Script generated by TTT

Title: Esparza: Diskrete Strukturen (07.11.2013)

Date: Thu Nov 07 10:15:25 CET 2013

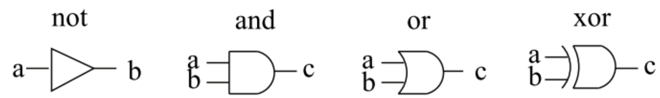
Duration: 88:46 min

Pages: 41

Modelling circuits with QBL

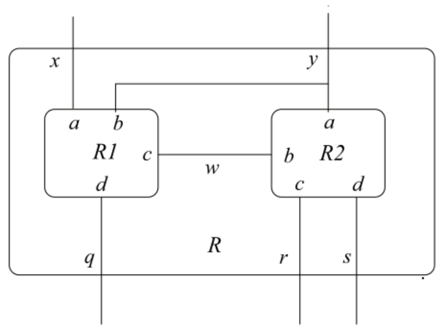
Gates as boolean formulas

Stable states as satisfying truth assignments



$$\mathbf{not}(a,b) \equiv \neg a \leftrightarrow b \qquad \mathbf{and}(a,b,c) \equiv (a \wedge b) \leftrightarrow c$$
$$\mathbf{or}(a,b,c) \equiv (a \vee b) \leftrightarrow c \qquad \mathbf{xor}(a,b,c) \equiv ((\neg a \wedge b) \vee (a \wedge \neg b)) \leftrightarrow c$$

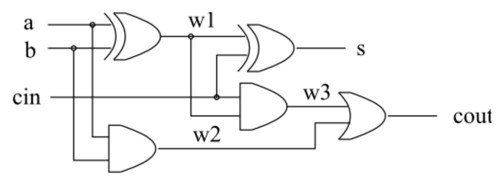
Combine gates with $\wedge$, $\exists$ (and renaming of variables)



$$R(x,y,q,r,s) = \exists w.\, R_1(x,y,w,q) \wedge R_2(y,w,r,s)$$

A full adder

	cout	*cin*	...
	...	*a*	...
+	...	*b*	...
	...	*s*	...

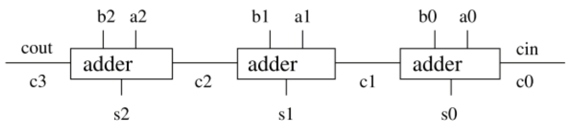


$$\mathbf{full\_adder}(a,b,s,cin,cout) \equiv$$
$$\exists w_1, w_2, w_3.\, \mathbf{xor}(a,b,w_1) \wedge \mathbf{xor}(w_1,cin,s) \wedge \mathbf{and}(a,b,w_2) \wedge$$
$$\mathbf{and}(cin,w_1,w_3) \wedge \mathbf{or}(w_3,w_2,cout)$$

An n-bit ripple-carry adder

$$
\begin{array}{cccc l}
 & \color{red}{c_2} & \color{red}{c_1} & \color{red}{cin} & (=0) \\
 & a_2 & a_1 & a_0 & \\
+ & b_2 & b_1 & b_0 & \\
\hline
cout & s_2 & s_1 & s_0 &
\end{array}
$$

Wire together n 1-bit adders where ith carry-out is i+1st carry-in, first carry is the carry-in and last is the carry-out.



We obtain the formula

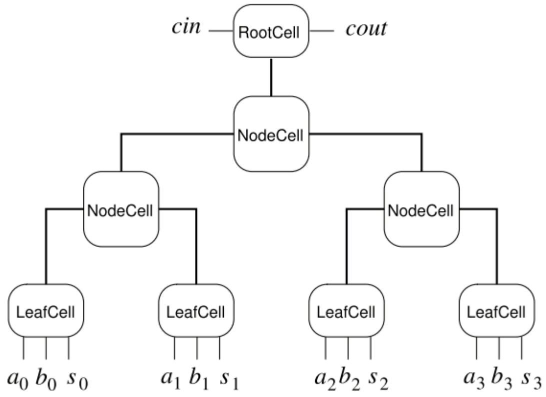
$$
\begin{aligned}
&\mathbf{adder}_n(a_0,\ldots,a_{n-1},b_0,\ldots,b_{n-1},s_0,\ldots,s_{n-1},\ cin, cout) \equiv \\
&\quad \exists c_0,\ldots,c_n.\,(c_0 \leftrightarrow cin) \wedge (c_n \leftrightarrow cout) \wedge \\
&\qquad \bigwedge_{i=1}^{n-1} \mathbf{full\_adder}(a_i,b_i,s_i,c_i,c_{i+1}))
\end{aligned}
$$

Problem: too slow!!

