Turingmaschinen

TECHNISCHE UNIVERSITÄT DARMSTADT

Eine Turingmaschine heißt Zähler, wenn sie, gestartet mit bin(p), $p \in \mathbb{N}$,

bin(p-1), bin(p-2),...bin(1),bin(0)

hintereinander, immer auf dem gleichen Bandbereich erzeugt
und dann stoppt. Sei n = | bin(p) | die Länge der Binärdarstellung von p.

Satz: Es gibt einen O(n) platz- und $O(2^n)$ zeitbeschränkten Zähler.

Beweis: gemeinsam in Übung

Turingmaschinen

Eingabe bin(p) = $x_{n-1}...x_0$

s = p;
while s <> 0 do
Berechne bin(s-1) aus bin(s) wie folgt:

- gehe zum rechten Rand der Eingabe
- solange eine 0 gelesen wird, überschreibe sie mit einer 1 und gehe nach links. Sobald eine 1 gelesen wird ersetze diese durch eine 0.
- gehe mit dem Kopf eine Stelle weiter nach links. Falls ein B gelesen wird, gehe nach rechts, ersetze die 0 durch ein B und gehe an den rechten Rand. Sonst gehe, ohne ein Zeichen zu ändern, an den rechten Rand.
- s = s-1;

Turingmaschinen

Beispiel: Eine 1 von 1011000 subtrahieren

B1011000B →	B1011001B
B1011011B ←	B1011111B
B1010111B	B1010111B
B1010111B	B1010111B
B1010111B	B1010111B
B1010111B	

- gehe zum rechten Rand der Eingabe
- solange eine 0 gelesen wird, überschreibe sie mit einer 1 und gehe nach links. Sobald eine 1 gelesen wird ersetze diese durch eine 0.

10 Schritte bzw, wenn a Nullen rechts stehen:
2a+4 Schritte

Turingmaschinen

Übergänge	Erläuterung
$\delta(q_0,0) = (q_0,B,R)$ $\delta(q_0,B) = (q_f,B,N)$ $\delta(q_0,1) = (q_1,1,R)$	ersetze führende Nullen duch Bs falls nur Nullen: fertig sonst:
$\delta(q_1,0) = (q_1,0,R)$ $\delta(q_1,1) = (q_1,1,R)$ $\delta(q_1,B) = (q_2,B,L)$	gehe an den rechten Rand der Eingabe, und dann vor das letzte B
$\delta(q_2,0) = (q_2,1,L)$ $\delta(q_2,1) = (q_3,0,L)$	solange eine 0 gelesen wird, wird 1 geschrieben wenn eine 1 gelesen wird, wird 0 geschrieben und ...
$\delta(q_3,B) = (q_0,B,R)$ $\delta(q_3,0) = (q_1,0,R)$ $\delta(q_3,1) = (q_1,1,R)$	falls dort ein B steht, prüfe, ob fertig. sonst gehe zum rechten Rand

Turingmaschinen

Beachte: mit $n = |\,bin(p)\,|$ gilt: $2^{n-1} \leq p < 2^n$, bzw. $n-1 \leq \log(p) < n$

$S_M(n) = n+2 = O(n)$

1x dekrementieren: falls rechts a Nullen stehen, $\leq 2a+4$ Schritte
-> p-mal dekrementieren bei Zeit höchstens 2n+4

Also: $T_M(n) = O(n\ 2^n)$

Frage: dauert das wirklich so lange?

Antwort: nein, denn die meisten Dekrements sind viel schneller.

Turingmaschinen

TECHNISCHE UNIVERSITÄT DARMSTADT

In 50% aller Fälle (beim runterzählen) steht eine 1 am Schluß. D.h. a=0
In 25% aller Fälle steht eine 10 am Schluß. D.h. a=1
In 12,5% aller Fälle steht eine 100 am Schluß. D.h. a=2
In 6,25% aller Fälle steht eine 1000 am Schluß. D.h. a=3
....

Im Durchschnitt sind das nicht mehr als $\sum_{a\geq 0} (2a+4)*2^{-a-1}$

$= \sum_{a\geq 0} 2a*2^{-a-1} + 4*2^{-a-1}$ viele Schritte.

$= \sum_{a\geq 0} a*2^{-a-1} + \sum_{a\geq 0} 2^{-a+1}$

$= \quad 2 \quad + \quad 4$

Also ein Dekrement in durchschnittlich 6 Schritten.
Laufzeit $O(2^n)$

Turingmaschinen

TECHNISCHE UNIVERSITÄT DARMSTADT

Def.:
Eine Sprache L heißt entscheidbar, wenn es eine Turingmaschine gibt, die zu jeder Eingabe $w \in \Sigma^*$ nach endlicher Zeit anhält, und genau dann in einem akzeptierenden Zustand endet, wenn $w \in L$ gilt.

Eine Sprache L heißt semi-entscheidbar, wenn es eine Turingmaschine gibt, die zu jeder Eingabe $w \in L$ nach endlicher Zeit in einem akzeptierenden Endzustand anhält.

Eine Funktion f heißt berechenbar, wenn es eine Turingmaschine gibt, die für alle Eingaben x, die aus dem Definitionsbereich von f stammen nach endlich vielen Schritten anhält und $f(x)$ auf das Band schreibt.

Unendscheidbarkeit

Gibt es unendscheidbare Sprachen?

Ja, denn es gibt nur abzählbar unendlich viele Turingmaschinen, aber überabzählbar viele Sprachen $L \subseteq \{0,1\}^*$

Begründung mit Hilfe des Cantor'schen Diagonalisierungsverfahrens:

	M_1	M_2	M_3	M_4	... →
0	n	j	j	n	
1	n	n	n	j	
01	j	j	j	n	
...					
x_i ↓					

Eintrag (M_i,x_k)=„j" bedeutet, dass x_k aus der Sprache $L(M_i)$ ist. Sei nun L die Sprache, die genau aus den Wörtern besteht, bei denen beim Eintrag (M_i,x_i) „n" steht. L gehört zu keiner der aufgeführten TMs.

Berechenbarkeit

Gibt es Funktionen, die nicht von einer Turingmaschine berechnet werden können?

Ja.

Die Busy-Beaver Funktion $\Sigma(n)$ ist definiert als die Anzahl der Einsen, die eine Champion-Turingmaschine auf ein zu Beginn leeres Band ausgibt, wobei n die Anzahl der erlaubten Zustände darstellt. Die TM muss irgendwann halten. Wir gehen weiterhin davon aus, dass diese Einsen alle zusammenhängend sein müssen.

Beweis:

Berechenbarkeit

TECHNISCHE UNIVERSITÄT DARMSTADT

Annahme: Die Busy-Beaver Funktion $\Sigma(n)$ ist berechnbar, und EVALΣ ist die TM, die $\Sigma(n)$ berechnet. Bei einer Eingabe von n Einsen schreibt sie $\Sigma(n)$ Einsen auf das Band und hält dann an.

Im folgenden definieren wir 4 Hilfs-TMs.

Sei INC eine TM, die bis zum ersten B nach rechts läuft, dort eine 1 schreibt und dann hält.

DOUBLE sein eine andere TM, die die Anzahl Einsen, die sich auf dem Band befinden verdoppelt. DOUBLE berechnet also zu Eingabe n n+n.

Wir bilden nun eine neue TM: DOUBLE | EVALΣ | INC
Die Anzahl der Zustände dieser Maschine sei n_0

Berechenbarkeit

Sei $CREATE_n_0$ eine weitere TM, welche n_0 Einsen auf ein leeres Band schreibt. Diese TM gibt es, trivialerweise eine mit n_0 vielen Zuständen.

