben

Definition 1.8 (Äquivalenz)

Zwei Formeln φ und ψ sind äquivalent, wenn für alle Belegungen V gilt, dass $V(\varphi) = V(\psi)$. Wir schreiben dann $\varphi \equiv \psi$.

Z.B: gilt

$$x \wedge y \equiv \neg(\neg x \vee \neg y)$$

Einfacher **Nachweis vom Äquivalenz** mittels Wahrheitstafeln für Formeln φ :

x	y	$x \wedge y$	x	y	$\neg(\neg x \vee \neg y)$
0	0	0	0	0	0
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	1	1	1

Links stehen die Variablen aus φ , bei n Variablen also 2^n Zeilen

Dieses Verfahren ist also ziemlich ineffizient!



Ersetzungslemma

Äquivalente Formeln sind austauschbar, auch innerhalb anderer Formeln

Zum Beispiel:

Wir wissen bereits

$$x \wedge y \equiv \neg(\neg x \vee \neg y)$$

Dann gilt auch

$$\neg z \vee (x \wedge y) \equiv \neg z \vee (\neg(\neg x \vee \neg y))$$

Lemma 1.9 (Ersetzungslemma)

Seien φ und ψ äquivalente Formeln, ϑ eine Formel mit $\varphi \in \text{TF}(\vartheta)$ und ϑ' eine Formel, die sich aus ϑ ergibt, indem ein beliebiges Vorkommen von φ durch ψ ersetzt wird. Dann gilt $\vartheta \equiv \vartheta'$.

Wir beweisen das Lemma per Induktion über den Aufbau von ϑ .



Ersetzungslemma

Lemma 1.9 (Ersetzungslemma)

Seien φ und ψ äquivalente Formeln, ϑ eine Formel mit $\varphi \in \text{TF}(\vartheta)$ und ϑ' eine Formel, die sich aus ϑ ergibt, indem ein beliebiges Vorkommen von φ durch ψ ersetzt wird. Dann gilt $\vartheta \equiv \vartheta'$.

Wir beweisen das Lemma per Induktion über den Aufbau von ϑ .

Induktionsanfang:

ϑ atomar

also $\vartheta = \varphi$ (weil φ Teilformel von ϑ)

also $\vartheta' = \psi$ (weil ϑ' aus ϑ durch Ersetzen von φ durch ψ entsteht)

also $\vartheta \equiv \vartheta'$ (weil $\varphi \equiv \psi$)



Ersetzungslemma

Lemma 1.9 (Ersetzungslemma)

Seien φ und ψ äquivalente Formeln, ϑ eine Formel mit $\varphi \in \text{TF}(\vartheta)$ und ϑ' eine Formel, die sich aus ϑ ergibt, indem ein beliebiges Vorkommen von φ durch ψ ersetzt wird. Dann gilt $\vartheta \equiv \vartheta'$.

Wir beweisen das Lemma per Induktion über den Aufbau von ϑ .

Induktionsschritt:

Wenn $\vartheta = \varphi$ argumentieren wir wie im Induktionsanfang; andernfalls:

Fall $\vartheta = \neg\vartheta_1$ $\neg\vartheta'_1$ aus ϑ_1 durch Ersetzen eines φ durch ψ

Dann hat ϑ' die Form $\neg\vartheta'_1$ (weil $\vartheta \neq \varphi$)

IV liefert $\vartheta_1 \equiv \vartheta'_1$

Semantik von “ \neg ” liefert $\vartheta \equiv \vartheta'$ (Beweis per Wahrheitstabelle)



Ersetzungslemma

Lemma 1.9 (Ersetzungslemma)

Seien φ und ψ äquivalente Formeln, ϑ eine Formel mit $\varphi \in \text{TF}(\vartheta)$ und ϑ' eine Formel, die sich aus ϑ ergibt, indem ein beliebiges Vorkommen von φ durch ψ ersetzt wird. Dann gilt $\vartheta \equiv \vartheta'$.

Wir beweisen das Lemma per Induktion über den Aufbau von ϑ .

Induktionsschritt:

Wenn $\vartheta = \varphi$ argumentieren wir wie im Induktionsanfang; andernfalls:

Fall $\vartheta = \vartheta_1 \vee \vartheta_2$

Dann $\vartheta' = \vartheta'_1 \vee \vartheta'_2$ oder $\vartheta' = \vartheta_1 \vee \vartheta'_2$ (weil $\vartheta \neq \varphi$)

IV liefert $\vartheta_i \equiv \vartheta'_i$ für $i = 1$ oder $i = 2$

Semantik von “ \vee ” liefert $\vartheta \equiv \vartheta'$

Fall $\vartheta = \vartheta_1 \wedge \vartheta_2$ analog

□



Nützliche Äquivalenzen

Im Folgenden wollen wir **einige nützliche Äquivalenzen** etablieren

Genauer gesagt handelt es sich um **Äquivalenzschemata**, z. B.:

Für alle Formeln φ gilt: $\varphi \equiv \neg\neg\varphi$

Eliminieren doppelter Negation

Die Gültigkeit der Äquivalenzen kann per Wahrheitstafel nachgewiesen werden



Nützliche Äquivalenzen

Folgende Äquivalenzen gelten für alle aussagenlogischen Formeln φ, ψ, ϑ :

- $\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$

De Morgansche Gesetze

- $\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$

- $\neg\neg\varphi \equiv \varphi$

Doppelte Negation

- $\varphi \wedge \varphi \equiv \varphi$

Idempotenz von Konjunktion

- $\varphi \vee \varphi \equiv \varphi$

und Disjunktion

- $\varphi \wedge \psi \equiv \psi \wedge \varphi$

Kommutativität von Konjunktion
und Disjunktion

- $\varphi \vee \psi \equiv \psi \vee \varphi$

- $\varphi \wedge (\psi \wedge \vartheta) \equiv (\varphi \wedge \psi) \wedge \vartheta$

Assoziativität von Konjunktion
und Disjunktion

- $\varphi \vee (\psi \vee \vartheta) \equiv (\varphi \vee \psi) \vee \vartheta$



Nützliche Äquivalenzen

Mehr nützliche Äquivalenzen:

- $\varphi \wedge (\psi \vee \vartheta) \equiv (\varphi \wedge \psi) \vee (\varphi \wedge \vartheta)$ Distributivgesetze
 $\varphi \vee (\psi \wedge \vartheta) \equiv (\varphi \vee \psi) \wedge (\varphi \vee \vartheta)$
- $\varphi \wedge (\varphi \vee \psi) \equiv \varphi \equiv \varphi \vee (\varphi \wedge \psi)$ Absorption
- $\varphi \wedge 1 \equiv \varphi$ Neutrales Element für Konjunktion
 $\varphi \vee 0 \equiv \varphi$ und Disjunktion
- $\varphi \wedge \neg\varphi \equiv 0$ Kontradiktion und
 $\varphi \vee \neg\varphi \equiv 1$ Tautologie

Auch für die (Bi)implikation gibt es interessante Äquivalenzen, z. B.:

- $\varphi \rightarrow \psi \equiv \neg\psi \rightarrow \neg\varphi$ Kontraposition



Nachweis weiterer Äquivalenzen

Mittels dieser Äquivalenzen und dem Ersetzungslemma (EL) kann man durch Umformung neue Äquivalenzen nachweisen.

Zum Beispiel $\neg x \wedge \neg y \equiv \neg(x \vee (\neg x \wedge y))$

$$\begin{aligned}\neg(x \vee (\neg x \wedge y)) &\equiv \neg x \wedge \neg(\neg x \wedge y) && \text{De Morgan} \\ &\equiv \neg x \wedge (\neg\neg x \vee \neg y) && \text{De Morgan + EL} \\ &\equiv \neg x \wedge (x \vee \neg y) && \text{doppelte Negation + EL} \\ &\equiv (\neg x \wedge x) \vee (\neg x \wedge \neg y) && \text{Distributivgesetz} \\ &\equiv 0 \vee (\neg x \wedge \neg y) && \text{Kontradiktion + EL} \\ &\equiv (\neg x \wedge \neg y) \vee 0 && \text{Kommutativgesetz} \\ &\equiv \neg x \wedge \neg y && \text{Neutrales Element Disjunktion}\end{aligned}$$



- 1.1 Syntax und Semantik
- 1.2 Auswertung und Äquivalenz
- 1.3 Normalformen und funktionale Vollständigkeit**
- 1.4 Erfüllbarkeit, Gültigkeit, Folgerbarkeit
- 1.5 Tseitin-Transformation
- 1.6 Horn Logik
- 1.7 Resolution
- 1.8 Kompaktheit

Next



Boolesche Funktionen

Definition 1.10 (Boolesche Funktion)

Eine n -stellige Boolesche Funktion ist eine Funktion $f : \{0, 1\}^n \rightarrow \{0, 1\}$.

Für $n \geq 0$ bezeichne

- \mathcal{B}^n die Menge aller n -stelligen Booleschen Funktionen
- \mathcal{B} die Menge $\mathcal{B}^0 \cup \mathcal{B}^1 \cup \dots$ aller Booleschen Funktionen

Zum Beispiel:

\mathcal{B}^0 besteht aus den beiden konstanten Funktionen 0 und 1.

\mathcal{B}^1 besteht aus vier Funktionen $f_{00}, f_{10}, f_{01}, f_{11}$:

Eingabe	f_{00}	f_{01}	f_{10}	f_{11}
0	0	0	1	1
1	0	1	0	1

Allgemein: \mathcal{B}^n besteht aus 2^{2^n} Funktionen

Boolesche Funktionen hängen sehr eng mit Aussagenlogik zusammen



AL Formeln \Rightarrow Boolesche Funktionen

Jede aussagenlogische Formel φ mit n Variablen repräsentiert eine n -stellige Boolesche Funktion f_φ :

- Seien x_1, \dots, x_n die Variablen in φ
- Jede Belegung V für φ entspricht einer Eingabe für f_φ :
 i -ter Eingabewert ist $V(x_i)$
- Der Wert von f_φ bei Eingabe/Belegung V ist $V(\varphi)$

Z.B. $\varphi = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$

x_1	x_2	φ
0	0	1
0	1	0
1	0	0
1	1	1

also

$$\begin{aligned}f_\varphi(0, 0) &= 1 \\f_\varphi(0, 1) &= 0 \\f_\varphi(1, 0) &= 0 \\f_\varphi(1, 1) &= 1\end{aligned}$$



Boolesche Funktionen \Rightarrow AL Formeln

Umgekehrt gibt es zu **jeder** Booleschen Funktion eine aussagenlogische Formel, die sie repräsentiert.

Theorem 1.11 (funktionale Vollständigkeit)

Zu jeder Booleschen Funktion $f \in \mathcal{B}$ gibt es eine Formel φ_f mit $f_{\varphi_f} = f$.

Beweis:

\mathcal{B}^0 enthält die konstante 0-Funktion und die konstante 1-Funktion
dann $\varphi_0 = 0$ und $\varphi_1 = 1$

Für $f \in \mathcal{B}^n$ mit $n > 0$ hat jede Eingabe die Form $t = (w_1, \dots, w_n) \in \{0, 1\}^n$

Darstellbar als Formel

$$\psi_t = \ell_1 \wedge \cdots \wedge \ell_n \quad \text{wobei} \quad \ell_i = \begin{cases} x_i & \text{wenn } w_i = 1 \\ \neg x_i & \text{wenn } w_i = 0 \end{cases}$$

Dann $\varphi_f = \bigvee_{\substack{t=(w_1, \dots, w_n) \in \{0, 1\}^n \\ f(t)=1}} \psi_t$

T1.4



Funktionale Vollständigkeit

Wir haben gerade gesehen:

Aussagenlogik kann als Sprache zur Repräsentation Boolescher Funktionen verstanden werden.

Die Tatsache, dass jede Boolesche Funktion als AL-Formel repräsentiert werden kann bezeichnet man auch als

Funktionale Vollständigkeit der Aussagenlogik

Das spielt zum Beispiel eine Rolle im Design von Chips

Die repräsentierenden Formeln können aber sehr groß werden:

Für $f \in \mathcal{B}^n$ hat die Formel φ_f bis zu 2^n viele Disjunkte

Wir zeigen im Folgenden, dass das im Allgemeinen nicht vermeidbar ist.



Boolesche Funktionen \Rightarrow AL Formeln

Lemma 1.12 (Exponentielle Größe unvermeidbar)

Sie ℓ eine Funktion, so dass es zu jeder Booleschen Funktion $f \in \mathcal{B}^n$ eine Formel φ_f der Größe $\ell(n)$ gibt mit $f_{\varphi_f} = f$. Dann $\ell \notin o(2^n)$.

D.h.: ℓ wächst nicht wesentlich langsamer als 2^n

Beweis (Idee):

Es gibt **wesentlich mehr** Boolesche Funktionen in \mathcal{B}^n als “kurze” Formeln mit n Variablen

Ließe sich jedes $f \in \mathcal{B}^n$ als “kurze” Formel darstellen, würden also zwei verschiedene $f, f' \in \mathcal{B}^n$ durch **dieselbe** Formel dargestellt.

Das ist aber offensichtlich nicht möglich.