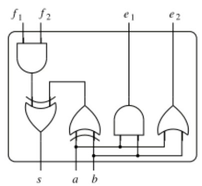
Each c_i can only be computed after all of $c_{i-1},\ldots,c_0$ have been computed

Delay: $2n+2$ time units for n-bit numbers

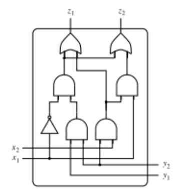
Description of the circuit (for 4 bits)



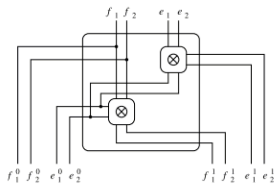
Description of the circuit II



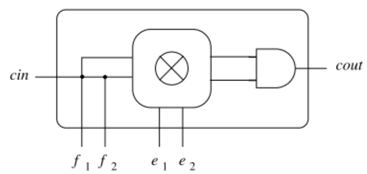
LeafCell circuit



$\otimes$ circuit



NodeCell circuit



RootCell circuit

Verification of the carry-look-ahead n-adder

Check if

$$\textbf{adder}_n(a_0,\ldots,a_{n-1},b_0,\ldots,b_{n-1},s_0,\ldots,s_{n-1},\ cin, cout)$$
$$\Leftrightarrow$$
$$\textbf{cla}_n(a_0,\ldots,a_{n-1},b_0,\ldots,b_{n-1},s_0,\ldots,s_{n-1},\ cin, cout)$$

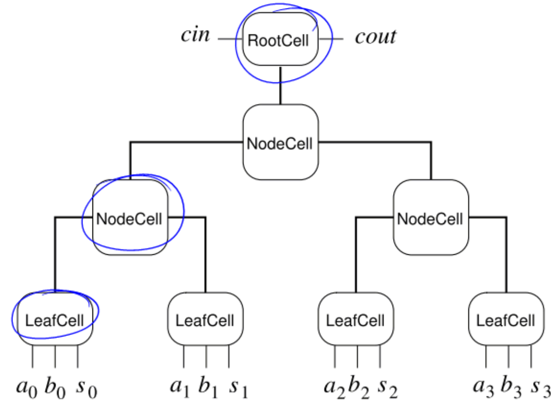
Use SAT solvers or QBF solvers

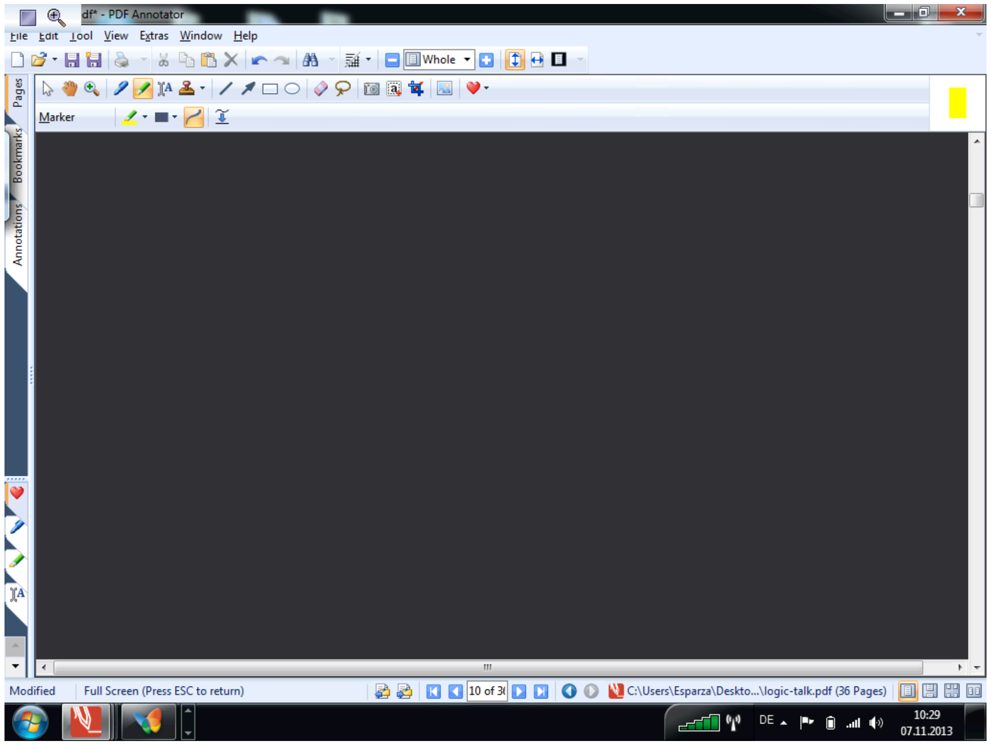
Results of the SAT 2002 competition on a variant of this problem:

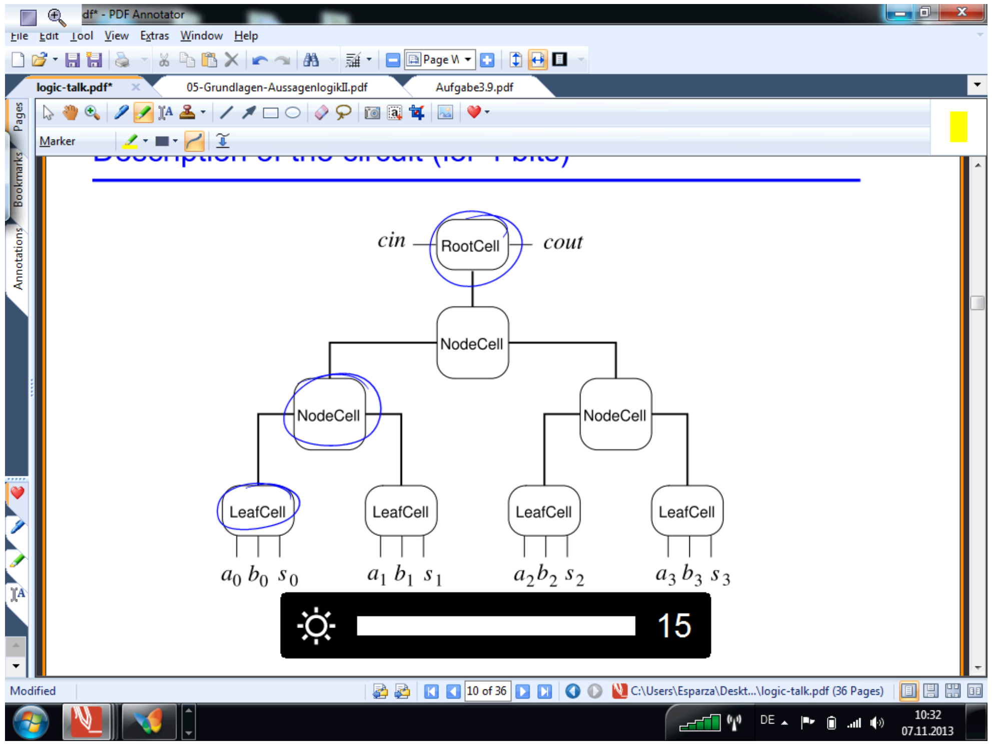
- Task was to compare 2, 4, 8, . . . , 256 bits adders (8 problems)
- From 26 variables and 70 3CNF clauses to 4584 variables and 13226 clauses
- Fastest solver (Zchaff) checked all 8 problems in 14 seconds
- More info at www.satlive.org/SATCompetition/2002/index.jsp

Rule-of-thumb: circuits with some hundreds of gates are routinely solved

Description of the circuit (for 4 bits)



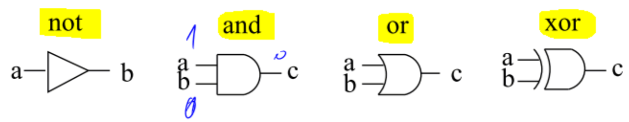




Modelling circuits with QBL

Gates as boolean formulas

Stable states as satisfying truth assignments



$\textbf{not}(a,b) \equiv \neg a \leftrightarrow b$ $\qquad$ $\textbf{and}(a,b,c) \equiv (a \wedge b) \leftrightarrow c$

$\textbf{or}(a,b,c) \equiv (a \vee b) \leftrightarrow c$ $\qquad$ $\textbf{xor}(a,b,c) \equiv ((\neg a \wedge b) \vee (a \wedge \neg b)) \leftrightarrow c$

Aufgabe 3.9

- Zu zeigen: NAND und NOR sind die einzigen vollständigen binären Operatoren
- Korrekte Lösungen von:
  - Maximilian Schmidt
  - Martin Wauligmann (?)
  - Nathanel Schilling (?)