Sei nun $N := n_0 + n_0$

Das Finale: Sei $BAD\Sigma$ folgende TM:

$$\underbrace{CREATE_n_0}_{n_0} \mid \underbrace{DOUBLE \mid EVAL\Sigma(N) \mid INC}_{n_0}$$

Diese Maschine hat N Zustände. Sie startet auf leerem Band, schreibt n_0 Einsen, verdoppelt diese, berechnet $\Sigma(N)$ und schreibt eine weitere 1.

$BAD\Sigma$ hat also eine 1 mehr als $\Sigma(N)$ geschrieben! Es folgt, dass die Annahme falsch war.

Busy Beaver

Interessanterweise sind einige Busy-Beaverwerte bekannt. Z.B. für TMs mit 2 Symbolen :

#Zustände	Anzahl Einsen des Siegers
1	1
2	4
3	6
4	13
5	>= 4098
6	>= 95.524.079

Schönheit in der Mathematik: (nach Prof. Hesse, Universität Stuttgart Fakultät für Mathematik und Physik, Dresden 2008)

- Präzision
- Klarheit
- Eleganz

Beispiel: Turnierproblem

Tennisturnier mit 128 Spielern nach K.O.-System.
Wie viele Begegnungen werden ausgetragen?

Turnierverlauf

1 2 3 4 5 6 7 8 …… 121 122 123 124 125 126 127 128

2 3 6 7 122 123 126 127

3 6 123 126

6 126

……

6 126

6

Gewinner

Turnierproblem: Lösung 1

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Runde: 128 Spieler, 64 Paare, **64** Begegnungen
2. Runde: 64 Spieler, 32 Paare, **32** Begegnungen
3. Runde: 32 Spieler, 16 Paare, **16** Begegnungen
4. Runde: 16 Spieler, 8 Paare, **8** Begegnungen
5. Runde: 8 Spieler, 4 Paare, **4** Begegnungen
6. Runde: 4 Spieler, 2 Paare, **2** Begegnungen
7. Runde: 2 Spieler, 1 Paar, **1** Begegnung

Insgesamt: **1** + **2** + **4** + **8** + **16** + **32** + **64** = **127** Begegnungen

Turnierproblem: Lösung 2

- Zahl der Spieler: Zweierpotenz, $128 = 2^7$
- Begegnungen je Runde: fortgesetzte Halbierung

Begegnungen gesamt

= Summe von Zweierpotenzen

$= 1 + 2 + 2^2 + ... + 2^6$

$= 2^7 - 1 = 127$

Allgemeiner

Für 2^n Spieler, 2^{n-1} Erstrunden-Begegnungen

2^{n-2} Zweitrunden-Begegnungen, etc.

$$1 + 2 + 2^2 + \ldots + 2^{n-1}$$

$$= (1 + 2 + 2^2 + \ldots + \ldots 2^{n-1})(2 - 1)$$

$$= 2 + 2^2 + \ldots + 2^{n-1} + 2^n - 1 - 2 - 2^2 - \ldots - 2^{n-1}$$

$$= 2^n - 1$$

Fortschritt: Tiefe, Verallgemeinerung, Verständnis

Turnierproblem: Lösung 3

a) Jede Begegnung hat einen Sieger und einen Verlierer.

b) Jeder Spieler spielt so lange, bis er verliert.

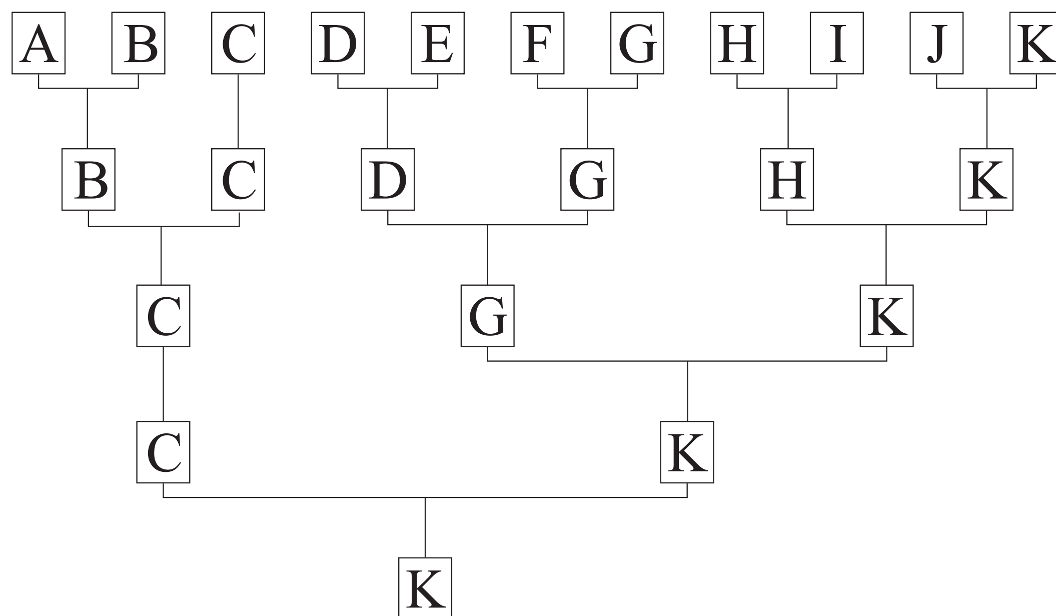
Also: es gibt genauso viele Begegnungen, wie es Verlierer gibt.

Jeder Spieler außer dem Champion ist ein Verlierer.

Also: Anzahl Begegnungen = Anzahl Verlierer
= Anzahl Spieler – 1.

Fortschritt: Vereinfachung, Tiefe, Verallgemeinerung , Ästhetik

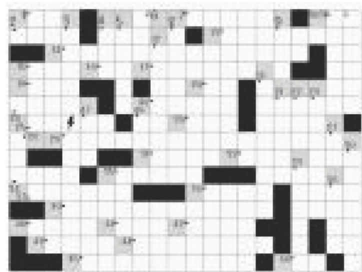
Tennisturnier mit 11-er Feld

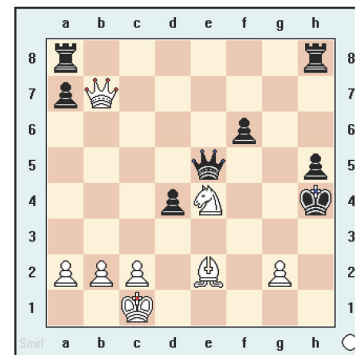


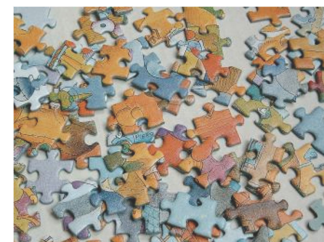
Probleme des täglichen Lebens

Im folgenden sind die Probleme lösbar. Die Frage ist nur in welcher Zeit und mit wieviel Speicherplatz.

- Was ist schwieriger?
  - Kopfrechnen
  - Kreuzworträtsel
  - Schach
  - Sokoban
  - Puzzle







Probleme und Problembeschreibungen, Wdh

TECHNISCHE UNIVERSITÄT DARMSTADT

- Eine Sprache $L \subseteq \Sigma^*$ muss nun irgendwie beschrieben werden.
  - z.B. durch einen *regulären Ausdruck*: (0*10*)
    - $\emptyset$ ist ein regulärer Ausdruck.
    - ε ist ein regulärer Ausdruck.
    - $\forall\, a_i \in \Sigma$ ist a_i ein regulärer Ausdruck.
    - Sind x und y reguläre Ausdrücke, so auch $x \cup y$, (xy) und x^*.
    - Es gibt keine weiteren regulären Ausdrücke.
  - z.B. durch eine **Problembeschreibung**:
    - **Definition**: Ein *Entscheidungsproblem* ist ein input-output Tupel mit

      geg.: Kodierung eines Inputs einer Instanz, mittels Alphabet $\sum$

      ges.: ja/nein
    - Die Teilmenge aller Inputs, für die die Antwort “ja” ist, ist offenbar eine Sprache

Ein Zeit-Komplexitätsmaß

- Definition: Komplexität eines Algorithmus
  - Sei A ein deterministischer (RAM-)Algorithmus, der auf allen Eingaben hält.
  - Die **Laufzeit (Zeitkomplexität)** von A ist eine Funktion f: $N \to N$,
    - wobei f(n) die maximale Anzahl von Schritten von A beschreibt **auf einer Eingabe der Länge n**.

