

Berechenbarkeits- und Komplexitätstheorie

Lerneinheit 4: Die Klasse P

Prof. Dr. Christoph Karg

Studiengang Informatik
Hochschule Aalen



Wintersemester 2015/2016



2.12.2015

Einleitung

Diese Lerneinheit beschäftigt sich mit **effizient lösbar**en Problemen.

Sie gliedert sich in:

- Beispiele für Sprachen in P
 - ▷ Erreichbarkeit in gerichteten Graphen
 - ▷ Kontextfreie Sprachen
 - ▷ Circuit Value Problem
 - ▷ Erfüllbarkeit von Formeln in 2-KNF
- Polynomialzeit many-one Reduktionen

Definition der Klasse P

Definition. Die Klasse P enthält alle in polynomialer Zeit lösbarer Entscheidungsprobleme, formal:

$$P = \bigcup_{k=1}^{\infty} \text{DTIME}(n^k)$$

Bemerkungen:

- P enthält alle Probleme, die aus realistischen Gesichtspunkten in vertretbarem Aufwand lösbar sind
- P ist robust, d.h., die Wahl eines deterministischen Berechnungsmodells beeinflusst nicht die Zugehörigkeit einer Sprache zu P

Erreichbarkeit in gerichteten Graphen

Erreichbarkeit in gerichteten Graphen (PATH)

Gegeben:

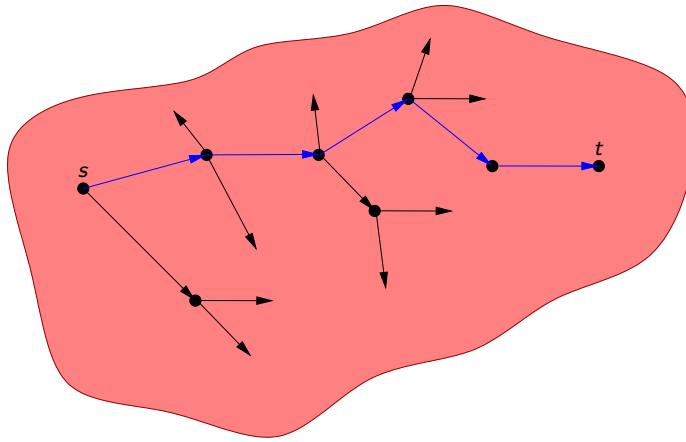
- Gerichteter Graph $G = (V, E)$
- Knoten $s, t \in V$

Gefragt: Ist t von s aus erreichbar? Oder anders gefragt: gibt es in G einen Pfad von s nach t ?

Satz. PATH $\in P$

Erreichbarkeit in gerichteten Graphen (Forts.)

Idee zum Beweis:



Beobachtung: Wenn t von s aus erreichbar ist, dann gibt es einen Pfad von s nach t der Länge $\leq n$

Erreichbarkeit in gerichteten Graphen (Forts.)

Beweis. Betrachte den folgenden nichtdeterministischen Algorithmus

PATH(G, s, t)

Input: Graph $G = (V, E)$, Knoten $s, t \in V$

Output: true, falls es in G einen Pfad von s nach t gibt
false, sonst.

- 1 $k := 1; v := s; n := \|V\|$
- 2 **while** ($k \leq n - 1$) **and** ($v \neq t$) **do**
- 3 Wähle nichtdeterministisch einen Knoten $v \in \text{Adj}[v]$
- 4 $k := k + 1$
- 5 **if** ($v = t$) **then**
- 6 **return** true
- 7 **else**
- 8 **return** false

Erreichbarkeit in gerichteten Graphen (Forts.)

Korrektheit: ✓ (wegen der obigen Beobachtung)

Speicherplatzbedarf: Angenommen, die Knoten sind durchnummiert, d.h., $V = \{1, \dots, n\}$

Konsequenz:

- Zur Speicherung von v werden $\log_2 n$ viele Bits benötigt
- Wenn k binär kodiert wird, dann liegt der Platzbedarf ebenfalls bei $\log_2 n$ Bits

Platzkomplexität $O(\log_2 n)$

$\rightsquigarrow \text{Path} \in \text{NL} \subseteq \text{P}$

Kontextfreie Sprachen

Satz. Für jede kontextfreie Sprache L gilt: $L \in \text{P}$

Beweis. Sei L eine beliebige kontextfreie Sprache.

Automatentheorie \rightsquigarrow es gibt eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ in Chomsky Normalform mit $L(G) = L$

Beachte: Die Grammatik G ist fest gewählt und nicht Teil der Eingabe

Ansatz: Lösung des Wortproblems unter Einsatz der Grammatik G

Programmiertechnik: **Dynamisches Programmieren**

\rightsquigarrow Algorithmus von Cocke, Younger und Kasami

Kontextfreie Sprachen (Forts.)

