Aussagenlogik

- Syntax
- Semantik
- Formeln, Modelle, Tautologien und Anwendungen
- **Folgerungen, Wissen**

Folgerungen (1)

φ **folgt aus** (ist **Folgerung** aus) Formelmenge *M* bzw.

$M \models \varphi$:⇔ Jedes für φ ausreichende Modell von *M* ist auch ein Modell von φ,

d.h. $bel \models M \Rightarrow bel \models \varphi$.

Bei einelementigem *M*: „$\psi \models \varphi$“ anstelle von „$\{\psi\} \models \varphi$“

- Natürlich folgt jede Formel **in** *M* auch **aus** *M*.
- Folgt eine Formel aus einer Formelmenge *M*, dann auch aus jeder **Obermenge** von *M*.
- Sei $M = \{\psi_1, \psi_2, ..., \psi_n\}$ eine endliche Menge von Formeln.
  $M \models \varphi \Leftrightarrow (\psi_1 \wedge \psi_2 \wedge ... \wedge \psi_n) \models \varphi$

Folgerungen (2)

$M \models \varphi$ erkennen wir bei **endlicher** Formelmenge M in der **Wahrheitstafel**:

In den Zeilen in denen **alle Formeln von *M*** den Wert W haben,
hat **auch** φ den Wert W.

Beispiel: $\{A \to B, \neg B\} \models \neg A$

A	B	$A \to B$	$\neg B$	$\neg A$
W	W	W	F	F
W	F	F	W	F
F	W	W	F	W
F	F	W	W	W

Ü26

φ ist Tautologie
$\Leftrightarrow$ φ folgt aus der leeren Formelmenge.
$\Leftrightarrow$ φ folgt aus jeder Formelmenge.
$\Leftrightarrow$ φ folgt aus einer Menge von Tautologien.

Folgerung und Unerfüllbarkeit

- $M \models \varphi \Leftrightarrow M \cup \{\neg\varphi\}$ unerfüllbar
- *M* ist genau dann unerfüllbar, wenn jede Formel eine Folgerung aus *M* ist.
- *M* ist genau dann unerfüllbar, wenn mindestens eine unerfüllbare Formel eine Folgerung aus *M* ist.

Folgerung und Implikation: Deduktionssätze (zunehmend allgemeiner)

- Für zwei Formeln φ, ψ gilt $\boxed{\varphi \models \psi}$ genau dann wenn $\boxed{\models \varphi \rightarrow \psi}$ gilt.

- Für $m \geq 1$ und Formeln $\varphi_1, \varphi_2, \ldots, \varphi_{m-1}, \varphi_m$ und ψ gilt:
  $\boxed{\varphi_1, \varphi_2, \ldots, \varphi_{m-1}, \varphi_m \models \psi}$ genau dann wenn $\boxed{\varphi_1, \varphi_2, \ldots, \varphi_{m-1} \models \varphi_m \rightarrow \psi}$.

- Für eine Formelmenge M und zwei Formeln φ, ψ gilt $\boxed{M \cup \{\varphi\} \models \psi}$ genau dann wenn $\boxed{M \models \varphi \rightarrow \psi}$ gilt.

Folgerung und Modelle

Sei N eine **Menge von Folgerungen** aus M, d.h.
für alle $\varphi \in N : M \models \varphi$,
und *bel* sei für M und N ausreichende Belegung.

Dann gilt: *bel* Modell von $M \Leftrightarrow$ *bel* Modell von $M \cup N$.

Wir können also M um Folgerungen daraus erweitern, ohne an M's eventueller Eigenschaft der

- Erfüllbarkeit,
- Unerfüllbarkeit oder
- Allgemeingültigkeit aller ihrer Formeln

etwas zu ändern.

Wissen

Sei M Formelmenge, $M \subseteq Form$.
Wenn man weiß, dass alle Formeln in M gelten,
d.h. wenn man über M als **Wissensbasis** verfügt,
dann weiß man auch, dass alle Folgerungen aus M gelten.
Letztere bilden das durch M gegebene **Wissen**.

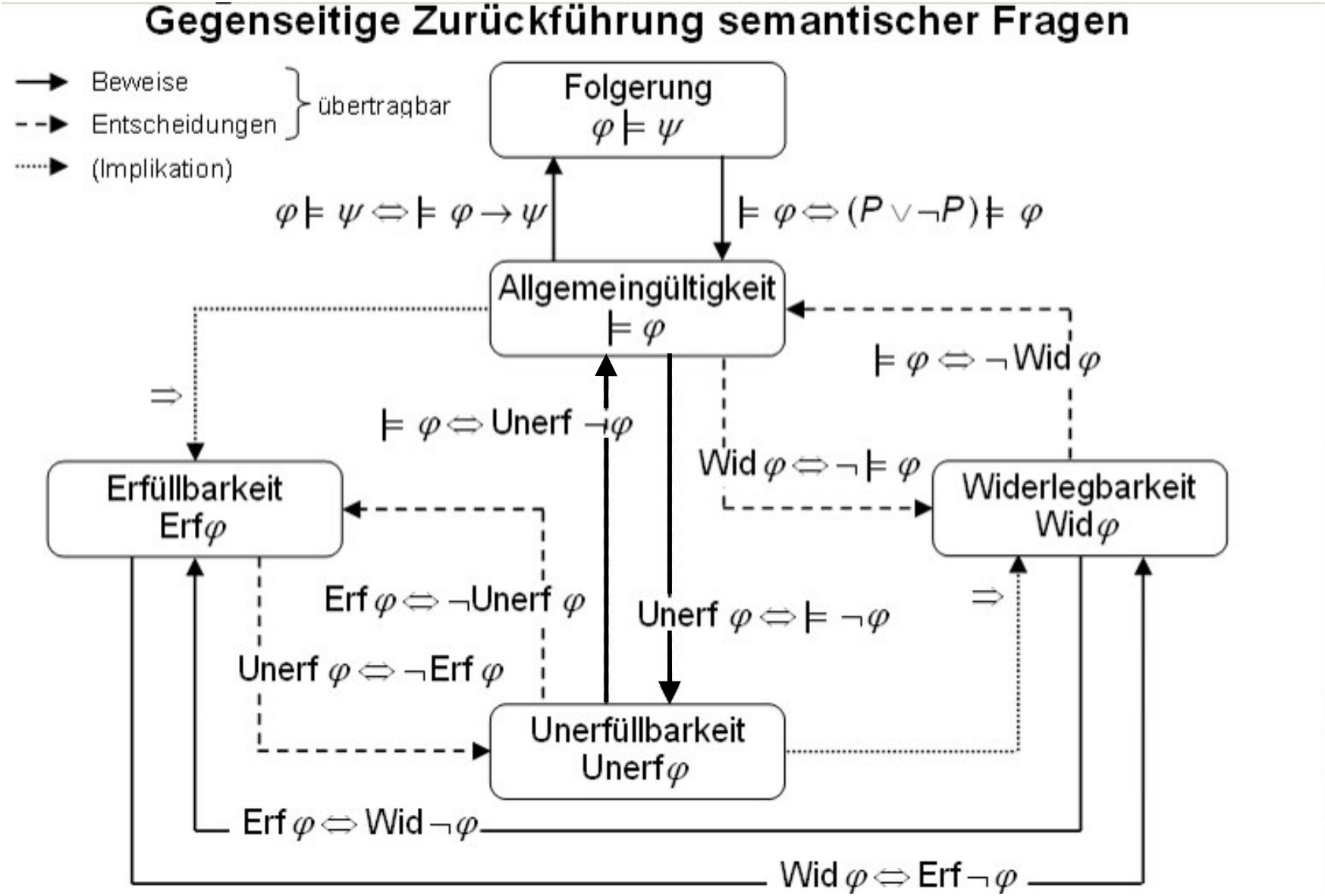
Alle anderen Formeln, sofern deren Negation nicht gewusst wird, könnten dann noch wahr oder falsch sein.

$$\text{Wissen}(M) := \{\varphi \in Form \mid M \models \varphi\}$$

Satz: **Wissen**
Seien M und N Formelmengen**.**

- Wissen($\emptyset$) ist die Menge aller Tautologien von AL;
- $M \subseteq N \Rightarrow$ Wissen(M) $\subseteq$ Wissen(N);
- Wissen(Wissen(M)) = Wissen(M);
- M ist unerfüllbar $\Leftrightarrow$ Wissen(M) = $Form$.*

*) Aber wenn man glaubt, eine widersprüchliche Formelmenge zu wissen, dann irrt man sich.



Aussagenlogik

- Syntax
- Semantik
- Formeln, Modelle, Tautologien und Anwendungen
- Folgerungen, Wissen
- **Syntaktisch besondere Formeln und Formelgraphen, wichtige Algorithmen**

Literale und Klauseln

Literal: eine Aussagevariable oder die Negation einer Aussagevariable
A, $\neg A$, ...

Klausel: eine Disjunktion (mit $\vee$ verknüpfte Kette) von Literalen
$A \vee B$, $A \vee \neg B \vee \neg C$, ... Auch: **disjunktive Klausel**.

Horn-Klausel Klausel, die höchstens ein positives Literal P enthält
$A \vee \neg B \vee \neg C$

Dualklausel Konjunktion (mit $\wedge$ verknüpfte Kette) von Literalen
$A \wedge B$, $A \wedge \neg B \wedge \neg C$, ... Auch: **konjunktive Klausel**.

Eine **Klausel** (bzw. eine ***Dualklausel***)
ist genau dann **allgemeingültig** (bzw. ***unerfüllbar***),
wenn sie für eine Aussagevariable P sowohl P als auch $\neg P$ enthält.

$A \vee \neg B \vee \neg A$

$A \wedge \neg B \wedge \neg A$

Konjunktive Normalform (1)

Formel in **konjunktiver Normalform** (**KNF-Formel**):
Konjunktion von Klauseln,

$$(Lit_{1,1} \vee ... \vee Lit_{1,k_1}) \wedge (Lit_{2,1} \vee ... \vee Lit_{2,k_2}) \wedge \ ... \ \wedge (Lit_{n,1} \vee Lit_{n,2} \vee ... \vee Lit_{n,k_n})$$

Andere **Schreibweise**: $\bigwedge_{i \in I} \bigvee_{k \in K_i} Lit_{ik}$ (mit endl. Indexmengen I und K_i, $i \in I$).

Beispiele:

- $(\neg A \vee B) \wedge (A \vee \neg B \vee C) \wedge A \wedge \neg C$,
- alle Klauseln,
- alle Dualklauseln.