    - Linear-Zeit-Algorithmus: $f(n) \leq c\, n$ für eine Konstante c
    - Polynom-Zeit-Algorithmus: $f(n) \leq c\, n^k$ für Konstanten c und k

- Definition: Komplexität eines Problems
  - Die Zeit- (Platz-) Komplexität eines Problems p ist die Laufzeit des schnellsten (am wenigsten Speicherplatz benötigenden) Algorithmus, der Problem p löst.
  - Ein Problem p ist "in Polynomzeit lösbar", wenn es Algorithmus A, Polynom Π und $n_0 \in \mathbb{N}$ gibt, so dass für alle $n > n_0$ **gilt : $f(n) <= \Pi(n)$**

P, NP, PSPACE

- **P:** Klasse aller Probleme, die von einer deterministischen RAM in Polynomzeit gelöst werden können

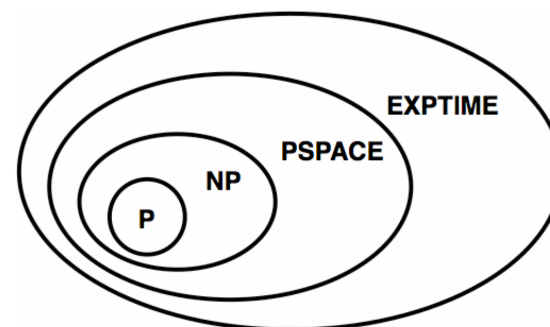
- **NP:** Klasse aller Probleme, die von einer nichtdeterministischen TM in Polynomzeit gelöst werden können.

- **PSPACE :** Klasse aller Probleme, die von einer deterministischen RAM mit polynomiell viel Platz gelöst werden können

- Man weiß nur, dass P ≠ EXPTIME und $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXPTIME}$

- EXPTIME = $\bigcup_k \mathbf{TIME}(2^{n^k})$

- Allgemein wird aber vermutet, dass alle Inklusionen echt sind, d.h.



Nichtdeterministische Turingmaschinen

- Eine nichtdeterministische Turingmaschine (NTM) ist definiert, wie eine deterministische Turingmaschine, nur dass δ eine Übergangsrelation und keine Funktion ist.

- $\delta: Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{R,N,L\}}$ ist die Übergangsrelation.

- Bsp.: Wenn die TM in Zustand q ist, und ein a liest, und $\delta(q,a) = \{(q',b,R), (q'', a, L)\}$ ist, dann ist die nichtdeterministische TM im nächsten Schritt entweder in Zustand q', nachdem sie ein b geschrieben hat, oder sie ist in Zustand q'' nachdem sie ein a schrieb.

- Die Laufzeit einer NTM ist definiert als die Länge des kürzesten Berechnungsweges, der in einem akzeptierenden Endzustand endet.

Nichtdeterministische Turingmaschinen und Verifizierer

Def. Es sei L eine Sprache. Ein Verifizierer für L ist ein deterministischer Algorithmus A, mit L = {w | es gibt ein c mit A akzeptiert wc}

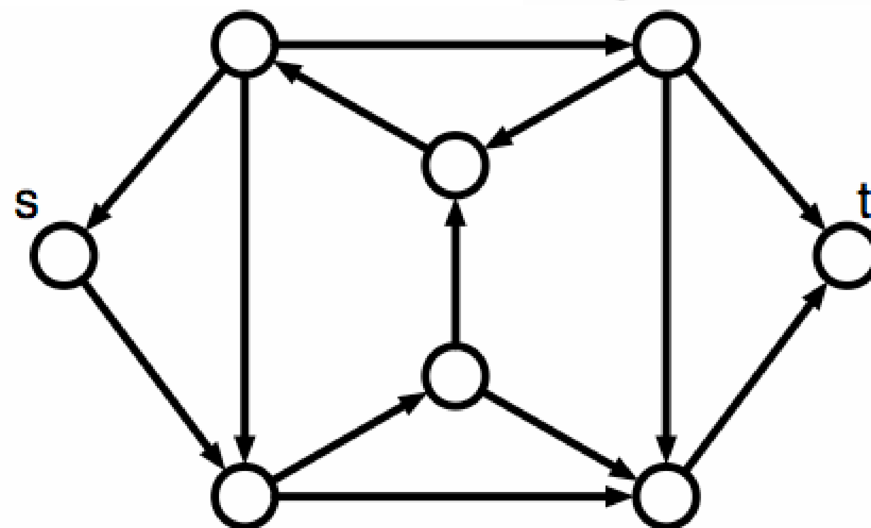
Der Zeitaufwand für einen Verifizierer wird abhängig von der Länge von w gemessen. L ist polynomiell prüfbar, wenn es einen Verifizierer mit polynomiellem Zeitaufwand gibt.

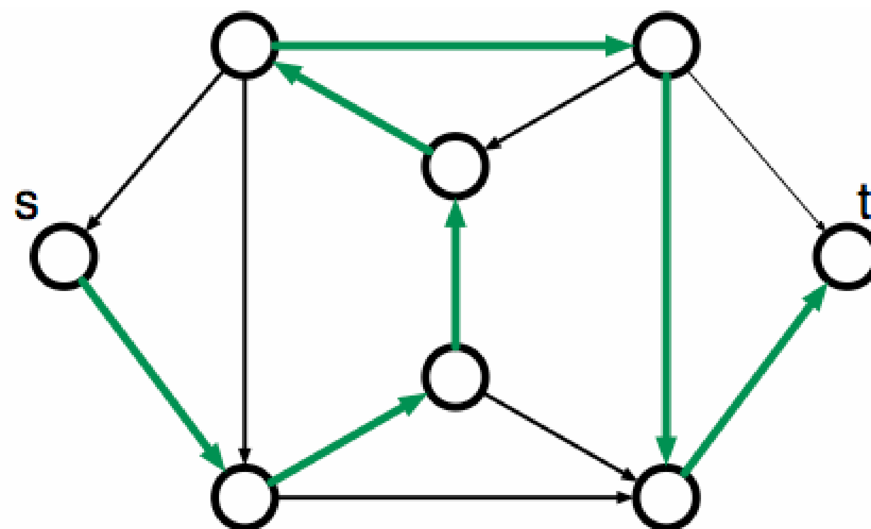
Satz: NP ist die Menge aller Probleme, für die es einen Verifzierer mit polynomiellem Zeitaufwand gibt.