$\text{CYK}_G(x)$

Input: Wort $x = a_1 \dots a_n \in \Sigma^*$

Output: true, falls $x \in L(G)$, false, sonst.

```
1 if ( $x = \varepsilon$ ) and ( $S \rightarrow \varepsilon \in P$ ) then
2   return true
3 for  $i := 1$  to  $n$  do
4   for jede Variable  $A \in V$  do
5     if ( $A \rightarrow a_i \in P$ ) then
6        $T[1, i] := T[1, i] \cup \{[A \rightarrow a_i, 0]\}$ 
```

Kontextfreie Sprachen (Forts.)

```
7 for  $\ell := 2$  to  $n$  do
8   for  $i := 1$  to  $n - \ell + 1$  do
9      $j := i + \ell - 1$ 
10    for  $k := i$  to  $j - 1$  do
11      for jede Regel  $A \rightarrow BC \in P$  do
12        if ( $B \in T[i, k]$ ) and ( $C \in T[k, j]$ ) then
13           $T[i, j] := T[i, j] \cup \{[A \rightarrow BC, k]\}$ 
14 if [ $S \rightarrow w, *$ ]  $\in T[1, n]$  then
15   return true
16 else
17   return false
```

Kontextfreie Sprachen (Forts.)

Korrektheit: ✓ (Beweis \rightsquigarrow Automatentheorie)

Laufzeit:

- Da die Grammatik G fest gewählt ist, ist die Laufzeit der Schleifen in Zeile 4–6 und Zeile 11–13 konstant
- Der wesentliche Aufwand entsteht durch die drei verschachtelten Schleifen in Zeile 7–13. Jede der Schleifen wird höchstens n -mal durchlaufen

Gesamt: Laufzeit $O(n^3)$

Somit: $L \in P$

Beispiel: CYK-Algorithmus

Beispiel. Betrachte die folgende Grammatik G Chomsky Normalform:

$$\begin{array}{l} S \rightarrow AB \\ S \rightarrow BC \\ A \rightarrow BA \\ A \rightarrow a \\ B \rightarrow CC \\ B \rightarrow b \\ C \rightarrow AB \\ C \rightarrow a \end{array}$$

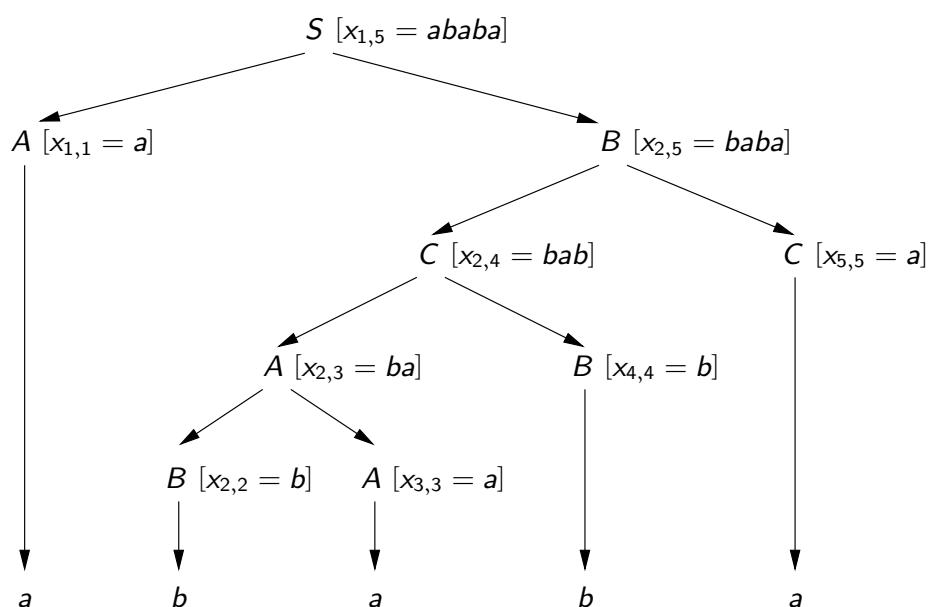
Beispiel: CYK-Algorithmus (Forts.)

Betrachte die Eingabe $x = ababa$:

i	1	2	3	4	5
	a	b	a	b	a
$j = 1$	$[A \rightarrow a, 0]$ $[C \rightarrow a, 0]$	$[B \rightarrow b, 0]$	$[A \rightarrow a, 0]$ $[C \rightarrow a, 0]$	$[B \rightarrow b, 0]$	$[A \rightarrow a, 0]$ $[C \rightarrow a, 0]$
$j = 2$	$[S \rightarrow AB, 1]$ $[C \rightarrow AB, 1]$	$[S \rightarrow BC, 2]$ $[A \rightarrow BA, 2]$	$[S \rightarrow AB, 3]$ $[C \rightarrow AB, 3]$	$[S \rightarrow BC, 4]$ $[A \rightarrow BA, 4]$	
$j = 3$	$[B \rightarrow CC, 2]$	$[S \rightarrow BC, 2]$ $[S \rightarrow AB, 3]$ $[C \rightarrow AB, 3]$	$[B \rightarrow CC, 4]$		
$j = 4$	$[B \rightarrow CC, 1]$ $[B \rightarrow CC, 2]$	$[B \rightarrow CC, 4]$			
$j = 5$	$[S \rightarrow AB, 1]$ $[C \rightarrow AB, 1]$ $[A \rightarrow BA, 3]$ $[A \rightarrow BA, 4]$ $[S \rightarrow BC, 4]$				

Beispiel: CYK-Algorithmus (Forts.)