Lemma 1.12 (Exponentielle Größe unvermeidbar)

Sie ℓ eine Funktion, so dass es zu jeder Booleschen Funktion $f \in \mathcal{B}^n$ eine Formel φ_f der Größe $\ell(n)$ gibt mit $f_{\varphi_f} = f$. Dann $\ell \notin o(2^n)$.

Beweis: Angenommen ℓ ist Funktion wie beschrieben, aber $\ell \in o(2^n)$.

Anzahl Formeln der Größe $\leq \ell(n)$ ist $\leq 8^{\ell(n)}$

(Anzahl Strings der Länge $\leq \ell(n)$ über 7 Symbolen $\neg, \wedge, \vee, (,), 0, 1$)

Wähle n **groß genug so** dass $3 \cdot \ell(n) < 2^n$

2^{2^n} Funktionen in \mathcal{B}^n , aber nur $8^{\ell(n)} = 2^{3 \cdot \ell(n)} < 2^{2^n}$ Formeln der Größe $\ell(n)$

Also $\varphi_f = \varphi_{f'}$ für **verschiedene** $f, f' \in \mathcal{B}^n$. **Widerspruch.** □

Normalformen

Der Beweis von Theorem 1.11 (Umwandlung Boolesche Funktion \Rightarrow AL Formel) hat eine interessante Konsequenz:

Jede Formel ist äquivalent zu einer Formel der Form

$$(\ell_{1,1} \wedge \cdots \wedge \ell_{1,m_1}) \vee \cdots \vee (\ell_{n,1} \wedge \cdots \wedge \ell_{n,m_n})$$

wobei die $\ell_{i,j}$ jeweils die Form x oder $\neg x$ haben.

Dies ist die sogenannte *disjunktive Normalform*.

Dual dazu gibt es auch die ebenso wichtige *konjunktive Normalform*.



Normalformen

Definition 1.13 (KNF, DNF)

Ein **Literal** ist eine Formel der Form

- x (**positives Literal**) oder
- $\neg x$ (**negatives Literal**)

Eine Formel φ ist in **konjunktiver Normalform (KNF)**, wenn sie eine Konjunktion von Disjunktionen von Literalen ist:

$$\varphi = \bigwedge_{i=1..n} \bigvee_{j=1..m_i} \ell_{i,j}$$

Eine Formel φ ist in **disjunktiver Normalform (DNF)**, wenn sie eine Disjunktion von Konjunktionen von Literalen ist:

$$\varphi = \bigvee_{i=1..n} \bigwedge_{j=1..m_i} \ell_{i,j}$$



Normalformen

Theorem 1.14 (KNF/DNF-Umwandlung)

Jede Formel lässt sich in äquivalente Formel in KNF bzw. DNF wandeln.

DNF: Gegeben Formel φ , wandle das beschriebene Verfahren auf
Funktion f_φ (= Wahrheitstafel für φ) an

Beispiel: $\varphi = (y \vee \neg(x \vee y)) \wedge \neg z$

betrachte jede Zeile mit $V(\varphi) = 1$

stelle als Konjunktion dar, z.B.

$$\neg x \wedge y \wedge \neg z$$

verknüpfe disjunktiv

x	y	z	φ
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

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



Normalformen

Theorem 1.14 (KNF/DNF-Umwandlung)

Jede Formel lässt sich in äquivalente Formel in KNF bzw. DNF wandeln.

KNF: $\varphi \equiv \neg \underline{\neg} \varphi$
in DNF wandeln

$$\equiv \neg \bigvee_{i=1}^n \bigwedge_{j=1}^{m_i} \ell_{ij}$$

$$\equiv \bigwedge_{i=1}^n \bigvee_{j=1}^{m_i} \neg \ell_{ij} \quad 2^* \text{ de Morgan}$$

ist in KNF (nach Eliminieren doppelter Negation)



Normalformen

Theorem 1.14 (KNF/DNF-Umwandlung)

Jede Formel lässt sich in äquivalente Formel in KNF bzw. DNF wandeln.

KNF (Alternative):

Verfahren wie bei DNF, aber dual

Beispiel: $\varphi = (y \vee \neg(x \vee y)) \wedge \neg z$

betrachte jede Zeile mit $V(\varphi) = 0$

Komplement als Disjunktion, z.B.

$$x \vee y \vee \neg z$$

verknüpfe konjunktiv

x	y	z	φ
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

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



Beachte:

- Sowohl DNF als auch KNF können **exponentiell groß** werden
(2^n viele Disjunkte / Konjunkte, wobei n Anzahl Variablen in φ)
- Das lässt sich auch nicht durch eine bessere Konstruktion verhindern

Konkret:

n -stellige Paritätsfunktion p_n : $p_n(t) = 1$ gdw. t ungeradzahlig oft 1 enthält

Man kann zeigen, dass:

1. es eine Formel für p_n gibt, deren Größe **polynomiell** in n ist
2. jede **DNF Formel** für p_n mindestens 2^{n-1} Disjunkte hat
3. jede **KNF Formel** für p_n mindestens 2^{n-1} Konjunkte hat



Größe KNF/DNF

Intuition dafür, dass DNF/KNF exponentiell groß werden:

Genau die **Hälfte** der Eingaben
liefert 1 (\Rightarrow DNF Konstruktion)
die andere **Hälfte** liefert 0
 $(\Rightarrow$ KNF Konstruktion)

x	y	z	p_3
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Man kann keine Disjunkte/Konjunkte “zusammenfassen”: resultierendes Disjunkt/Konjunkt hat weniger Variablen, aber Parität wird durch **alle** Variablen bestimmt

Das ist natürlich **kein** formaler Beweis!

Der findet sich z.B. im Buch „Komplexitätstheorie“ von Ingo Wegener



Größe KNF/DNF

Idee Paritätsfunktion von polynomieller Größe: Divide and Conquer
(Parität von $x_0x_1x_2x_3$ ergibt sich aus der von x_0x_1 und x_2x_3 mittels Parität!)

- Für 2 Variablen: $\varphi_{01} = (\neg x_0 \wedge x_1) \vee (x_0 \wedge \neg x_1)$
- Für 4 Variablen: $\varphi_{0123} = (\neg \varphi_{01} \wedge \varphi_{23}) \vee (\varphi_{01} \wedge \neg \varphi_{23})$


wie φ_{01} , aber mit Variablen x_2, x_3
- Für 8 Variablen: $\varphi_{01234567} = (\neg \varphi_{0123} \wedge \varphi_{4567}) \vee (\varphi_{0123} \wedge \neg \varphi_{4567})$

Für $n = 2^m$ Variablen ist m -maliges Einsetzen erforderlich

Es ergibt sich Syntaxbaum mit

- Tiefe $3m$ und Verzweigung ≤ 2
- also mit Anzahl Knoten $< 2^{3m} = (2^m)^3 = n^3$

Kleine Modifikationen,
wenn Anzahl Variablen
keine Zweierpotenz



Funktionale Vollständigkeit

Wir haben gesehen:

Mittels der Junktoren \neg, \wedge, \vee kann man für jede Boolesche Funktion f eine „äquivalente“ Formel φ_f konstruieren

soll heißen: die Wahrheitstafel von φ_f ist f

Aus den De Morganschen Gesetzen folgt

$$\varphi \wedge \psi \equiv \neg(\neg\varphi \vee \neg\psi)$$

$$\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi)$$

also gilt dasselbe für die Junktormengen \neg, \wedge und \neg, \vee

Allgemein stellt sich die Frage:

Welche Junktorenmengen sind in diesem Sinne vollständig?



Funktionale Vollständigkeit von Junktorenmengen

Die Junktoren \neg, \wedge, \vee können als Boolesche Funktionen aus \mathcal{B}^1 bzw. \mathcal{B}^2 aufgefasst werden.

Umgekehrt liefert jede Boolesche Funktion $f \in \mathcal{B}^n$ einen n -ären Junktor:
Zeile $t = (w_1, \dots, w_n) \in \{0, 1\}^n$ in Wahrheitstafel hat Wert $f(t)$.

T1.5

Wir werden im Folgenden beliebige Boolesche Funktionen als Junktoren betrachten.

Weiterer interessanter Junktor neben $0, 1, \neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus$ z.B.:

φ	ψ	$\varphi \mid \psi$
0	0	1
0	1	1
1	0	1
1	1	0

Nand / „Sheffer-Strich“



Funktionale Vollständigkeit von Junktorenmengen

Definition 1.15 (funktionale Vollständigkeit)

Eine Menge $\Omega \subseteq \mathcal{B}$ von Booleschen Funktionen ist **funktional vollständig** wenn es für jede Boolesche Funktion f eine Formel φ mit Junktoren aus Ω gibt, so dass $f_\varphi = f$.

Wir wissen bereits, dass folgende Mengen funktional vollständig sind:

$$\{\neg, \wedge, \vee\}$$

$$\{\neg, \wedge\}$$

$$\{\neg, \vee\}$$

Weitere funktional vollständige Mengen:

- $\{\neg, \rightarrow\}$

Da $\{\neg, \vee\}$ funktional vollständig und $\varphi \vee \psi \equiv \neg\varphi \rightarrow \psi$

- $\{\wedge, \oplus, 1\}$

Da $\{\neg, \wedge\}$ funktional vollständig und $\neg\varphi \equiv 1 \oplus \varphi$



Funktionale Vollständigkeit von Junktorenmengen

Weitere funktional vollständige Menge:

- $\{| \}$

Mit Nand kann man \neg ausdrücken:

$$\begin{aligned}\varphi | \varphi &\equiv \neg(\varphi \wedge \varphi) && (\text{Definition von } |) \\ &\equiv \neg\varphi && (\text{Idempotenz})\end{aligned}$$

Mit Nand kann man \wedge ausdrücken:

$$\begin{aligned}(\varphi | \psi) | (\varphi | \psi) &\equiv \neg(\varphi | \psi) && (\text{gerade gezeigt}) \\ &\equiv \neg\neg(\varphi \wedge \psi) && (\text{Definition von } |) \\ &\equiv \varphi \wedge \psi && (\text{Doppelnegation})\end{aligned}$$

Da $\{\neg, \wedge\}$ funktional vollständig, also auch $\{| \}$



Funktionale Vollständigkeit von Junktorenmengen

Nicht funktional vollständig z.B. $\{\wedge, \vee, \rightarrow\}$:

- Jede mit $\wedge, \vee, \rightarrow$ gebildete Formel φ erfüllt $f_\varphi(1, \dots, 1) = 1$

Beweis per Induktion über den Aufbau von φ :

- wenn $\varphi = x$, dann $V_1(\varphi) = 1$
- wenn $\varphi = \psi \wedge \vartheta$ und $V_1(\psi) = V_1(\vartheta) = 1$, dann $V_1(\varphi) = 1$
- analog für $\varphi = \psi \vee \vartheta$
und $\varphi = \psi \rightarrow \vartheta$
- Es gibt also keine zu $\neg x$ äquivalente Formel



- 1.1 Syntax und Semantik
- 1.2 Auswertung und Äquivalenz
- 1.3 Normalformen und funktionale Vollständigkeit
- 1.4 Erfüllbarkeit, Gültigkeit, Folgerbarkeit**
- 1.5 Tseitin-Transformation
- 1.6 Horn Logik
- 1.7 Resolution
- 1.8 Kompaktheit

Next



Erfüllbarkeit, Gültigkeit

Definition 1.16 (Erfüllbarkeit, Gültigkeit)

Eine Formel heißt

- **gültig** oder **Tautologie**, wenn sie von *jeder* Belegung wahr gemacht wird
- **erfüllbar**, wenn eine Belegung *existiert*, die sie wahr macht (sonst **unerfüllbar**)

Beispiele für **gültige** Formeln:

$$1 \quad x \vee \neg x \quad \neg(x \wedge y) \leftrightarrow \neg x \vee \neg y$$
$$(x \wedge y) \vee (\neg x \wedge \neg y) \vee (\neg x \wedge y) \vee (x \wedge \neg y)$$

Beispiele für **unerfüllbare** Formeln:

$$0 \quad x \wedge \neg x \quad x \wedge \neg y \wedge (x \rightarrow y) \quad (x \vee y) \wedge (\neg x \vee \neg y) \wedge (\neg x \vee y) \wedge (x \vee \neg y)$$



