Informatik III

4.2 NP

Christian Schindelhauer

Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Rechnernetze und Telematik
Wintersemester 2007/08

Komplexitätstheorie

P und NP

2

Zwei wichtige Komplexitätsklassen

- **Definition:**
  - $\mathrm{P} = \bigcup_k \mathrm{TIME}(n^k)$
  - $\mathrm{NP} = \bigcup_k \mathrm{NTIME}(n^k)$

- **In Worten**
  - P: Klasse aller Sprachen, die von einer Polynom-Zeit DTM entschieden werden
  - NP: Klasse aller Sprachen, die von einer Polynom-Zeit NTM entschieden werden können.

Probleme in P

Binärrechnung	**Laufzeit in Anzahl der Stellen der Binärzahl (Mehrband-TM)**
Addition	$O(n)$
Vergleich	$O(n)$
Multiplikation	$O(n^2)$
Matrixmultiplikation zweier n x n-Matrizen mit n-stelligen Zahlen	$O(n^5)$
Berechnung der Determinante einer n x n-Matrix mit n-stelligen Einträge	$O(n^5)$
Division mit Rest	$O(n^2)$
ggT	$O(n^3)$
kgV	$O(n^3)$

Bitte nicht die Laufzeiten lernen!

Probleme in P

Operationen auf n Zahlen mit je m Stellen	**Laufzeit (Mehrband-TM)**
Minimum, Maximum, Median, Mittelwert	O(nm)
Sortieren	O(nm log n)
Sortieren mit Bubble-Sort/Quick-Sort	$O(mn^2)$
Permutiere gemäß gegebener Permutation	O(nm log n)

Bitte nicht die Laufzeiten lernen!

*Laufzeit ist nicht unbedingt optimal

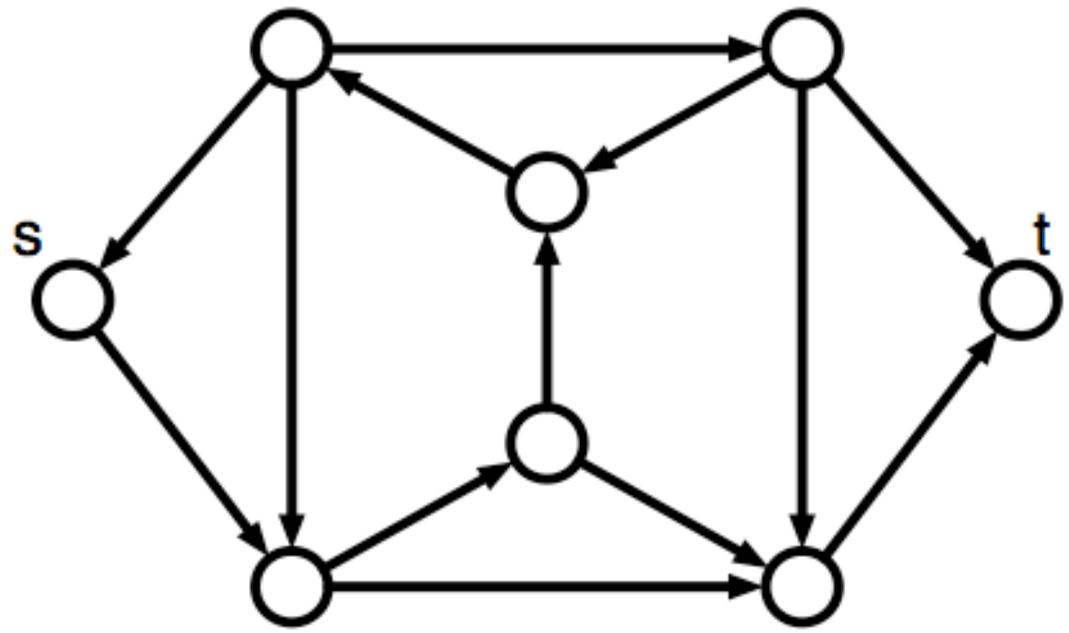
Probleme in P

Graphenprobleme n Knoten, m Kanten Graph gegeben als Adjazenzliste	**Laufzeit (Mehrband-TM)**
Gibt es Weg von Start zu Ziel	O((n+m) log n)
Kürzester Weg von Start zu Ziel	O(nm log n)
Gibt es einen Kreis im Graph	O((n+m) log n)
Maximaler Grad	O(m log n)
Durchmesser	$O(n^3 m \log n)$
Zusammenhangskomponenten	$O((n+m)n^2 \log n)$