Ein Ableitungsbaum für $x = ababa$ ist:



Circuit Value Problem

Circuit Value Problem (CVP)

Gegeben:

- Aussagenlogische Formel $F(x_1, \dots, x_n)$
- Belegung $B = (b_1, \dots, b_n)$

Gefragt: Ist B eine erfüllende Belegung für F ?

Satz. CVP $\in P$

Beweis. **Idee:** rekursive Auswertung der Formel

Circuit Value Problem (Forts.)

EVALUATE(F, B)

Input: Formel F , Belegung B

Output: true, falls B eine erfüllende Belegung für F ist,
false, sonst.

- 1 **while** $F = (G)$ **do** $F := G$
- 2 **if** $F = x_i$ **then**
3 **return** $B[x_i]$
- 4 *Suche in F den bindungsschwächsten Operator*
- 5 **if** $F = \neg G$ **then**
6 **return** **not** EVALUATE(G, B)
- 7 **elseif** $F = G \wedge H$ **then**
8 **return** EVALUATE(G, B) **and** EVALUATE(H, B)
- 9 **elseif** $F = G \vee H$ **then**
10 **return** EVALUATE(G, B) **or** EVALUATE(H, B)

Beispiel Circuit Value Problem (Forts.)

Formel: $x_1 \wedge \neg x_2 \vee (x_3 \wedge x_1)$

Belegung: $x_1 = 1, x_2 = 0, x_3 = 0$

$$x_1 \quad \wedge \quad \neg x_2 \quad \vee \quad (\quad x_3 \quad \wedge \quad x_1 \quad)$$

Beispiel Circuit Value Problem (Forts.)

Formel: $x_1 \wedge \neg x_2 \vee (x_3 \wedge x_1)$

Belegung: $x_1 = 1, x_2 = 0, x_3 = 0$

The diagram shows a logical expression with a circled OR operator (\vee) and a blue arrow pointing to a simplified expression below.

Original expression:

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

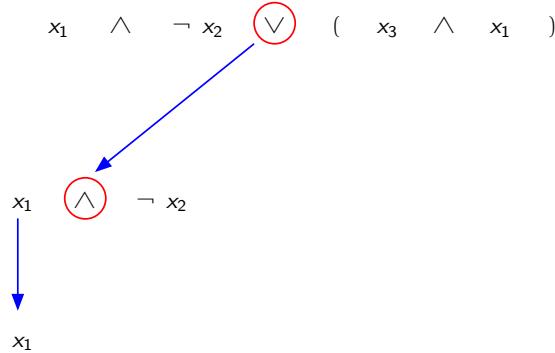
Simplified expression:

$$x_1 \wedge \neg x_2$$

Beispiel Circuit Value Problem (Forts.)

Formel: $x_1 \wedge \neg x_2 \vee (x_3 \wedge x_1)$

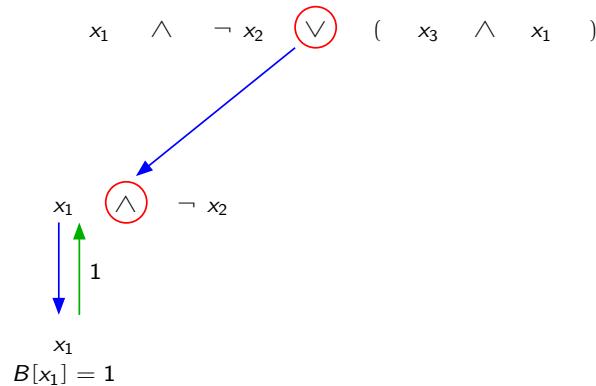
Belegung: $x_1 = 1, x_2 = 0, x_3 = 0$



Beispiel Circuit Value Problem (Forts.)

Formel: $x_1 \wedge \neg x_2 \vee (x_3 \wedge x_1)$

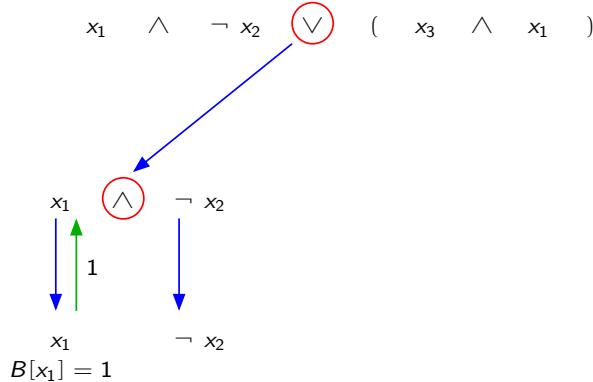
Belegung: $x_1 = 1, x_2 = 0, x_3 = 0$



Beispiel Circuit Value Problem (Forts.)