Grammatik der KNF-Formeln (ohne äußere Klammern):

KNF	$\rightarrow$	(Kl) | (Kl) $\wedge$ KNF
Kl	$\rightarrow$	Lit | Lit $\vee$ Kl
Lit	$\rightarrow$	AV | $\neg$ AV
AV	$\rightarrow$	A_1 | A_2 | A_3 | ...

Konjunktive Normalform (2)

Man betrachtet oft eine KNF-Formel als **Menge von Klauseln**

und jede dieser Klauseln als **Menge von Literalen**:

KNFset$((\neg A \vee B) \wedge (A \vee \neg B \vee C) \wedge A \wedge \neg C) = \{\{\neg A, B\}, \{A, \neg B, C\}, \{A\}, \{\neg C\}\}_{KNF}$.

Rechtfertigung:

Wie die Mengen ist auch der Wahrheitswert
der Original-KNF-Formel
invariant unter **Umordnung** und **Wiederholung**
innerhalb der Mengen/Ketten.

Vereinbarung:

- **Leere** KNF-Formel $\{\}_{KNF}$ immer **wahr;**
- **leere** Klausel $\{\}_{Kl}$ immer **falsch**.

KNF: Allgemeingültigkeit und Normalformensatz

KNF-Formel φ ist **allgemeingültig** (1)

$\Leftrightarrow$ alle Klauseln von φ sind allgemeingültig (2)

$\Leftrightarrow$ **jede Klausel** von φ enthält für mindestens eine Variable P die Literale P und $\neg P$. (3)

... und dies (sowie **Widerlegbarkeit**) ist dann **unmittelbar abzulesen**!

Beweisidee:

(1) $\Rightarrow$ (2): Wäre $KNF = Kl \wedge Rest$, und Kl unter bel falsch, dann auch $Kl \wedge Rest$.

(2) $\Rightarrow$ (3): Wären alle Variablen in Klausel $\{Lit_1, Lit_2, ..., Lit_k\}_{Kl}$ unterschiedlich, dann belege alle „anders als in der Klausel“, und sie würde falsch.

(3) $\Rightarrow$ (1): $P \vee \neg P$ ist stets wahr, also auch $P \vee \neg P \vee Rest$, also jede Klausel, also auch $\bigwedge(Klauseln)$

Ü27

Normalformensatz (für KNF)
Zu jeder AL-Formel gibt es mindestens eine äquivalente KNF-Formel.

Begründung **konstruktiv**:
Angabe (und Beweis) eines **Umformungsalgorithmus** – hier sogar zwei

Umformung in KNF (1)

Algorithmus **KNF1**:

Synthese „der KNF einer Formel φ“ **über den Wahrheitswerteverlauf**

1. Berechne den Wahrheitswerteverlauf von φ mit Hilfe der **Wahrheitstafel**.
2. Ist die Formel **allgemeingültig** (alle Formelwerte = W), so wähle eine Aussagevariable P und bilde $P \vee \neg P$ – fertig!
   In der **Mengenform** reicht auch die leere KNF-Menge $\{\}_{KNF}$ – (2) wird Teil von (3). Andernfalls gibt es Zeilen mit dem Formelwert F.
3. Für jede **Zeile** mit dem **Formelwert F** bilde eine Klausel, die für jede vorkommende Aussagevariable P folgendes enthält:
   $\boldsymbol{P}$, wenn der P-Wert in der Zeile **F** ist, und $\neg \boldsymbol{P}$, wenn er **W** ist.
   Die Menge bzw. Konjunktion dieser Klauseln ist eine KNF- Darstellung von φ.

Umformung in KNF (1a)

Beispiel für Schritt 3:

A	B		$\neg(A \rightarrow B) \vee \neg(B \rightarrow \neg A)$
W	W		W
W	F		W
F	**W**		**F**
F	**F**		**F**

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

bzw. $\{ \{ A, \neg B \}, \{ A, B \} \}_{KNF}$

Achtung: Es gibt hierbei auch eine **kürzere** Lösung: ***A*** !

Umformung in KNF (2)

Algorithmus KNF2: Äquivalente syntaktische Umformung in KNF

Wende, so lange es geht, immer wieder irgendeine der folgenden Ersetzungen auf Teilformeln an:

1. Elimination von Äquivalenz und Implikation

1.1. $\varphi \leftrightarrow \psi \quad \mapsto \quad (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$

1.2. $\varphi \rightarrow \psi \quad \mapsto \quad \neg\varphi \vee \psi$

2. Negation beseitigen oder näher an die Aussagevariablen rücken

2.1. $\neg\neg\varphi \quad \mapsto \quad \varphi$

2.2. $\neg(\varphi \wedge \psi) \quad \mapsto \quad \neg\varphi \vee \neg\psi$

2.3. $\neg(\varphi \vee \psi) \quad \mapsto \quad \neg\varphi \wedge \neg\psi$

} … auch mit $\wedge$/$\vee$-Ketten

3. Konjunktion von den Literalen wegrücken

3.1. $\varphi \vee (\psi \wedge \rho) \quad \mapsto \quad (\varphi \vee \psi) \wedge (\varphi \vee \rho)$

3.2. $(\psi \wedge \rho) \vee \varphi \quad \mapsto \quad (\psi \vee \varphi) \wedge (\rho \vee \varphi)$

} … auch mit $\wedge$-Ketten

In #3 **nur $\vee$ über $\wedge$** verteilen, **nicht umgekehrt!**

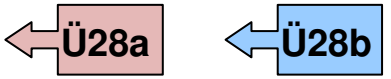
beschleunigt

Umformung in KNF

KNF1 und KNF2 sind korrekt:

Die Algorithmen **terminieren stets**.

Die durch Umformung erhaltene Formel ist in KNF
(ggf. durch geeignetes Weglassen von Klammern)
und zur Ausgangsformel **äquivalent**.



Erfüllbarkeit von KNF-Formeln: Resolution

… ein **effizienter** Algorithmus zur **Entscheidung** über die **Erfüllbarkeit oder Unerfüllbarkeit von KNF-Formeln**.

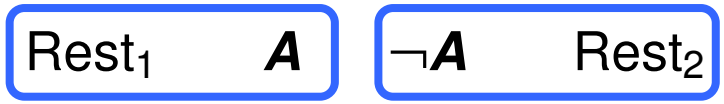
Algorithmus: **AL-Resolution von KNF-Formeln**
Sei φ KNFset-Formel. Wir setzen $\varphi_0 := \varphi$.
Für k = 0,1,... (bis zum unten definierten Abbruch) gehen wir wie folgt vor:

- **Resolutionsschritt:** Sind zwei der Klauseln, κ_1, κ_2, von φ_k von der Form $\kappa_1 = \{P, Rest_1\}$ und $\kappa_2 = \{\neg P, Rest_2\}$ (auch mit evtl. leerem $Rest_{1/2}$), und ist $Rest_1 \cup Rest_2 \notin \varphi_k$,
  so setzen wir $\varphi_{k+1} := \varphi_k \cup \{Rest_1 \cup Rest_2\}$ (**Resolvente**) ;
  (**Variante**: Ist $Rest_1 \cup Rest_2 = \varnothing$ $\longrightarrow$ **STOP**.)
  (In der Original-Formelschreibweise erzeugen wir
  mit $\kappa_1 = A \vee Rest_1$ und $\kappa_2 = \neg A \vee Rest_2$
  die erweiterte Formel $\varphi_{k+1} := \varphi \wedge (Rest_1 \vee Rest_2)$.)
- Ist mit φ_k stattdessen kein Resolutionsschritt mehr möglich: **STOP**.

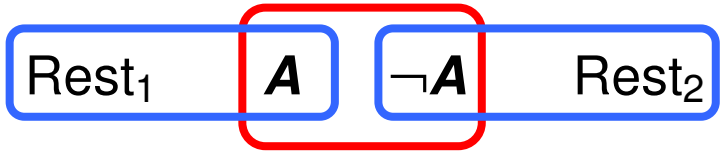
Das zuletzt erhaltene φ_i ist eine (bzw. *die*, s.u.) **Resolventenmenge Res(φ)**.

Resolutionsschritt

2 Klauseln



mit passenden Literalen



→ neue Klausel
(Resolvente)

Resolution – Beispiel

KNF-Formel

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

mögliche Resolutionsschritte in willkürlicher Reihenfolge

$\varphi_0 = \{\{\neg A, B\}, \{\neg B, C\}, \{A\}, \{\neg C\}\}$

$\varphi_1 = \{\{\neg A, B\}, \{\neg B, C\}, \{A\}, \{\neg C\}, \{\neg A, C\}\}$

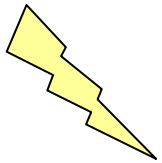
$\varphi_2 = \{\{\neg A, B\}, \{\neg B, C\}, \{A\}, \{\neg C\}, \{\neg A, C\}, \{B\}\}$

$\varphi_3 = \{\{\neg A, B\}, \{\neg B, C\}, \{A\}, \{\neg C\}, \{\neg A, C\}, \{B\}, \{\neg B\}\}$

$\varphi_4 = \{\{\neg A, B\}, \{\neg B, C\}, \{A\}, \{\neg C\}, \{\neg A, C\}, \{B\}, \{\neg B\}, \{\neg A\}\}$

$\varphi_5 = \{\{\neg A, B\}, \{\neg B, C\}, \{A\}, \{\neg C\}, \{\neg A, C\}, \{B\}, \{\neg B\}, \{\neg A\}, \varnothing\}$

$\varphi_6 = \{\{\neg A, B\}, \{\neg B, C\}, \{A\}, \{\neg C\}, \{\neg A, C\}, \{B\}, \{\neg B\}, \{\neg A\}, \varnothing, \{C\}\}$



Bei manueller Berechnung natürlich nur die neuen Formeln hinschreiben, vgl. Skript: Folgenform!