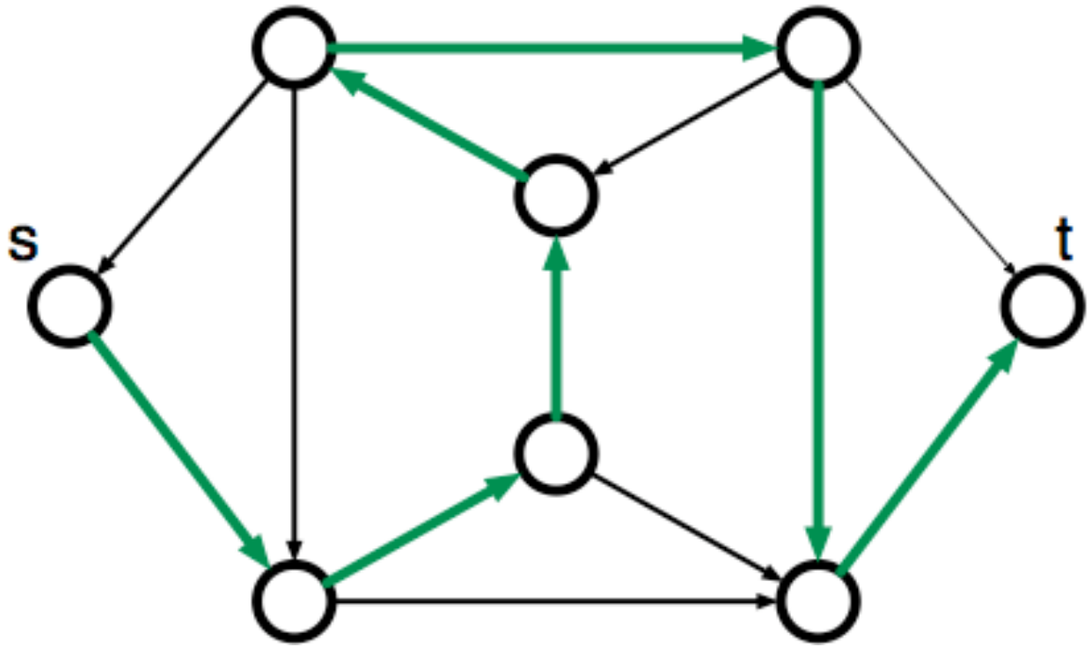
Bitte nicht die Laufzeiten lernen!

Laufzeiten sind nicht unbedingt optimal

Hamiltonsche Pfade
Ein Problem in NP



- **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 $(s,v_1,v_2,...,v_{n-2},t)$
  - Teste, ob Permutation ein Pfad ist
    - falls ja, akzeptiere
    - falls nein, verwerfe
- **Also: HamPath $\in$ NP**



Die Nicht-Primzahlen

- **Definition: COMPOSITES**
  - Geg.: x (als Binärzahl)
  - Ges.: Gibt es ganze Zahlen p,q>1, so dass x = p q
- **COMPOSITES := {x | x = p q, für ganze Zahlen p,q>1}**
- **NTM für COMPOSITES :**
  - Rate p,q>1
  - Berechne p q
  - Akzeptiere, falls p q = x
  - Verwerfe sonst
- **Also ist COMPOSITES $\in$ NP**

Komplexitätstheorie

Verifizierer und NP

9

Der Verifizierer

- **Definition**
  - Ein **Verifizierer** für eine Sprache A ist eine DTM V, wobei
    - A = {w | V akzeptiert <w,c> für ein Wort c}
  - Ein **Polynom-Zeit-Verifizierer** hat eine Laufzeit die durch ein Polynom $|w|^k$ beschränkt ist.
  - Eine Sprache ist in **Polynom-Zeit verifizierbar**, falls sie einen Polynom-Zeit-Verifizierer hat.
- **Theorem**
  - NP beschreibt genau die Sprachen, die in Polynom-Zeit verifiziert werden können.