(ohne Beweis)

Beispiele

- **Definition: *HAMPATH***
  - Das Hamiltonsche Pfadproblem
    - Geg.:
      - ein gerichteter Graph
      - Zwei Knoten s,t
    - Ges.: existiert ein Hamitonscher Pfad von s nach t
      - d.h. ein gerichteter Pfad, der alle Knoten besucht, aber keine Kante zweimal benutzt
- **Algorithmus für Hamiltonscher Pfad:**
  - Rate eine Permutation $(s, v_1, v_2, \ldots, v_{n-2}, t)$
  - Teste, ob Permutation ein Pfad ist
    - falls ja, akzeptiere
    - falls nein, verwerfe
- **Also: HamPath $\in$ NP**





Das SAT Problem

- Eine Boolesche Funktion $f(x_1,x_2,..,x_n)$ ist erfüllbar, wenn es eine Wertebelegung für $x_1,x_2,..,x_n$ gibt, so dass $f(x_1,x_2,..,x_n) = 1$
    - $(x \vee y) \wedge (z \vee \neg x \vee \neg y) \wedge (x \vee \neg z)$ ist erfüllbar, da
        - die Belegung $x = 1$, $y = 0$, $z = 0$
    - $(1 \vee 0) \wedge (0 \vee 0 \vee 1) \wedge (1 \vee 1) = 1 \wedge 1 \wedge 1 = 1$ liefert.
- Definition (SAT Problem, die Mutter aller NPc Probleme)
    - **Gegeben**:
        - Boolesche Funktion ϕ
    - **Gesucht**:
        - Gibt es $x_1,x_2,..,x_n$ so dass $\phi(x_1,x_2,..,x_n) = 1$
- SAT ist in NP. Man vermutet, dass SAT nicht in P ist.

Das QSAT Problem

- Eine quantifizierte Boolesche Formel (QBF) besteht aus
  - Einer Folge von Quantoren $\exists x$, $\forall y$ mit daran gebundenen Variablen; obdA seien genau alle x_i mit ungeradem i existenzquantifiziert
  - Einer Booelschen Funktion $F(x_1,x_2,...,x_m)$
  - Jede Variable der Funktion ist genau einmal an einem Quantor gebunden
- Die quantifizierte Boolesche Formel ist erfüllbar falls
  - Im Falle eines Existenzquantors: $\exists x\ F(x) \Leftrightarrow F(0) \vee F(1)$
  - Im Falle eines Allquantors: $\forall x\ F(x) \Leftrightarrow F(0) \wedge F(1)$
- Definition (QSAT Problem, die Mutter aller PSPACEc Probleme)
  - **Gegeben**: Quantifizierte Boolesche Funktion ϕ
  - **Frage**: Gibt es x_1, so dass es für alle x_2 ein x_3 gibt, so dass ... so dass $\phi(x_1,x_2,..,x_n) = 1$
- QSAT ist in PSPACE. Man vermutet, dass QSAT nicht in NP ist.

Beispiele:

- $\exists x\ \forall y\ (x \wedge y) \vee (\neg x \wedge \neg y)$

  $= (\forall y\ (0 \wedge y) \vee (\neg 0 \wedge \neg y)) \quad \vee (\forall y\ (1 \wedge y) \vee (\neg 1 \wedge \neg y))$

  $= (\forall y: \neg y) \quad \vee (\forall y: y)$

  $= (\neg 0 \wedge \neg 1) \quad \vee (0 \wedge 1)$

  $= 0 \vee 0$

  $= 0$

- $\forall y\ \exists x\ (x \wedge y) \vee (\neg x \wedge \neg y)$

  $= (\exists x: (x \wedge 0) \vee (\neg x \wedge \neg 0)) \wedge (\exists x: (x \wedge 1) \vee (\neg x \wedge \neg 1))$

  $= (\exists x: : \neg x) \quad \wedge (\exists x: x)$

  $= (\neg 0 \vee \neg 1) \quad \wedge (0 \vee 1)$

  $= 1 \wedge 1$

  $= 1$

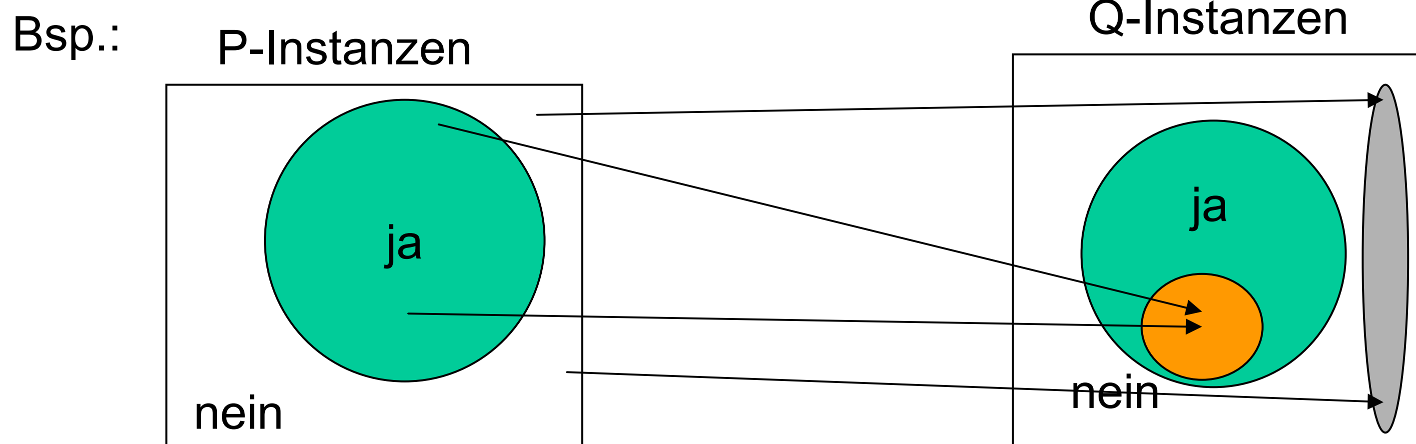
Einordnung von Problemen in P, NP, PSPACE

- **Angabe eines Algorithmus für Problem**
- **Reduktionstechniken u.a.**

Definition: Seien P,Q Probleme. Sei L_P (L_Q) die Menge der Instanzen des Problems P (Q), für die die Antwort „ja“ ist. P heißt auf Q polynomiell reduzierbar ($P \leq_p Q$), wenn es eine von einem deterministischen Algorithmus in Polynomzeit berechenbare Funktion f: $\Sigma^* \rightarrow \Sigma^*$ gibt, so dass

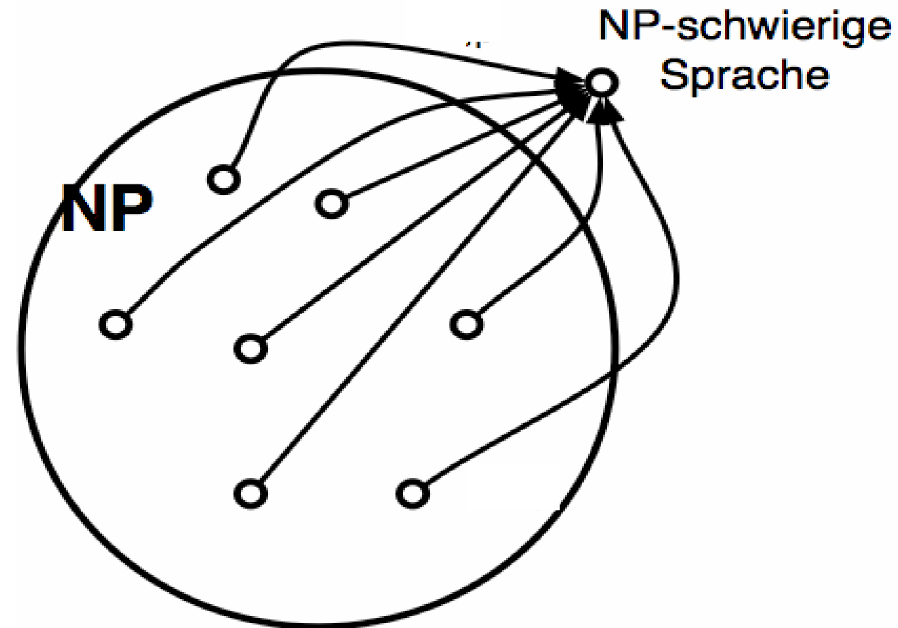
$$\underline{x \in L_P \Leftrightarrow f(x) \in L_Q}$$