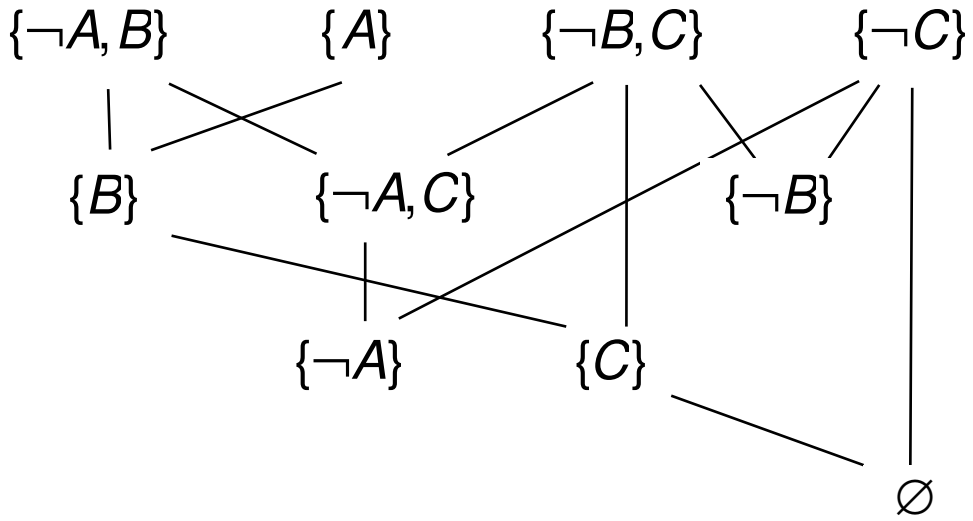
Resolventen-menge

Resolution – als Graph

KNF-Formel

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

Ein möglicher Resolutionsgraph



Bei manueller Berechnung am besten **als Graph** hinschreiben!

Andere willkürliche Reihenfolge

→ evtl. **anderer** Resolutionsgraph

→ **gleiche** Resolventenmenge

Resolutionssatz (1)

Resolutionssatz der Aussagenlogik – 4 Teile:

1. Es sind stets nur endlich viele Resolutionsschritte möglich.
Der Algorithmus **terminiert**.

Bew.:
φ enthalte m Literale. Jeder RS erzeugt „neue“ Teilmenge,
$\leq 2^m$ Möglichkeiten.

2. Alle konstruierbaren Resolventen von φ sind (in ihrer Menge) identisch,
also logisch äquivalent. Man spricht daher von **der Resolventenmenge**.

Bew.:
Jeder einmal mögliche Resolutionsschritt
bleibt weiter möglich, bis seine Resolvente erzeugt ist,
also höchstens Reihenfolgeunterschiede in deren Erzeugung.

Resolutionssatz (2)

3. φ ist **semantisch äquivalent** zu Res(φ).

Bew.: Res. erzeugt nur Folgerungen aus φ:
Resolutionsregel $((P \vee Q) \wedge (\neg P \vee R)) \rightarrow (Q \vee R)$.

„Sonder-Stop"

4. φ ist genau dann **unerfüllbar**, wenn die Res(φ) die **leere Klausel enthält** bzw. – in Formelschreibweise – zwei Klauseln der Form P und $\neg P$ enthält.

Bew.idee: $\Leftarrow$: einfach
$\Rightarrow$: siehe Literatur/Web

Resolution ist auch zur Entscheidung über
- Folgerung und
- Allgemeingültigkeit

verwendbar**, nämlich wie?**

also z.B.
$(\neg A \vee B) \wedge (\neg B \vee C) \wedge A \wedge \neg C$

Ü29

Disjunktive Normalform (1)

Formel in **disjunktiver Normalform** (**DNF-Formel**):

Disjunktion von Dualklauseln.

$$(\mathit{Lit}_{1,1} \wedge ... \wedge \mathit{Lit}_{1,k_1}) \vee \ ... \vee (\mathit{Lit}_{n,1} \wedge ... \wedge \mathit{Lit}_{n,k_n}), \quad \text{bzw. } \bigvee_{i \in I} \bigwedge_{k \in K_i} \mathit{Lit}_{ik}$$

Beispiele:

- $(\neg A \wedge B) \vee (\neg B \wedge C) \vee A \vee \neg C$
- alle Klauseln,
- alle Dualklauseln.

Grammatik der DNF-Formeln (ohne äußere Klammen):

DNF $\rightarrow$ (Dkl) | (Dkl) $\vee$ DNF
Dkl $\rightarrow$ Lit | Lit $\wedge$ Dkl
Lit $\rightarrow$ AV | $\neg$ AV
AV $\rightarrow$ A_1 | A_2 | A_3 | ...

Mengenschreibweise:

DNFset$((\neg A \wedge B) \vee (\neg B \wedge C) \vee A \vee \neg C) = \{\{\neg A, B\}, \{\neg B, C\}, \{A\}, \{\neg C\}\}_{DNF}$.

Disjunktive Normalform (2)

Vereinbarung:

Leere Konjunktion $\Lambda\varepsilon$ (z.B. KNF-Formel, Dualklausel) immer **wahr**;

leere Disjunktion $V\varepsilon$ (z.B. DNF-Formel, Klausel) immer **falsch**.

(→ $\perp$-T-Dialekt)

DNF-Formel φ ist **widersprüchlich**

$\Leftrightarrow$ alle Dualklauseln von φ (auch leeres φ) sind unerfüllbar

$\Leftrightarrow$ **jede Dualklausel** von φ enthält für mindestens eine Variable P die Literale P und $\neg P$.

... und dies (sowie **Erfüllbarkeit**) ist dann **unmittelbar abzulesen!**

Ü30

Erinnerung: Dualitätsaspekt

Bei **KNF**-Formeln sind

- **Allgemeingültigkeit** und
- **Widerlegbarkeit**

unmittelbar abzulesen.

Bei **DNF**-Formeln sind

- **Widersprüchlichkeit (= Unerfüllbarkeit)** und
- **Erfüllbarkeit**

unmittelbar abzulesen.

Normalformensatz und Umformung in DNF (1)

Normalformensatz (für DNF)

Zu jeder AL-Formel gibt es mindestens eine äquivalente DNF-Formel.
Sowohl mit DNF1 als auch mit DNF2 erhält man eine solche.

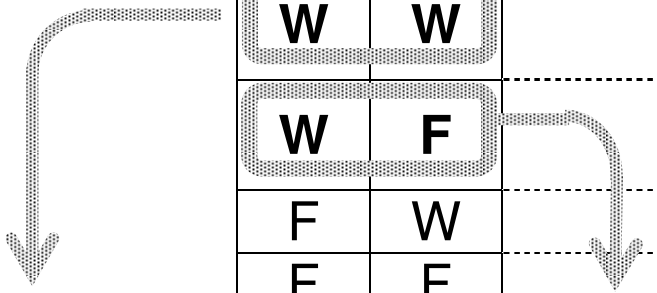
Algorithmus DNF1: **Begründung/ Beweis überlegen!**

Synthese der DNF einer Formel φ **über den Wahrheitswerteverlauf**

1. **Wahrheitstafel** von φ.
2. Sind alle Formelwerte = F, wähle $P \in Vars$ und bilde $P \wedge \neg P$ – fertig!
   In der Mengenform reicht auch die leere DNF-Menge $\{\}_{DNF}$; dann wird (2) Teil von (3). Andernfalls gibt es Zeilen mit dem Formelwert W.
3. Für jede Zeile mit dem Formelwert W bilde eine Klausel mit:
   $\boldsymbol{P}$, wenn der P-Wert in der Zeile **W** ist, und $\neg\boldsymbol{P}$, wenn er **F** ist.
4. Bilde die Menge bzw. Disjunktion dieser Dualklauseln

Umformung in DNF (1a)

Beispiel für Schritt 3:



A	B	$\neg(A \rightarrow B) \vee \neg(B \rightarrow \neg A)$
W	**W**	**W**
W	**F**	**W**
F	W	F
F	F	F

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

Achtung, **kürzere** Lösung: ***A*** !

bzw. $\{ \{ A, \neg B \}, \{ A, B \} \}_{DNF}$

Umformung in DNF (2)

Algorithmus DNF2: Begründung/ Beweis überlegen!

Äquivalente syntaktische Umformung in DNF

So lange es geht:

1. Elimination von Äquivalenz und Implikation
2. Negation beseitigen oder näher an die Aussagevariablen rücken

} **wie bei KNF**

3. <u>Disjunktion</u> von den Literalen wegrücken

3.1. $\varphi \wedge (\psi \vee \rho) \quad \mapsto \quad (\varphi \wedge \psi) \vee (\varphi \wedge \rho)$

3.2. $(\psi \vee \rho) \wedge \varphi \quad \mapsto \quad (\psi \wedge \varphi) \vee (\rho \wedge \varphi)$

} … auch mit $\wedge$/$\vee$-Ketten

In #3 **nur $\wedge$ über $\vee$** verteilen, **nicht umgekehrt!**

beschleunigt

(Auch für DNF: eventuelle Klammern aus $\wedge$- bzw. $\vee$-Ketten weglassen)

Ü31a Ü31b

Formeln und Bäume

Wir suchen nach **Graphen** (z.B. Bäumen oder Automaten), mit denen **Formeln** (ihre Syntax), bzw. ihr **Werteverlauf** (ihre Semantik) **kodiert** werden können.

Das führt zum
- **Tableauverfahren** und zu den
- **binären Entscheidungsgraphen (BDDs)**.