Verifizierbare Sprachen sind in NP

- **Ein Polynom-Zeit-Verifizierer für eine Sprache A ist eine DTM V, wobei**
  - A = {w | V akzeptiert <w,c> für ein Wort c}
  - und V mit Polynom-Laufzeit $O(|w|^k)$ beschränkt ist.
- **Theorem**
  - NP beschreibt genau die die Sprachen, die in Polynom-Zeit verifiziert werden können.
- **Beweis des 1. Teil:**
  - Die Sprachen, die in Polynom-Zeit verifiziert werden können, sind in NP.

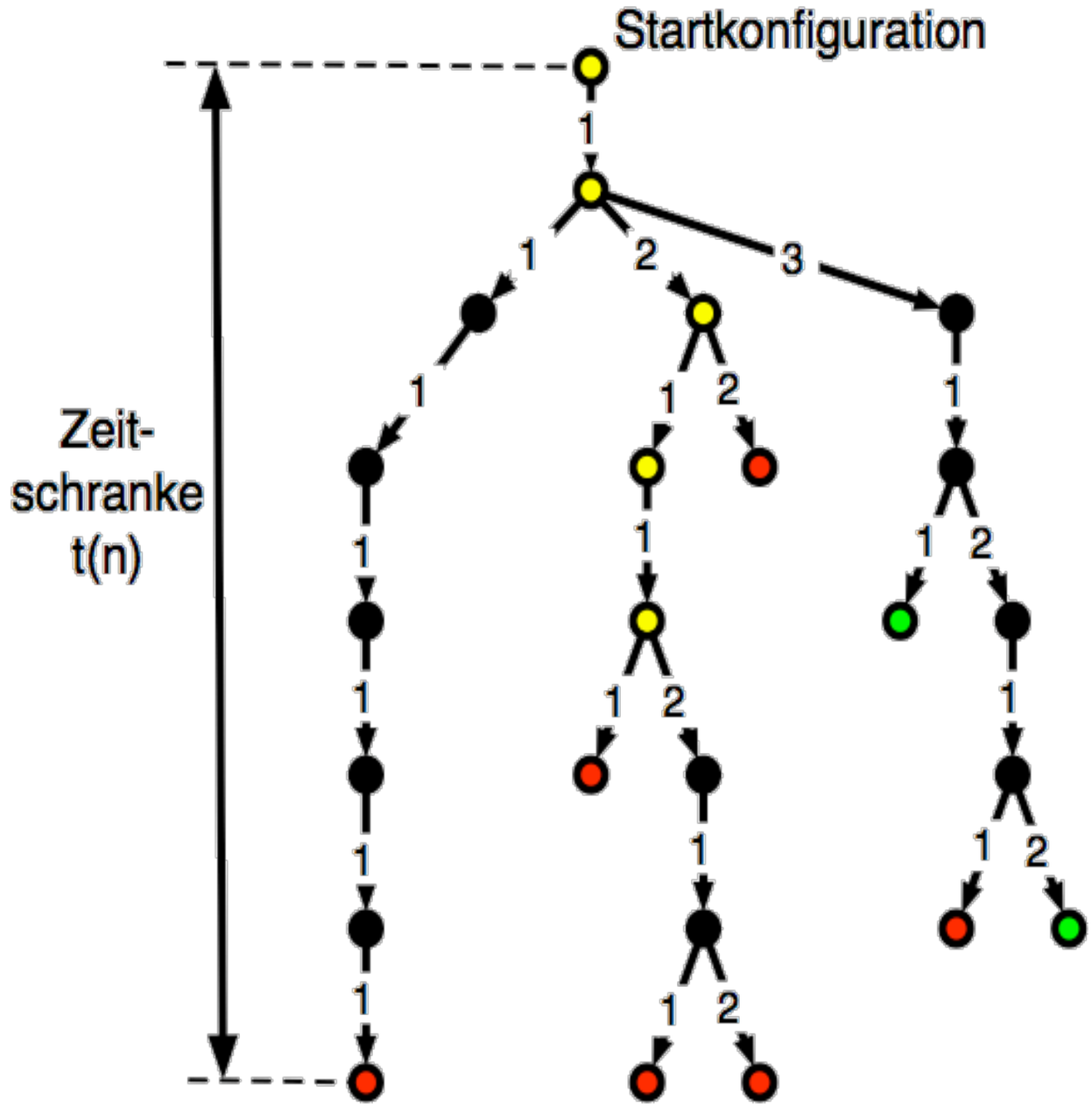
- **Konstruiere NTM M die A in Polynom-Zeit akzeptiert:**
  - M = “Auf Eingabe w,
    - Rate ein Wort c der Länge $\leq |w|^k$
    - Führe Berechnung von V auf Eingabe <w,c> durch
    - Akzeptiere, wenn V akzeptiert”
- **M akzeptiert genau die Worte in A**
  - da V nur Worte <w,c> der Länge $|w|^k$ bearbeiten kann, wird auch das relevante Wort c berücksichtigt, wenn V auf <w,c> akzeptiert.
- **M rechnet in Polynom-Zeit:**
  - Laufzeit für Raten $\leq |w|^k$
  - Laufzeit für Verifizieren $\leq |w|^k$