Aufgabe 3.9

- Ein binärer Operator $\boldsymbol{Op}$ hat eine Wahrheitstabelle der Gestalt

F	G	$F\ \boldsymbol{Op}\ G$
0	0	x
0	1	y
1	0	z
1	1	w

Aufgabe 3.9

- Sei H eine beliebige Formel, die nur aus Variablen und (vielleicht) $\boldsymbol{Op}$ besteht.

F	G	$F\ \boldsymbol{Op}\ G$
0	0	0
0	1	y
1	0	z
1	1	w

Fall 1. $x = 0$.
Sei β_0 die minimale Belegung von H, die allen Variablen 0 zuordnet. Es gilt $[H](\beta_0) = 0$.
Damit gilt z.B. $F \not\equiv \neg p$

Aufgabe 3.9

- Sei H eine beliebige Formel, die nur aus Variablen und (vielleicht) $\boldsymbol{Op}$ besteht.

F	G	$F\ \boldsymbol{Op}\ G$
0	0	1
0	1	0
1	0	0
1	1	0

F	G	$F\ \boldsymbol{Op}\ G$
0	0	1
0	1	1
1	0	1
1	1	0

Fall 3. $x = 1, w = 0, y = z.$

NAND und NOR.

Aufgabe 3.9

- Sei H eine beliebige Formel, die nur aus Variablen und (vielleicht) $\boldsymbol{Op}$ besteht.

F	G	$F\ \boldsymbol{Op}$
0 1	0 1	1
0 1	1 0	0
1 0	0 1	1
1 0	1 0	0

F	G	$F\ \boldsymbol{Op}$
0	0	1
0	1	1
1	0	0
1	1	0

New
Size
Current page size
Style
5mm
7mm
Orientation
Pages
1
OK
Cancel

Fall 4. $x = 1, w = 0, y \neq z.$

Die Tabellen bleiben invariant durch den Tausch $0 \mapsto 1, 1 \mapsto 0$.

Es gilt dann $[H](\beta_0) \neq [H](\beta_1)$

und so z.B $H \not\equiv p \rightarrow q$

Aufgabe 3.9

- Sei H eine beliebige Formel, die nur aus Variablen und (vielleicht) $\boldsymbol{Op}$ besteht.

F	G	$F\ \boldsymbol{Op}\ G$
0 1	0 1	1 0
0 1	1 0	0 1
1 0	0 1	1 0
1 0	1 0	0 1

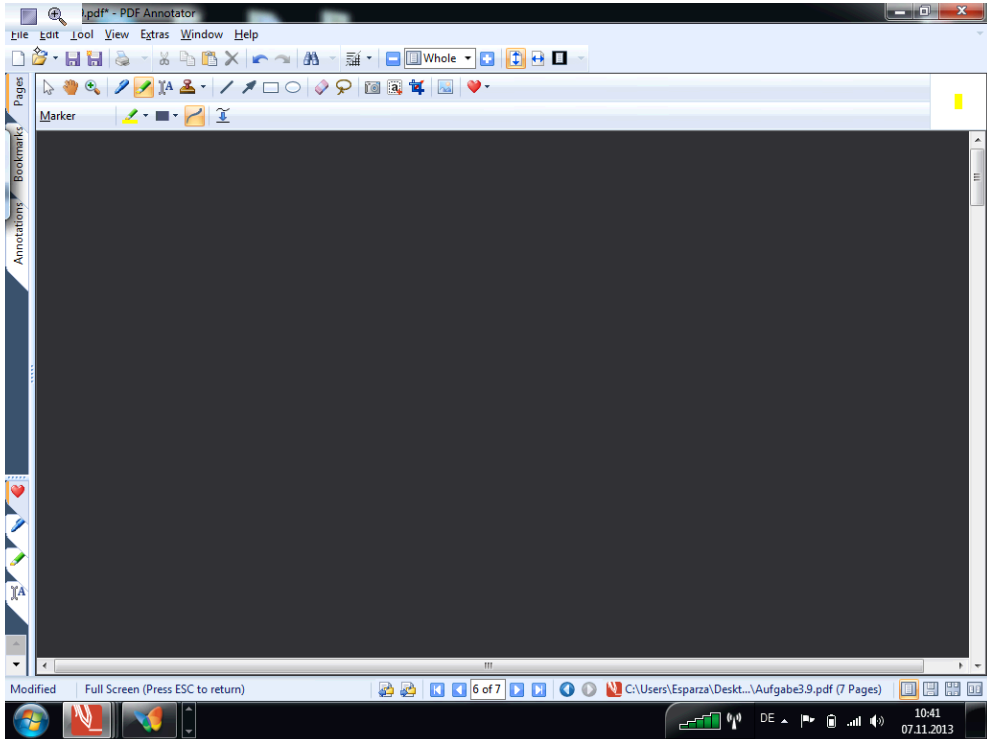
F	G	$F\ \boldsymbol{Op}\ G$
0	0	1
0	1	1
1	0	0
1	1	0