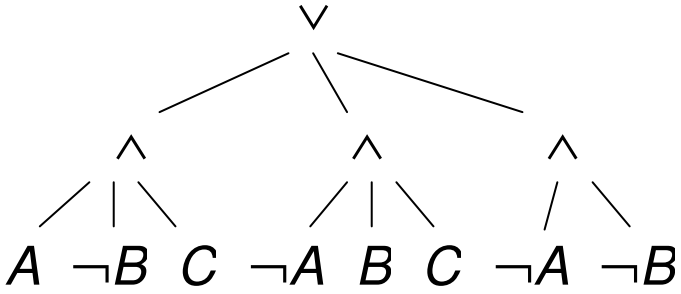
Erinnerung: Assoziativität ⇒ $\wedge-$ und $\vee-$Verkettungen mit Länge >2 sinnvoll

⇓

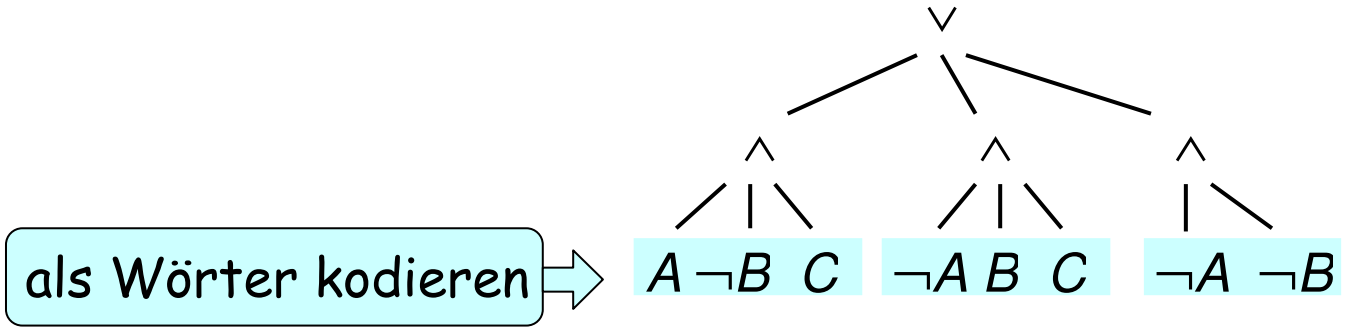
Baumdarstellung einer DNF-Formel

DNF-Formel ⇒ **Syntaxbaum(φ)**

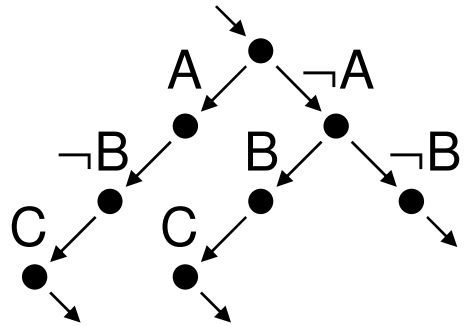
$$\varphi = (A \wedge \neg B \wedge C) \vee (\neg A \wedge B \wedge C) \vee (\neg A \wedge \neg B)$$



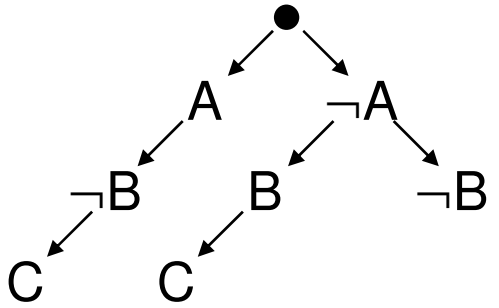
Formeln, Bäume und Automaten (1)



sparsamer: ein* **Automat** der φ-Dualklauseln



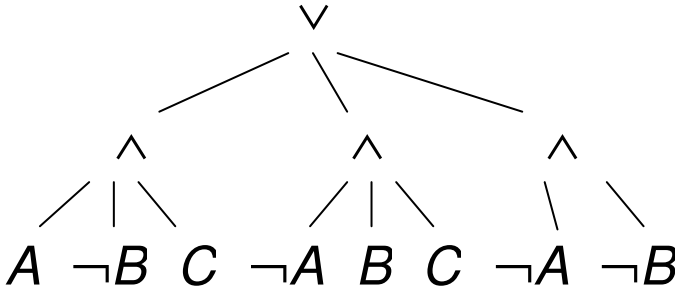
Knoten-Kanten-Tausch: ein „**Tableaubaum**“(φ)



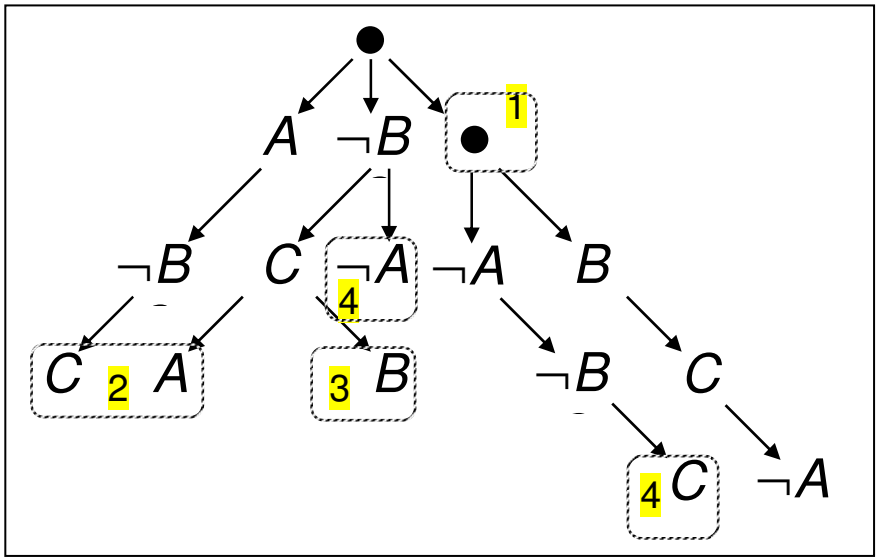
*) pro Variablen-Reihenfolge

Formeln, Bäume und Automaten (2)

Erinnerung:
Syntaxbaum



ein **anderer Tableaubaum**** für (φ)



**) mit mehr Freiheiten:
Leerknoten 1 ,
Doppelproduktionen 2 ,
Widerspruchzweigen 3 ,
„Oberbelegungen“ 4 ,
gemischten Reihenfolgen.

Tableauregeln (1)

- *tableau*, pl. *tableaux* = Französisch für *Tabelle*

Hauptgedanke:

Welche Voraussetzungen müssen gelten, damit eine Formel gilt?

Dazu 9 **Tautologien**, leicht verändert als **Tableauregeln** geschrieben:

1. $\dfrac{\neg(\varphi \to \psi)}{\begin{array}{l}\varphi \\ \neg\psi\end{array}}$ 2. $\dfrac{\varphi \wedge \psi}{\begin{array}{l}\varphi \\ \psi\end{array}}$ 3. $\dfrac{\neg(\varphi \vee \psi)}{\begin{array}{l}\neg\varphi \\ \neg\psi\end{array}}$ 4. $\dfrac{\varphi \leftrightarrow \psi}{\begin{array}{l}\varphi \to \psi \\ \psi \to \varphi\end{array}}$ 5. $\dfrac{\neg\neg\varphi}{\varphi}$

6. $\dfrac{\varphi \to \psi}{\neg\varphi \mid \psi}$ 7. $\dfrac{\neg(\varphi \wedge \psi)}{\neg\varphi \mid \neg\psi}$ 8. $\dfrac{\varphi \vee \psi}{\varphi \mid \psi}$ 9. $\dfrac{\neg(\varphi \leftrightarrow \psi)}{\neg(\varphi \to \psi) \mid \neg(\psi \to \varphi)}$

Beispielsweise die #1: $\neg(\varphi \to \psi) \equiv (\varphi \wedge \neg\psi)$: **übereinander** $\cong$ **UND**,
und die #6: $(\varphi \to \psi) \equiv (\neg\varphi \vee \psi)$: **nebeneinander** $\cong$ **ODER**.

Tableauregeln (2)

Beschleunigungsmöglichkeiten

- #2 und #7 auf längere $\wedge$**-Ketten** verallgemeinern
- #3 und #8 auf längere $\vee$**-Ketten**

verallgemeinern.

Bei #2 und #3 entstehen dann mehr als 2 Nachkommensgenerationen.
Bei #7 und #8 entstehen dann mehr als 2 Geschwister.

Beispiel mit #8:

$$A \vee B \vee C \quad \xrightarrow{\dfrac{\varphi_1 \vee \ldots \vee \varphi_n}{\varphi_1 | \ldots | \varphi_n}} \quad \begin{array}{c} A \vee B \vee C \\ \diagup \quad | \quad \diagdown \\ A \quad B \quad C \end{array}$$

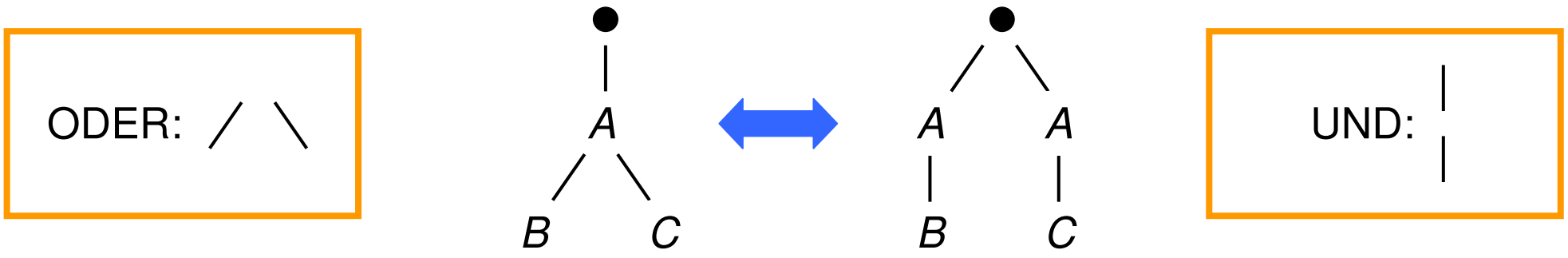
Tableaux zur DNF-Umformung (1)

Wiederholte Anwendung der Tableauregeln erzeugt einen **Baum**,

- „ersetzt“ Biimplikation durch Implikationen,
- „ersetzt“ Implikation durch UND oder ODER,
- bringt Negationen „nach innen“, vor die Aussagevariablen.

Tableauverfahren ähneln also der **äquivalenten Umformung** in DNF.
Wie dort kann man ...

- alle Modelle ablesen und
- über die Erfüllbarkeit entscheiden.



Wegen der Distributivität $A \wedge (B \vee C) \quad \equiv \quad (A \wedge B) \vee (A \wedge C)$ können wir alle Zweige zu den Blättern „**auseinanderzupfen**“, **nur** die **Literale** verwenden und erhalten → **DNF**!

Tableau-Algorithmus zur DNF-Umformung (2)

Erstes Ziel: Aus der Ausgangsformel φ einen **Tableaubaum Tab(φ)** konstruieren, dessen **Knoten** mit je einer Formel **beschriftet** und am Ende als erledigt **markiert** sind. Jeder Knoten ist dabei zunächst **unerledigt**.

(1) ***Baum*** := **Wurzelknoten**, mit φ beschriftet (*und unerledigt, Startbaum*)

(2) So lange es geht, **wähle** einen **unerledigten** Knoten *k* in ***Baum***.
a) Ist *k* **kein Literal**, dann **expandiere** *k* (→ *nächste Folie*)
b) Markiere Knoten *k* als **erledigt**. (*Literale sind also gleich erledigt.*)

(3) **Tab(φ)** := ***Baum***

Zweites Ziel: Aus dem **Tableaubaum** eine **DNF-Darstellung** ablesen.

