

# Formale Systeme

Prof. Dr. Bernhard Beckert, WS 2016/2017

## Das Erfüllbarkeitsproblem

KIT – INSTITUT FÜR THEORETISCHE INFORMATIK

# Das SAT Problem

oder Erfüllbarkeitsproblem

## SAT

Instanz: Eine aussagenlogische Formel  $F \in \text{For}_0$

Frage: Ist  $F$  erfüllbar?

Gibt es eine Interpretation  $I$  mit  $\text{val}_I(F) = \mathbf{W}$ ?

# Das SAT Problem

oder Erfüllbarkeitsproblem

## SAT

Instanz: Eine aussagenlogische Formel  $F \in \text{For0}$

Frage: Ist  $F$  erfüllbar?

Gibt es eine Interpretation  $I$  mit  $\text{val}_I(F) = \mathbf{W}$ ?

SAT ist ein *NP-vollständiges* Problem:

# Das SAT Problem

oder Erfüllbarkeitsproblem

## SAT

Instanz: Eine aussagenlogische Formel  $F \in \text{For}_0$

Frage: Ist  $F$  erfüllbar?

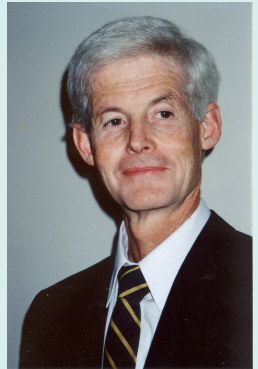
Gibt es eine Interpretation  $I$  mit  $\text{val}_I(F) = \mathbf{W}$ ?

SAT ist ein *NP-vollständiges* Problem:

Gäbe es einen (deterministischen) polynomiellen Entscheidungsalgorithmus für die Erfüllbarkeit, dann wäre  $NP = P$ , d.h. jedes nichtdeterministisch-polynomielle Entscheidungsproblem auch deterministisch-polynomiell.

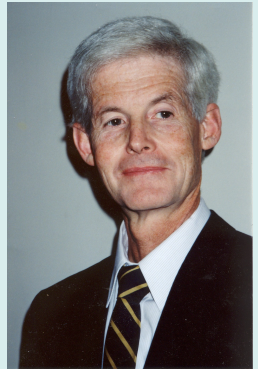
Stephen A. Cook ★ 1939

- Informatik-Professor an der Universität Toronto



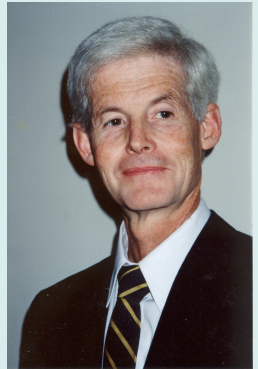
Stephen A. Cook ★ 1939

- ▶ Informatik-Professor an der Universität Toronto
- ▶ 1971: „Das Erfüllbarkeitsproblem der Aussagenlogik (SAT) ist NP-vollständig“



Stephen A. Cook ★ 1939

- ▶ Informatik-Professor an der Universität Toronto
- ▶ 1971: „Das Erfüllbarkeitsproblem der Aussagenlogik (SAT) ist NP-vollständig“
- ▶ **Turing-Preisträger**



## Das Erfüllbarkeitsproblem für Formeln

- ▶ in KNF ist NP-vollständig



## Das Erfüllbarkeitsproblem für Formeln

- ▶ in KNF ist NP-vollständig
- ▶ in 3-KNF ist NP-vollständig

## Das Erfüllbarkeitsproblem für Formeln

- ▶ in KNF ist NP-vollständig
- ▶ in 3-KNF ist NP-vollständig
- ▶ in 2-KNF ist polynomiell entscheidbar

## Das Erfüllbarkeitsproblem für Formeln

- ▶ in KNF ist NP-vollständig
- ▶ in 3-KNF ist NP-vollständig
- ▶ in 2-KNF ist polynomiell entscheidbar
- ▶ in DNF ist polynomiell entscheidbar ( $O(n \log n)$ )

## Das Erfüllbarkeitsproblem für Formeln

- ▶ in KNF ist NP-vollständig
- ▶ in 3-KNF ist NP-vollständig
- ▶ in 2-KNF ist polynomiell entscheidbar
- ▶ in DNF ist polynomiell entscheidbar ( $O(n \log n)$ )
- ▶ für Horn-Formeln ist polynomiell entscheidbar ( $O(n^2)$ )