Bsp.:



NP-Schwierig

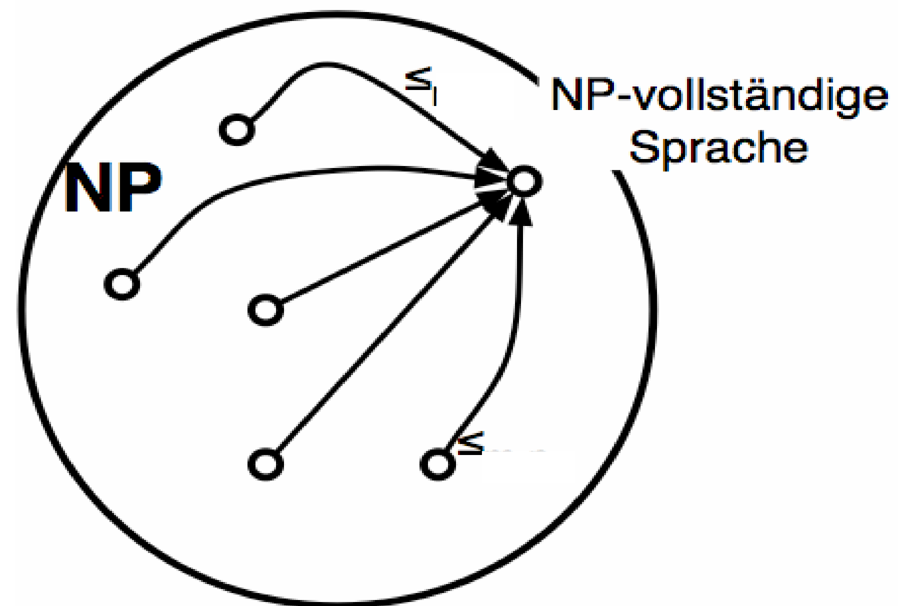
- Definition:
  - Eine Sprache S ist **NP-schwierig** (NP-hard) wenn:
    - jede Sprache aus NP mit einer Polynom-Zeit-Abbildungsreduktion auf S reduziert werden kann, d.h.
    - für alle $L \in NP$: $L \leq_p S$
- Theorem
  - Falls eine NP-schwierige Sprache in P ist, ist P=NP
- Beweis
  - Falls $S \in P$ und $L \leq_p S$ gilt $L \in P$.



NP-Vollständigkeit

- Definition:
  - Eine Sprache S ist **NP-vollständig** (NP-complete) wenn:
    - S $\in$ NP
    - S ist NP-schwierig

- Korollar:
  - Ist eine NP-vollständige Sprache in P, dann ist P=NP
- Beweis:
  - folgt aus der NP-Schwierigkeit der NP-vollständigen Sprache.

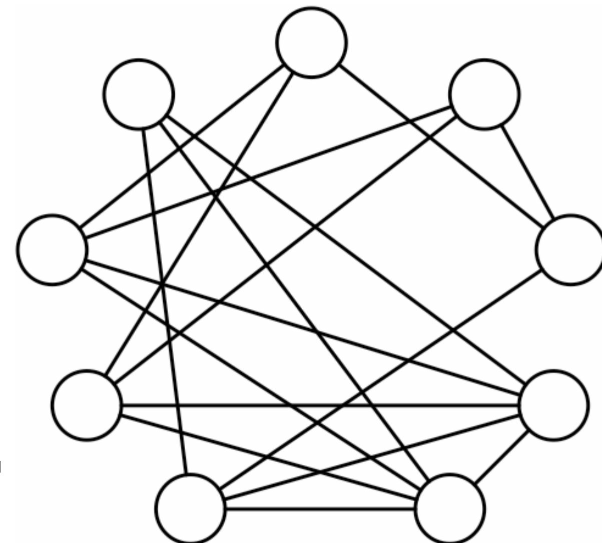


Das 3-SAT-Problem und das Clique-Problem

- 3-SAT:
  - **Gegeben**:
    - Eine Boolesche Formel in 3-CNF
  - **Gesucht**:
    - Gibt es eine erfüllende Belegung
- Definition k-Clique
  - Ein ungerichteter Graph G=(V,E) hat eine k-Clique,
    - falls es k verschiedene Knoten gibt,
    - so dass jeder mit jedem anderen eine Kante in G verbindet
- CLIQUE:
  - **Gegeben**:
    - Ein ungerichteter Graph G
    - Eine Zahl k
  - **Gesucht**:
    - Hat der Graph G eine Clique der Größe k?