Alle Sprachen in NP sind verifizierbar

- **Ein Polynom-Zeit-Verifizierer für eine Sprache A ist eine DTM V, wobei**
  - A = {w | V akzeptiert <w,c> für ein Wort c}
  - und V mit Polynom-Laufzeit $O(|w|^k)$ beschränkt ist.
- **Theorem**
  - NP beschreibt genau die die Sprachen, die in Polynom-Zeit verifiziert werden können.
- **Beweis des 2. Teil:**
  - Die Sprachen in NP können in Polynom-Zeit verifiziert werden
- **Gegeben:**
  - NTM M mit Polynom-Laufzeit $|w|^k$
- **Konstruiere Polynom-Zeit-Verifizierer**
  - Beschreibe c den Pfad eines Berechnungsbaums von M
  - V = “Auf Eingabe <w,c>,
    - Führe Berechnung von M auf Eingabe w durch
    - Falls M in Schritt t nichtdeterministisch verzweigt, gibt der Buchstabe c_t an, welcher Folgezustand von A genommen wird.
    - Akzeptiere, wenn M akzeptiert”
- **V verifiziert genau die Worte in L(M)**
  - da M nur $|w|^k$ Schritte rechnen kann, kann auch der Berechnungspfad beschrieben von c von V eingelesen werden, für den M auf w akzeptert (wenn M jemals akzeptiert)
- **V rechnet in Polynom-Zeit:**
  - da M in Polynom-Zeit rechnet

Alle Sprachen in NP sind verifizierbar

- **Gegeben:**
  - NTM M mit Polynom-Laufteit $|w|^k$
  - Konstruiere Polynom-Zeit-Verifizierer
- **Beschreibe c den Pfad eines Berechnungsbaums von M**
- **V = “Auf Eingabe <w,c>,**
  - Führe Berechnung von M auf Eingabe w durch
  - Falls M in Schritt t nichtdeterministisch verzweigt, gibt der Buchstabe c_t an, welcher Folgezustand von A genommen wird.
  - Akzeptiere, wenn M akzeptiert”
- **V verifiziert genau die Worte in L(M)**
  - da M nur $|w|^k$ Schritte rechnen kann, kann auch der Berechnungspfad beschrieben von c von V eingelesen werden, für den M auf w akzeptert (wenn M jemals akzeptiert)
- **V rechnet in Polynom-Zeit:**
  - da M in Polynom-Zeit rechnet