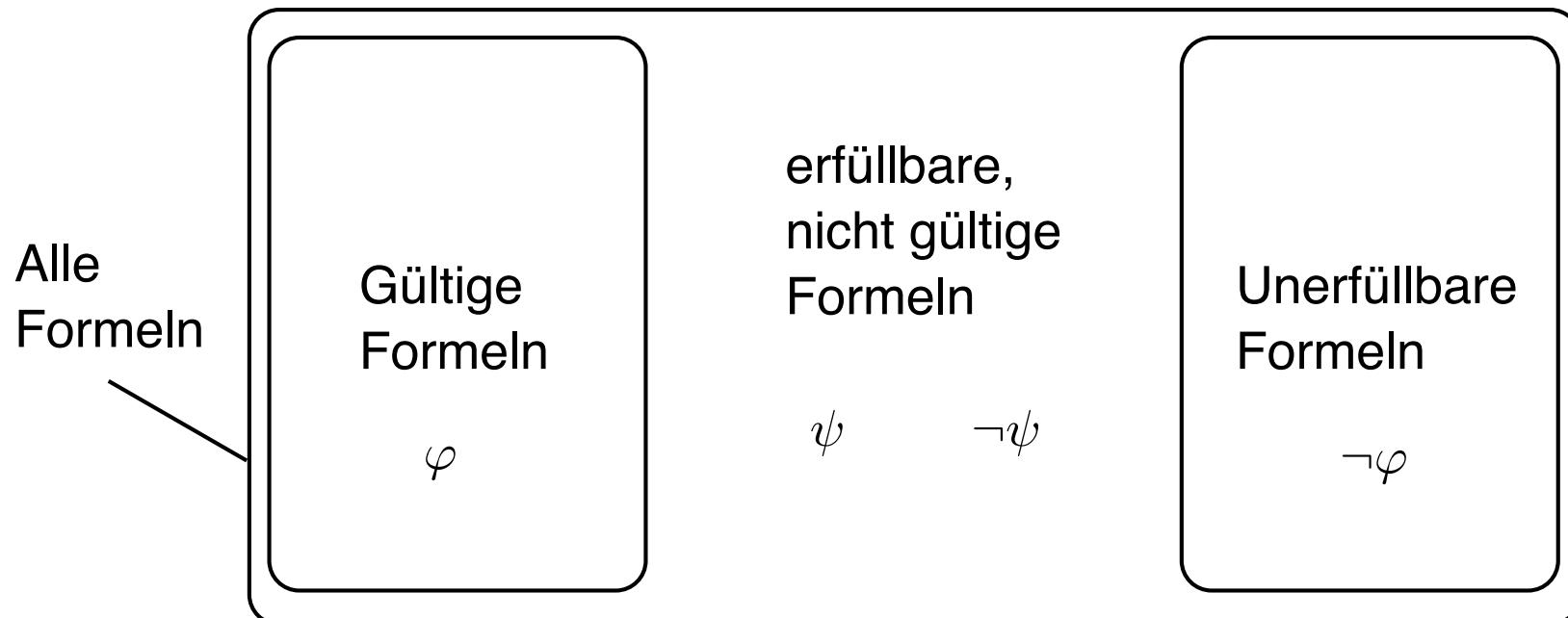
Dualität

Folgt direkt aus Definition Erfüllbarkeit/Tautologie + Semantik Negation:

Lemma 1.17 (Dualität Erfüllbarkeit, Gültigkeit)

Eine Formel φ ist

- gültig gdw. $\neg\varphi$ unerfüllbar ist.
- erfüllbar gdw. $\neg\varphi$ nicht gültig ist.



Erfüllbarkeits-, Gültigkeitsproblem

Definition 1.18 (Erfüllbarkeitsproblem, Gültigkeitsproblem)

Das Erfüllbarkeitsproblem der Aussagenlogik ist:

Gegeben: Aussagenlogische Formel φ

Frage: Ist φ erfüllbar?

Das Gültigkeitsproblem der Aussagenlogik ist:

Gegeben: Aussagenlogische Formel φ

Frage: Ist φ eine Tautologie?

Offensichtlicher, naiver Algorithmus für das Erfüllbarkeitsproblem:

Zähle alle 2^n Belegungen V für φ auf (wobei n Anzahl Variablen in φ).

Prüfe für jedes V in Linearzeit, ob $V \models \varphi$

Ähnlich für das Gültigkeitsproblem



SAT Problem

Das Erfüllbarkeitsproblem der Aussagenlogik

- wird auch „SAT“ genannt (von engl. *satisfiability*)
- ist eines der wichtigsten Probleme der Informatik, von dem vermutet wird, dass es sich nicht effizient lösen lässt

Wobei „effizient“ mit „in polynomieller Zeit“ gleichgesetzt wird

Eng verknüpft mit dem berühmtesten ungelöste Problem der theoretischen Informatik:

Das „P vs NP Problem“

Wird in der VL “Berechenbarkeit” näher behandelt

Im Folgenden die eng damit verwandte Exponentialzeitvermutung



Exponentialzeitvermutung (ETH)

Wichtige Vermutung aus der Algorithmen- und Komplexitätstheorie:

ETH Vermutung (Exponential Time Hypothesis)

Es gibt keinen Algorithmus für das Erfüllbarkeitsproblem von KNF-Formeln mit ≤ 3 Literalen pro Konjunkt und Laufzeit $2^{o(n)}$, wobei n die Anzahl der Variablen in der Eingabeformel ist.

Funktionen in $2^{o(n)}$ zum Beispiel: n^2 $2^{(\log n)^c}$ $2^{\frac{n}{\log n}}$ $2^{\sqrt{n}}$

Aber nicht zum Beispiel: 1.4^n

Grundlage für viele interessante Resultate in der theoretischen Informatik

Die ETH impliziert $P \neq NP$, umgekehrt ist das nicht bekannt



Erfüllbarkeit/Gültigkeit in KNF/DNF

Erfüllbarkeit von KNF-Formeln ist (wahrscheinlich) nicht leichter zu lösen als für beliebige Formeln, ebenso Gültigkeit von DNF-Formeln

Jeweils umgekehrt ist das aber nicht so:

Lemma 1.19 (Einfache Fälle)

DNF-Formel ist erfüllbar

gdw. es ein Disjunkt gibt, das erfüllbar ist

gdw. es ein Disjunkt gibt, das keine Literale der Form $x, \neg x$ enthält.

KNF-Formel ist gültig

gdw. jedes Konjunkt gültig ist

gdw. jedes Konjunkt zwei Literale der Form $x, \neg x$ enthält.

Beweis: Übung

T1.6



Folgerbarkeit

Definition 1.20 (Folgerbarkeit, Konsequenz)

Formel ψ folgt aus Formel φ wenn $V \models \varphi$ impliziert $V \models \psi$ für alle Beleg. V .
Wir nennen ψ dann Konsequenz von φ und schreiben $\varphi \models \psi$.

Beispiele: $x \wedge y \models x$ $x \models x \vee y$

Offensichtlich: $\varphi \equiv \psi$ gdw. $\varphi \models \psi$ und $\psi \models \varphi$

Lemma 1.21 (Folgerbarkeit und Gültigkeit)

Für alle Formeln φ, ψ gilt:

1. $\varphi \models \psi$ gdw. $\varphi \rightarrow \psi$ gültig ist (aka Deduktionstheorem)
2. φ ist gültig gdw. $1 \models \varphi$.



Modus Ponens

Wir definieren Folgerbarkeit auch für (potentiell unendliche) Formelmengen Γ :

$\Gamma \models \varphi$ wenn $V \models \Gamma$ impliziert $V \models \varphi$ für alle V .
d.h. $V \models \varphi$ für alle $\varphi \in \Gamma$.

Wir lesen Formelmengen also als **Konjunktion** der enthaltenen Formeln!

Zum Beispiel:

$$\{\varphi, \varphi \rightarrow \psi\} \models \psi \quad (\text{Modus Ponens})$$

Also: wenn φ und $\varphi \rightarrow \psi$ wahr sind, dann ist ψ wahr.

Der Modus Ponens ist **sehr bekannte Schlussregel**, ähnlich Syllogismen

Er spielt eine wichtige Rolle in der Logik, etwa im **Hilbert-Kalkül**



- 1.1 Syntax und Semantik
- 1.2 Auswertung und Äquivalenz
- 1.3 Normalformen und funktionale Vollständigkeit
- 1.4 Erfüllbarkeit, Gültigkeit, Folgerbarkeit
- 1.5 Tseitin-Transformation**
- 1.6 Horn-Logik
- 1.7 Resolution
- 1.8 Kompaktheit

Next



Tseitin-Transformation

Wir betrachten das Wandeln einer Formel φ in KNF ψ nochmal aus der Perspektive der Erfüllbarkeit (statt der Äquivalenz)

Manchmal genügt es, dass φ und ψ nur erfüllbarkeitsäquivalent sind:

$$\varphi \text{ erfüllbar gdw. } \psi \text{ erfüllbar}$$

Das ist ein großer Unterschied, z.B. sind

$$(\neg x \vee y) \wedge x \text{ und } z \vee \neg z$$

erfüllbarkeitsäquivalent, aber offensichtlich nicht äquivalent!

Erfüllbarkeitsäquivalenz ist beispielsweise hinreichend wenn es uns nur darum geht, die Erfüllbarkeit der ursprünglichen Formeln zu entscheiden

Eine erfüllbarkeitsäquivalente Formeln in NNF können wir in polynomieller Zeit konstruieren; die entstehende Formel ist nur linear groß



Tseitin-Transformation

Tseitin-Transformation:

Beispiel $\varphi = \neg(x \wedge y) \vee \neg z$

- Schritt 1: Einführen von neuen Variablen für nicht-atomare Teilformeln

x_1 für $x \wedge y$

x_2 für $\neg(x \wedge y)$

x_3 für $\neg z$

x_4 für $\neg(x \wedge y) \vee \neg z$

- Schritt 2: Bedeutung der neuen Variablen festlegen und Teilformeln ersetzen

$x_1 \leftrightarrow x \wedge y$

$x_2 \leftrightarrow \neg x_1$

$x_3 \leftrightarrow \neg z$

$x_4 \leftrightarrow x_2 \vee x_3$

- Schritt 3: Konjunktion bilden

$x_4 \wedge (x_1 \leftrightarrow x \wedge y) \wedge (x_2 \leftrightarrow \neg x_1) \wedge (x_3 \leftrightarrow \neg z) \wedge (x_4 \leftrightarrow x_2 \vee x_3)$



Tseitin-Transformation

Tseitin-Transformation:

Beispiel $\varphi = \neg(x \wedge y) \vee \neg z$

- Schritt 3: Konjunktion bilden

$$x_4 \wedge (x_1 \leftrightarrow x \wedge y) \wedge (x_2 \leftrightarrow \neg x_1) \wedge (x_3 \leftrightarrow \neg z) \wedge (x_4 \leftrightarrow x_2 \vee x_3)$$

- Schritt 4: jedes Konjunkt einzeln in **KNF** umformen

Zum Beispiel mittels Wahrheitstafel

Größe der konstruierten Formel ψ :

- Anzahl Konjunkte \leq Anzahl Teilformeln von $\varphi + 1$, also $\leq |\varphi| + 1$
- Jedes Konjunkt hat ≤ 3 Variablen, erzeugt also ≤ 8 Konjunkte mit je ≤ 3 Literalen - also 24 Literalvorkommen

Insgesamt hat ψ also Größe $\approx 24 \cdot (|\varphi| + 1) \in \mathcal{O}(|\varphi|)$

(wenn man die Länge der Variablen als 1 zählt)



Tseitin-Transformation

Theorem 1.27

Wenn ψ aus φ durch die Tseitin-Transformation hervorgeht, dann sind φ und ψ erfüllbarkeitsäquivalent (d.h. φ erfüllbar gdw. ψ erfüllbar).

Beweis: Im Folgenden sei x_ϑ die für Teilformel ϑ eingeführte Variable

“ \Rightarrow ”. Angenommen, φ ist erfüllbar in Modell V

Sei V' Erweiterung von V durch Setzen von

$$V'(x_\vartheta) = \begin{cases} 1 & \text{wenn } V \models \vartheta \\ 0 & \text{sonst} \end{cases} \quad \text{für alle } \vartheta \in \text{TF}(\varphi)$$

Wir zeigen, dass V' Modell von ψ ist (alle Konjunkte erfüllt).

T1.9

□



Tseitin-Transformation

Theorem 1.27

Wenn ψ aus φ durch die Tseitin-Transformation hervorgeht, dann sind φ und ψ erfüllbarkeitsäquivalent (d.h. φ erfüllbar gdw. ψ erfüllbar).

Beweis: Im Folgenden sei x_ϑ die für Teilformel ϑ eingeführte Variable

“ \Leftarrow ”. Angenommen, ψ ist erfüllbar; sei V Modell von ψ

Man zeigt per Induktion über die Struktur aller nicht-atomaren Teilformeln $\vartheta \in \text{TF}(\varphi)$:

Wenn $V(x_\vartheta) = 1$, dann $V \models \vartheta$

Details: Übung

Da x_φ Konjunkt von ψ ist gilt $V(x_\varphi) = 1$, also $V \models \varphi$.



Nachbemerkung Tseitin-Transformation

Warum sind φ und ψ **nicht** äquivalent?

Beispiel $\varphi = \neg(x \wedge y) \vee \neg z$

Ergebnis der Tseitin-Transformation:

$$x_4 \wedge \underline{(x_1 \leftrightarrow x \wedge y) \wedge (x_2 \leftrightarrow \neg x_1) \wedge (x_3 \leftrightarrow \neg z)} \wedge (x_4 \leftrightarrow x_2 \vee x_3)$$

Die folgende Belegung V :

$$x \mapsto 0 \quad y \mapsto 0 \quad z \mapsto 0 \quad x_i \mapsto 1 \text{ for } i \in \{1, \dots, 4\}$$

ist offensichtlich

- Modell von φ
- **kein** Modell von ψ , denn z.B. $V \not\models x_1 \leftrightarrow x \wedge y$



- 1.1 Syntax und Semantik
- 1.2 Auswertung und Äquivalenz
- 1.3 Normalformen und funktionale Vollständigkeit
- 1.4 Erfüllbarkeit, Gültigkeit, Folgerbarkeit
- 1.5 Tseitin-Transformation
- 1.6 Horn-Logik**
- 1.7 Resolution
- 1.8 Kompaktheit

Next



Horn-Formeln

Eine wichtige Klasse von Formeln mit guten Berechnungseigenschaften sind die Horn-Formeln (nach Alfred Horn).