$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$

k=4

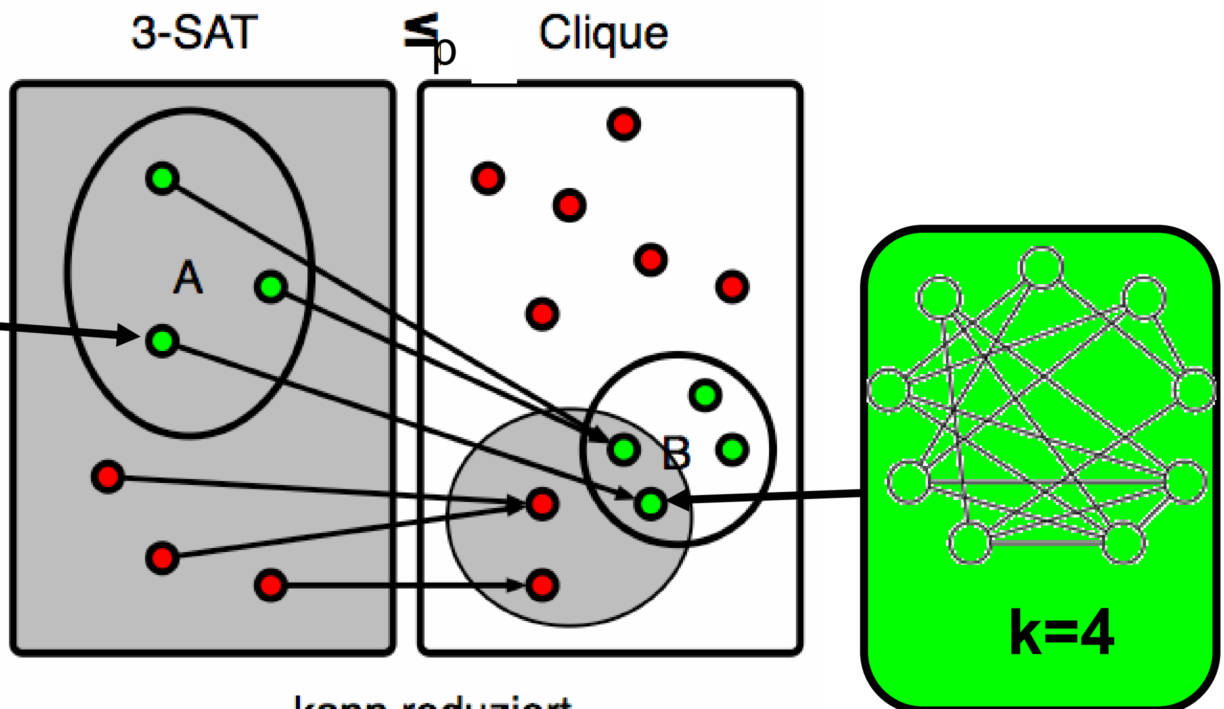


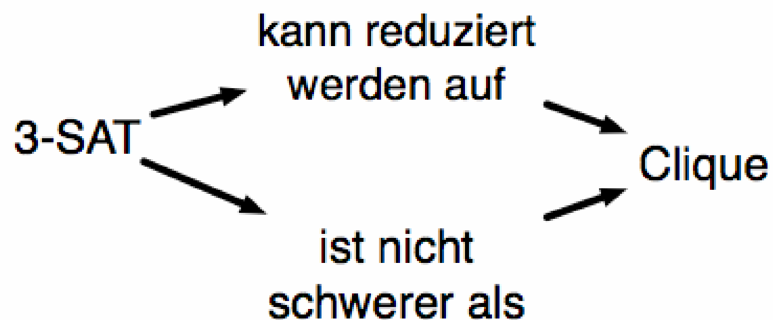
3-SAT läßt sich auf Clique reduzieren

TECHNISCHE UNIVERSITÄT DARMSTADT

- Theorem: 3-SAT $\leq_p$ CLIQUE

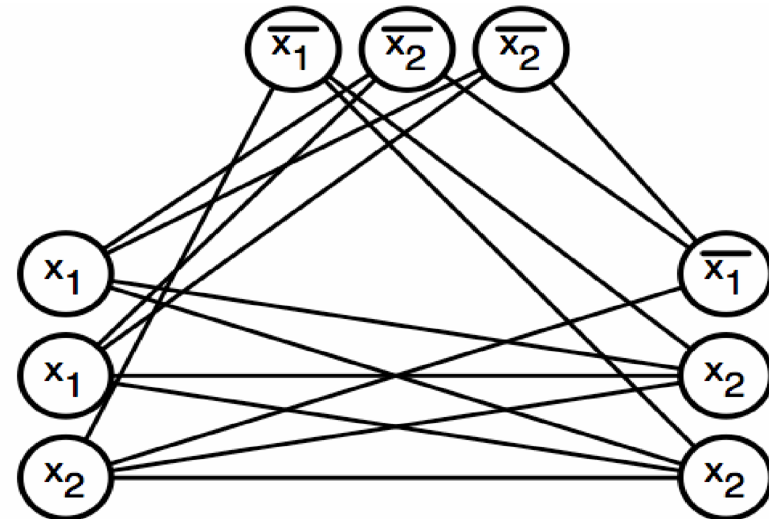
$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



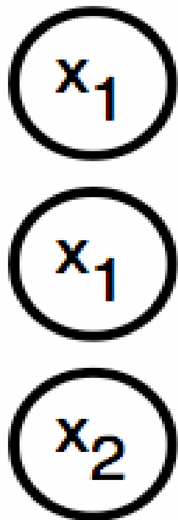


3-SAT läßt sich auf Clique reduzieren

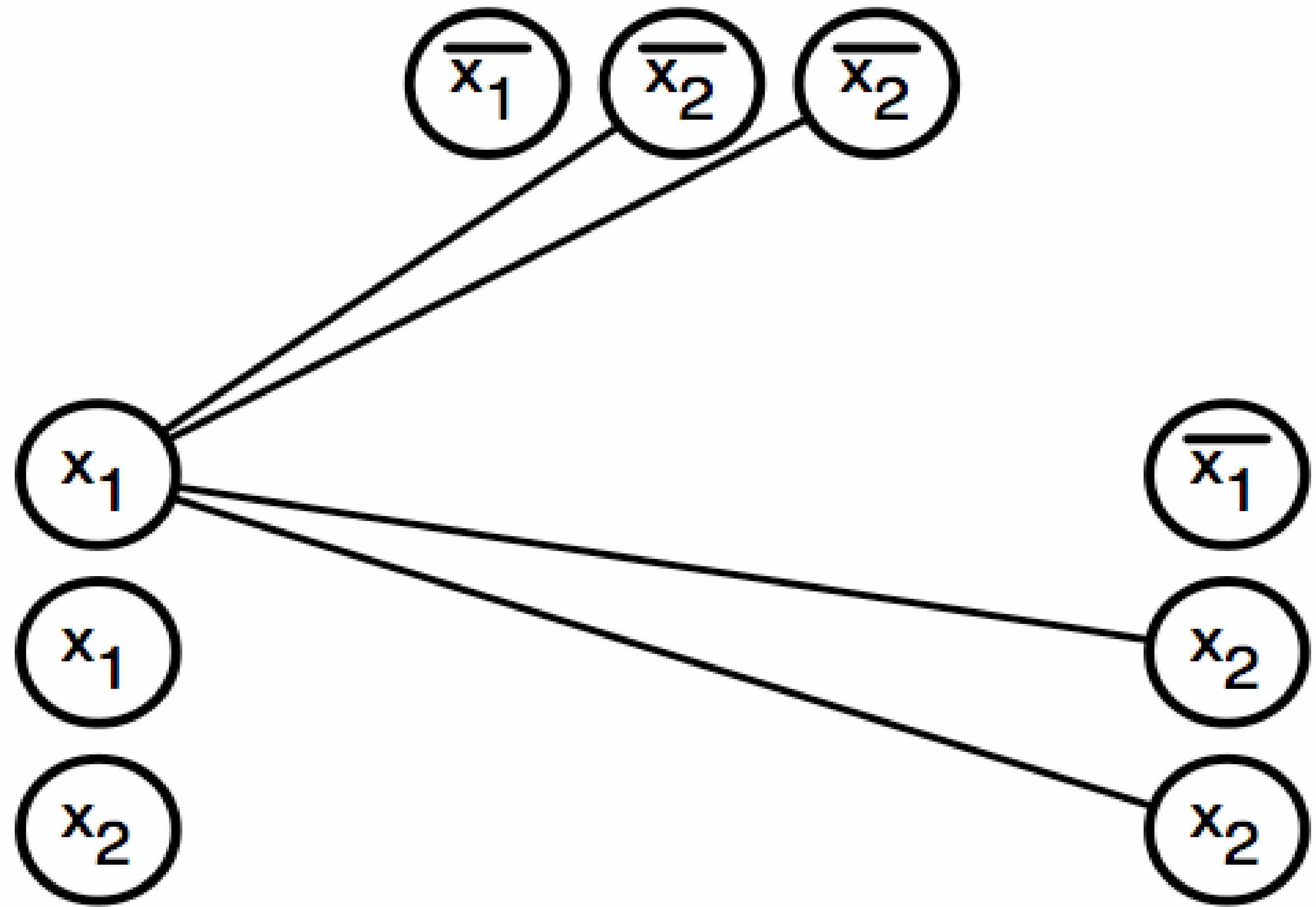
- Theorem: 3-SAT $\leq_p$ CLIQUE
- Beweis
  - Konstruiere Reduktionsfunktion f wie folgt:
  - f(ϕ) = <G,k>
  - k = Anzahl der Klauseln
  - Für jede Klausel C in ϕ werden drei Knoten angelegt, die mit den Literalen der Klausel bezeichnet werden
  - Füge Kante zwischen zwei Knoten ein, gdw.
    - die beiden Knoten nicht zur selben Klausel gehören und
    - die beiden Knoten nicht einer Variable und der selben negierten Variable entsprechen.