## Das Erfüllbarkeitsproblem für Formeln

- ▶ in KNF ist NP-vollständig
- ▶ in 3-KNF ist NP-vollständig
- ▶ in 2-KNF ist polynomiell entscheidbar
- ▶ in DNF ist polynomiell entscheidbar ( $O(n \log n)$ )
- ▶ für Horn-Formeln ist polynomiell entscheidbar ( $O(n^2)$ )

## Das Erfüllbarkeitsproblem für Formeln

- ▶ in KNF ist NP-vollständig
- ▶ in 3-KNF ist NP-vollständig
- ▶ in 2-KNF ist polynomiell entscheidbar
- ▶ in DNF ist polynomiell entscheidbar ( $O(n \log n)$ )
- ▶ für Horn-Formeln ist polynomiell entscheidbar ( $O(n^2)$ )

## Bemerkungen:

- ▶  $k$ -KNF-Formeln sind Konjunktionen von Disjunktionen mit höchstens  $k$  Literalen.

## Das Erfüllbarkeitsproblem für Formeln

- ▶ in KNF ist NP-vollständig
- ▶ in 3-KNF ist NP-vollständig
- ▶ in 2-KNF ist polynomiell entscheidbar
- ▶ in DNF ist polynomiell entscheidbar ( $O(n \log n)$ )
- ▶ für Horn-Formeln ist polynomiell entscheidbar ( $O(n^2)$ )

## Bemerkungen:

- ▶  $k$ -KNF-Formeln sind Konjunktionen von Disjunktionen mit höchstens  $k$  Literalen.
- ▶ 2-KNF-Formeln heißen auch Krom-Formeln.

# Horn-Formeln

Wichtigste Teilklasse mit nicht NP-vollständigem  
Erfüllbarkeitsproblem



# Horn-Formeln

Wichtigste Teilklasse mit nicht NP-vollständigem  
Erfüllbarkeitsproblem

## Definition

Eine aussagenlogische Formel  $A$  ist eine *Horn-Formel*, wenn

- ▶  $A$  in KNF ist,
- ▶ jede Disjunktion in  $A$  höchstens ein positives Literal enthält

# Horn-Formeln

Wichtigste Teilklasse mit nicht NP-vollständigem Erfüllbarkeitsproblem

## Definition

Eine aussagenlogische Formel  $A$  ist eine *Horn-Formel*, wenn

- ▶  $A$  in KNF ist,
- ▶ jede Disjunktion in  $A$  höchstens ein positives Literal enthält

Alternative Schreibweise:

|                                            |                                                      |
|--------------------------------------------|------------------------------------------------------|
| $\neg B_1 \vee \dots \vee \neg B_m \vee A$ | $B_1 \wedge \dots \wedge B_m \rightarrow A$          |
| $\neg B_1 \vee \dots \vee \neg B_m$        | $B_1 \wedge \dots \wedge B_m \rightarrow \mathbf{0}$ |
| $A$                                        | $A$                                                  |
| leere Disjunktion                          | $\square$                                            |

# Horn-Formeln

Wichtigste Teilklasse mit nicht NP-vollständigem Erfüllbarkeitsproblem

## Definition

Eine aussagenlogische Formel  $A$  ist eine *Horn-Formel*, wenn

- ▶  $A$  in KNF ist,
- ▶ jede Disjunktion in  $A$  höchstens ein positives Literal enthält

Alternative Schreibweise:

|                                            |                                                      |
|--------------------------------------------|------------------------------------------------------|
| $\neg B_1 \vee \dots \vee \neg B_m \vee A$ | $B_1 \wedge \dots \wedge B_m \rightarrow A$          |
| $\neg B_1 \vee \dots \vee \neg B_m$        | $B_1 \wedge \dots \wedge B_m \rightarrow \mathbf{0}$ |
| $A$                                        | $A$                                                  |
| leere Disjunktion                          | $\square$                                            |

Bezeichnungen:

$B_1 \wedge \dots \wedge B_m$ : „Rumpf“       $A$ : „Kopf“ (bei leerem Rumpf: „Fakt“)

# Beispiel einer Horn-Formel

$$\begin{aligned} & \neg P \\ \wedge & (Q \vee \neg R \vee \neg S) \\ \wedge & (\neg Q \vee \neg S) \\ \wedge & R \\ \wedge & S \\ \wedge & (\neg Q \vee P) \end{aligned}$$

Alternative Schreibweise

$$\begin{aligned} & (P \rightarrow 0) \\ \wedge & (R \wedge S \rightarrow Q) \\ \wedge & (Q \wedge S \rightarrow 0) \\ \wedge & R \\ \wedge & S \\ \wedge & (Q \rightarrow P) \end{aligned}$$

# Erfüllbarkeitsproblem für Horn-Formeln

## Theorem

*Für Horn-Formeln ist die Erfüllbarkeit in quadratischer Zeit entscheidbar.*

Sei  $C = D_1 \wedge \dots \wedge D_m$  eine Hornformel.