Definition 1.22 (Horn-Formel)

Eine aussagenlogische Horn-Formel ist eine KNF-Formel $\varphi = \bigwedge_i \bigvee_j \ell_{i,j}$, wobei jedes Konjunktum $\bigvee_j \ell_{i,j}$ höchstens ein positives Literal enthält.

Beispiel: $(\neg x \vee \neg y \vee z) \wedge (\neg y \vee \neg z) \wedge x$

Vier mögliche Formen von Konjunkten (Horn-Klauseln):

Negative Literale + 1 positives Literal

Nur ein positives Literal

Nur negative Literale

(Gar keine Literale
 $\equiv 0$, daher uninteressant)



Horn-Formeln

Anschaulicher:

x		Fakt
$\neg x_1 \vee \dots \vee \neg x_k \vee x$	\equiv	Regel
$\neg x_1 \vee \dots \vee \neg x_k$	\equiv	Constraint

Beispiel Horn-Formel: Konjunktion von

$$\begin{array}{ll} \text{Regen} & \text{Schnee} \\ \text{Regen} \rightarrow \text{Niederschlag} & \text{Schnee} \rightarrow \text{Niederschlag} \\ \text{Regen} \rightarrow \text{Temp} \geq 0 & \text{Schnee} \rightarrow \text{Temp} < 0 \\ \text{Temp} \geq 0 \wedge \text{Temp} < 0 \rightarrow 0 & \end{array}$$

Hierbei sind „Regen“, „Schnee“, „Temp>0“, ... Aussagenvariablen



Erfüllbarkeitsproblem für Horn-Formeln

Theorem 1.23 (Effiziente Erfüllbarkeit)

Das Erfüllbarkeitsproblem für Horn-Formeln kann in Linearzeit gelöst werden.

Polyzeit-Algorithmus für Eingabe φ : („Markierungsalgorithmus“)

$V := \{x \in \text{VAR} \mid x \text{ ist Konjunkt von } \varphi\}$

while es gibt Konjunkt $x_1 \wedge \dots \wedge x_k \rightarrow x$ mit $\{x_1, \dots, x_k\} \subseteq V$ und $x \notin V$ **do**

$V := V \cup \{x\}$

Hier wird x als wahr „markiert“.

done

if es gibt ein Konjunkt $x_1 \wedge \dots \wedge x_k \rightarrow 0$ mit $\{x_1, \dots, x_k\} \subseteq V$ **then**

return „unerfüllbar“

else

return „erfüllbar“



Erfüllbarkeitsproblem für Horn-Formeln

Regen		Schnee		
Regen	\rightarrow	Niederschlag	\rightarrow	Niederschlag
Regen	\rightarrow	Temp ≥ 0	\rightarrow	Temp < 0
Temp $\geq 0 \wedge$	Temp < 0	\rightarrow	0	

$V := \{x \in \text{VAR} \mid x \text{ ist Konjunkt von } \varphi\}$

while es gibt Konjunkt $x_1 \wedge \dots \wedge x_k \rightarrow x$ mit $\{x_1, \dots, x_k\} \subseteq V$ und $x \notin V$ **do**

$V := V \cup \{x\}$

done

if es gibt ein Konjunkt $x_1 \wedge \dots \wedge x_k \rightarrow 0$ mit $\{x_1, \dots, x_k\} \subseteq V$ **then**

return „unerfüllbar“

else

return „erfüllbar“

T1.7



Erfüllbarkeitsproblem für Horn-Formeln

$V := \{x \in \text{VAR} \mid x \text{ ist Konjunkt von } \varphi\}$

while es gibt Konjunkt $x_1 \wedge \cdots \wedge x_k \rightarrow x$ mit $\{x_1, \dots, x_k\} \subseteq V$ und $x \notin V$ **do**

$V := V \cup \{x\}$

done

if es gibt ein Konjunkt $x_1 \wedge \cdots \wedge x_k \rightarrow 0$ mit $\{x_1, \dots, x_k\} \subseteq V$ **then**

return „unerfüllbar“

else

return „erfüllbar“

Bei Eingabe von Formel φ mit n Variablen terminiert der Algorithmus offensichtlich nach max. n Durchläufen der while-Schleife.

Er läuft damit in polynomieller Zeit



Erfüllbarkeitsproblem für Horn-Formeln

Lemma 1.24

Der Algorithmus antwortet „erfüllbar“ gdw. die Eingabeformel erfüllbar ist.

Wir unterscheiden im Folgenden nicht zwischen einer Belegung V (Abbildung $\text{VAR} \rightarrow \{0, 1\}$) und der Menge $\{x \mid V(x) = 1\}$

Die vom Algorithmus berechnete Menge V ist also eine Belegung



Erfüllbarkeitsproblem für Horn-Formeln

Lemma 1.24

Der Algorithmus antwortet „erfüllbar“ gdw. die Eingabeformel erfüllbar ist.

“ \Rightarrow ”

Angenommen, der Alg. berechnet Menge V und antwortet “erfüllbar”

Wir zeigen $V \models \varphi$

Die Konjunkte ψ von φ haben drei mögliche Formen:

1. $\psi = x$. Dann $x \in V$ (Anfang Alg.), also $V \models x$

2. $\psi = x_1 \wedge \dots \wedge x_k \rightarrow x$. Wenn $\{x_1, \dots, x_k\} \not\subseteq V$, dann $V \models \psi$

Andernfalls $x \in V$ (while-Schleife) und damit ebenfalls $V \models \psi$.

3. $\psi = x_1 \wedge \dots \wedge x_k \rightarrow 0$. Dann $\{x_1, \dots, x_k\} \not\subseteq V$ wegen

Antwort “erfüllbar”, also $V \models \psi$



Erfüllbarkeitsproblem für Horn-Formeln

Lemma 1.24

Der Algorithmus antwortet „erfüllbar“ gdw. die Eingabeformel erfüllbar ist.

“ \Leftarrow ”

Angenommen, φ ist erfüllbar

Man zeigt leicht per Induktion über die Anzahl Durchläufe der while-Schleife:

(*) $V \subseteq \widehat{V}$ für **alle** Modelle \widehat{V} von φ

$V := \{x \in \text{VAR} \mid x \text{ ist Konjunkt von } \varphi\}$

while es gibt Konjunkt $x_1 \wedge \dots \wedge x_k \rightarrow x$ mit $\{x_1, \dots, x_k\} \subseteq V$ und $x \notin V$ **do**

$V := V \cup \{x\}$

done



Erfüllbarkeitsproblem für Horn-Formeln

Lemma 1.24

Der Algorithmus antwortet „erfüllbar“ gdw. die Eingabeformel erfüllbar ist.

“ \Leftarrow ”

Angenommen, φ ist erfüllbar

Man zeigt leicht per Induktion über die Anzahl Durchläufe der while-Schleife:

(*) $V \subseteq \widehat{V}$ für alle Modelle \widehat{V} von φ

Sei $x_1 \wedge \dots \wedge x_k \rightarrow 0$ Konjunkt in φ . Zu zeigen: $\{x_1, \dots, x_k\} \not\subseteq V$.

Nimm im Gegenteil an, dass $\{x_1, \dots, x_k\} \subseteq V$ ⇒ Alg. antwortet
“erfüllbar”

Da φ erfüllbar, existiert ein Modell \widehat{V} von φ

Mit (*) gilt $\widehat{V} \models x_1 \wedge \dots \wedge x_k$, also $\widehat{V} \not\models x_1 \wedge \dots \wedge x_k \rightarrow 0$

Dies ist ein Widerspruch zu $\widehat{V} \models \varphi$ □



Minimale Modelle

Ein Modell V on φ ist **minimal** wenn gilt:

Für jedes Modell \widehat{V} von φ gilt: $V \subseteq \widehat{V}$

(alle Variablen, die V wahr macht, sind in **jedem** Modell von φ wahr)

Wenn Eingabeformel φ erfüllbar, dann ist die vom Algorithmus berechnete Belegung V ein **minimales** Modell

(wir haben das gezeigt als Teil des Korrektheitsbeweises)

Es folgt:

Korollar 1.25

Jede erfüllbare Horn-Formel hat ein minimales Modell.

Minimale Modelle haben zahlreiche interessante Eigenschaften.

Wir können sie z.B. für Beweise der **Nichtausdrückbarkeit** verwenden.



Ausdrucksstärke

Ausdrucksstärke von Horn-Formeln:

Welche AL-Formeln kann man als Horn-Formel ausdrücken, welche nicht?

Ausdrückbar z. B.:

$$x \rightarrow y \wedge z \equiv (x \rightarrow y) \wedge (x \rightarrow z)$$

$$x \vee y \rightarrow z \equiv (x \rightarrow z) \wedge (y \rightarrow z)$$

Nicht ausdrückbar z. B. $x \vee y$

(Beispiel: wir können ausdrücken, dass $\text{Temp} < 0 \wedge \text{Temp} \geq 0 \rightarrow 0$,
nicht aber, dass $\text{Temp} < 0 \vee \text{Temp} \geq 0$)

Lemma 1.26 (Nicht-Horn-Ausdrückbarkeit)

Keine Horn-Formel ist äquivalent zu $x \vee y$.

T1.8

Intuitiv: Horn-Formeln sind der **disjunktionsfreie Teil** von Aussagenlogik.



- 1.1 Syntax und Semantik
- 1.2 Auswertung und Äquivalenz
- 1.3 Normalformen und funktionale Vollständigkeit
- 1.4 Erfüllbarkeit, Gültigkeit, Folgerbarkeit
- 1.5 Tseitin-Transformation
- 1.6 Horn Logik
- 1.7 Resolution**
- 1.8 Kompaktheit

Next



Resolution

Wir lernen im Folgenden ein Verfahren kennen, mit dem man die Unerfüllbarkeit aussagenlogischer Formeln prüfen kann: **Resolution**

Es handelt sich dabei um einen sogenannten **Kalkül**, in dem mittels **Schlussfolgerungsregeln** neue Formeln hergeleitet werden.

bei Resolution nur eine einzige: **Resolventenbildung**

Es gibt viele verschiedene Arten von Kalkülen und wir werden noch zwei weitere kennenlernen: den Hilbert-Kalkül und den Sequenzenkalkül

Man kann Kalküle **algorithmisch implementieren** oder zu **theoretischen Zwecken** verwenden.



Resolution

Wir lernen im Folgenden ein Verfahren kennen, mit dem man die Unerfüllbarkeit aussagenlogischer Formeln prüfen kann: **Resolution**

Für Resolution ist es notwendig, dass die **Eingabeformel in KNF** ist.

Wie wir bereits wissen ist dies

1. keine Einschränkung der Allgemeinheit und
2. mit nur polynomiellem Mehraufwand zu erreichen
(Tseitin-Transformation)

Die Resolution stellt KNF Formeln jedoch in leicht anderer Form dar:
als **Klauselmengen**



Klauseln und Klauselmengen

Definition 1.28 (Klausel, Klauselmenge)

Eine **Klausel** ist eine endliche Menge von Literalen. Die leere Klausel bezeichnen wir mit \square .

Einer KNF-Formel $\varphi = \bigwedge_{i=1..n} \bigvee_{j=1..m_i} \ell_{ij}$ wird **Klauselmenge** $M(\varphi)$ wie folgt zugeordnet:

- i -te Disjunktion $\bigvee_{j=1..m_i} \ell_{ij}$ erzeugt Klausel $C_i = \{\ell_{i1}, \dots, \ell_{im_i}\}$
- $M(\varphi) = \{C_1, \dots, C_n\}$.

Beispiel: die Formeln

$$(x_1 \vee \neg x_2) \wedge x_3 \quad \text{und} \quad (\neg x_2 \vee x_1 \vee x_1) \wedge (x_3 \vee x_3)$$

entsprechen der Klauselmenge $M = \{\{x_1, \neg x_2\}, \{x_3\}\}$.



Klauseln und Klauselmengen

Praktischerweise haben **Klauseln**, da sie Mengen sind, verschiedene wichtige Eigenschaften von \vee bereits “eingebaut”:

Kommutativität:

$x_1 \vee x_2$ und $x_2 \vee x_1$ entsprechen beide $\{x_1, x_2\}$

Assoziativität:

$(x_1 \vee x_2) \vee x_3$ und $x_1 \vee (x_2 \vee x_3)$ entsprechen beide $\{x_1, x_2, x_3\}$

Idempotenz:

$(x_1 \vee x_1)$ entspricht $\{x_1\}$

Dasselbe gilt für **Klauselmengen** und \wedge .



Klauseln und Klauselmengen

Umgekehrt entspricht eine **Klausel C** der Formel $\bigvee_{\ell \in C} \ell$ und eine endliche **Klauselmenge M** der Formel $\bigwedge_{C \in M} \bigvee_{\ell \in C} \ell$.

Dies gibt uns in offensichtlicher Weise auch eine **Semantik für Klauseln und Klauselmengen**.

Wir können also Begriffe wie Erfüllbarkeit und Äquivalenz für Klauseln und Klauselmengen verwenden.