(*Nach Schritt 3 sind alle Knoten erledigt.*)

(4) Bilde für jeden **Zweig** von Tab(φ) eine **Dualklausel**, nämlich die **Konjunktion aller „Literal-Knoten“** des Zweiges.

(5) Bilde die **Disjunktion** dieser Dualklauseln für alle Zweige bzw. Blätter.

Das Verfahren ist im **Ablauf** vor allem wegen der möglichen unterschiedlichen Wahlen in **Schritt 2 nichtdeterministisch**.

Tableaux zur DNF-Umformung (3)

Expansion von *k*:

- Hat *k*'s Formel die Struktur einer Kopf-Formel in der **oberen** Tableauregelreihe (1–5), dann **hänge an alle Blätter** unterhalb *k* die „entsprechenden" Formeln **untereinander** (*als Kind + Enkel etc., im Falle* $\neg\neg\psi$ *nur* ψ *als Kind*) an.
- Hat *k*'s Formel die Struktur einer Kopf-Formel in der ***unteren*** Tableauregelreihe (6–9), dann **hänge an alle Blätter** unterhalb *k* die „entsprechenden" Formeln ***nebeneinander*** (als Kinder) an.

Satz: **Mit Tableaux zu DNF und Modellen**

Das obige Verfahren **terminiert** und
liefert mit Schritt 5 eine **DNF-Darstellung** der Ausgangsformel.
Die Dualklauseln aus Schritt 3 liefern (mit beliebiger Fortsetzung auf Vars(φ)) die **Modelle** von φ.

Ü32a Ü32bc

Tableaux für Erfüllbarkeit, Allgemeingültigkeit (1)

Wir nennen während der Ausführung des Tableau-Algorithmus

- einen Knoten **abgeschlossen,** wenn auf seinem **Pfad** ab der Wurzel zwei erkennbar **widersprüchliche** Formeln, („persönlicher Nichtdeterminismus“ oder spätestens erkennbar: P und $\neg P$) an Knoten stehen; ➡ **unterstreichen**
- ein Blatt **offen**, wenn es **nicht abgeschlossen** ist und **alle** Knoten auf seinem Zweig als **erledigt** markiert sind.

Algorithmus zur **Überprüfung der Erfüllbarkeit** von Formeln**: wie oben**, aber

- nach dem **ersten offenen** Blatt **aufhören**: die Formel ist dann **erfüllbar**.
- Die Formel ist genau dann **unerfüllbar**, wenn **nach Terminierung** des Algorithmus **alle** Blätter **abgeschlossen** sind.

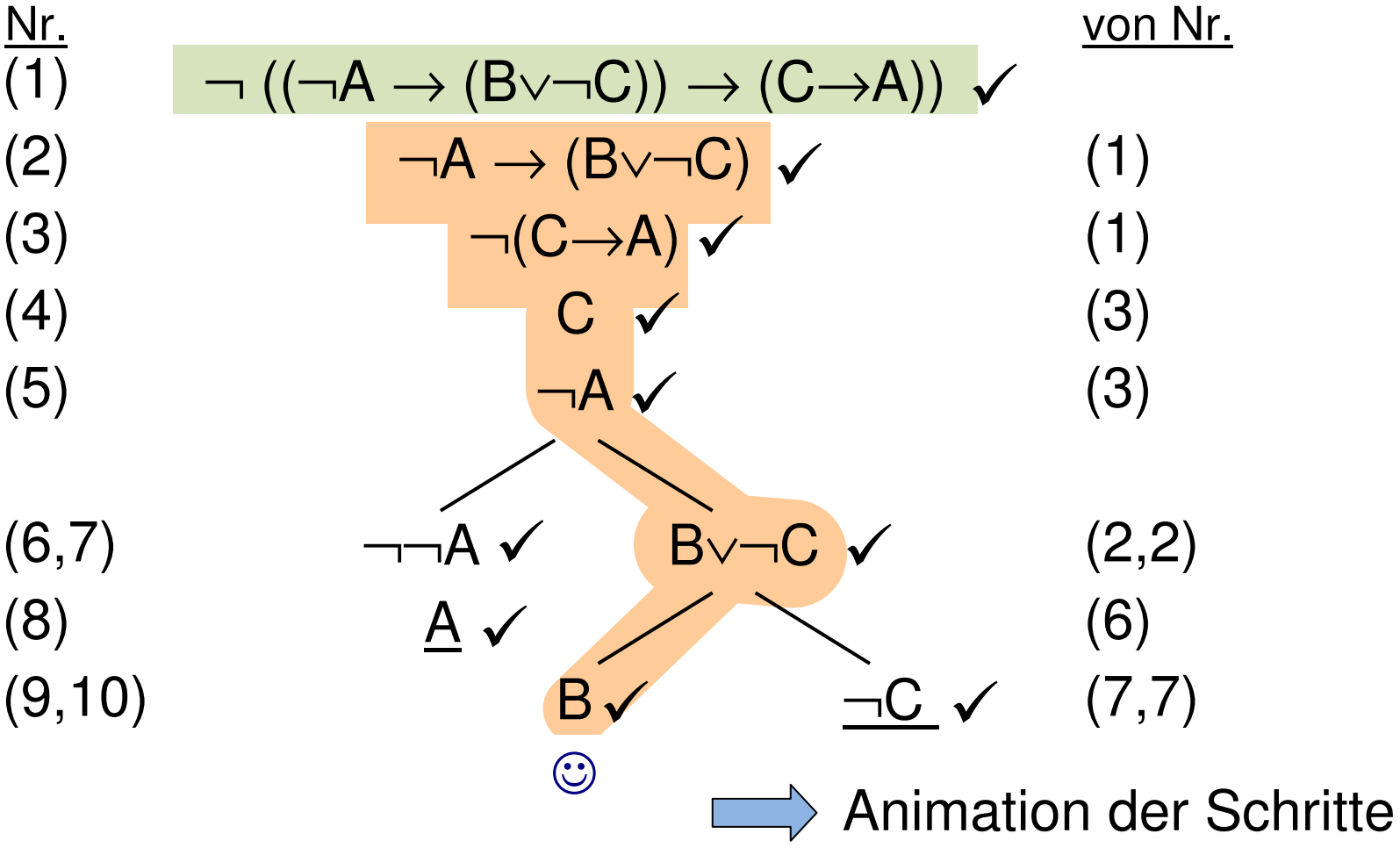
Algorithmus zur **Überprüfung der Allgemeingültigkeit** von Formeln:

- die **Negation** bilden
- und deren **Unerfüllbarkeit** per Tableau überprüfen.

Ü33

Tableaux für Erfüllbarkeit, Allgemeingültigkeit (2)

Der folgende Tableaubaum zeigt die Erfüllbarkeit der Wurzelformel :



Animation der Schritte

Tableaux: Modelle, Effizienzsteigerung, Erfüllbarkeit

Verbesserungen zwecks Beschleunigung, Raumersparnis, Übersichtlichkeit:

- **Übergehe** abgeschlossene Blätter beim Anhängen von Kindern, denn unerfüllbare Alternativen können in der DNF entfallen.
- **Wähle** Knoten mit **Tableau-Schritten** der **oberen Reihe** (ohne weitere Aufspaltung des Baumes) **vor** den Knoten mit **Aufspaltung** aus.

Manuell sind kleinere Tableau-Bäume sehr hilfreich, aber größere Exemplare rasch unhandlich. Dann evtl. → syntaktische DNF-Umformung.

Beweisprogramme arbeiten teils mit **Resolutions**-, teils mit **Tableau**-Methoden.

SAT = Entscheidung der AL-Erfüllbarkeit

Was geht am schnellsten?

- Äquivalente **DNF-Umformung** + Inspektion
- Äquivalente **KNF**-Umformung + **Resolution**
- Wahrheitstafel + Inspektion
- **Tableau** + Inspektion
- (später noch: **Deduktion**)

Das hängt von den Formeln ab!

- DNF-nah oder KNF-nah?
- Formel lang oder kurz?
- Viele oder wenige Aussagevariablen?
- Sowie von den nichtdeterministischen Auswahlen und Reihenfolgen.

Alle 4 (oder 5) Methoden entscheiden zwischen

- **Erfüllbarkeit** und **Widersprüchlichkeit**,
- also auch über (endliche) **Folgerung**, sowie zwischen
- **Tautologie** und **Widerlegbarkeit**.(vgl. Reduktionsdiagramm).

Bedeutung von SAT

- Alle bekannten Algorithmen zur Entscheidung der Erfüllbarkeit (SAT) erfordern (bis zu) **exponentiell** wachsenden Zeit- bzw. Rechenaufwand in Abhängigkeit von der Komplexität der Formel.
- Das Problem SAT ist **NP-vollständig**, d.h. NP und mindestens so komplex wie jedes andere NP-Problem.
- **NP**: Polynomialer Aufwand, die richtige Lösung zu **verifizieren**;
  sie lässt sich aber evtl. nicht mit polynomialem Aufwand **bestimmen**.
- **Polynomial**er Aufwand: Existiert Algorithmus: Problemgröße g $\Rightarrow$ Anzahl der Lösungsschritte geringer als Polynom n-ten Grades von g.
- Wer beweist, dass SAT nicht mit polynomialem Aufwand zu lösen ist, beweist damit **P≠NP** und wird berühmt. Wer einen polynomial komplexen Algorithmus für SAT findet, beweist **P=NP** und wird berühmt – aber nicht unbedingt beliebt. (**kryptologische Katastrophe**).

- Auf jeden Fall wird man auf beide Arten **Dollarmillionär**:
  http://www.claymath.org/millennium-problems/p-vs-np-problem

Logik-Programmierung („LP“)

Logik-Programmierung
verwendet Programmiersprachen,
in denen wir

- uns Logik-nah ausdrücken,
- unser Wissen in Formeln eingeben und
- Fragen nach Formeln stellen

können.