Formel: $x_1 \wedge \neg x_2 \vee (x_3 \wedge x_1)$

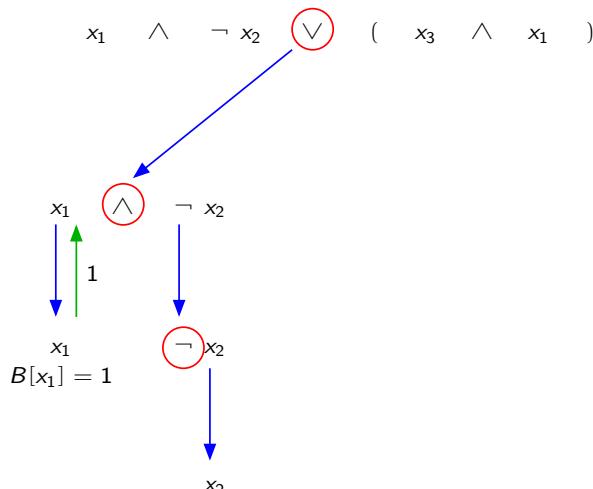
Belegung: $x_1 = 1, x_2 = 0, x_3 = 0$



Beispiel Circuit Value Problem (Forts.)

Formel: $x_1 \wedge \neg x_2 \vee (x_3 \wedge x_1)$

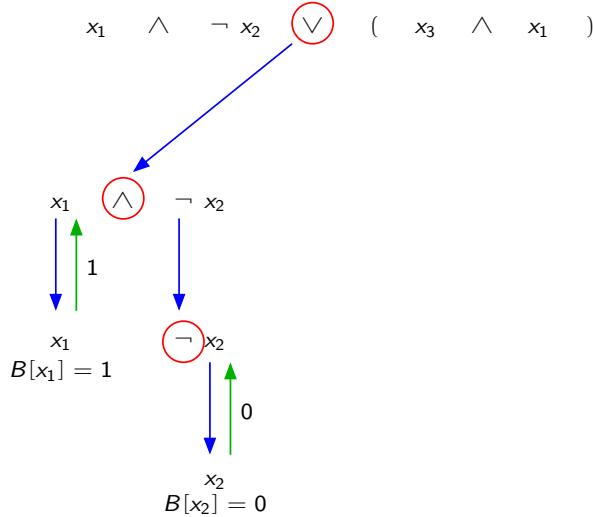
Belegung: $x_1 = 1, x_2 = 0, x_3 = 0$



Beispiel Circuit Value Problem (Forts.)

Formel: $x_1 \wedge \neg x_2 \vee (x_3 \wedge x_1)$

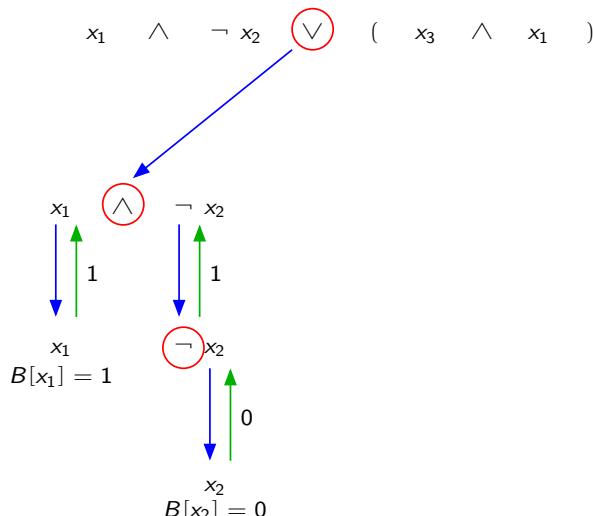
Belegung: $x_1 = 1, x_2 = 0, x_3 = 0$



Beispiel Circuit Value Problem (Forts.)

Formel: $x_1 \wedge \neg x_2 \vee (x_3 \wedge x_1)$

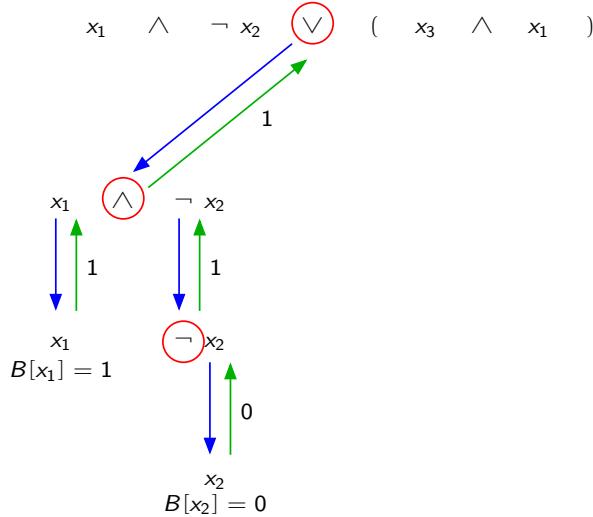
Belegung: $x_1 = 1, x_2 = 0, x_3 = 0$



Beispiel Circuit Value Problem (Forts.)