Beachte:

- \square entspricht der „leeren Disjunktion“ und ist unerfüllbar
- jede Klauselmenge, die \square enthält, ist unerfüllbar



Komplementäre Literale

Eine zentrale Beobachtung:

Betrachte zwei Klauseln C_1 und C_2 mit $x \in C_1$ und $\neg x \in C_2$.

Wir nennen x und $\neg x$ komplementäre Literale.

Aus der Klauselmenge $\{C_1, C_2\}$ folgt die Klausel $(C_1 \setminus \{x\}) \cup (C_2 \setminus \{\neg x\})$.

Beispiel: aus $C_1 = \{x_1, x_3, x_4\}$ und $C_2 = \{\neg x_2, \neg x_4\}$ folgt $\{x_1, x_3, \neg x_2\}$

Denn für jede Belegung V , die C_1 und C_2 wahr macht, gilt:

$V(x_4) = 0$ Weil $V \models C_1$ gilt dann: $V \models x_1 \vee x_3$

oder $V(x_4) = 1$ Weil $V \models C_2$ gilt dann: $V \models \neg x_2$

Insgesamt also $V \models x_1 \vee x_3 \vee \neg x_2$.

Notation für die
Negation von Literalen:

$$\neg \ell := \begin{cases} \neg x & \text{falls } \ell = x \\ x & \text{falls } \ell = \neg x \quad (\text{statt } \neg \neg x!) \end{cases}$$

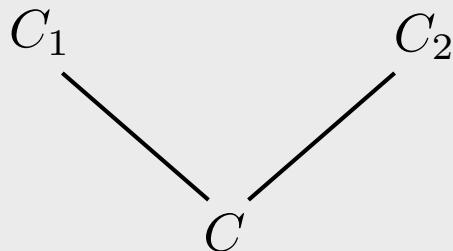


Resolventen

Definition 1.29 (Resolvente)

Seien C_1, C_2 Klauseln. Klausel C ist **Resolvente** von C_1 und C_2 gdw. es Literal ℓ gibt mit $\ell \in C_1$, $\neg\ell \in C_2$ und $C = (C_1 \setminus \{\ell\}) \cup (C_2 \setminus \{\neg\ell\})$.

Wir schreiben dann:



Beispiele:

$$\{x_1, x_3, x_4\}$$

$$\{\neg x_2, \neg x_4\}$$

$$\{x_1\}$$

$$\{\neg x_1\}$$

$$\{x_1, x_3, \neg x_2\}$$

□

$$\{x_1, x_2\}$$

$$\{\neg x_1, \neg x_2\}$$

$$\{x_2, \neg x_2\}$$

$$\{x_2, \neg x_2\}$$

(stets nur ein
kompl. Literal
resolvieren)

$$\{x_2, \neg x_2\}$$

(mögl., aber
nicht sinnvoll)



Resolutionslemma

Lemma 1.30 (Resolutionslemma)

Sei M eine Klauselmenge, $C_1, C_2 \in M$ und C Resolvente von C_1 und C_2 . Dann $M \equiv M \cup \{C\}$.

Beweis: zu zeigen ist, dass für all Belegungen V gilt:

$$V \models M \quad \text{gdw.} \quad V \models M \cup \{C\}$$

“ \Leftarrow ” ist trivial, weil $M \subseteq M \cup \{C\}$

“ \Rightarrow ” Angenommen $V \models M$. Es ist

$$C = (C_1 \setminus \{\ell\}) \cup (C_2 \setminus \{\neg\ell\}) \text{ mit } \ell \in C_1 \text{ und } \neg\ell \in C_2$$

Unterscheide zwei Fälle:

1. $V(\ell) = 1$. Dann folgt aus $V \models C_2$ auch $V \models C_2 \setminus \{\neg\ell\}$, also $V \models C$.
2. $V(\ell) = 0$. Dann folgt aus $V \models C_1$ auch $V \models C_1 \setminus \{\ell\}$, also $V \models C$.

□



Resolventenbildung

Definition 1.31 (Res)

Für jede Klauselmenge M sei

- $\text{Res}(M) := M \cup \{C \mid C \text{ Resolvente zweier Klauseln aus } M\}$
- $\text{Res}^0(M) := M, \quad \text{Res}^{i+1}(M) := \text{Res}(\text{Res}^i(M))$
- $\text{Res}^*(M) := \bigcup_{i \geq 0} \text{Res}^i(M)$

Beispiel: $\varphi = x_1 \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge \neg x_3$

T1.10



Analyse des Resolutionskalküls

Resolution ist Algorithmus für (Un)Erfüllbarkeit:

$\square \in \text{Res}^*(M)$ gdw. Eingabemenge M unerfüllbar

Im Allgemeinen:

- Ein Kalkül **terminiert**,
wenn sich für jede Eingabe nur endlich viele Formeln erzeugen lassen.
- Ein Kalkül heißt **korrekt**,
wenn sich nur gewünschte Formeln erzeugen lassen.
- Ein Kalkül heißt **vollständig**,
wenn sich jede gewünschte Formel erzeugen lässt.

Im Fall von Resolution also:

\square gewünscht gdw. Eingabeformel unerfüllbar



Terminierung und Korrektheit

Lemma 1.32 (Terminierung)

Für jede endliche Klauselmenge M gilt:

Es gibt ein $i \geq 1$ mit $\text{Res}^i(M) = \text{Res}^{i+1}(M) = \text{Res}^*(M)$.

Beweis: $\text{Res}^0(M) \subseteq \text{Res}^1(M) \subseteq \dots$ und

aus n Literalen lassen sich nur 2^n Klauseln bilden

Theorem 1.33 (Korrektheit)

Wenn M endliche Klauselmenge mit $\square \in \text{Res}^*(M)$, dann M unerfüllbar.

Beweis:

Wegen $\square \in \text{Res}^*(M)$ ist $\text{Res}^*(M)$ unerfüllbar

Nach Lemma 1.32 gibt es k mit $\text{Res}^k(M) = \text{Res}^*(M) \Rightarrow \text{Res}^k(M)$ unerfüllbar

Aus Resolutionslemma folgt leicht:

$\text{Res}^i(M) \equiv M$ für alle $i \geq 0 \Rightarrow M$ unerfüllbar \square

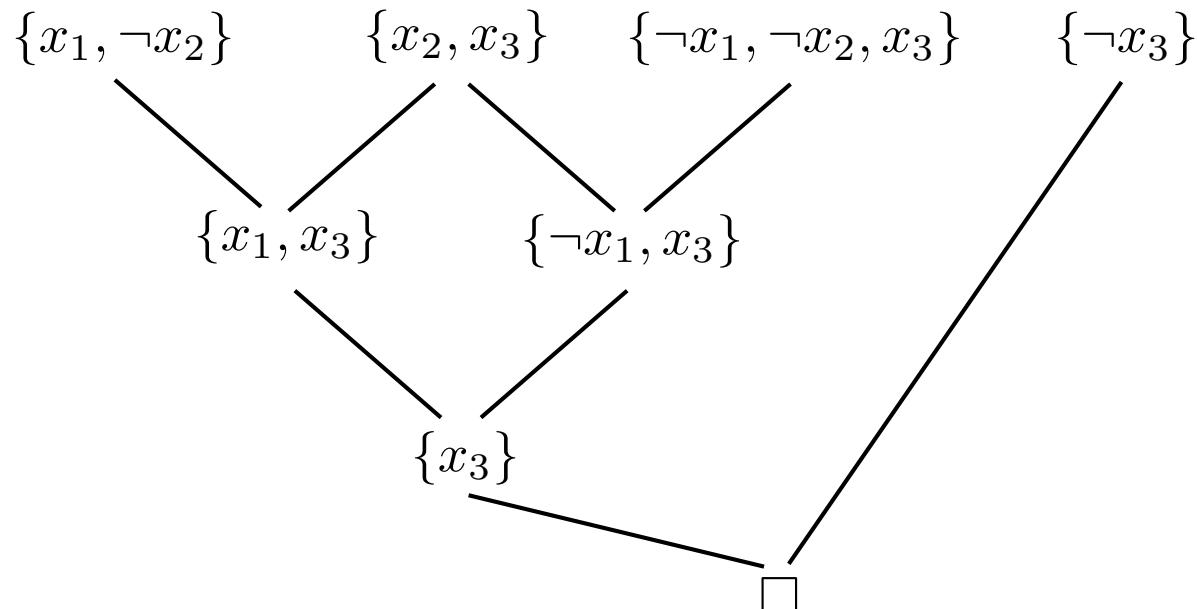


Resolutionsbeweise

Resolutionsbeweis:

Darstellung der Ableitung von \square mittels Resolventen als Graph

$$\varphi = (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge \neg x_3$$



Beachte:

$\text{Res}^*(M)$ entspricht nicht **einem** Resolutionsbeweis, sondern enthält **alle** Resolutionsbeweise für die Unerfüllbarkeit von M
(und auch Klauseln, die in keinem Resolutionsbeweis vorkommen)

Resolutionsbeweise

Definition 1.34 (Resolutionsbeweis)

Sei M eine endliche Klauselmenge. Ein Resolutionsbeweis für M ist eine Folge C_1, \dots, C_m von Klauseln, für die gilt:

- Für alle $i \leq m$ ist entweder $C_i \in M$
oder C_i ist Resolvente zweier C_j, C_k mit $j, k < i$.
- $C_m = \square$.

Die Länge eines Resolutionsbeweises C_1, \dots, C_m ist m .

Folgendes ist sehr leicht zu beweisen:

Lemma 1.35

Für alle endlichen Klauselmengen M gilt:

Es gibt einen Resolutionsbeweis für M gdw. $\square \in \text{Res}^*(M)$.



Vollständigkeit

Theorem 1.36 (Resolutionssatz, Robinson 1965)

Wenn endliche Klauselmenge M unerfüllbar ist, dann $\square \in \text{Res}^*(M)$.

Beweis:

Für Klauselmenge M und Variable x definiere neue Klauselmenge

$$M_x^+ := \{C \setminus \{\neg x\} \mid C \in M \text{ und } x \notin C\}$$

Entspricht dem Fall $V(x) = 1$:

- lösche alle Klauseln C mit $x \in C$ und
- lösche $\neg x$ aus den verbliebenen Klauseln

kein x mehr!



Z.B.: $M = \{\{x, y\}, \{\neg x, z\}, \{y, z\}\}$ $M_x^+ = \{\{z\}, \{y, z\}\}$

Analog entspricht M_x^- dem Fall $V(x) = 0$:

$$M_x^- := \{C \setminus \{x\} \mid C \in M \text{ und } \neg x \notin C\}$$



Vollständigkeit

Wir beweisen den Resolutionssatz nicht direkt sondern stattdessen folgendes Lemma:

Lemma 1.37

Wenn M unerfüllbar ist, dann gilt für jede Variable x , die in M vorkommt:

- (1) M_x^+ **und** M_x^- sind unerfüllbar .
- (2) $\square \in \text{Res}^*(M)$ **oder** $\{\neg x\} \in \text{Res}^*(M)$
- (3) $\square \in \text{Res}^*(M)$ **oder** $\{x\} \in \text{Res}^*(M)$

Daraus folgt leicht der Resolutionssatz:

Sei M unerfüllbar. Konsequenz von (2) und (3):

$$\square \in \text{Res}^*(M) \text{ oder } \{x\}, \{\neg x\} \in \text{Res}^*(M)$$

Das impliziert aber (per Resolventenbildung) auch $\square \in \text{Res}^*(M)$

T1.11



Vollständigkeit

- (1) Wenn M unerfüllbar, dann M_x^+ und M_x^- unerfüllbar
- (2) Wenn M unerfüllbar, dann $\square \in \text{Res}^*(M)$ oder $\{\neg x\} \in \text{Res}^*(M)$
- (3) Wenn M unerfüllbar, dann $\square \in \text{Res}^*(M)$ oder $\{x\} \in \text{Res}^*(M)$

Beweis von (2) + (3) **simultan** per Induktion über $|\text{Var}(M)|$

Angenommen, M ist unerfüllbar.