Das Teilsummenproblem (Subset-Sum-Problem)

- **Definition SUBSET-SUM:**
  - Gegeben:
    - Menge von natürlichen Zahlen $S = \{x_1,..., x_k\}$
    - Eine natürliche Zahl t
  - Gesucht:
    - Gibt es eine Teilmenge $\{y_1,...,y_m\} \subseteq \{x_1,..., x_k\}$ so dass

$$\sum_{i=1}^{m} y_i = t$$

- **Theorem**
  - Das Teilsummenproblem ist in NP

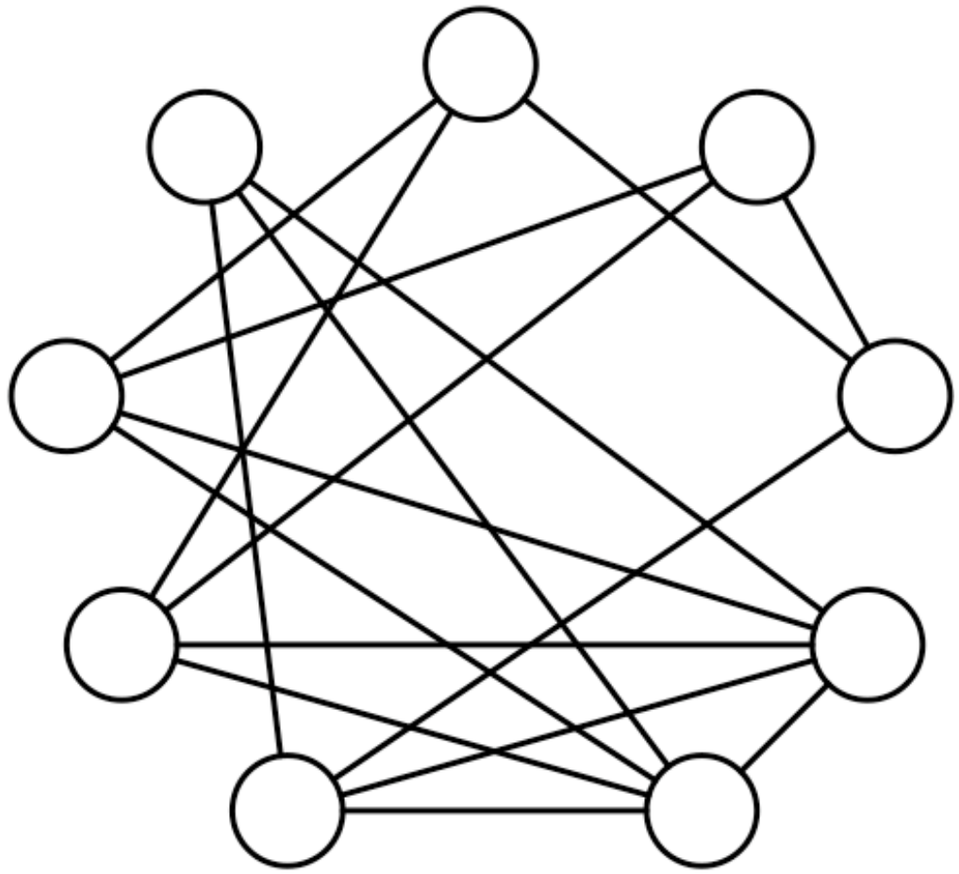
- **Beweis**
  - Betrachte
    $A = \{<x_1,..., x_k,t> \mid$
    es gibt $\{y_1,...,y_m\} \subseteq \{x_1,..., x_k\}$ und

$$\sum_{i=1}^{m} y_i = t \ \ \}$$

  - Verifizierer testet, ob die Teilmengenbeziehung gilt und ob die Summe der Teilmenge t entspricht
  - Laufzeit: O(n log n)