Ein Atom in  $C$  *markieren*, bedeutet, es an allen Stellen seines Auftretens in  $C$  zu markieren.

# Erfüllbarkeitstest

Sei  $C = D_1 \wedge \dots \wedge D_m$  eine Hornformel.

Ein Atom in  $C$  *markieren*, bedeutet, es an allen Stellen seines Auftretens in  $C$  zu markieren.

- 0: Falls keine Fakten existieren: Ausgabe „erfüllbar“. STOP.  
Andernfalls: Markiere alle Fakten.

# Erfüllbarkeitstest

Sei  $C = D_1 \wedge \dots \wedge D_m$  eine Hornformel.

Ein Atom in  $C$  *markieren*, bedeutet, es an allen Stellen seines Auftretens in  $C$  zu markieren.

- 0: Falls keine Fakten existieren: Ausgabe „erfüllbar“. STOP.  
Andernfalls: Markiere alle Fakten.
- 1: Falls kein  $B_1 \wedge \dots \wedge B_m \rightarrow K$  existiert, so dass alle  $B_i$  im Rumpf markiert sind: Ausgabe „erfüllbar“. STOP.



# Erfüllbarkeitstest

Sei  $C = D_1 \wedge \dots \wedge D_m$  eine Hornformel.

Ein Atom in  $C$  *markieren*, bedeutet, es an allen Stellen seines Auftretens in  $C$  zu markieren.

0: Falls keine Fakten existieren: Ausgabe „erfüllbar“. STOP.  
Andernfalls: Markiere alle Fakten.

1: Falls kein  $B_1 \wedge \dots \wedge B_m \rightarrow K$  existiert, so dass alle  $B_i$  im Rumpf markiert sind: Ausgabe „erfüllbar“. STOP.

Falls ein  $B_1 \wedge \dots \wedge B_m \rightarrow \mathbf{0}$  existiert mit Kopf  $\mathbf{0}$ , so dass alle Atome  $B_i$  im Rumpf markiert sind:  
Ausgabe „unerfüllbar“. STOP.

# Erfüllbarkeitstest

Sei  $C = D_1 \wedge \dots \wedge D_m$  eine Hornformel.

Ein Atom in  $C$  *markieren*, bedeutet, es an allen Stellen seines Auftretens in  $C$  zu markieren.

0: Falls keine Fakten existieren: Ausgabe „erfüllbar“. STOP.  
Andernfalls: Markiere alle Fakten.

1: Falls kein  $B_1 \wedge \dots \wedge B_m \rightarrow K$  existiert, so dass alle  $B_i$  im Rumpf markiert sind: Ausgabe „erfüllbar“. STOP.

Falls ein  $B_1 \wedge \dots \wedge B_m \rightarrow \mathbf{0}$  existiert mit Kopf  $\mathbf{0}$ , so dass alle Atome  $B_i$  im Rumpf markiert sind:  
Ausgabe „unerfüllbar“. STOP.

Falls ein  $B_1 \wedge \dots \wedge B_m \rightarrow A$  existiert, so dass alle Atome  $B_i$  im Rumpf markiert sind aber der Kopf  $A$  nicht:  
Markiere  $A$ . Gehe zu 1.

# Erfüllbarkeitstest

Sei  $C = D_1 \wedge \dots \wedge D_m$  eine Hornformel.

Ein Atom in  $C$  *markieren*, bedeutet, es an allen Stellen seines Auftretens in  $C$  zu markieren.

0: Falls keine Fakten existieren: Ausgabe „erfüllbar“. STOP.  
Andernfalls: Markiere alle Fakten.

1: Falls kein  $B_1 \wedge \dots \wedge B_m \rightarrow K$  existiert, so dass alle  $B_i$  im Rumpf markiert sind: Ausgabe „erfüllbar“. STOP.

Falls ein  $B_1 \wedge \dots \wedge B_m \rightarrow \mathbf{0}$  existiert mit Kopf  $\mathbf{0}$ , so dass alle Atome  $B_i$  im Rumpf markiert sind:  
Ausgabe „unerfüllbar“. STOP.