Anzahl Variablen in M

Anfang (2): $|\text{Var}(M)| = 0$

Dann $M = \emptyset$ oder $M = \{\square\}$

\emptyset ist aber erfüllbar, also $M = \{\square\}$ ✓

Schritt (2): $|\text{Var}(M)| = n + 1$

Wähle beliebige Variable x , die in M vorkommt, betrachte M_x^+

Es gilt $|\text{Var}(M_x^+)| = n$ weil x eliminiert wurde

Punkt (1) liefert: M_x^+ ist unerfüllbar

IV für Punkte (2) + (3) liefert also: $\square \in \text{Res}^*(M_x^+)$

Also gibt es Resolutionsbeweis C_1, \dots, C_m für M_x^+

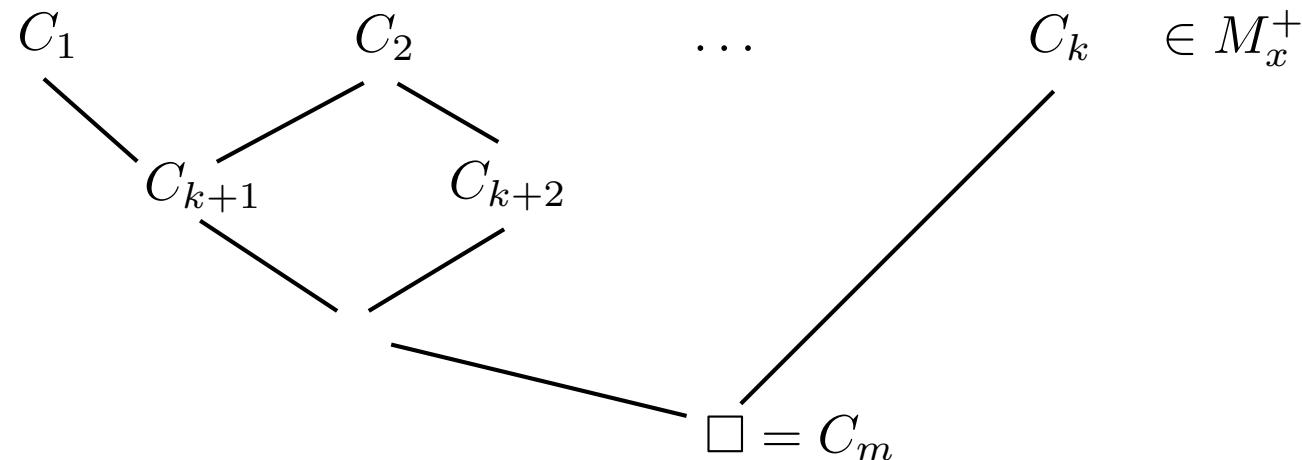


Vollständigkeit

(2) Wenn M unerfüllbar, dann $\square \in \text{Res}^*(M)$ oder $\{\neg x\} \in \text{Res}^*(M)$

(3) Wenn M unerfüllbar, dann $\square \in \text{Res}^*(M)$ oder $\{x\} \in \text{Res}^*(M)$

Resolutionsbeweis C_1, \dots, C_m für M_x^+ :



Nach Definition von M_x^+ gilt für jedes C_i aus der obersten Zeile:

$$C_i \in M \text{ oder } C_i \cup \{\neg x\} \in M$$

Fall 1: $\{C_1, \dots, C_k\} \subseteq M$

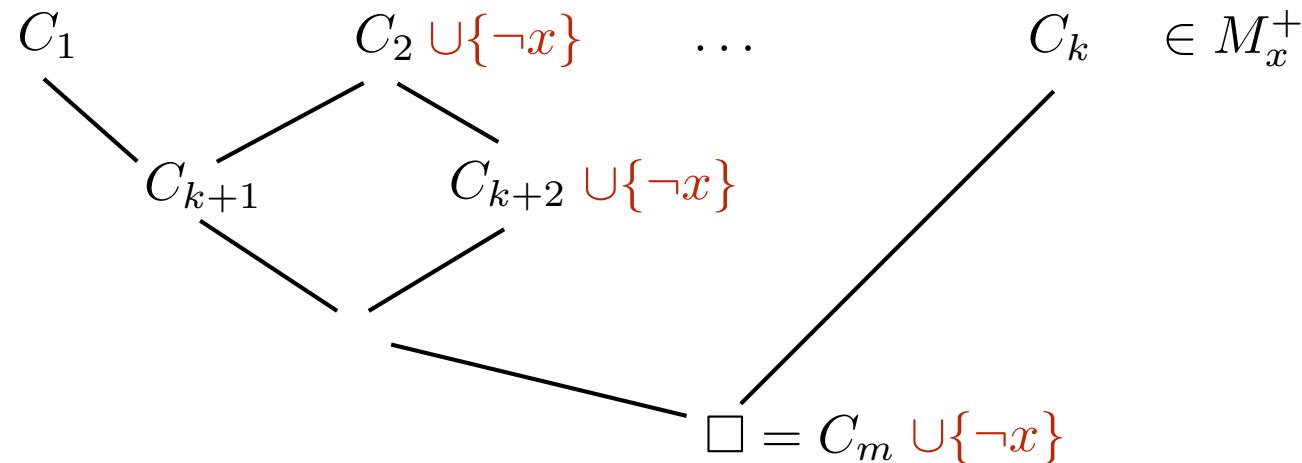
Dann ist C_1, \dots, C_m auch Resolutionsbeweis für M , also $\square \in \text{Res}^*(M)$ ✓

Vollständigkeit

(2) Wenn M unerfüllbar, dann $\square \in \text{Res}^*(M)$ oder $\{\neg x\} \in \text{Res}^*(M)$

(3) Wenn M unerfüllbar, dann $\square \in \text{Res}^*(M)$ oder $\{x\} \in \text{Res}^*(M)$

Resolutionsbeweis C_1, \dots, C_m für M_x^+ :



Fall 2: Für mind. ein C_i ist $C_i \cup \{\neg x\} \in M$ ($1 \leq i \leq k$)

Wiedereinführen von $\neg x$ im Resolutionsbeweis!

Aus Resolutionsbeweis von \square wird Resolutionsbeweis von $\neg x$

Es folgt $\{\neg x\} \in \text{Res}^*(M)$. ✓

Beweis von (3) analog via M_x^-

□

Resolution als Algorithmus

Wir erhalten folgenden Algorithmus für Erfüllbarkeit in der Aussagenlogik:

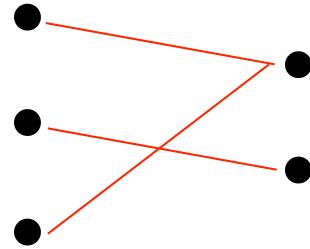
```
R0 := M  
i := 0  
repeat  
    i := i + 1  
    Ri := Res(Ri-1)  
    if  $\square \in R_i$  then return „unerfüllbar“  
until Ri = Ri-1  
return „erfüllbar“
```



Exkurs: Beweislänge

Es gibt Klauselmengen, für die jeder Resolutionsbeweis exponentiell lang ist.

Schubfachprinzip: wenn man $n + 1$ Objekte auf n Schubladen verteilt,
enthält mindestens eine Schublade 2 Objekte



Als aussagenlogische Formel: $(x_{ij} = \text{Objekt } i \text{ ist in Schubfach } j)$

$$(x_{11} \vee x_{12}) \wedge (x_{21} \vee x_{22}) \wedge (x_{31} \vee x_{32}) \rightarrow (x_{11} \wedge x_{21}) \vee (x_{11} \wedge x_{31}) \vee (x_{21} \wedge x_{31}) \\ \vee (x_{12} \wedge x_{22}) \vee (x_{12} \wedge x_{32}) \vee (x_{22} \wedge x_{32})$$

Da das Schubfachprinzip gültig ist, ist die Negation dieser Formel unerfüllbar.

Exkurs: Beweislänge

Für $n + 1$ Objekte und n Schubfächer, negiert, in KNF gewandelt:

$$\varphi_n = \bigwedge_{i=1..n+1} \bigvee_{j=1..n} x_{ij} \wedge \bigwedge_{j=1..n} \bigwedge_{1 \leq i < i' \leq n+1} (\neg x_{ij} \vee \neg x_{i'j})$$

Gibt Folge von Formeln $\varphi_1, \varphi_2, \varphi_3, \dots$

Formellänge wächst nur polynomiell:

φ_n hat $\mathcal{O}(n^3)$ Klauseln mit je $\leq n$ Variablen

Beweislänge wächst aber exponentiell:

Theorem 1.38 (Haken 1985)

Es gibt Konstanten $k_1, k_2 > 1$ so dass für alle $n \geq k_1$:

jeder Resolutionsbeweis für φ_n hat Länge $\geq (k_2)^n$

Andere Kalküle haben aber u. U. kurze Beweise für diese Formelklasse.



Exkurs: Beweislänge

Aber selbst wenn es kurze Beweise gibt, sind sie i. allg. nicht leicht zu finden:

Es ist nicht möglich, einen kürzesten Resolutionsbeweis effizient zu finden, d.h. in Zeit **polynomiell in der Beweislänge**

Theorem 1.39 (AtseriasMüller 2019)

Es gibt keinen Algorithmus, der

1. gegeben eine unerfüllbare Klauselmenge M einen Resolutionsbeweis für M ausgibt und
2. dessen Zeitbedarf polynomiell in der Größe von M und dem kürzesten Resolutionsbeweis für M ist.

es sei denn P=NP. (d.h.: mit sehr großer Wahrscheinlichkeit nicht)



Einheitsresolution

Wir betrachten nun eine Variante: die sogenannte **Einheitsresolution**

Dabei muss in jedem Resolutionsschritt eines der beteiligten Literale ein **positives Einheitsliteral** sein.

Einheitsresolution ist

- immernoch **korrekt** aber
- im allgemeinen **nicht** mehr **vollständig**

Für Mengen von **Hornklauseln** **ist** Einheitsresolution aber vollständig!

In der Tat besteht ein enger Zusammenhang zum Markierungsalgorithmus

Mit einer weiteren Modifikation (**geordnete** Einheitsresolution) entsteht ein **Polynomialzeitalgorithmus** für die Erfüllbarkeit von Hornformeln



Einheitsresolution

Definition 1.40 (Hornklausel)

Eine Klausel ist eine Hornklausel, wenn sie höchstens ein positives Literal enthält.

Beachte: \square , $\{x\}$, $\{\neg x\}$ sind also (spezielle) Horn-Klauseln

Definition 1.41 (Einheitsresolvente)

Klausel C ist Einheitsresolvente von Klauseln C_1 und C_2 wenn folgendes gilt:

1. C ist Resolvente von C_1 und C_2 und
2. C_1 hat die Form $\{x\}$.

Wir setzen

$$\text{ERes}(M) := M \cup \{C \mid C \text{ Einheitsresolvente zweier Klauseln aus } M\}$$

und definieren $\text{ERes}^i(M)$ und $\text{ERes}^*(M)$ analog zu $\text{Res}^i(M)$ und $\text{Res}^*(M)$.



Einheitsresolution

Für Mengen von Hornklauseln ist Einheitsresolution vollständig:

Theorem 1.42 (Resolutionssatz für Einheitsresolution)

Eine endliche Menge M von Hornklauseln ist unerfüllbar gdw. $\square \in \text{ERes}^*(M)$

„ \Leftarrow “ Folgt aus der Korrektheit der „normalen“ Resolution.

„ \Rightarrow “ Wir verwenden Vollständigkeit des Markierungsalgorithmus. Sei

$$V^0 = \{x \mid M \text{ enthält } \{x\}\}$$

$$V^{i+1} = V^i \cup \{x \mid \exists x_1, \dots, x_k \in V^i : M \text{ enthält } \{\neg x_1, \dots, \neg x_k, x\}\}$$

$$V^* = \bigcup_{i \geq 0} V^i$$

$$V := \{x \in \text{VAR} \mid x \text{ ist Konjunkt von } \varphi\}$$

while es gibt Konjunkt $x_1 \wedge \dots \wedge x_k \rightarrow x$ mit $\{x_1, \dots, x_k\} \subseteq V$ und $x \notin V$ **do**

$$V := V \cup \{x\}$$

done



LEIPZIG

Einheitsresolution

Für Mengen von Hornklauseln ist Einheitsresolution vollständig:

Theorem 1.42 (Resolutionssatz für Einheitsresolution)

Eine endliche Menge M von Hornklauseln ist unerfüllbar gdw. $\square \in \text{ERes}^*(M)$

„ \Leftarrow “ Folgt aus der Korrektheit der „normalen“ Resolution.

„ \Rightarrow “ Wir verwenden Vollständigkeit des Markierungsalgorithmus. Sei

$$V^0 = \{x \mid M \text{ enthält } \{x\}\}$$

$$V^{i+1} = V^i \cup \{x \mid \exists x_1, \dots, x_k \in V^i : M \text{ enthält } \{\neg x_1, \dots, \neg x_k, x\}\}$$

$$V^* = \bigcup_{i \geq 0} V^i$$

Korrektheit + Vollständigkeit des Markierungsalgorithmus liefert:

(a) M ist unerfüllbar gdw. $\exists x_1, \dots, x_k \in V^* : M \text{ enthält } \{\neg x_1, \dots, \neg x_k\}$.

if es gibt ein Konjunkt $x_1 \wedge \dots \wedge x_k \rightarrow 0$ mit $\{x_1, \dots, x_k\} \subseteq V$ **then**
return „unerfüllbar“



Einheitsresolution

Auf Hornklauseln ist Einheitsresolution ausreichend:

Theorem 1.42 (Resolutionssatz für Einheitsresolution)

Eine endliche Menge M von Hornklauseln ist unerfüllbar gdw. $\square \in \text{ERes}^*(M)$

„ \Rightarrow “ Wir verwenden Vollständigkeit des Markierungsalgorithmus.

(a) M ist unerfüllbar gdw. $\exists x_1, \dots, x_k \in V^* : M$ enthält $\{\neg x_1, \dots, \neg x_k\}$.

Wir zeigen als nächstes:

(b) $x \in V^* \Rightarrow \{x\} \in \text{ERes}^*(M)$

In Worten:

Wenn der Markierungsalgorithmus x wahr macht, dann kann die Klausel $\{x\}$ mittels Einheitsresolution erzeugt werden.