Das Cliquen-Problem ist in NP

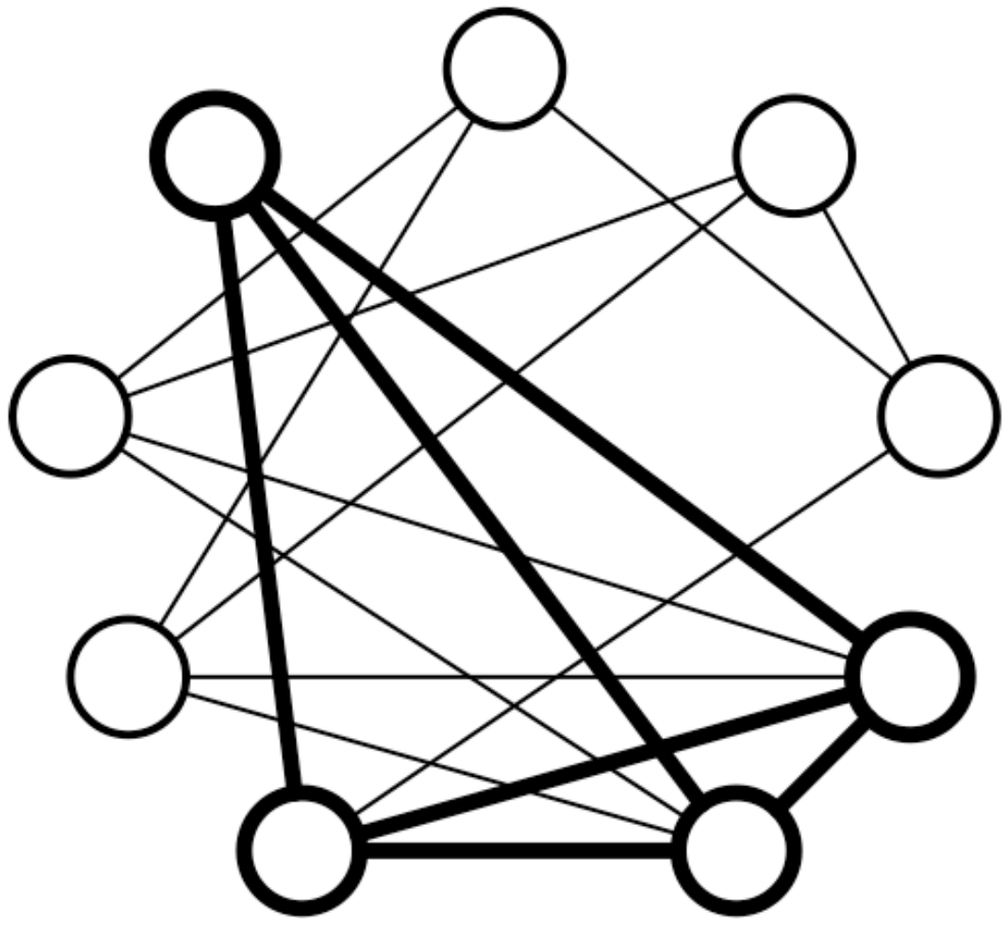
- **Definition k-Clique**
  - Ein Graph G=(V,E) hat eine k-Clique,
    - falls es k verschiedene Knoten gibt,
    - so dass jeder mit jedem anderen eine Kante in G verbindet
- **Das Cliquen-Problem**
  - Gegeben:
    - Ein ungerichteter Graph G
    - Eine Zahl k
  - Gesucht:
    - Hat der Graph G eine Clique der Größe k?