Logik-Programmierung – Beispiel

Beispieldialog | **Zustandsänderungen**

Anfangszustand: $Wissen := \emptyset$

Eingaben: `es_regnet.`

$Wissen := Wissen \cup \{es\_regnet\} = \{es\_regnet\}$

`(es_regnet ∧ kein_schirm) → werde_nass.`

$Wissen := Wissen \cup \{(es\_regnet \wedge kein\_schirm) \rightarrow werde\_nass\}$

`werde_nass?`

$(Wissen \models werde\_nass)?$

Ausgabe: `nein.`

nicht für: $Wissen \models \neg\, werde\_nass$ **sondern:** $Wissen \not\models werde\_nass$

Eingabe: `kein_schirm.`

$Wissen := Wissen \cup \{kein\_schirm\}$

Eingabe: `werde_nass?`

Ausgabe: `ja.`

denn jetzt: $Wissen \models werde\_nass$

Diese Logik-Programmierung …

- akzeptiert drei Arten von Eingaben:
  - **Fakt** (Aussagevariable.)
  - **Regel** (Konjunktion von Aussagevar‘en → Aussagevariable.)
  - **Frage** oder **Ziel** (Aussagevariable?)
- sammelt die eingegebenen Fakten und Regeln in einer Wissensbasis: Alle bisher eingegebenen Fakten und Regeln sollen wahr sein.
- beantwortet die Frage nach einer Aussagevariable P …
  - mit **ja**, wenn aus der Wissensbasis P folgt;
  - mit **nein**, wenn aus der Wissensbasis (noch nicht) P folgt;

beschränkte Ausdrucksmittel ⟹ besonders **effiziente Beantwortung der Fragen (folgerbar oder nicht?)**

Erweiterungsmöglichkeiten (1)

Die Prämisse (linke Seite) einer Regel

darf **Disjunktion von** Konjunktionen von Aussagevariablen sein,

denn z.B. ist

$((A_1 \wedge A_2) \vee ((B_1 \wedge B_2 \wedge B_3)) \rightarrow C.$

ersetzbar durch

$(A_1 \wedge A_2) \rightarrow C.$
$(B_1 \wedge B_2 \wedge B_3) \rightarrow C.$

Vgl.

i. Wenn es regnet oder jemand mit der Gießkanne gießt, ist die Erde nass.
ii. Wenn es regnet, ist die Erde nass, und wenn jemand mit der Gießkanne gießt, dann auch.

Erweiterungsmöglichkeiten (2)

Die Konklusion (rechte Seite) einer Regel, sowie eine Frage

darf **Konjunktion von** Aussagevariablen sein,

denn z.B. ist

$((A_1 \wedge A_2) \rightarrow C \wedge D.$

ersetzbar durch

$(A_1 \wedge A_2) \rightarrow C.$

$(A_1 \wedge A_2) \rightarrow D.$

sowie

$E \wedge F$?

ersetzbar durch

$E \wedge F \rightarrow G$ (*G neu*)

G ?

Vgl.

i. Wenn es regnet, ist die Erde nass und Regenwürmer kommen heraus.
ii. Wenn es regnet, ist die Erde nass,
und außerdem kommen dann Regenwürmer heraus.

Markierungsalgorithmus (1)

Gegeben sei eine Wissensbasis aus Fakten und Regeln, gefolgt von einer Frage (Ziel) $A_1 \wedge \ldots \wedge A_n$? .
Modelliere das Ziel als weitere Regel $A_1 \wedge \ldots \wedge A_n \rightarrow$ **ja**.

1. Schreibe eine **Liste der** vorkommenden **Aussagevariablen** plus **ja**. Jede von ihnen ist zunächst **unmarkiert**.
2. Markiere für jeden Fakt A_n. dessen Aussagevariable A_n .
3. Wenn **ja** markiert ist, so endet die Berechnung mit dem Ergebnis **JA**.
4. Wenn
   - eine **Regel** $A_1 \wedge \ldots \wedge A_n \rightarrow A_k$ bzw.$\rightarrow$**ja** in der Wissensbasis steht,
   - und alle $A_1, ..., A_n$ sind markiert,
   - und A_k bzw. **ja** ist noch nicht markiert,

   dann **markiere** auch A_k bzw. **ja** und gehe nach 3.
5. (*Wenn also die Berechnung nicht in 3. stoppte und 4. nicht (mehr) anwendbar war*)
   Die Berechnung endet mit dem **Ergebnis *NEIN***.

Markierungsalgorithmus (2)

Satz: **Markierungsalgorithmus**

Der Markierungsalgorithmus **terminiert**
und **beantwortet** die Frage, ob das Ziel
eine logische Folge der bisherigen Wissensbasis –
d.h. der eingegebenen Fakten und Regeln – ist.

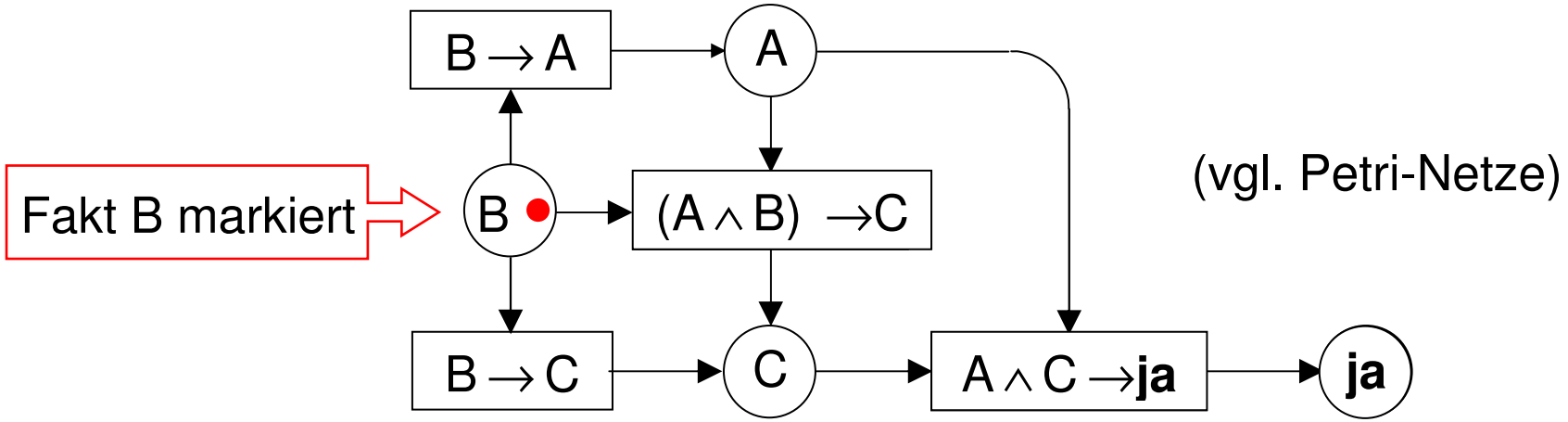
Markierungsalgorithmus: grafisches Beispiel

$B \rightarrow A.\quad (A \wedge B) \rightarrow C.\quad B \rightarrow C.\quad B.\quad A \wedge C\ ?$

Stellen (◯) modellieren Aussagevariablen und **ja**.
Man markiert sie mit •, sobald man „sie weiß“.

Jede **Transition** (▭) (eine pro Regel) markiert ihre Ausgangsstellen, sobald alle ihre Eingangsstellen markiert sind.

ja markiert? → Antwort **ja**.



Wieso werden der Reihe nach alle Stellen markiert?

(insbesondere auch **ja**)

Ü34

Literaturverständnis: PROLOG-Stil

Es gibt verschiedene, leicht unterschiedliche PROLOG-Formen, hier ein **Stil-Beispiel**:

```
es_regnet.                                          es_regnet.

(es_regnet ∧ kein_schirm) → werde_nass.
                        werde_nass :- es_regnet, kein_schirm.

werde_nass?                                         ?- werde_nass.
```

Literaturverständnis: Horn-Logik-Stil

Traditionell werden die hinter

- den programmiersprachlichen „Befehlen“
  (Fakten, Regeln, Ziele)
  stehenden logischen Formeln

und die hinter

- der Leistung der „Programme“
  (Ziel ableitbar?)
  stehenden semantischen Fragen

etwas anders (aber im Ergebnis nicht wirklich anders!) gewählt:

Alle **Befehle** werden als sog. **Horn-Klauseln** interpretiert.

Eine Horn-Klausel ist eine
Klausel mit höchstens einem positiven Literal,

Die Frage nach einem Ziel $A_1 \wedge \ldots \wedge A_n$

wird angesehen als die Frage, ob die Wissensbasis,
erweitert um die Negation des Ziels $\neg(A_1 \wedge \ldots \wedge A_n)$,
unerfüllbar ist

Arten von Horn-Klauseln

Regeln: 1 positives + mindestens ein negatives Literal

Beispiel: $\boxed{\neg \text{es\_regnet} \vee \neg \text{kein\_schirm} \vee \text{werde\_nass}}$

Fakten: 1 positives Literal, keine negativen

Beispiel: $\boxed{\text{es\_regnet.}}$

Fragen/Ziele negiert: nur negative Literale

Beispiel: $\boxed{\neg \text{kein\_schirm} \vee \neg \text{werde\_nass}}$

M Formelmenge: $M \models \varphi \;\Leftrightarrow\; M \cup \{\neg\varphi\}$ unerfüllbar $\;\Leftrightarrow\; M \cup \{\neg\varphi\} \models \bot$

Horn-Klauseln als Implikationen

Regeln: 1 positives + mindestens ein negatives Literal

Beispiel: $\neg$es_regnet $\vee$ $\neg$kein_schirm $\vee$ werde_nass.

$\neg$(es_regnet $\wedge$ kein_schirm) $\vee$ werde_nass

de Morgan

als **Implikation**: **es_regnet $\wedge$ kein_schirm $\rightarrow$ werde_nass**

Fakten: 1 positives Literal, keine negativen

Beispiel: es_regnet.

als **Implikation**: $\top \rightarrow$ **es_regnet**

Fragen, Ziele: nur negative Literale

Beispiel: $\neg$kein_schirm $\vee \neg$werde_nass.

als **Implikation**: **kein_schirm $\wedge$ werde_nass** $\rightarrow \bot$.

Vor- und Nachteile der PROLOG/Horn-Logik

Vorteil

- Der Markierungsalgorithmus erfordert nur **polynomialen** Aufwand.

Nachteil

- **Nicht** zu **allen Formeln** existiert eine äquivalente PROLOG-Formel.
- **Gegenbeispiel**: $A \vee B$!

OBDD-Vorbereitung – Erinnerung an die ITE-Form

ITE-Form-Satz

Zu jeder Formel gibt es mindestens eine äquivalente (sogar **geordnete)** ITE-Formel.

Unser ITE-Algorithmus (→ Junktorenbasen!) liefert eine **spezielle ITE-Formel**, nämlich

- mit ausschließlich **Konstanten** in den **THEN/ELSE-Blatt-Positionen**,
- mit ausschließlich **Variablen** in den **IF-Positionen**,
- und alle **Variablen** werden in jedem Zweig des Formelbaumes in **derselben** (willkürlich gewählten) **Reihenfolge** abgefragt.