Fall 4. $x = 1, w = 0, y \neq z.$

Die Tabellen bleiben invariant durch den Tausch $0 \mapsto 1, 1 \mapsto 0$.

Es gilt dann $[H](\beta_0) \neq [H](\beta_1)$

und so z.B $H \not\equiv p \rightarrow q$



- DPLL (Davis-Putnam-Logemann-Loveland).

Definition. Sei F eine KNF-Formel und sei p eine Variable von F.

$F[p\backslash\mathbf{true}]$ bezeichnet die Formel, die ensteht, in dem jedem Vorkommis von p in F durch **true** ersetzt wird und das Ergebnis mit Hilfe der Regeln

$$F \wedge \mathbf{true} \equiv F \qquad F \vee \mathbf{true} \equiv \mathbf{true} \qquad \neg\mathbf{true} \equiv \mathbf{false}$$

$$F \wedge \mathbf{false} \equiv \mathbf{false} \qquad F \vee \mathbf{false} \equiv F \qquad \neg\mathbf{false} \equiv \mathbf{true}$$

vereinfacht wird.

$F[p\backslash\mathbf{false}]$ wird analog definiert.

- Normalformen
  - Ein Literal ist eine Aussagenvariable oder die Negation einer Aussagenvariable.
  - Eine Klausel ist eine Disjunktion von Literalen oder die Formel **false** (Disjunktion von 0 Literalen)
  - Eine Formel in konjunktiver Normalform (KNF) ist
    - eine Konjunktion von Klauseln $(1_{23} \vee 1_{24} \vee 1_{2f}$

      (d.h., eine Formel der Form $(\bigwedge_{i=1}^{n}(\bigvee_{j=1}^{m_i} L_{i,j}))$

      mit $L_{i,j} \in \{A_1, A_2, \ldots\} \cup \{\neg A_1, \neg A_2, \ldots\}$)
    - oder die Formel **true** (Konjunktion von 0 Klauseln).

- Normalformen $p \quad \neg p \quad (p \vee q) \quad (p \vee q \vee \neg r) \quad (\neg p \vee \neg p)$
  - Ein Literal ist eine Aussagenvariable oder die Negation einer Aussagenvariable. p
  - Eine Klausel ist eine Disjunktion von Literalen oder die Formel **false** (Disjunktion von 0 Literalen) p
  - Eine Formel in konjunktiver Normalform (KNF) ist
    - eine Konjunktion von Klauseln $(1_{23} \vee 1_{24} \vee 1_{2f}$

      (d.h., eine Formel der Form $(\bigwedge_{i=1}^{n}(\bigvee_{j=1}^{m_i} L_{i,j}))$

      mit $L_{i,j} \in \{A_1, A_2, \ldots\} \cup \{\neg A_1, \neg A_2, \ldots\}$)
    - oder die Formel **true** (Konjunktion von 0 Klauseln).

- Normalformen
  - Eine Formel ist in KNF falls in allen Pfaden ihres Syntaxbaums Konjunktionen vor Disjunktionen vor Negationen vorkommen.

Kapitel II – Grundlagen; Logik

- Normalformen: Umformunsmethode
  - Formeln aus der Praxis sind oft in KNF
  - Außerdem, das folgende Verfahren konstruiert für eine beliebige Formel F eine äquivalente KNF-Formel:
  1. Ersetze in F jedes Vorkommen einer Teilformel der Bauart

     $\neg\neg G$ durch G

     $\neg(G \wedge H)$ durch $(\neg G \vee \neg H)$

     $\neg(G \vee H)$ durch $(\neg G \wedge \neg H)$

     bis keine derartige Teilformel mehr vorkommt.