Formel: $x_1 \wedge \neg x_2 \vee (x_3 \wedge x_1)$

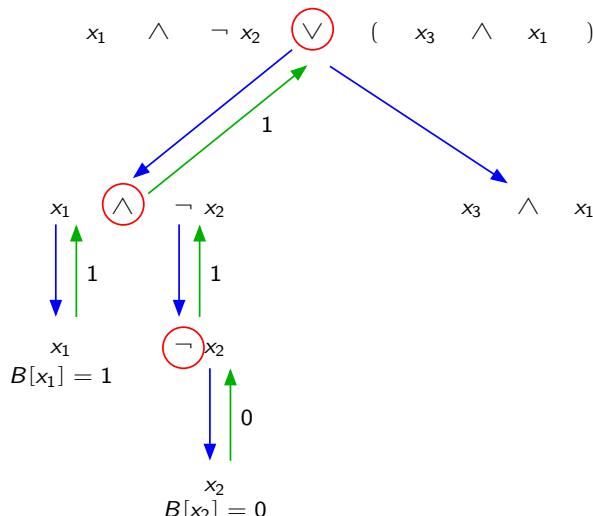
Belegung: $x_1 = 1, x_2 = 0, x_3 = 0$



Beispiel Circuit Value Problem (Forts.)

Formel: $x_1 \wedge \neg x_2 \vee (x_3 \wedge x_1)$

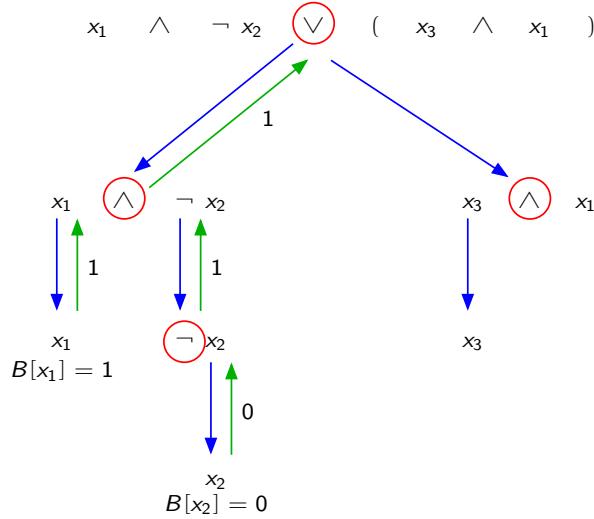
Belegung: $x_1 = 1, x_2 = 0, x_3 = 0$



Beispiel Circuit Value Problem (Forts.)

Formel: $x_1 \wedge \neg x_2 \vee (x_3 \wedge x_1)$

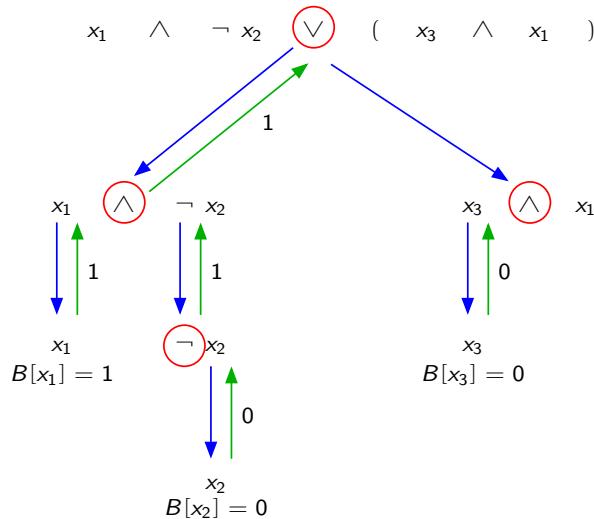
Belegung: $x_1 = 1, x_2 = 0, x_3 = 0$



Beispiel Circuit Value Problem (Forts.)