Falls ein  $B_1 \wedge \dots \wedge B_m \rightarrow A$  existiert, so dass alle Atome  $B_i$  im Rumpf markiert sind aber der Kopf  $A$  nicht:  
Markiere  $A$ . Gehe zu 1.

Andernfalls: Ausgabe „erfüllbar“. STOP.

## Beispiel

$\textcircled{p}$

$\textcircled{q}$

$\textcircled{q} \wedge \textcircled{r} \rightarrow \textcircled{s}$

$\textcircled{p} \rightarrow \textcircled{r}$

$\textcircled{s} \rightarrow \textcircled{0}$

## Beispiel

**p**

**q**

$$(q) \wedge (r) \rightarrow (s)$$

$$(p) \rightarrow (r)$$

$$(s) \rightarrow (0)$$

## Beispiel

**p**

**q**

**q**  $\wedge$  **r**  $\rightarrow$  **s**

**p**  $\rightarrow$  **r**

**s**  $\rightarrow$  **0**

## Beispiel

$p$

$q$

$q \wedge r \rightarrow s$

$p \rightarrow r$

$s \rightarrow 0$

## Beispiel

$p$

$q$

$q \wedge r \rightarrow s$

$p \rightarrow r$

$s \rightarrow 0$



## Beispiel

$p$

$q$

$q \wedge r \rightarrow s$

$p \rightarrow r$

$s \rightarrow 0$

## Beispiel

$p$

$q$

$q \wedge r \rightarrow s$

$p \rightarrow r$

$s \rightarrow 0$

## Beispiel

$p$

$q$

$q \wedge r \rightarrow s$

$p \rightarrow r$

$s \rightarrow 0$

Formelmenge nicht erfüllbar

# Davis-Putnam-Logemann-Loveland-Verfahren

## DPLL-Verfahren

Wichtigstes Verfahren zur Entscheidung des allgemeinen Erfüllbarkeitsproblems (SAT-Problem)

## DPLL-Verfahren

Wichtigstes Verfahren zur Entscheidung des allgemeinen Erfüllbarkeitsproblems (SAT-Problem)

Das zur Zeit schnellste und fast ausschließlich benutzte Verfahren

## DPLL-Verfahren

Wichtigstes Verfahren zur Entscheidung des allgemeinen Erfüllbarkeitsproblems (SAT-Problem)

Das zur Zeit schnellste und fast ausschließlich benutzte Verfahren

Eingabe: Formel in KNF

In Disjunktionen und Konjunktionen  
kommt es nicht auf

- ▶ die Reihenfolge der Teilformeln an

In Disjunktionen und Konjunktionen  
kommt es nicht auf

- ▶ die Reihenfolge der Teilformeln an
- ▶ die Multiplizität identischer Teilformeln an



In Disjunktionen und Konjunktionen  
kommt es nicht auf

- ▶ die Reihenfolge der Teilformeln an
- ▶ die Multiplizität identischer Teilformeln an

In Disjunktionen und Konjunktionen  
kommt es nicht auf

- ▶ die Reihenfolge der Teilformeln an
- ▶ die Multiplizität identischer Teilformeln an

## Mengenschreibweise

In Disjunktionen und Konjunktionen  
kommt es nicht auf

- ▶ die Reihenfolge der Teilformeln an
- ▶ die Multiplizität identischer Teilformeln an

## Mengenschreibweise

Disjunktion als Menge: „Klausel“

KNF-Formel als Menge: „Klauselmenge“

In Disjunktionen und Konjunktionen kommt es nicht auf

- ▶ die Reihenfolge der Teilformeln an
- ▶ die Multiplizität identischer Teilformeln an

## Mengenschreibweise

Disjunktion als Menge: „Klausel“

KNF-Formel als Menge: „Klauselmenge“

Schreibweise:

- ☐ für die leere Klausel

# Exkurs: Klauselschreibweise

In Disjunktionen und Konjunktionen kommt es nicht auf

- ▶ die Reihenfolge der Teilformeln an
- ▶ die Multiplizität identischer Teilformeln an

## Mengenschreibweise

Disjunktion als Menge: „Klausel“

KNF-Formel als Menge: „Klauselmenge“

Schreibweise:

- $\square$  für die leere Klausel
- $\emptyset$  für die leere Klauselmenge

## Semantik

/ Interpretation,  $S$  Menge von Klauseln,  $C$  Klausel.