ITE-Form – Beispiel

$(A \vee B) \rightarrow C$

(1) Shannon-Expansion nach C

$\equiv C \rightarrow ((A \vee B) \rightarrow \mathrm{T}) \ / \ ((A \vee B) \rightarrow \bot))$

(2) Shannon-Expansion nach B

$\equiv B \rightarrow (C \rightarrow ((A \vee \mathrm{T}) \rightarrow \mathrm{T}) / ((A \vee \mathrm{T}) \rightarrow \bot)) / (C \rightarrow ((A \vee \bot) \rightarrow \mathrm{T}) / ((A \vee \bot) \rightarrow \bot))$

(3) Shannon-Expansion nach A

usw.

$\equiv A \rightarrow (B \rightarrow (C \rightarrow ((\mathrm{T} \vee \mathrm{T}) \rightarrow \mathrm{T}) \ / \ldots$ usw.

(4) konstante Terme auswerten

$\equiv A \rightarrow (B \rightarrow (C \rightarrow \mathrm{T}) / \ldots$ usw.

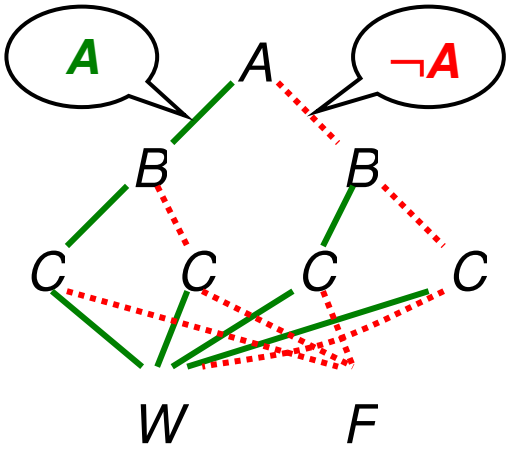
W-Fälle:
ABC, A-BC, -ABC, -A-BC, -A-B-C

F-Fälle:
AB-C, A-B-C, -AB-C

Darstellung als Baum*: ...

*) T‘s bzw. $\bot$‘s als W bzw. F übereinander gelegt

Binäre Entscheidungsgraphen – Vorschau

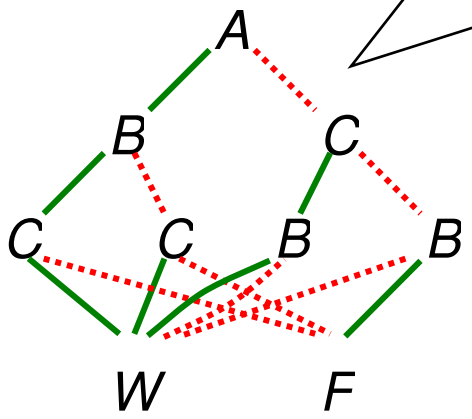


Formelbaum+Auswertung,
ein **OBDD**,
Ordered Binary Decision Diagram

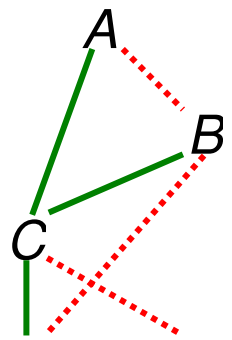
Ordnung:
hier A vor B vor C –
müssen aber nicht auf allen Wegen alle vorkommen!

W-Fälle:
ABC, A-BC, -ABC, -A-BC, -A-B-C

F-Fälle:
AB-C, A-B-C, -AB-C

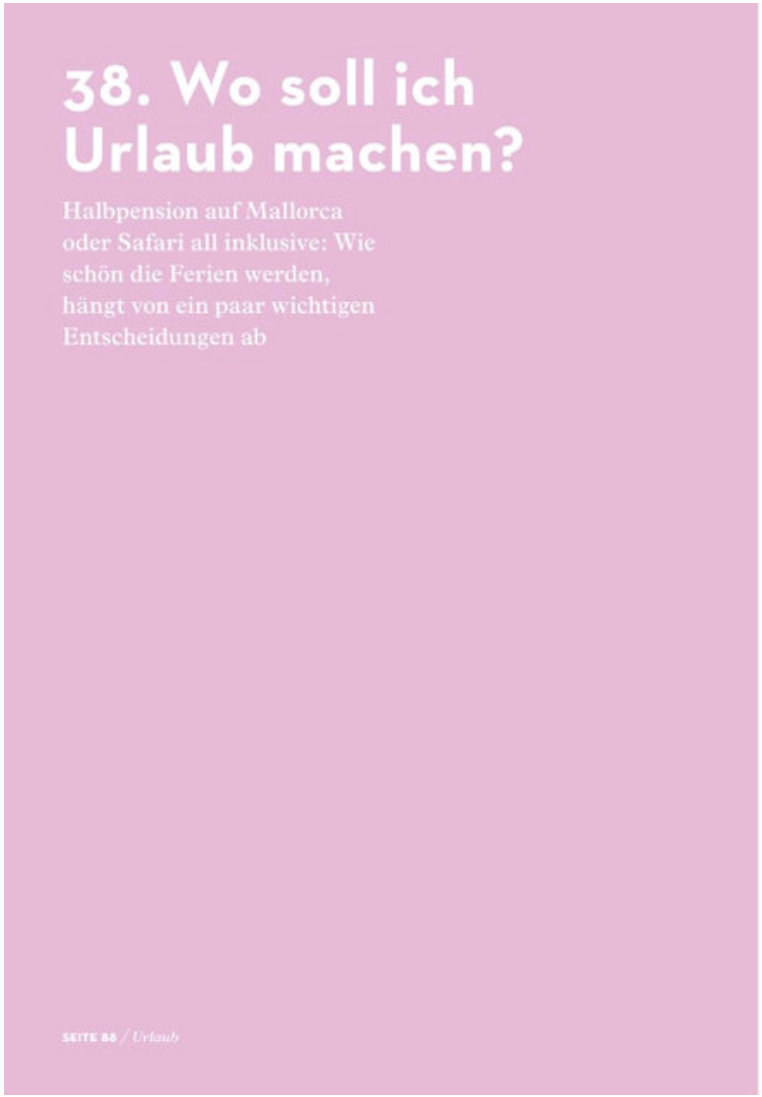


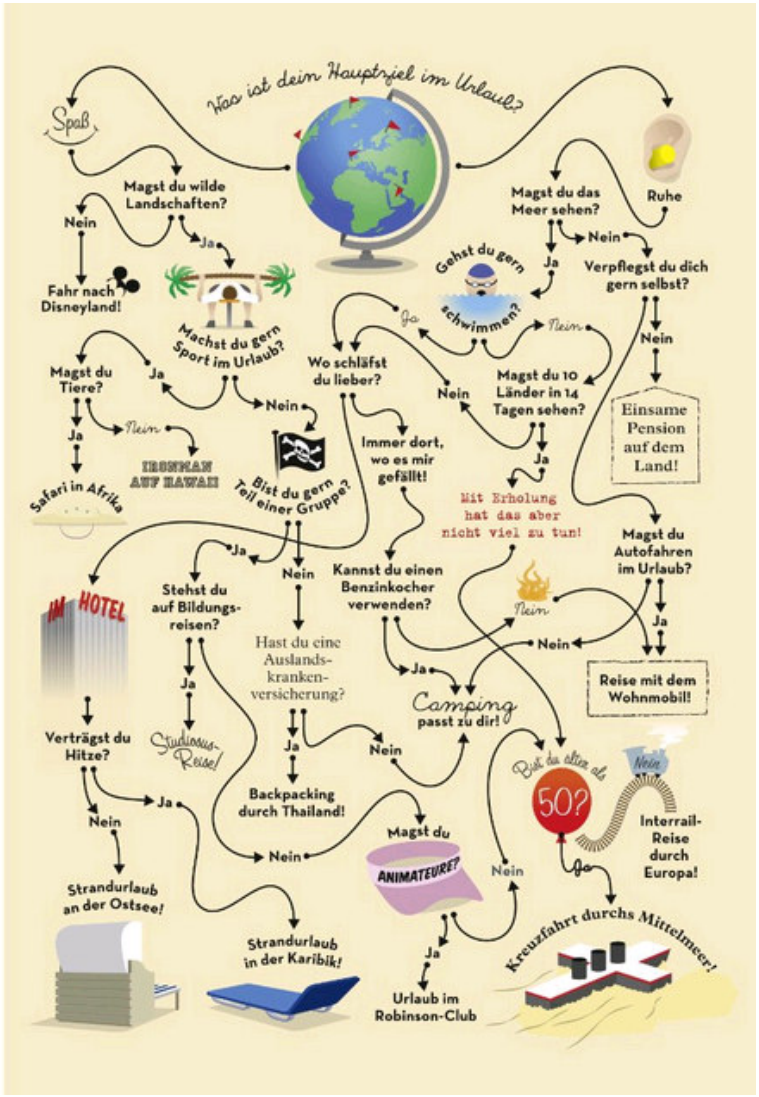
ein äquivalenter **BDD**
(**ungeordnet**, aber ohne Variablenwiederholung, -auslassung und „unmögliche Kanten“)



ein äquivalenter reduzierter **RO**BDD

Binäre Entscheidungsgraphen – unterhaltsam





Heike Kottmann: Entweder vielleicht oder doch lieber ja. Knaur, 2012

BDDs, OBDDs, ROBDDs (1)

Ein (aussagenlogisches) **BDD** – **Binary Decision Diagram** –
ist ein Kanten- und Knoten-beschrifteter
verwurzelter zyklenfreier gerichteter Graph *G* mit:

- Jeder terminale Knoten ist mit W oder mit F beschriftet.
- Jeder nicht-terminale Knoten
  - ist mit einer Aussagevariablen beschriftet und
  - hat zwei Ausgangskanten, eine mit W, eine mit F beschriftet.
- Er ist widerspruchsfrei: z.B nicht P-F-Q-w-W und Q-w-P-F-F

BDDs, OBDDs, ROBDDs (2)

Zeichenkonventionen

- Alle Kanten laufen abwärts, ohne Pfeilspitzen
- **W**-Kanten (ohne „W“) **durchgehend** und **F**-Kanten **gestrichelt**.
- (Nichtterminalknoten werden oft durch Kreise, Terminalknoten durch Quadrate eingerahmt.)