Formel: $x_1 \wedge \neg x_2 \vee (x_3 \wedge x_1)$

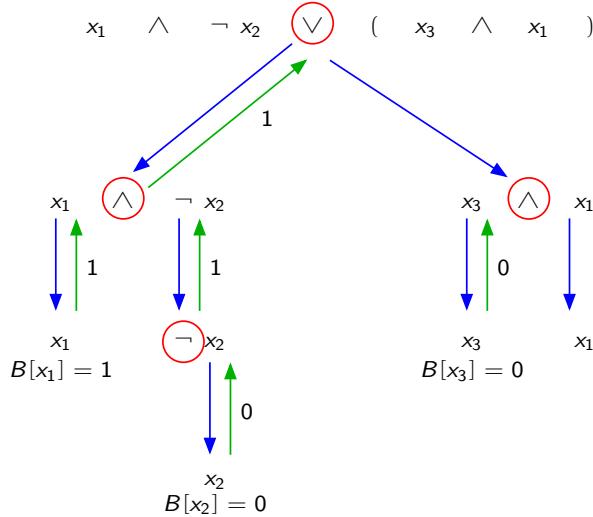
Belegung: $x_1 = 1, x_2 = 0, x_3 = 0$



Beispiel Circuit Value Problem (Forts.)

Formel: $x_1 \wedge \neg x_2 \vee (x_3 \wedge x_1)$

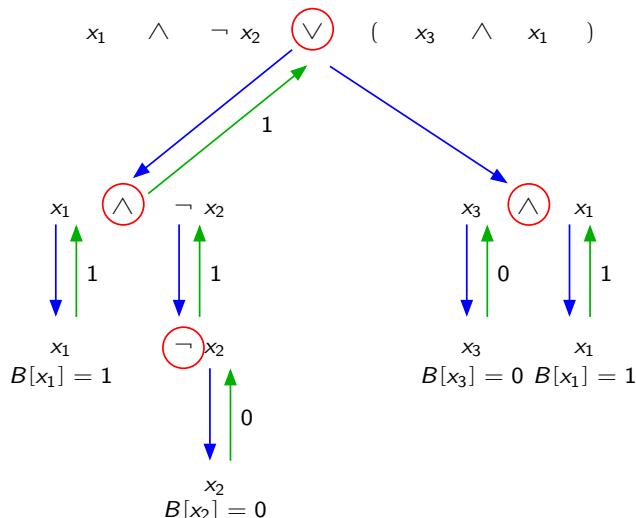
Belegung: $x_1 = 1, x_2 = 0, x_3 = 0$



Beispiel Circuit Value Problem (Forts.)

Formel: $x_1 \wedge \neg x_2 \vee (x_3 \wedge x_1)$

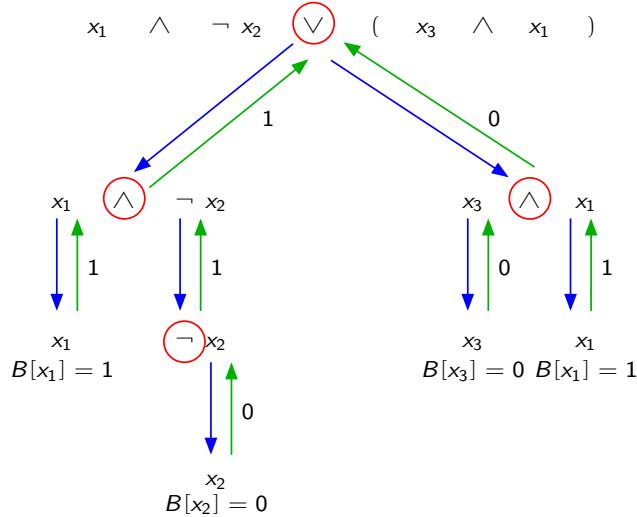
Belegung: $x_1 = 1, x_2 = 0, x_3 = 0$



Beispiel Circuit Value Problem (Forts.)

Formel: $x_1 \wedge \neg x_2 \vee (x_3 \wedge x_1)$

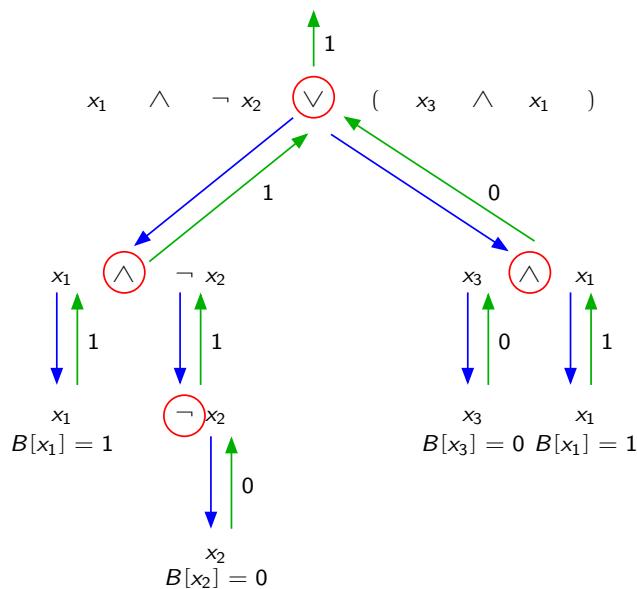
Belegung: $x_1 = 1, x_2 = 0, x_3 = 0$



Beispiel Circuit Value Problem (Forts.)