## Semantik

/ Interpretation,  $S$  Menge von Klauseln,  $C$  Klausel.

$$1. \text{val}_I(S) = \begin{cases} \mathbf{W} & \text{falls für alle } C \in S \text{ gilt: } \text{val}_I(C) = \mathbf{W} \\ \mathbf{F} & \text{sonst} \end{cases}$$

## Semantik

$I$  Interpretation,  $S$  Menge von Klauseln,  $C$  Klausel.

$$1. \text{val}_I(S) = \begin{cases} \mathbf{W} & \text{falls f\"ur alle } C \in S \text{ gilt: } \text{val}_I(C) = \mathbf{W} \\ \mathbf{F} & \text{sonst} \end{cases}$$

$$2. \text{val}_I(C) = \begin{cases} \mathbf{W} & \text{falls ein } L \in C \text{ existiert mit } \text{val}_I(L) = \mathbf{W} \\ \mathbf{F} & \text{sonst} \end{cases}$$



## Semantik

$I$  Interpretation,  $S$  Menge von Klauseln,  $C$  Klausel.

1.  $val_I(S) = \begin{cases} \mathbf{W} & \text{falls für alle } C \in S \text{ gilt: } val_I(C) = \mathbf{W} \\ \mathbf{F} & \text{sonst} \end{cases}$
2.  $val_I(C) = \begin{cases} \mathbf{W} & \text{falls ein } L \in C \text{ existiert mit } val_I(L) = \mathbf{W} \\ \mathbf{F} & \text{sonst} \end{cases}$
3.  $I(\emptyset) = \mathbf{W}$ .

## Semantik

$I$  Interpretation,  $S$  Menge von Klauseln,  $C$  Klausel.

1.  $val_I(S) = \begin{cases} \mathbf{W} & \text{falls für alle } C \in S \text{ gilt: } val_I(C) = \mathbf{W} \\ \mathbf{F} & \text{sonst} \end{cases}$
2.  $val_I(C) = \begin{cases} \mathbf{W} & \text{falls ein } L \in C \text{ existiert mit } val_I(L) = \mathbf{W} \\ \mathbf{F} & \text{sonst} \end{cases}$
3.  $I(\emptyset) = \mathbf{W}.$
4.  $I(\square) = \mathbf{F}.$

procedure DPLL(Klauselmenge  $S$ )

- 1 **if**  $S = \emptyset$  **then return** 1;
- 2 **if**  $\square \in S$  **then return** 0;
- 3 **if**  $S$  enthält Einerklausel
- 4     **then choose** Einerklausel  $\{L\} \in S$ ;
- 5         **return** DPLL( $\text{red}_{\{L\}}(S)$ );
- 6     **else choose**  $P \in \text{atom}(S)$
- 7         **return**  $\max\{\text{DPLL}(S_P), \text{DPLL}(S_{\neg P})\}$ ;

procedure DPLL(Klauselmenge  $S$ )

```
1  if  $S = \emptyset$  then return 1;  
2  if  $\square \in S$  then return 0;  
3  if  $S$  enthält Einerklausel  
4    then choose Einerklausel  $\{L\} \in S$ ;  
5    return DPLL( $\text{red}_{\{L\}}(S)$ );  
6  else choose  $P \in \text{atom}(S)$   
7    return  $\max\{\text{DPLL}(S_P), \text{DPLL}(S_{\neg P})\}$ ;
```

- $\text{atom}(S) = \bigcup_{C \in S} \text{atom}(C)$ ,  
wobei  $\text{atom}(C) = \{P \in \Sigma \mid P \in C \text{ oder } \neg P \in C\}$

procedure DPLL(Klauselmenge  $S$ )

```
1  if  $S = \emptyset$  then return 1;  
2  if  $\square \in S$  then return 0;  
3  if  $S$  enthält Einerklausel  
4    then choose Einerklausel  $\{L\} \in S$ ;  
5    return DPLL( $\text{red}_{\{L\}}(S)$ );  
6  else choose  $P \in \text{atom}(S)$   
7    return  $\max\{\text{DPLL}(S_P), \text{DPLL}(S_{\neg P})\}$ ;
```

►  $\text{atom}(S) = \bigcup_{C \in S} \text{atom}(C)$ ,  
wobei  $\text{atom}(C) = \{P \in \Sigma \mid P \in C \text{ oder } \neg P \in C\}$

►  $\text{red}_{\{L\}}(S) = \{\text{red}_{\{L\}}(C) \mid C \in S \text{ mit } L \notin C\}$