T1.13



Einheitsresolution

Auf Hornklauseln ist Einheitsresolution ausreichend:

Theorem 1.42 (Resolutionssatz für Einheitsresolution)

Eine endliche Menge M von Hornklauseln ist unerfüllbar gdw. $\square \in \text{ERes}^*(M)$

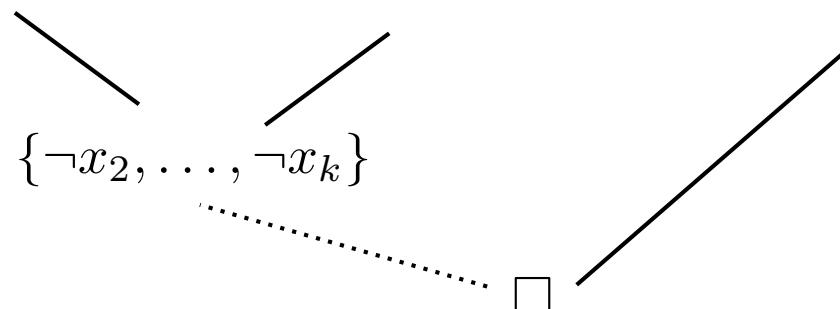
„ \Rightarrow “ Wir verwenden Vollständigkeit des Markierungsalgorithmus.

- (a) M ist unerfüllbar gdw. $\exists x_1, \dots, x_k \in V^* : M$ enthält $\{\neg x_1, \dots, \neg x_k\}$.
- (b) $x \in V^* \Rightarrow \{x\} \in \text{ERes}^*(M)$

Sei M unerfüllbar

- (a) + (b): es gibt $\{\neg x_1, \dots, \neg x_k\} \in M$ mit $\{x_1\}, \dots, \{x_k\} \in \text{ERes}^*(M)$

Dann: $\{\neg x_1, \dots, \neg x_k\}$ $\{x_1\}$ \dots $\{x_k\}$



□



Einheitsresolution

Auf Hornklauseln ist Einheitsresolution ausreichend:

Theorem 1.42 (Resolutionssatz für Einheitsresolution)

Eine endliche Menge M von Hornklauseln ist unerfüllbar gdw. $\square \in \text{ERes}^*(M)$

Der Beweis zeigt auch, dass es für jede unerfüllbare Horn-Formel φ einen (Einheits)Resolutionsbeweis gibt, der höchstens $|\varphi|^2$ Schritte hat:

- die Anzahl Variablen in V^* ist begrenzt durch $|\varphi|$
- für jede Variable in V^* und für \square Resolutionsbeweis, dessen Länge durch max. Klauselgröße beschränkt ist.

Einheitsresolution ist aber dennoch **kein** Polynomialzeit-Verfahren!

Das liegt daran, dass $\text{ERes}^*(M)$ „alle“ Resolutionsbeweise für M enthält“

T1.14



Geordnete Einheitsresolution

Sei $<$ beliebige totale Ordnung der Menge aller Variablen

Wir modifizieren Einheitsresolution so, dass nur mit dem bzgl. ' $<$ ' kleinsten Literal einer Klausel resoviert werden darf

T1.15

Definition 1.43 (Geordnete Einheitsresolvente)

Klausel C ist geordnete Einheitsresolvente von Klauseln C_1 und $\{x\}$ wenn folgendes gilt:

1. C ist Einheitsresolvente von C und $\{x\}$ und
2. für jedes $\neg y \in C$ gilt: $x < y$.

OERes(M), OEResⁱ(M) und OERes^{*}(M) sind wie erwartet definiert

Effekt: Jede Klausel kann nur ein einziges mal resoviert werden!

Beachte: die Variablenordnung $<$ ist beliebig aber fest!



Geordnete Einheitsresolution

Theorem 1.44 (Geordnete Einheitsresolution in Polynomialzeit)

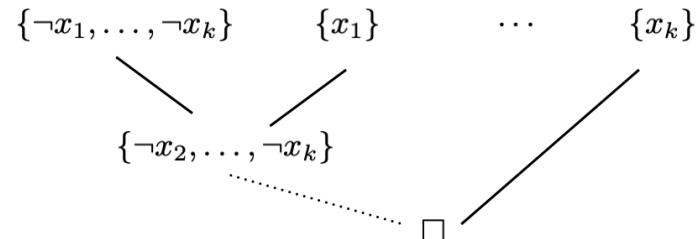
Sei M endliche Menge von Hornklauseln. Dann ist

1. M unerfüllbar gdw. $\square \in \text{OERes}^*(M)$
2. $\text{OERes}^*(M)$ in polynomieller Zeit berechenbar.

Beweis:

1. Derselbe Beweis wie für ungeordnete Einheitsresolution funktioniert:

Alle verwendeten Resolutionsbeweise immernoch möglich,
müssen nur ggf. in der Reihenfolge angepasst werden:



2. Da jede Klausel nur einmal resovliert werden kann, können maximal so viele Klauseln erzeugt werden, wie es Literalvorkommen in M gibt \square



SAT-Solver

Erfüllbarkeit in Aussagenlogik nennt man auch das **SAT-Problem**.

Obwohl SAT ein schwieriges Problem ist, gibt es heute **SAT-Solver**, die auch große Formeln (Tausende von Variablen) lösen können.

Dies ist deshalb von großer Bedeutung, weil sich viele **kombinatorische Probleme** in sehr **natürlicher Weise** in SAT kodieren lassen, z.B.:

- Produktkonfigurierung
- Hardware-Verifikation (Bounded Model Checking)
- Software-Verifikation
- Automatisches Planen
- Scheduling (z.B. Stundenpläne)



Beispiel Repräsentation

		Klasse a)	Klasse b)
Müller	Mathe	Mathe	Deutsch
Schmidt	Deutsch	Deutsch	Deutsch
Körner		Mathe	Mathe

Jeder Lehrer mindestens 2 Stunden

Wir verwenden Variablen der Form

$$x_{sk}^L \text{ mit } L \in \{\text{M, S, K}\}, s \in \{\text{I, II, III}\}, k \in \{a, b\}$$

Zum Beispiel repräsentiert die Variable $x_{\text{II}a}^{\text{M}}$ die Aussage

„Müller unterrichtet Klasse a) in Stunde II“

Wir finden aussagenlogische Formel φ deren erfüllende Belegungen genau den Lösungen des Zeitplanungsproblems entsprechen

SAT-Solver

Moderne SAT-Solver basieren auf einem Algorithmus namens **DPLL** (benannt nach den Entwicklern Davis-Putnam-Logemann-Loveland)

DPLL ist nicht dasselbe wie Resolution, hat sich aber daraus entwickelt

Wirklich effizient werden SAT-Solver erst durch zahlreiche raffinierte (und teils recht komplizierte) **Optimierungen**

Konkrete Systeme wie **Lingeling**, **Minisat**, **Glucose**, **zchaff**, **precosat**, **Sat4J** treten in jährlichen **SAT competitions** gegeneinander an.



In der Basisversion ist DPLL ein einfacher rekursiver Algorithmus

Wie auch die Resolution arbeitet er auf **Klauselmengen**

Gestartet auf Klauselmenge M :

- Wähle Variable x , setze $x \mapsto 1$
- Vereinfache M zu M_x^+ : (siehe Beweis Resolutionssatz)
 - entferne alle Klauseln, die x enthalten
 - entferne $\neg x$ aus verbleibenden Klauseln
- Prüfe M_x^+ auf Erfüllbarkeit (rekursiver Aufruf)
 - wenn ja, gib „erfüllbar“ aus
- sonst probiere $x \mapsto 0$ und M_x^-

Der DPLL-Algorithmus

Hier nochmal konkret als Pseudocode:

Function DPLL(M):

```
    if  $\square \in M$  then return unerfüllbar
    if  $M = \emptyset$  then return erfüllbar

    Wähle Variable  $x$ 
    if DPLL( $M_x^+$ ) then return erfüllbar
    else if DPLL( $M_x^-$ ) then return erfüllbar
    else return unerfüllbar
```

Optimierungen beschränken den Suchraum geeignet



DPLL – Hauptideen

Elementare DPLL-Optimierungen:

Unit Propagation (Einheitsresolution)

- Belege so früh wie möglich Einheitsklauseln $\{\ell\}$ entsprechend
 - ~~ Lösche alle Klauseln, die ℓ enthalten
 - ~~ Lösche $\neg\ell$ aus allen übrigen Klauseln

Pure Literal Elimination

- Literal ℓ ist *pur* in M , wenn M nur ℓ und nicht $\neg\ell$ enthält
- Pure Literale tragen nichts zur Unerfüllbarkeit von M bei
(Setzen von $V(\ell) = 1$ macht alle Klauseln mit ℓ wahr)
 - ~~ Lösche alle Klauseln, die ℓ enthalten

Optimierungen werden in **jedem** Unteraufruf angewendet („Dominoeffekt“)



DPLL mit einfachen Optimierungen

Function DPLL(M):

```
while  $M$  enthält Einheitsklausel  $\{\ell\}$  do
    Lösche alle Klauseln aus  $M$ , die  $\ell$  enthalten           //Unit Prop.
    | Lösche  $\neg\ell$  aus allen übrigen Klauseln
if  $\square \in M$  then return unerfüllbar
while  $M$  enthält pures Literal  $\ell$  do
    Lösche alle Klauseln aus  $M$ , die  $\ell$  enthalten       //Pure Lit Elim.
if  $M = \emptyset$  then return erfüllbar
Wähle Variable  $x$ 
if DPLL( $M \cup \{\{x\}\}$ ) then return erfüllbar
else if DPLL( $M \cup \{\{\neg x\}\}$ ) then return erfüllbar
else return unerfüllbar
```

T1.16



Einige Komplexere Optimierungen

- **Variablenheuristik**

Versuche, die **nächste Entscheidungsvariable geschickt so zu wählen**, dass möglichst schnell erfüllende Belegung gefunden wird (und ob zuerst 0 oder 1 probiert wird) Z.B.: **oft verwendete Variable zuerst**

- **Klausel-Lernen**

Wenn **nicht-erfüllende Belegung** entstanden ist, ist es manchmal möglich, **wenige Variablen zu identifizieren**, die dafür „verantwortlich“ sind
Z.B.: Belegung erfüllt Formel nicht weil

$$x_3 = 0 \text{ und } x_{17} = 1 \text{ und } x_{42} = 1$$

Negation von Teilbelegung liefert Klausel: $x_3 \vee \neg x_{17} \vee \neg x_{42}$

Hinzufügen der **Klausel** verhindert Wiederholen desselben "Fehlers"

- **2-Watched Literal Scheme**

Clevere Implementierung und Datenstruktur für Unit Propagation

Viel effizienter als jedes mal alle Klauseln durchzugehen



Hilbert-Kalkül

Wir betrachten noch kurz ein weiteres Beispiel für einen Kalkül, den sogenannten Hilbert-Kalkül.

Dieser geht zurück auf den Logiker David Hilbert.



Es gibt einige Unterschiede zur Resolution:

- der Hilbert-Kalkül erhält keine Eingabe
- er erzeugt nacheinander alle gültigen Formeln
(und terminiert also nie)
- er modelliert in gewisser Weise mathematisches Beweisen:
ausgehend von Axiomen werden durch Anwendung von Schlussregeln
zunehmend komplexere Aussagen (= gültige Formeln) bewiesen

Hilbert-Kalkül

Der **Hilbert-Kalkül** verwendet Formeln über der Junktormenge $\{\rightarrow, \neg\}$ und basiert auf den folgenden Axiomenschemata:

1. $\varphi \rightarrow (\psi \rightarrow \varphi)$
2. $(\varphi \rightarrow (\psi \rightarrow \vartheta)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \vartheta))$
3. $(\varphi \rightarrow \psi) \rightarrow (\neg\psi \rightarrow \neg\varphi)$
4. $\varphi \rightarrow (\neg\varphi \rightarrow \psi)$
5. $(\neg\varphi \rightarrow \varphi) \rightarrow \varphi$

Aus diesen Axiomenschemata kann man mittels einer einzigen Schlussfolgerungsregel, dem **Modus Ponens**, alle gültigen Formeln herleiten



Hilbert-Kalkül

Definition 1.45 (Herleitbarkeit im Hilbert-Kalkül)

Die Menge der *herleitbaren Formeln* ist die kleinste Menge, so dass:

- jede Instanz der Axiomenschemata 1–5 ist herleitbar
(Instanz: Teilformeln φ , ψ , ϑ beliebig ersetzen)
- wenn φ herleitbar und $\varphi \rightarrow \psi$ herleitbar, dann ψ herleitbar
(*Modus Ponens*)

Beispiel: die Formel $x \rightarrow x$ ist herleitbar

T1.17

Ohne Beweis:

Theorem 1.46 (Korrektheit & Vollständigkeit Hilbert-Kalkül)

Eine Formel φ ist gültig gdw. sie im Hilbert-Kalkül herleitbar ist.