k=4

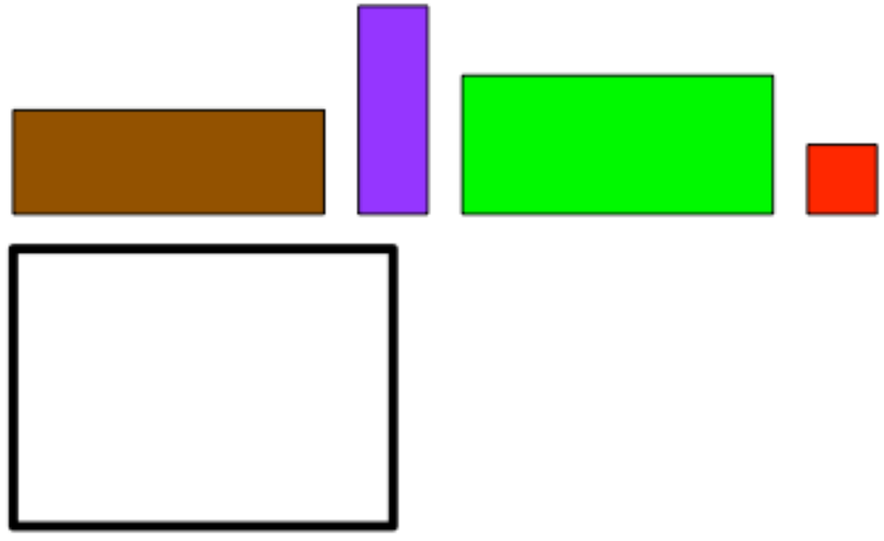
Das Cliquen-Problem ist in NP

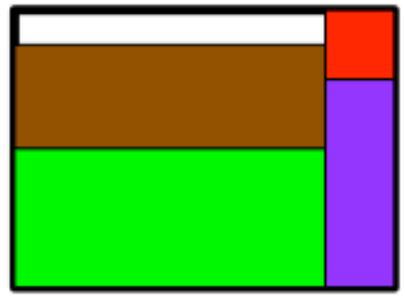
- **Theorem:**
  - Das Cliquen-Problem ist in NP
- **Beweis:**
  - Betrachte
    A = {<G,k> | es gibt Knoten $v_1,v_2,..,v_k$
    die eine k-Clique in G sind}.
  - Verifizierer testet,
    - ob die k Knoten unterschiedlich sind
    - und ob zwischen allen Knoten eine Kante existiert.
  - Laufzeit: $O(n^2)$



Das Koffer-Pack-Problem ist in NP

- **Definition Koffer-Pack-Problem:**
  - Gegeben:
    - Einge Menge M von kleinen Rechtecken mit geradzahliger Kantenlänge
    - Ein großes Rechteck R
  - Gesucht:
    - Passen die kleinen Rechtecke orthogonal überschneidungsfrei in das große Rechteck?
- **Theorem**
  - Das Koffer-Pack-Problem ist in NP
- **Beweis**
  - A = {<M,R>| es gibt eine Lagebeschreibung in der die Rechtecke sich nicht überschneiden und innerhalb von R liegen}