$\text{red}_{\{L\}}(C) = C \setminus \{\bar{L}\}$ , wobei  $\bar{A} = \neg A$  und  $\overline{\neg A} = A$  für  $A \in \Sigma$

procedure DPLL(Klauselmenge  $S$ )

```
1  if  $S = \emptyset$  then return 1;  
2  if  $\square \in S$  then return 0;  
3  if  $S$  enthält Einerklausel  
4    then choose Einerklausel  $\{L\} \in S$ ;  
5    return DPLL( $\text{red}_{\{L\}}(S)$ );  
6  else choose  $P \in \text{atom}(S)$   
7    return  $\max\{\text{DPLL}(S_P), \text{DPLL}(S_{\neg P})\}$ ;
```

- ▶  $\text{atom}(S) = \bigcup_{C \in S} \text{atom}(C)$ ,  
wobei  $\text{atom}(C) = \{P \in \Sigma \mid P \in C \text{ oder } \neg P \in C\}$
- ▶  $\text{red}_{\{L\}}(S) = \{\text{red}_{\{L\}}(C) \mid C \in S \text{ mit } L \notin C\}$   
 $\text{red}_{\{L\}}(C) = C \setminus \{\bar{L}\}$ , wobei  $\bar{A} = \neg A$  und  $\overline{\neg A} = A$  für  $A \in \Sigma$
- ▶  $S_P = S \cup \{\{P\}\}$  und  $S_{\neg P} = S \cup \{\{\neg P\}\}$ .

Wir beginnen mit der Klauselmenge  $S$

$$\begin{array}{ll} P_1 \vee P_2 \vee P_3 & \neg P_1 \vee P_2 \vee \neg P_4 \\ \neg P_1 \vee P_3 & \neg P_1 \vee \neg P_3 \vee P_4 \\ P_1 \vee \neg P_3 & \neg P_2 \end{array}$$

Wir beginnen mit der Klauselmenge  $S$

$$\begin{array}{ll} P_1 \vee P_2 \vee P_3 & \neg P_1 \vee P_2 \vee \neg P_4 \\ \neg P_1 \vee P_3 & \neg P_1 \vee \neg P_3 \vee P_4 \\ P_1 \vee \neg P_3 & \neg P_2 \end{array}$$

Beim ersten Aufruf von  $\text{DPLL}(S)$  wird das Unterprogramm  $\text{red}_{\{\neg P_2\}}(S)$  aufgerufen.



# DPLL Beispiel

Wir beginnen mit der Klauselmenge  $S$

$$\begin{array}{ll}
 P_1 \vee P_2 \vee P_3 & \neg P_1 \vee P_2 \vee \neg P_4 \\
 \neg P_1 \vee P_3 & \neg P_1 \vee \neg P_3 \vee P_4 \\
 P_1 \vee \neg P_3 & \neg P_2
 \end{array}$$

Beim ersten Aufruf von DPLL( $S$ ) wird das Unterprogramm  $red_{\{\neg P_2\}}(S)$  aufgerufen und liefert  $S_1$ :

$$\begin{array}{ll}
 P_1 \vee P_3 & \neg P_1 \vee \neg P_4 \\
 \neg P_1 \vee P_3 & \neg P_1 \vee \neg P_3 \vee P_4 \\
 P_1 \vee \neg P_3 &
 \end{array}$$

|            |   |                              |
|------------|---|------------------------------|
| rot        | = | ganze Klausel entfernt       |
| dunkelgrün | = | Literal aus Klausel entfernt |
| blau       | = | unverändert                  |

$$\begin{array}{ll} P_1 \vee P_3 & \neg P_1 \vee \neg P_4 \\ \neg P_1 \vee P_3 & \neg P_1 \vee \neg P_3 \vee P_4 \\ P_1 \vee \neg P_3 & \end{array}$$

$S_1$  enthält keine Einerklausel.

$$\begin{array}{l} P_1 \vee P_3 \quad \neg P_1 \vee \neg P_4 \\ \neg P_1 \vee P_3 \quad \neg P_1 \vee \neg P_3 \vee P_4 \\ P_1 \vee \neg P_3 \end{array}$$

$S_1$  enthält keine Einerklausel. Die Variable  $P_1$  wird gewählt und  $\text{DPLL}(S_{1,0})$  und  $\text{DPLL}(S_{1,1})$  werden aufgerufen.