Die Frage nach Terminierung stellt sich hier nicht (aber man kann wieder Beweislängen studieren)



Resolutionskalkül vs. Hilbert-Kalkül

	Resolutionskalkül	Hilbert-Kalkül
Ziel	erzeugt leere Klausel wenn gegebene Formel unerfüllbar	erzeugt alle gültigen Formeln
Formeln	in KNF	über Junktormenge $\{\rightarrow, \neg\}$
Arbeitsweise	Herleitung der leeren Klausel mittels Resolventenbildung	Herleitung neuer Formeln aus Axiomen mittels Modus Ponens
Vollständ.-beweis	recht einfach	recht aufwändig
Anwendung	automatisches Entscheiden von Erfüllbarkeit	Modellierung mathematischen Schließens



- 1.1 Syntax und Semantik
- 1.2 Auswertung und Äquivalenz
- 1.3 Normalformen und funktionale Vollständigkeit
- 1.4 Erfüllbarkeit, Gültigkeit, Folgerbarkeit
- 1.5 Tseitin-Transformation
- 1.6 Horn Logik
- 1.7 Resolution
- 1.8 Kompaktheit**

Next



Kompaktheit

Manchmal ist es nützlich, mit **unendlichen Mengen** aussagenlogischer Formeln zu arbeiten.

Unendliche Formelmenge Γ ist **erfüllbar**, wenn es Belegung V gibt mit $V \models \varphi$ für alle $\varphi \in \Gamma$.

Ein zentrales Resultat zum Verständnis unendlicher Formelmengen ist der **Kompaktheitssatz** (auch **Endlichkeitssatz** genannt):

Theorem 1.47 (Kompaktheitssatz)

Für alle (potentiell unendlichen) Mengen Γ von AL-Formeln gilt:

Γ ist erfüllbar gdw. jede endliche Teilmenge von Γ erfüllbar ist.

Wir betrachten zunächst eine Beispielanwendung.



Kompaktheit – Beispielanwendung

Definition 1.48 (4-Färbbarkeit)

Ein ungerichteter *Graph* $G = (V, E)$ heißt **4-färbbar**, wenn es Abbildung $f : V \rightarrow \{R, G, B, W\}$ gibt, so dass $f(v) \neq f(v')$ für alle $\{v, v'\} \in E$. So ein f heißt **4-Färbung**.

Der bekannte 4-Farben-Satz (Appel und Haken 1976):

Theorem 1.49 (4-Farben-Satz, endliche Graphen)

Jeder endliche planare Graph ist 4-färbbar.

planar = kann ohne sich überkreuzende Kanten gezeichnet werden

(Dieser Satz hat übrigens eine interessante Geschichte:

1. „Beweis“ 1879

Fehler gefunden 1890

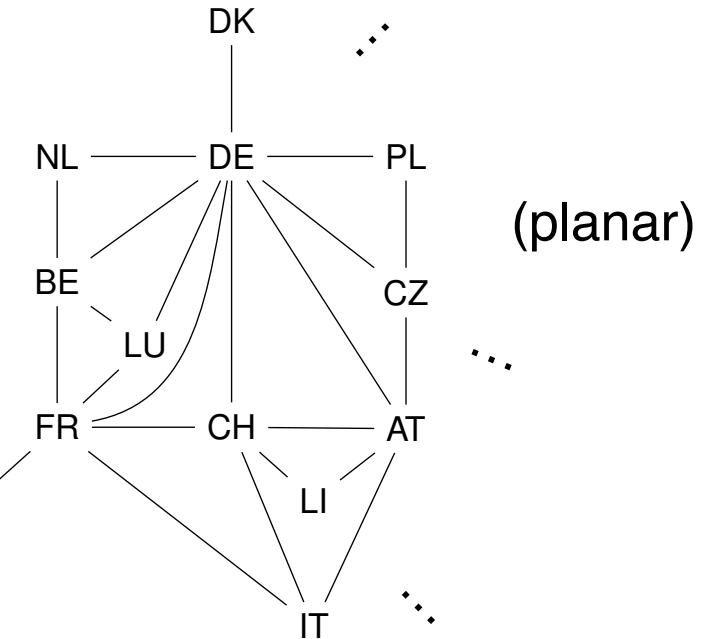
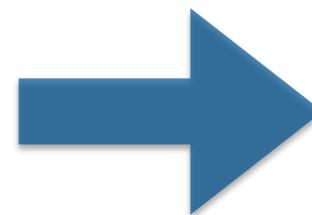
wirklich bewiesen 1976, erster wichtiger computergestützter Beweis)



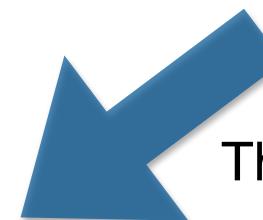
4-Farben-Satz: Beispiel



CC BY-SA 3.0 Wikimedia, Alexrk2



CC BY-SA 2.5 Wikimedia, User:mjchael



Thm. 1.49



UNIVERSITÄT
LEIPZIG

Kompaktheit – Beispielanwendung

Mittels des Kompaktheitssatzes kann man den 4-Farben-Satz von endlichen auf unendliche Graphen übertragen:

Theorem 1.50 (4-Farben-Satz, beliebige Graphen)

Jeder (möglicherweise unendliche) planare Graph ist 4-färbbar.

Beweis.

Sei $G = (V, E)$ ein (möglicherweise unendlicher) planarer Graph.

Definiere Formelmenge

$$\begin{aligned}\Gamma = & \{x_{v1} \vee x_{v2} \vee x_{v3} \vee x_{v4} \mid v \in V\} \\ & \cup \{\neg(x_{vi} \wedge x_{vj}) \mid v \in V, 1 \leq i < j \leq 4\} \\ & \cup \{\neg(x_{vi} \wedge x_{wi}) \mid \{v, w\} \in E, 1 \leq i \leq 4\}\end{aligned}$$

Behauptung: Jede endliche Teilmenge $\Delta \subseteq \Gamma$ ist erfüllbar.

T1.18

Mit Kompaktheitssatz folgt: Γ ist erfüllbar.

Jede erfüllende Belegung liefert eine 4-Färbung von G .



Kompaktheit

Wir beweisen nun den Kompaktheitssatz.

Theorem 1.47 (Kompaktheitssatz)

Für alle (potentiell unendlichen) Mengen Γ von AL-Formeln gilt:
 Γ ist erfüllbar gdw. jede endliche Teilmenge von Γ erfüllbar ist.

„ \Rightarrow “ ist offensichtlich, gesucht also Beweis von „ \Leftarrow “

Im Prinzip ist ein **Beweis über Resolution** möglich. Skizze:

- Erweitere den Resolutionssatz auf **unendliche Formelmengen**
- Wenn Γ unerfüllbar gibt es also einen Resolutionsbeweis
- Ein Resolutionsbeweis ist ein **endliches Objekt**, kann nur endlich viele Klauseln aus Γ verwenden
- Das identifiziert **endliche unerfüllbare Teilmenge von Γ**

Im Folgenden ein unabhängiger, rein „semantischer“ Beweis



Kompaktheit

Theorem 1.47 (Kompaktheitssatz)

Für alle (potentiell unendlichen) Mengen Γ von AL-Formeln gilt:

Γ ist erfüllbar \Leftarrow jede endliche Teilmenge von Γ ist erfüllbar.

Wir nennen $\Gamma \subseteq \text{AL}$ **limit-erfüllbar** wenn jedes endliche $\Gamma' \subseteq \Gamma$ erfüllbar

Zu zeigen also: wenn Γ limit-erfüllbar, dann Γ erfüllbar

Betrachte **Aufzählung** $\varphi_1, \varphi_2, \dots$ aller AL-Formeln (existiert weil VAR abzählbar)

Konstruiere **Folge** $\Gamma_0, \Gamma_1, \dots$ mit $\Gamma_0 = \Gamma$ und

$\Gamma_{i+1} = \Gamma_i \cup \{\varphi_{i+1}\}$ wenn limit erfüllbar, $\Gamma_{i+1} = \Gamma_i \cup \{\neg\varphi_{i+1}\}$ sonst

- Wir zeigen:
1. Alle Γ_i sind limit-erfüllbar
 2. $\Gamma_\omega := \bigcup_{i \geq 0} \Gamma_i$ ist limit-erfüllbar
 3. Γ_ω ist erfüllbar (also auch Γ !)

“Vervollständigung”

T1.19



Zwei Äquivalente Formulierungen

Theorem 1.47 (Kompaktheitssatz)

Für alle (potentiell unendlichen) Mengen Γ von AL-Formeln gilt:

Γ ist erfüllbar gdw. jede endliche Teilmenge von Γ erfüllbar ist.

Äquivalent (und manchmal natürlicher) ist die folgende Variante:

Theorem 1.51 (Kompaktheitssatz Variante 2)

Für alle (potentiell unendlichen) Mengen Γ von AL-Formeln und alle aussagenlogischen Formeln φ gilt:

$\Gamma \models \varphi$ gdw. ein endliches $\Delta \subseteq \Gamma$ existiert so dass $\Delta \models \varphi$.

Übung: Zeige, dass Theorem 1.51 aus Theorem 1.47 folgt und umgekehrt.
(verwende die Definition von „erfüllbar“ und „ \models “)



Übersicht der Vorlesung

Einführung

Teil 1: Aussagenlogik

Next



Teil 2: Prädikatenlogik Grundlagen

Teil 3: Mehr zur Prädikatenlogik erster Stufe

Teil 4: Prädikatenlogik zweiter Stufe



Prädikatenlogik

Für viele Zwecke in der Informatik (und Mathematik) **abstrahiert** die Aussagenlogik zu stark.

Betrachte z. B. die Beispiele aus der Einleitung:

Alle Menschen sind sterblich
Sokrates ist ein Mensch

Sokrates ist sterblich

Jedes P ist auch ein Q
 x ist ein P

x ist ein Q

- $\forall n \in \mathbb{N} : \exists n' \in \mathbb{N} : n' = \text{nf}(n)$
- $\forall n \in \mathbb{N} : \text{nf}(n) \neq 0$
- ...

Bei diesen Aussagen geht es nicht nur um Wahrheitswerte:

Objekte (Menschen, natürliche Zahlen) und **Quantifizierung** sind zentral!



Die Prädikatenlogik wurde von Frege gegen Ende des 19. Jh. eingeführt.

Zentrale Elemente

1. Formeln zusammengesetzt aus Objektvariablen, Booleschen Operatoren und Quantoren
2. eine Semantik, die Objekte sowie deren Eigenschaften und Beziehungen erfasst

Prädikatenlogik spielt zentrale Rolle in Informatik, Mathematik, Philosophie

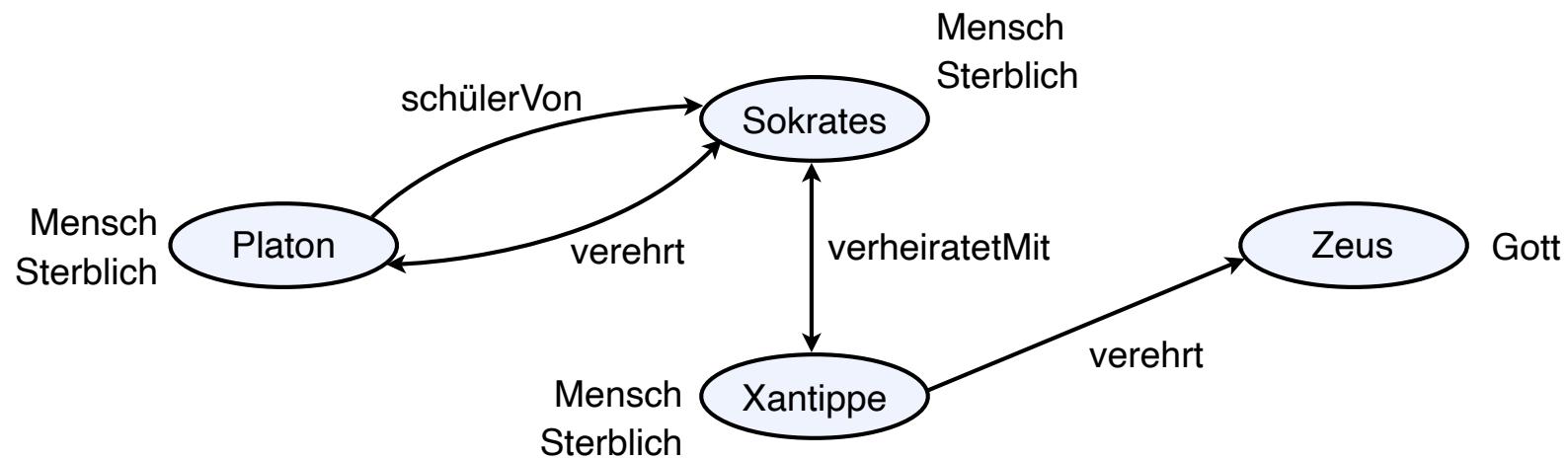
Andere Namen: Logik erster Stufe, First-order Logic, Predicate calculus

Abkürzung: FO



Beispiel

Eine semantische Struktur der Logik erster Stufe:



Zu dieser Struktur passende Beispielformeln:

$$\forall x (\text{Mensch}(x) \rightarrow \text{Sterblich}(x))$$

$$\begin{aligned} \exists x (\exists y (\text{verehrt}(x, y) \wedge \text{Gott}(y)) \wedge \\ \exists y (\text{verheiratetMit}(x, y) \wedge \forall z (\text{verehrt}(y, z) \rightarrow \neg \text{Gott}(z)))) \end{aligned}$$