Formel: $x_1 \wedge \neg x_2 \vee (x_3 \wedge x_1)$

Belegung: $x_1 = 1, x_2 = 0, x_3 = 0$



Circuit Value Problem (Forts.)

Korrektheit: ✓

Laufzeit:

- Die Suche nach dem bindungsschwächsten Operator in einer Formel der Länge n ist in $O(n)$ Rechenschritten möglich
- Die Zerlegung von F in Teilformeln ist in $O(n)$ Rechenschritten durchführbar
- Bei einer Formel der Länge n gibt es insgesamt höchstens n rekursive Aufrufe

Gesamt: Laufzeit von $O(n^2)$

Erfüllbarkeit von Formeln in 2-KNF

Erfüllbarkeit von aussagenlogischen Formeln in 2-KNF (2-SAT)

Gegeben: Aussagenlogische Formel F in 2-KNF

Gefragt: Ist F erfüllbar?

Satz. 2-SAT $\in P$

Beweis. Betrachte den Algorithmus TwoSAT(F):

Erfüllbarkeit von Formeln in 2-KNF (Forts.)

$\text{TwoSAT}(F)$

Input: Aussagenlogische Formel F in 2-KNF

Output: true, falls F erfüllbar ist, false, sonst.

```
1  $V :=$  Menge der Variablen in  $F$ 
2  $C :=$  Menge der Klauseln in  $F$ 
3 for jede Variable  $x \in V$  do
4    $color(x) :=$  white
5 for jede Klausel  $K \in C$  do
6    $color(K) :=$  white
7 while  $V \neq \emptyset$  do
8   Wähle  $x \in V$ 
9    $value(x) := 1; color(x) :=$  black
```

Erfüllbarkeit von Formeln in 2-KNF (Forts.)

```
10  $first :=$  true
11 while  $C$  enthält eine Klausel  $K = (\ell_1 \vee \ell_2)$  mit
     $color(K) =$  white und mindestens einem
    belegtem Literal do
12   if  $(\ell_1 = 1)$  or  $(\ell_2 = 1)$  then
13      $color(K) :=$  black
14   else if  $(\ell_1 = 0)$  and  $(\ell_2 = 0)$  then
15     if  $(first =$  true) then
16       for jede Variable  $x' \in V$  do
17          $color(x') :=$  white
18       for jede Klausel  $K' \in C$  do
19          $color(K') :=$  white
```

Erfüllbarkeit von Formeln in 2-KNF (Forts.)

```
20      value( $x$ ) := 0; color( $x$ ) := black
21      first := false
22      else
23          return false
24      else if ( $\ell_1 = 0$ ) then
25          value( $\ell_2$ ) := 1; color( $\ell_2$ ) := black
26      else
27          value( $\ell_1$ ) := 1; color( $\ell_1$ ) := black
28      for jede Variable  $x' \in V$  do
29          if (color( $x'$ ) = black) then  $V := V - \{x'\}$ 
30      for jede Klausel  $K' \in C$  do
31          if (color( $K'$ ) = black) then  $C := C - \{K'\}$ 
32      return true
```

Erfüllbarkeit von Formeln in 2-KNF (Forts.)

Bemerkungen:

- Der Algorithmus belegt Schritt für Schritt eine Variable mit 1 und verarbeitet die Auswirkungen
- Wird ein Widerspruch festgestellt, dann wird die Variable mit 0 belegt und der Vorgang wiederholt
- Kann die Variable nicht widerspruchsfrei belegt werden, dann ist die Formel nicht erfüllbar
- Die Laufzeit von TwoSAT(F) bei einer Formel mit n Variablen und m Klauseln ist $O(n \cdot m)$

Beispiel zu TWO-SAT(F)

Beispiel. $F = (\neg x_2 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_4 \vee x_5)$

first	x_1	x_2	x_3	x_4	x_5	$\neg x_2 \vee x_3$	$x_1 \vee x_2$	$\neg x_2 \vee \neg x_3$	$x_4 \vee x_5$
true	1	✓	.	.
true	(1)	(✓)	.	.
true	(1)	1	.	.	.	$\rightsquigarrow x_3 = 1$	(✓)	.	.
true	(1)	1	1	.	.	✓	(✓)	✗	.
false	(1)	0	.	.	.	✓	(✓)	✓	.
false	(1)	(0)	.	.	.	(✓)	(✓)	(✓)	.
true	(1)	(0)	1	.	.	(✓)	(✓)	(✓)	.
true	(1)	(0)	(1)	1	.	(✓)	(✓)	(✓)	✓
true	(1)	(0)	(1)	(1)	.	(✓)	(✓)	(✓)	(✓)
true	(1)	(0)	(1)	(1)	1	(✓)	(✓)	(✓)	(✓)
true	(1)	(0)	(1)	(1)	(1)	(✓)	(✓)	(✓)	(✓)

Ergebnis: F ist erfüllbar durch die Belegung $x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1, x_5 = 1$

Weitere Erfüllbarkeitsprobleme

Erfüllbarkeit von Formeln in 3-KNF (3-SAT)

Gegeben: Aussagenlogische Formel F in 3-KNF

Gefragt: Ist F erfüllbar?