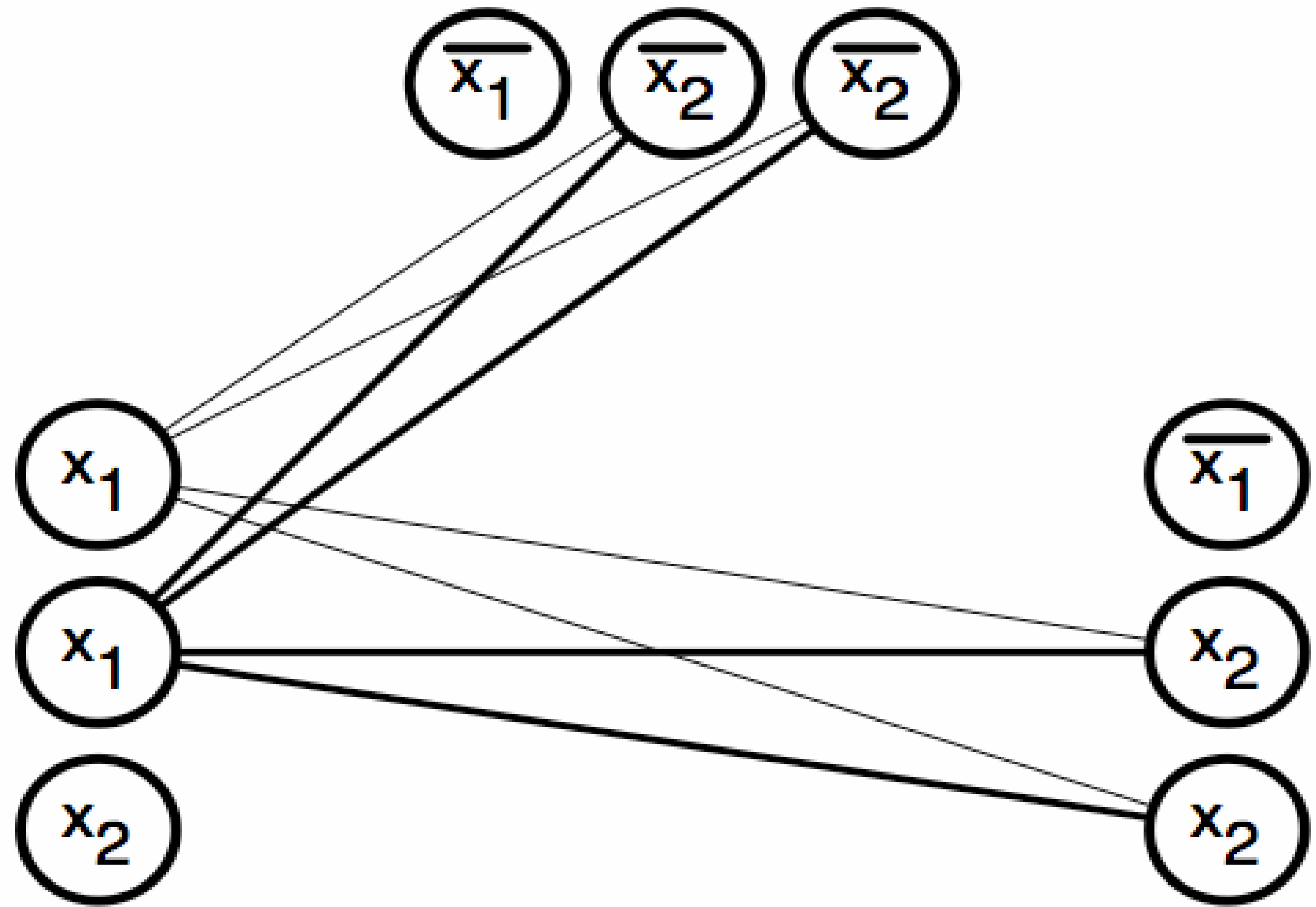
$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



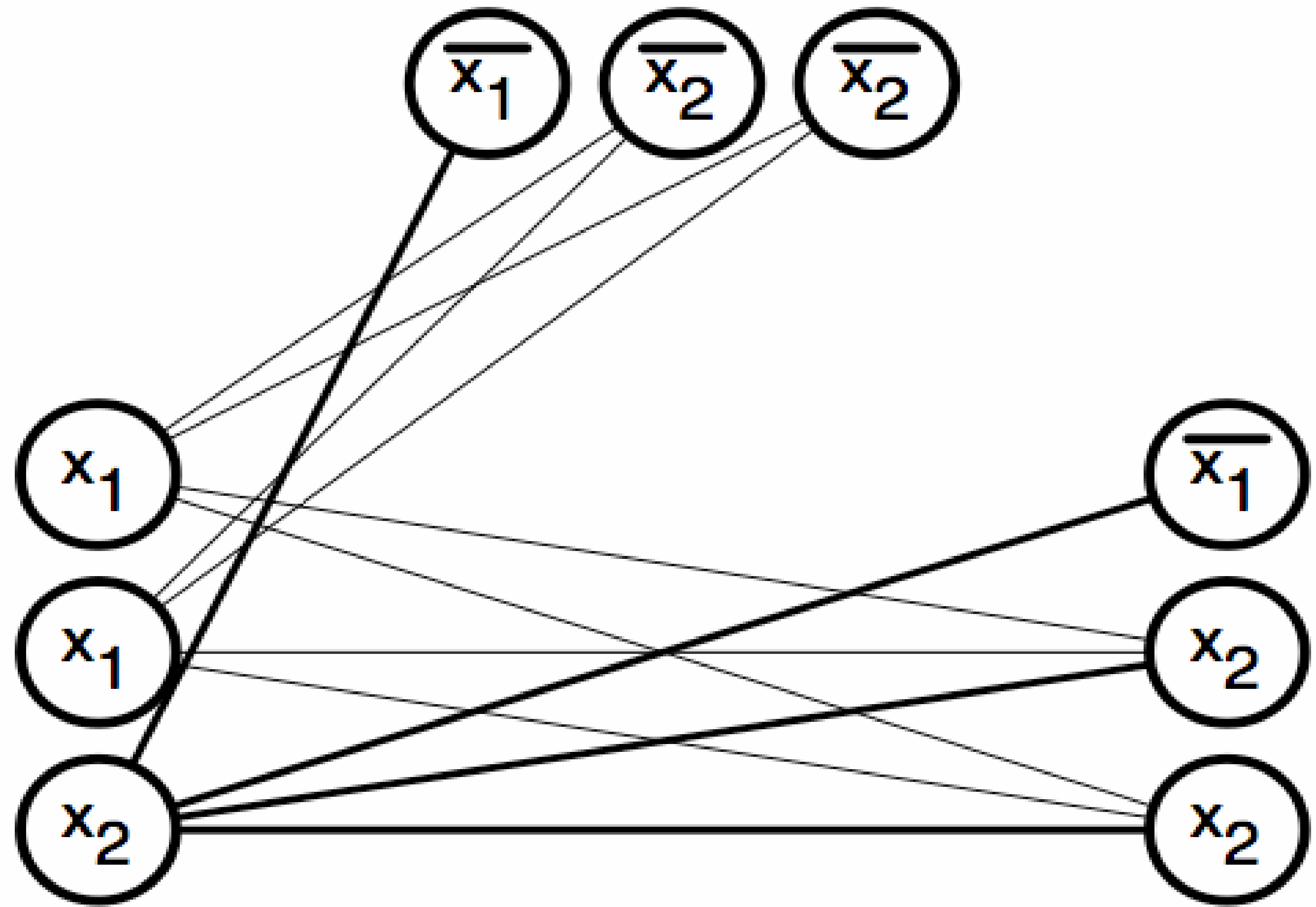
$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