Kapitel II – Grundlagen; Logik

2. Ersetze jedes Vorkommen einer Teilformel der Bauart

   $(F \vee (G \wedge H))$ durch $((F \vee G) \wedge (F \vee H))$

   $((G \wedge H) \wedge F)$ durch $((F \vee G) \wedge (F \vee H))$

   bis keine derartige Teilformel mehr vorkommt.

Alle Schritte erhalten äquivalenz.

Nach Schritt 1. kommen im Syntaxbaum Konjunktionen und Disjunktionen vor Negationen vor.

Nach Schritt 2 kommen im Syntaxbaum Konjunktionen vor Disjunktionen vor.

Kapitel II – Grundlagen; Logik

- Algorithmen für KNF-SAT
  - Wir kennen bisher nur den folgenden Algorithmus:
    - Erzeuge alle minimalen Belegungen, die zur Formel passen.
    - Für jede solche Belegung, prüfe, ob sie die Formel erfüllt.
  - Zwei Probleme
    - Es müssen 2^n Belegungen geprüft werden !!
    - Wenn die Formel unerfüllbar ist, dann werden mit Sicherheit **alle** Belegungen geprüft.

Kapitel II – Grundlagen; Logik

- Algorithmen für KNF-SAT
  - Wir präsentieren zwei Algorithmen:
    - DPLL (Davis-Putnam-Logemann-Loveland).
      - Versucht, die Anzahl der zu prüfenden Belegungen zu reduzieren
      - Auf Erfüllbarkeit gerichtet: kann früh terminieren, wenn die Formel erfüllbar ist.
      - Wir präsentieren nur eine vereinfachte Version.
    - Resolution (Robinson).
      - Auf Nicht-Erfüllbarkeit gerichtet: kann früh Terminieren, wenn die Formel nicht erfüllbar ist.

Kapitel II – Grundlagen; Logik

- DPLL (Davis-Putnam-Logemann-Loveland).

Definition. Sei F eine KNF-Formel und sei p eine Variable von F.
$F[p\backslash\mathbf{true}]$ bezeichnet die Formel, die ensteht, in dem jedem Vorkommis von p in F durch **true** ersetzt wird und das Ergebnis mit Hilfe der Regeln

$$F \wedge \mathbf{true} \equiv F \qquad F \vee \mathbf{true} \equiv \mathbf{true} \qquad \neg\mathbf{true} \equiv \mathbf{false}$$

$$F \wedge \mathbf{false} \equiv \mathbf{false} \qquad F \vee \mathbf{false} \equiv F \qquad \neg\mathbf{false} \equiv \mathbf{true}$$

vereinfacht wird.
$F[p\backslash\mathbf{false}]$ wird analog definiert.

Kapitel II – Grundlagen; Logik

- DPLL (Davis-Putnam-Logemann-Loveland).

Beispiel:

$$F = (p \vee \neg q \vee r) \wedge (\neg p \vee \neg r) \wedge (q \vee \neg r) \wedge p$$

$$F[p\backslash\mathbf{true}] = (\mathbf{true} \vee \neg q \vee r) \wedge (\neg\mathbf{true} \vee \neg r) \wedge (q \vee \neg r) \wedge \mathbf{true}$$

$$\equiv (\mathbf{true} \vee \neg q \vee r) \wedge (\mathbf{false} \vee \neg r) \wedge (q \vee \neg r)$$

$$\equiv \mathbf{true} \;\wedge \neg r \wedge (q \vee \neg r)$$

$$\equiv \neg r \wedge (q \vee \neg r)$$

Kapitel II – Grundlagen; Logik

- DPLL (Davis-Putnam-Logemann-Loveland).

Fakt. F ist erfüllbar gdw. $F[p\backslash\mathbf{true}]$ oder $F[p\backslash\mathbf{false}]$ erfüllbar sind.

Begründung.