$S_{1,0}$  :

$$P_1 \vee P_3$$

$$\neg P_1 \vee \neg P_4$$

$$\neg P_1 \vee P_3$$

$$\neg P_1 \vee \neg P_3 \vee P_4$$

$$P_1 \vee \neg P_3$$

$$P_1$$

$S_{1,1}$  :

$$P_1 \vee P_3$$

$$\neg P_1 \vee \neg P_4$$

$$\neg P_1 \vee P_3$$

$$\neg P_1 \vee \neg P_3 \vee P_4$$

$$P_1 \vee \neg P_3$$

$$\neg P_1$$

$$\begin{aligned} S_{1,0} : \\ & P_1 \vee P_3 \\ & \neg P_1 \vee \neg P_4 \\ & \neg P_1 \vee P_3 \\ & \neg P_1 \vee \neg P_3 \vee P_4 \\ & P_1 \vee \neg P_3 \\ & P_1 \end{aligned}$$

$$\text{red}_{\{P_1\}}(S_{1,0})$$

$S_{1,0} :$

$P_1 \vee P_3$

$\neg P_1 \vee \neg P_4$

$\neg P_1 \vee P_3$

$\neg P_1 \vee \neg P_3 \vee P_4$

$P_1 \vee \neg P_3$

$P_1$

$red_{\{P_1\}}(S_{1,0})$ : Die **Klauseln** werden gestrichen.  
Das **Literal** wird entfernt.

$$\begin{aligned} red_{\{P_1\}}(S_{1,0}) &= S_{2,0} \\ &= \{\neg P_4, P_3, \neg P_3 \vee P_4\} \end{aligned}$$

$$S_{2,0} = \{\neg P_4, P_3, \neg P_3 \vee P_4\}$$

Der Aufruf von  $red_{\{P_3\}}(S_{2,0})$  liefert

$$\{\neg P_4, P_4\}$$

$$S_{2,0} = \{\neg P_4, P_3, \neg P_3 \vee P_4\}$$

Der Aufruf von  $red_{\{P_3\}}(S_{2,0})$  liefert

$$\{\neg P_4, P_4\}$$

Dann:

$$red_{\{P_4\}}(\{P_4, \neg P_4\}) = \{\square\}$$

woraus die Unerfüllbarkeit von  $S_{1,0}$  folgt.

$S_{1,0}$  :

$P_1 \vee P_3$

$\neg P_1 \vee \neg P_4$

$\neg P_1 \vee P_3$

$\neg P_1 \vee \neg P_3 \vee P_4$

$P_1 \vee \neg P_3$

$P_1$

$S_{1,1}$  :

$P_1 \vee P_3$

$\neg P_1 \vee \neg P_4$

$\neg P_1 \vee P_3$

$\neg P_1 \vee \neg P_3 \vee P_4$

$P_1 \vee \neg P_3$

$\neg P_1$

Jetzt kommt die Abarbeitung von  $\text{DPLL}(S_{1,1})$  an die Reihe.



$$\begin{aligned} S_{1,1} : \\ P_1 \vee P_3 \\ \neg P_1 \vee \neg P_4 \\ \neg P_1 \vee P_3 \\ \neg P_1 \vee \neg P_3 \vee P_4 \\ P_1 \vee \neg P_3 \\ \neg P_1 \end{aligned}$$

$red_{\{\neg P_1\}}(S_{1,1})$  entfernt die Klauseln, in denen  $\neg P_1$  vorkommt ...

$$\begin{aligned} S_{1,1} : \\ P_1 \vee P_3 \\ P_1 \vee \neg P_3 \end{aligned}$$

... und streicht in den restlichen  $P_1$ .

$$\begin{aligned} S_{1,1} : \\ P_1 \vee P_3 \\ P_1 \vee \neg P_3 \end{aligned}$$

... und streicht in den restlichen  $P_1$ .  
Das liefert

$$\{P_3, \neg P_3\}$$

$$\begin{aligned} S_{1,1} : \\ P_1 \vee P_3 \\ P_1 \vee \neg P_3 \end{aligned}$$

... und streicht in den restlichen  $P_1$ .  
Das liefert

$$\{P_3, \neg P_3\}$$

woraus im nächsten Schritt

$$\{\square\}$$

entsteht,

$$\begin{aligned} S_{1,1} : \\ P_1 \vee P_3 \\ P_1 \vee \neg P_3 \end{aligned}$$

... und streicht in den restlichen  $P_1$ .  
Das liefert

$$\{P_3, \neg P_3\}$$

woraus im nächsten Schritt

$$\{\square\}$$

entsteht,  
woraus die Unerfüllbarkeit von  $S_{1,1}$  und damit insgesamt die  
Unerfüllbarkeit von  $S$  folgt.