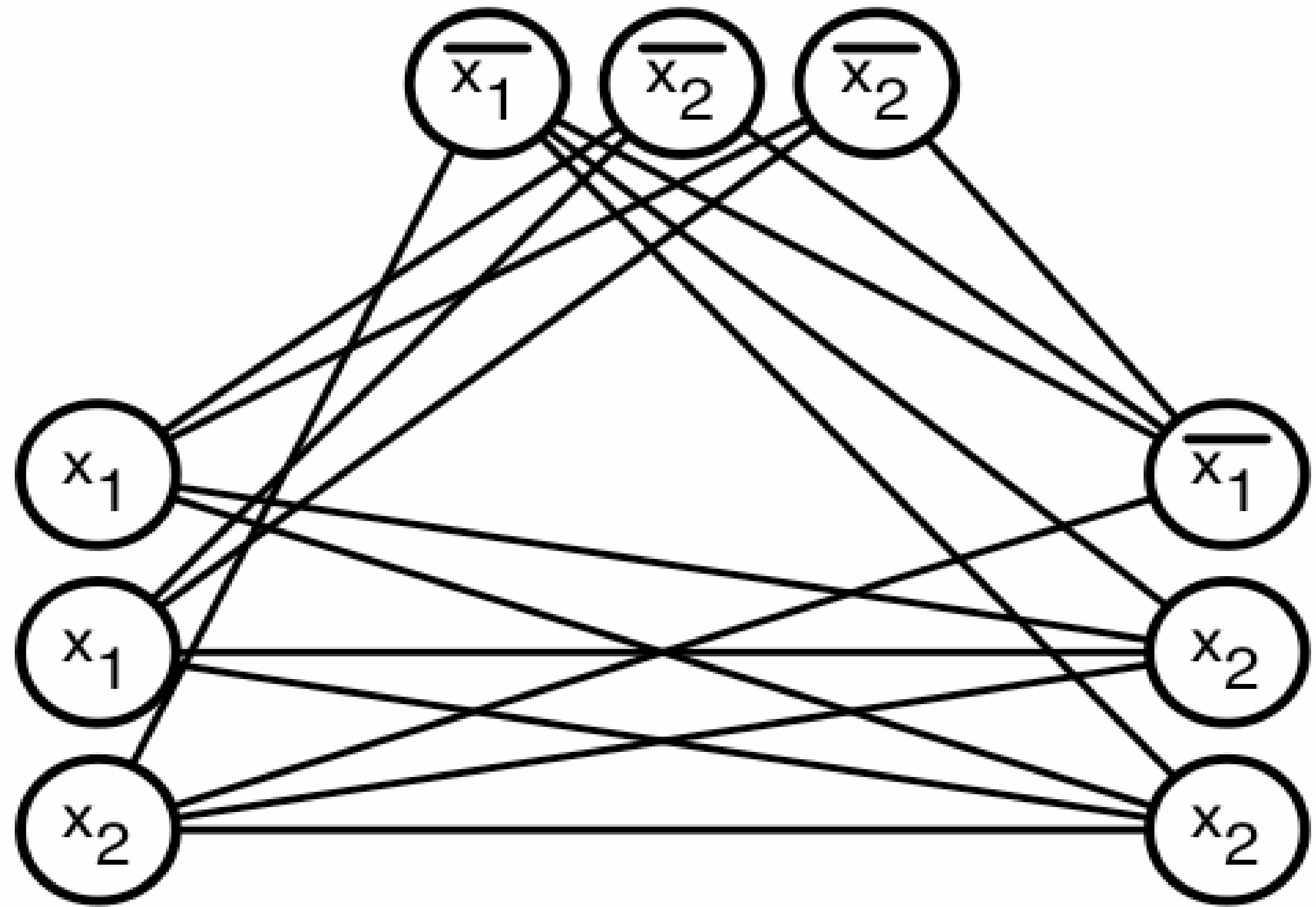
$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



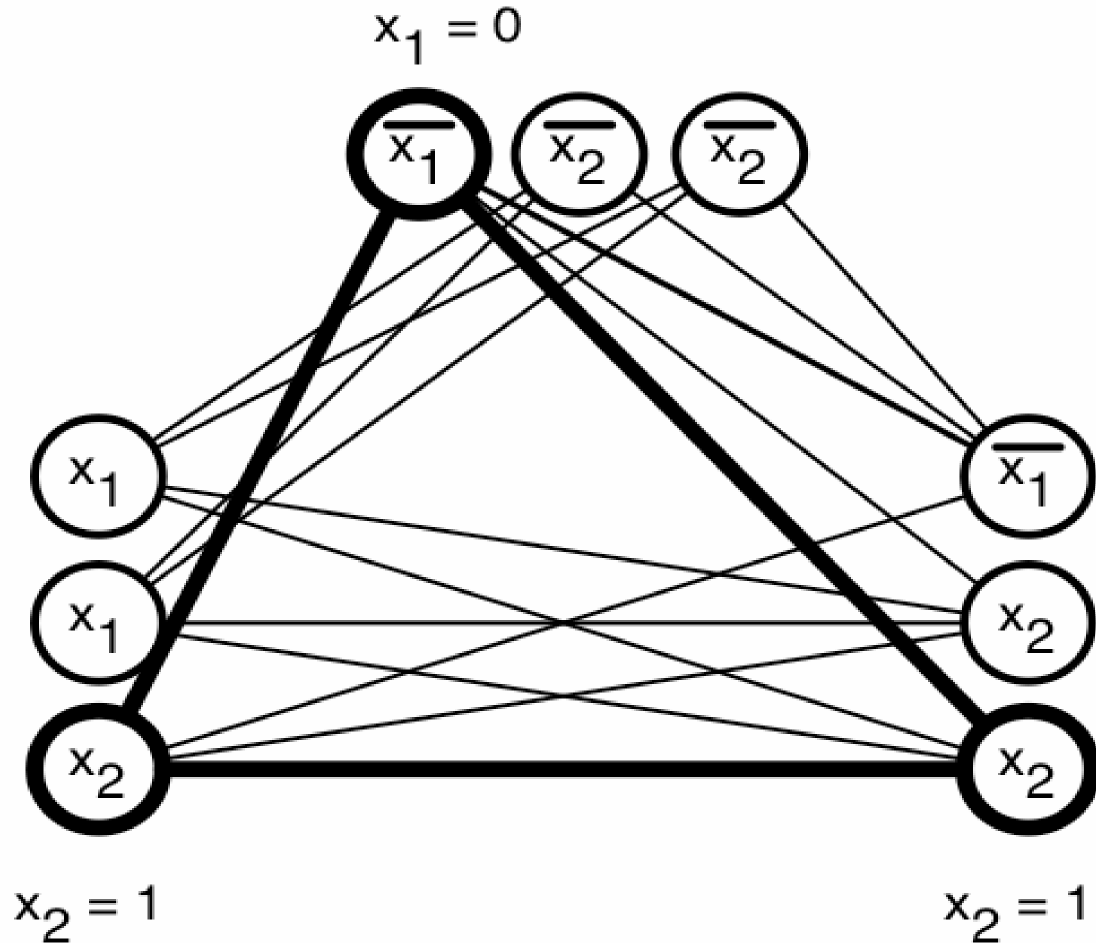
$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$


x1 = 0
x1
x2
x2
x1
x1
x2
x1
x2
x2
x2 = 1
x2 = 1
ψ = (x1 ∨ x1 ∨ x2) ∧ (x1 ∨ x2 ∨ x2) ∧ (x1 ∨ x2 ∨ x2)
0 0 1
1 0 0
1 1 1

Beweis der Korrektheit der Reduktionsfunktion

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die Reduktionsfunktion ist korrekt:
- Behauptung;
  - Eine erfüllende Belegung in ϕ existert gdw. eine k-Clique in G existiert
- 1. Fall: eine erfüllende Belegung existiert in ϕ
  - Dann liefert die Belegung in jeder Klausel mindestens ein Literal mit Wert 1
  - Wähle aus der Knotenmenge einer Klausel ein beliebiges solches Literal
  - Die gewählte Knotenmenge besteht dann aus k Knoten
  - Zwischen allen Knoten existiert eine Kante, da Variable und negierte Variable nicht gleichzeitig 1 sein können
- 2. Fall: eine k-Clique existiert in G
  - Jeder der Knoten der Clique gehört zu einer anderen Klausel
  - Setze die entsprechenden Literale auf 1
  - Bestimmte daraus die Variablen-Belegung
  - Das führt zu keinem Widerspruch, da keine Kanten zwischen einem Literal und seiner negierten Version existieren
- Laufzeit:
  - Konstruktion des Graphens und der Kanten benötigt höchstens quadratische Zeit.