Sei β Belegung mit $[F](\beta) = 1$.
Wenn $\beta(p) = 1$ dann gilt $[F](\beta) = [F[p\backslash\mathbf{true}]](\beta) = 1$.
Wenn $\beta(p) = 0$ dann gilt $[F](\beta) = [F[p\backslash\mathbf{false}]](\beta) = 1$.
Sei β Belegung mit $[F[p\backslash\mathbf{true}]](\beta) = 1$. Sei β' die Belegung mit $\beta'(p) = 1$, und $\beta'(q) = \beta(q)$ für $q \neq p$. Dann gilt $F[\beta] = 1$.
Sei β Belegung mit $[F[p\backslash\mathbf{false}]](\beta) = 1$. Sei β' die Belegung mit $\beta'(p) = 0$, und $\beta'(q) = \beta(q)$ für $q \neq p$. Dann gilt $F[\beta] = 1$.

Kapitel II – Grundlagen; Logik

- DPLL (Davis-Putnam-Logemann-Loveland).

Algorithmus 1:

Wenn $F = \mathbf{true}$ dann antworte „erfüllbar“
Wenn $F = \mathbf{false}$ dann antworte „unerfüllbar“
Wenn $\mathbf{true} \neq F \neq \mathbf{false}$ dann
wähle eine Variable p, die in F vorkommt;
prüfe, ob $F[p\backslash\mathbf{true}]$ oder $F[p\backslash\mathbf{false}]$ erfüllbar sind;
wenn mindestens eine von den beiden erfüllbar ist, antworte „erfüllbar“, sonst „unerfüllbar“

- DPLL (Davis-Putnam-Logemann-Loveland).
  - Algorithmus 1 kann durch eine geschickte Wahl der Variablen p beschleunigt werden.
  - Eine One-Literal-Klausel ist eine Klausel, die nur ein Literal enthält, i.e., eine Klausel der Gestalt $\{p\}$ oder der Gestalt $\{\neg p\}$.

Kapitel II – Grundlagen; Logik

- DPLL (Davis-Putnam-Logemann-Loveland).

Algorithmus 2:

Wenn $F = \mathbf{true}$ dann antworte „erfüllbar"
Wenn $F = \mathbf{false}$ dann antworte „unerfüllbar"
Wenn $\mathbf{true} \neq F \neq \mathbf{false}$ dann
  wenn F eine One-Literal-Klausel $\{p\}$ enthält
    dann prüfe, ob $F[p\backslash\mathbf{true}]$ erfüllbar ist;
    antworte „erfüllbar" gdw. $F[p\backslash\mathbf{true}]$ erfüllbar
  wenn F eine One-Literal-Klausel $\{\neg p\}$ enthält
    dann prüfe, ob $F[p\backslash\mathbf{false}]$ erfüllbar ist;
    antworte „erfüllbar" gdw. $F[p\backslash\mathbf{false}]$ erfüllbar
  wenn F keine One-Literal-Klausel enthält
    dann wähle eine Variable p, die in F vorkommt;
    prüfe, ob $F[p\backslash\mathbf{true}]$ oder $F[p\backslash\mathbf{false}]$ erfüllbar sind;
    wenn mindestens eine von den beiden erfüllbar ist,
    dann antworte „erfüllbar", sonst „unerfüllbar"

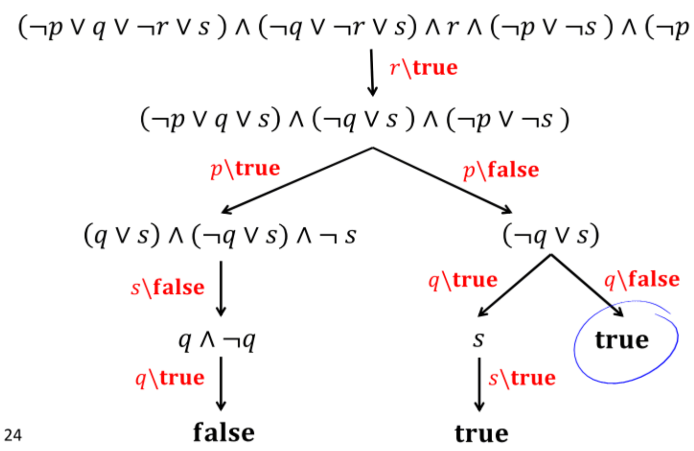
Kapitel II – Grundlagen; Logik

$$(\neg p \vee q \vee \neg r \vee s) \wedge (\neg q \vee \neg r \vee s) \wedge r \wedge (\neg p \vee \neg s) \wedge (\neg p \vee r)$$