## Theorem

## Theorem

1. Der DPLL Algorithmus terminiert für jede Eingabe.

## Theorem

1. Der DPLL Algorithmus terminiert für jede Eingabe.
2. Der DPLL Algorithmus ist korrekt und vollständig.



## Theorem

1. Der DPLL Algorithmus terminiert für jede Eingabe.
2. Der DPLL Algorithmus ist korrekt und vollständig.

## Theorem

1. Der DPLL Algorithmus terminiert für jede Eingabe.
2. Der DPLL Algorithmus ist korrekt und vollständig.
  - 2.1 aus  $DPLL(S) = 1$  folgt, daß  $S$  erfüllbar ist und

## Theorem

1. Der DPLL Algorithmus terminiert für jede Eingabe.
2. Der DPLL Algorithmus ist korrekt und vollständig.
  - 2.1 aus  $DPLL(S) = 1$  folgt, daß  $S$  erfüllbar ist und
  - 2.2 ist  $S$  erfüllbar, dann gilt  $DPLL(S) = 1$ .

## Theorem

1. Der DPLL Algorithmus terminiert für jede Eingabe.
2. Der DPLL Algorithmus ist korrekt und vollständig.
  - 2.1 aus  $DPLL(S) = 1$  folgt, daß  $S$  erfüllbar ist und
  - 2.2 ist  $S$  erfüllbar, dann gilt  $DPLL(S) = 1$ .

## Theorem

1. Der DPLL Algorithmus terminiert für jede Eingabe.
2. Der DPLL Algorithmus ist korrekt und vollständig.
  - 2.1 aus  $\text{DPLL}(S) = 1$  folgt, daß  $S$  erfüllbar ist und
  - 2.2 ist  $S$  erfüllbar, dann gilt  $\text{DPLL}(S) = 1$ .

### Beweisidee:

Terminierung:

- Bei jeder Reduktion fällt ein Atom weg

## Theorem

1. Der DPLL Algorithmus terminiert für jede Eingabe.
2. Der DPLL Algorithmus ist korrekt und vollständig.
  - 2.1 aus  $\text{DPLL}(S) = 1$  folgt, daß  $S$  erfüllbar ist und
  - 2.2 ist  $S$  erfüllbar, dann gilt  $\text{DPLL}(S) = 1$ .

### Beweisidee:

Terminierung:

- Bei jeder Reduktion fällt ein Atom weg

## Theorem

1. Der DPLL Algorithmus terminiert für jede Eingabe.
2. Der DPLL Algorithmus ist korrekt und vollständig.
  - 2.1 aus  $\text{DPLL}(S) = 1$  folgt, daß  $S$  erfüllbar ist und
  - 2.2 ist  $S$  erfüllbar, dann gilt  $\text{DPLL}(S) = 1$ .

### Beweisidee:

Terminierung:

- ▶ Bei jeder Reduktion fällt ein Atom weg

Korrektheit/Vollständigkeit

- ▶ Jeder Schritt führt zu einer erfüllbarkeitsäquivalenten Klauselmenge

## Theorem

1. Der DPLL Algorithmus terminiert für jede Eingabe.
2. Der DPLL Algorithmus ist korrekt und vollständig.
  - 2.1 aus  $DPLL(S) = 1$  folgt, daß  $S$  erfüllbar ist und
  - 2.2 ist  $S$  erfüllbar, dann gilt  $DPLL(S) = 1$ .

### Beweisidee:

Terminierung:

- ▶ Bei jeder Reduktion fällt ein Atom weg

Korrektheit/Vollständigkeit

- ▶ Jeder Schritt führt zu einer erfüllbarkeitsäquivalenten Klauselmenge
- ▶  $\emptyset$  ist erfüllbar



## Theorem

1. Der DPLL Algorithmus terminiert für jede Eingabe.
2. Der DPLL Algorithmus ist korrekt und vollständig.
  - 2.1 aus  $DPLL(S) = 1$  folgt, daß  $S$  erfüllbar ist und
  - 2.2 ist  $S$  erfüllbar, dann gilt  $DPLL(S) = 1$ .

## Beweisidee:

Terminierung:

- ▶ Bei jeder Reduktion fällt ein Atom weg

Korrektheit/Vollständigkeit

- ▶ Jeder Schritt führt zu einer erfüllbarkeitsäquivalenten Klauselmenge
- ▶  $\emptyset$  ist erfüllbar
- ▶  $\{\square\}$  ist unerfüllbar