Ein BDD heißt **geordnet** – **Ordered Binary Decision Diagram, OBDD** – wenn:

- auf den Aussagevariablen eine lineare Ordnung π der Art existiert, dass für jede Kante von einem nicht-terminale Knoten mit Beschriftung P zu einem nicht-terminalen Knoten mit Beschriftung Q gilt: $P <_\pi Q$.

Ein OBDD heißt **reduziert** – Reduced **Ordered Binary Decision Diagram, ROBDD**, wenn er die **kleinstmögliche Knotenzahl** (für diese Ordnung π !) hat.

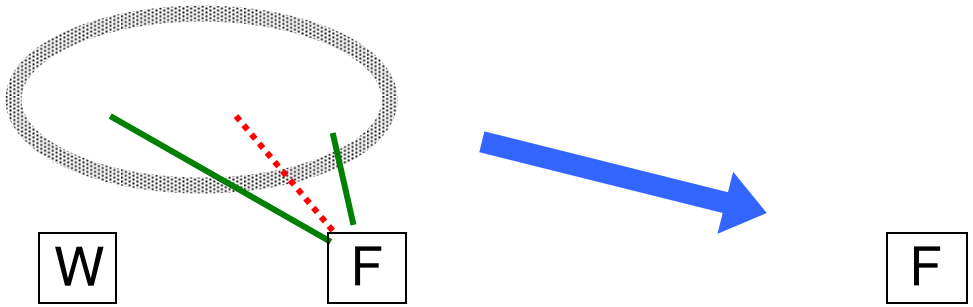
Erzeugung von ROBDDs (1)

- **Wie verschaffen wir uns eine reduzierte Form eines gegebenen OBDDs?**

Algorithmus RedOBDD: **Reduktion eines OBDD**

Zunächst:

0. Wenn der W- oder F-Knoten **isoliert** ist,
   dann wird sein **Gegenteil** der **einzige Knoten**.



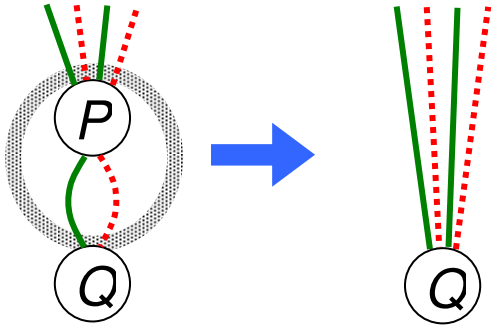
Erzeugung von ROBDDs (2)

- **Wie verschaffen wir uns eine reduzierte Form eines gegebenen OBDDs?**

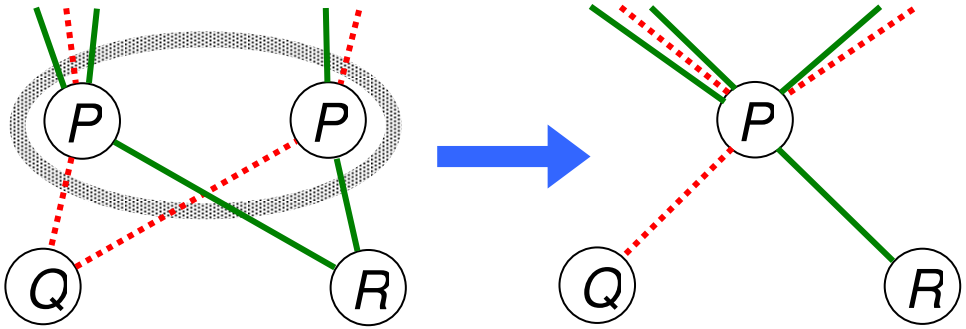
Algorithmus RedOBDD:
Solange möglich:

1. **Überspringe** eine „folgenlose Fallunterscheidung“ – bzw.

Reduktion eines OBDD



2. **verschmelze** zwei Knoten mit „gleicher Fallunterscheidung“



Eigenschaften von ROBDDs

Korrektheit und mehr …

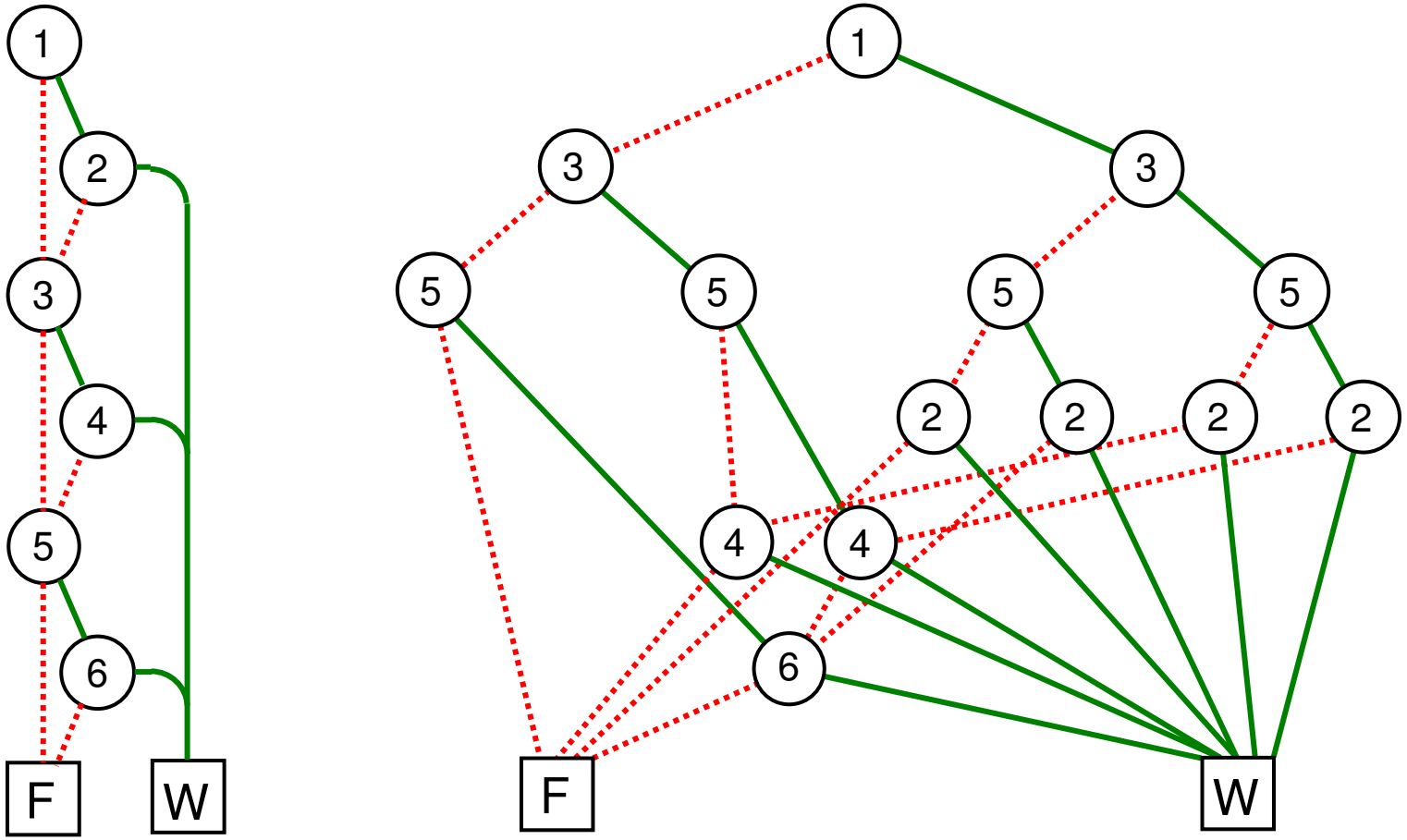
- Der Algorithmus **RedOBDD terminiert** stets.
- Das durch Umformung dann erhaltene BDD ist
  - ein **R**OBDD,
  - **eindeutig** (**kanonisch**) für Ordnung π und
  - zum Ausgangs-OBDD **äquivalent**.

Ü35

BDD – die Variablenreihenfolge

Effizienz ist **reihenfolgeabhängig**, linear bis exponentiell

Beispiel $\varphi = (A_1 \wedge A_2) \vee (A_3 \wedge A_4) \vee (A_5 \wedge A_6)$



BDD – Anwendungen

ROBDDs als Datenstruktur erlauben **oft** rechnerisch **effiziente** Darstellungen von Formeln und effiziente Berechnungen von Erfüllbarkeit und Äquivalenz.

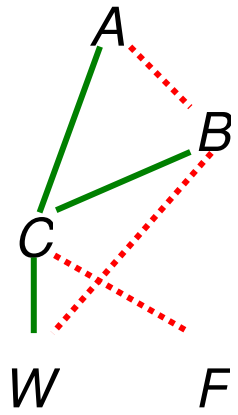
- **Äquivalent** sind Formeln genau dann, wenn sie **den gleichen ROBDD** haben.
- **Erfüllbar** ist eine Formel genau dann, wenn **F im ROBDD nicht einziger** Knoten ist.

- Man kann Junktoren auch **auf OBDDs** anwenden.
- Optimierungsmöglichkeiten, z.B. mehrere Zeiger.
- **SAT** bleibt **allgemein exponentiell aufwendig**, wird aber in vielen praktischen Fällen machbar.
- **Nutzung: VLSI-CAD**, **Model Checking**

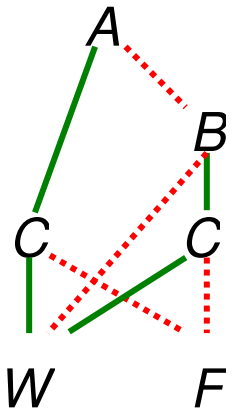
OBDDs und ITE-T-⊥-Formeln

Jedes OBDD lässt sich äquivalent als ITE-Formel schreiben, und umgekehrt: Der hier gewählte Übergang (das „Rezept“) ist bezüglich der ITE-Standardform, d.h. IF nur mit AV‘n, Blätter nur Konstanten) surjektiv aber nicht injektiv:

Beispielsweise ist das OBDD …



aber auch das OBDD …



„offenbar“ äquivalent zu der ITE-Formel

$$\varphi = A \to (C \to \mathrm{T}/\bot)/(B \to (C \to \mathrm{T}/\bot)/\bot)$$

Nach welchem Rezept wird die Formel/der Graph gebildet?

und daraus lässt sich leicht automatisch das rechte OBDD konstruieren. Beide Richtungen sind (natürlich) über **rekursive Abbildungen** definiert.