Kapitel II – Grundlagen; Logik

- DPLL (Davis-Putnam-Logemann-Loveland).
  - Korrektheit des DPLL-Verfahrens.
    1. Das Verfahren terminiert für jede Eingabeformel.
       Folgt daraus, dass $F[p\backslash\mathbf{true}]$ und $F[p\backslash\mathbf{false}]$ mindestens eine Variable weniger als F enthalten. Wenn das Verfahren nicht früher terminiert, dann kommen wir bei Formeln mit 0 Variablen an. Die einzigen solchen Formeln sind **true** und **false**
    2. Wenn die Formel erfüllbar ist, dann antwortet das Verfahren „erfüllbar". Folgt aus Fakt.
    3. Wenn die Formel unerfüllbar ist, dann antwortet das Verfahren „unerfüllbar". Folgt aus Fakt.
  - Ausführlicher Beweis später in der Vorlesung.

Kapitel II – Grundlagen; Logik



- DPLL (Davis-Putnam-Logemann-Loveland).
  - Algorithmus 1 kann durch eine geschickte Wahl der Variablen *p* beschleunigt werden.
  - Eine One-Literal-Klausel ist eine Klausel, die nur ein Literal enthält, i.e., eine Klausel der Gestalt $\{p\}$ oder der Gestalt $\{\neg p\}$.

Kapitel II – Grundlagen; Logik

- Mengendarstellung von KNF-Formeln
  - Eine Klausel wird durch die Menge ihrer Literale dargestellt
    - Die Menge $\{p, \neg q, s\}$ stellt die Klausel $(p \vee \neg q \vee s)$ dar.
    - Die leere Menge von stellt die Formel **false** dar.
  - Eine KNF-Formel wird durch die Menge ihrer Klauseln (genauer: die Menge der Darstellungen ihrer Klauseln)
    - Die Klauselmenge $\{\{\neg p\}, \{p, \neg q, s\}\}$ stellt die Formel $\neg p \wedge (p \vee \neg q \vee s)$ dar.
    - Die leere Klauselmenge stellt die Formel **true** dar.

Kapitel II – Grundlagen; Logik

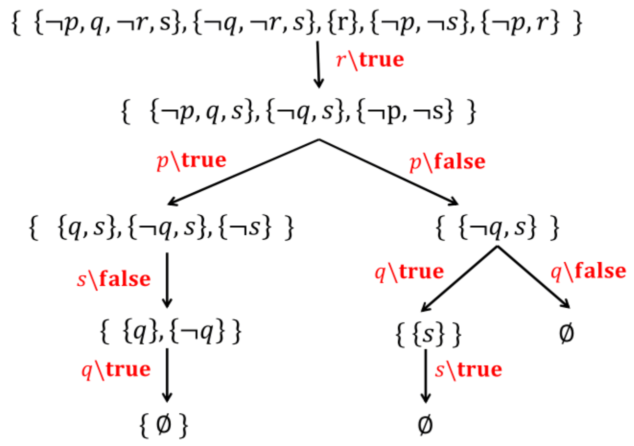
- DPLL mit Mengendarstellung

  Algorithmen 1 und 2 werden aufgrund der folgenden Eigenschaft mit Hilfe der Mengendarstellung implementiert:

  Sei F eine KNF-Formel und sei p eine Variable von F. Die Mengendarstellung von $F[p\backslash\mathbf{true}]$ ensteht, in dem:

  - alle Klauseln von F, die das Literal p enthalten, gestrichen werden, und
  - in allen Klauseln die das Literal $\neg p$ enthalten, dieses gestrichen wird.

  Die Mengendarstellung von $F[p\backslash\mathbf{false}]$ erhält man analog.



- Resolution.
  - Grundidee: füge iterativ neue Klauseln zur Formel hinzu, die logische Konsequenzen der ursprunglichen Klauseln sind.
  - Beispiel: wenn die Formel Klauseln $(\neg p \vee q \vee r)$ und $(\neg r \vee s \vee t)$ enthält, dann kann die Klausel $(\neg p \vee q \vee s \vee t)$ hinzugefügt werden.

    Die Klauseln sind äquivalent zu

    $$(p \wedge \neg q) \rightarrow r \qquad r \rightarrow (s \vee t) \qquad (p \wedge \neg q) \rightarrow (s \vee t)$$