Erfüllbarkeit von Formeln in KNF (KNF-SAT)

Gegeben: Aussagenlogische Formel F in KNF

Gefragt: Ist F erfüllbar?

Offen: Weder für 3-SAT noch für KNF-SAT ist ein effizienter Algorithmus bekannt

Polynomialzeit many-one Reduktionen

Definition. Seien A und B Sprachen über dem Alphabet Σ .

A ist in **Polynomialzeit many-one reduzierbar** auf B , symbolisch $A \leq_m^p B$, falls es eine Funktion $f : \Sigma^* \mapsto \Sigma^*$ gibt mit folgenden Eigenschaften:

- Für alle $x \in \Sigma^*$ gilt: $x \in A \iff f(x) \in B$
- f ist mittels einer Turing Maschine berechenbar, die auf allen Eingaben stoppt und für jede Eingabe x den Wert $f(x)$ zurückliefert
- Die Laufzeit der Turing Maschine bei Eingaben der Länge n ist $O(n^k)$ für eine Konstante $k > 0$.

Wichtige Eigenschaft der \leq_m^p Reduktion

Satz. Falls $A \leq_m^p B$ und $B \in P$, dann ist $A \in P$.

Beweis. Sei $A \leq_m^p B$ mittels der Reduktionsfunktion f , die in $O(n^k)$ Schritten berechenbar ist. Sei M eine Turing Maschine mit $L(M) = B$ und einer Zeitkomplexität von $O(n^\ell)$

Betrachte die Turing Maschine M' , die auf Eingabe x zuerst $y = f(x)$ berechnet und dann M auf Eingabe y ausführt.

Es gilt: $x \in A \iff M'$ akzeptiert x

Laufzeit für Eingaben der Länge n :

$$O(n^k) + O(O(n^k)^\ell) = O(n^{k+\ell})$$

Somit: $A \in P$

Anwendung der \leq_m^p Reduktion

Klar: 3-SAT \leq_m^p KNF-SAT

Satz. KNF-SAT \leq_m^p 3-SAT

Beweis. Sei F eine beliebige aussagenlogische Formel in KNF

Ansatz: Bearbeite jede Klausel K in F separat

Fall 1: $K = \ell_1$. Ersetze K durch die Klausel $(\ell_1 \vee \ell_1 \vee \ell_1)$

Fall 2: $K = (\ell_1 \vee \ell_2)$. Ersetze K durch die Klausel
 $(\ell_1 \vee \ell_2 \vee \ell_2)$

Fall 3: $K = (\ell_1 \vee \ell_2 \vee \ell_3)$. ✓

Anwendung der \leq_m^p Reduktion (Forts.)

Fall 4: $K = (\ell_1 \vee \ell_2 \vee \dots \vee \ell_k)$, wobei $k \geq 4$. Ersetze K durch die Klausel

$$(\ell_1 \vee \ell_2 \vee z_1) \wedge (\neg z_1 \vee \ell_3 \vee z_2) \\ \wedge (\neg z_2 \vee \ell_4 \vee z_3) \wedge \dots \wedge (\neg z_{k-3} \vee \ell_{k-1} \vee \ell_k)$$

wobei z_1, z_2, \dots, z_{k-3} neue Variablen sind

Es gilt:

- F ist erfüllbar genau dann, wenn die modifizierte Formel F' erfüllbar ist
- Die Berechnung der modifizierten Formel F' ist in Polynomialzeit durchführbar

Beispiel zur Reduktion KNF-SAT \leq_m^p 3-SAT

Beispiel. Gegeben ist die Formel

$$F = (x_1 \vee \neg x_2 \vee \neg x_3 \vee x_5 \vee \neg x_4) \wedge (x_1 \vee x_3)$$

Die Reduktion liefert die Formel

$$\begin{aligned} F' = & (x_1 \vee \neg x_2 \vee z_1) \wedge (\neg z_1 \vee \neg x_3 \vee z_2) \\ & \wedge (\neg z_2 \vee x_5 \vee \neg x_4) \wedge (x_1 \vee x_3 \vee x_3) \end{aligned}$$

Zusammenfassung

- P steht für die Klasse der effizient lösbarer Entscheidungsprobleme
- Effizient lösbar ist äquivalent zu in Polynomialzeit entscheidbar
- Polynomialzeit Many-one Reduktionen dienen zur (effizienten) Transformation des Wortproblems von einer Sprache auf eine andere
- Es gibt Probleme, von denen nicht bekannt ist, ob sie effizient lösbar sind