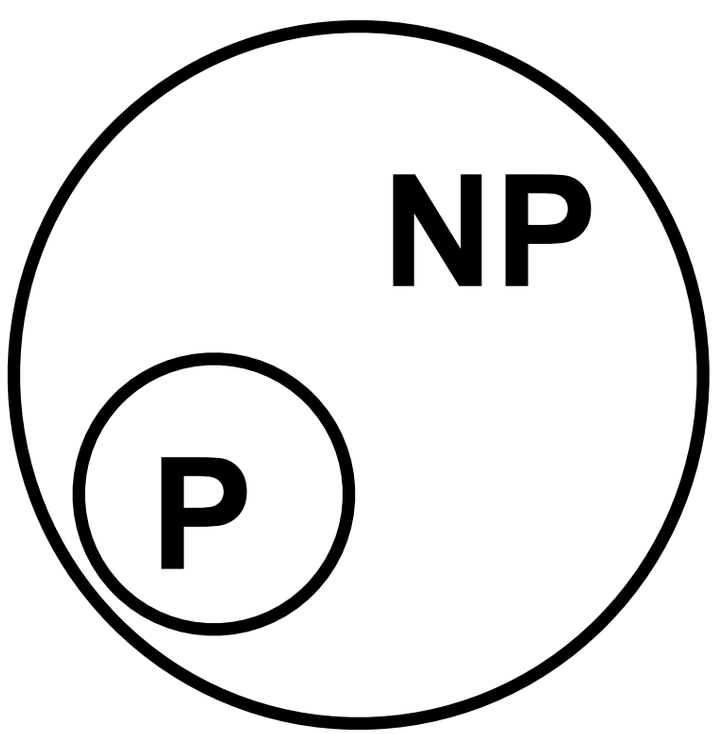
Komplexitätstheorie

P versus NP

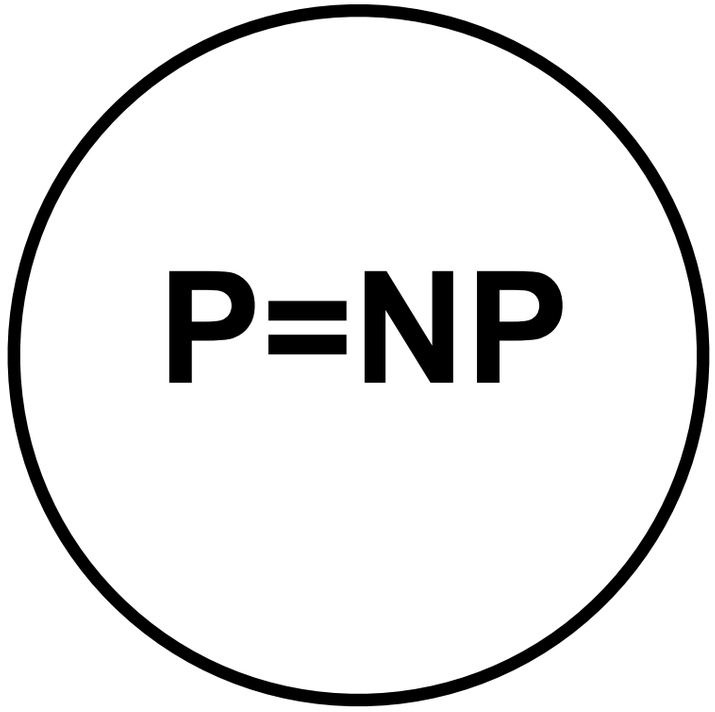
18

Die Frage P versus NP (I)

- **P = Klasse aller Probleme, die effizient *entschieden* werden können**
- **NP = Klasse aller Problem, die effizient *verifiziert* werden können**
  - Beispiele: Hamiltonscher Pfad, Clique, Teilsummenproblem, Koffer-Pack-Problem
- **Es gibt jetzt zwei Möglichkeiten**



- **Was weiß man?**
  - **P** $\subseteq$ **NP** $\subseteq \bigcup_k \mathrm{TIME}(2^{n^k})$ **= EXPTIME**



Auswirkungen auf Kryptographie

- **Was folgt aus P=NP?**
  - Schwierige kombinatorische Probleme sind lösbar
  - Viele kryptographische Systeme werden in Polynomzeit lösbar:
    - A = { Code | es gibt einen Originaltext der zum Code passt}
    - A' = { Code | es gibt einen Originaltext, der mit b_1 anfängt und zum Code passt}
    - A" = { Code | es gibt einen Originaltext, der mit b_1 b_2 anfängt und zum Code passt}
    - ...
    - Wäre jeweils in P. Damit kann der Originaltext Bit für Bit aufgedröselt werden

- **Was folgt aus P≠NP.**
  - Dann bleibt gewisse Probleme in NP praktisch unlösbar.

- **Was weiß man?**
  - Wenig, außer:
  - Cook-Levin: Genau dann wenn man ein bestimmtes Problem in Polynomzeit lösen kann, dann ist P=NP.

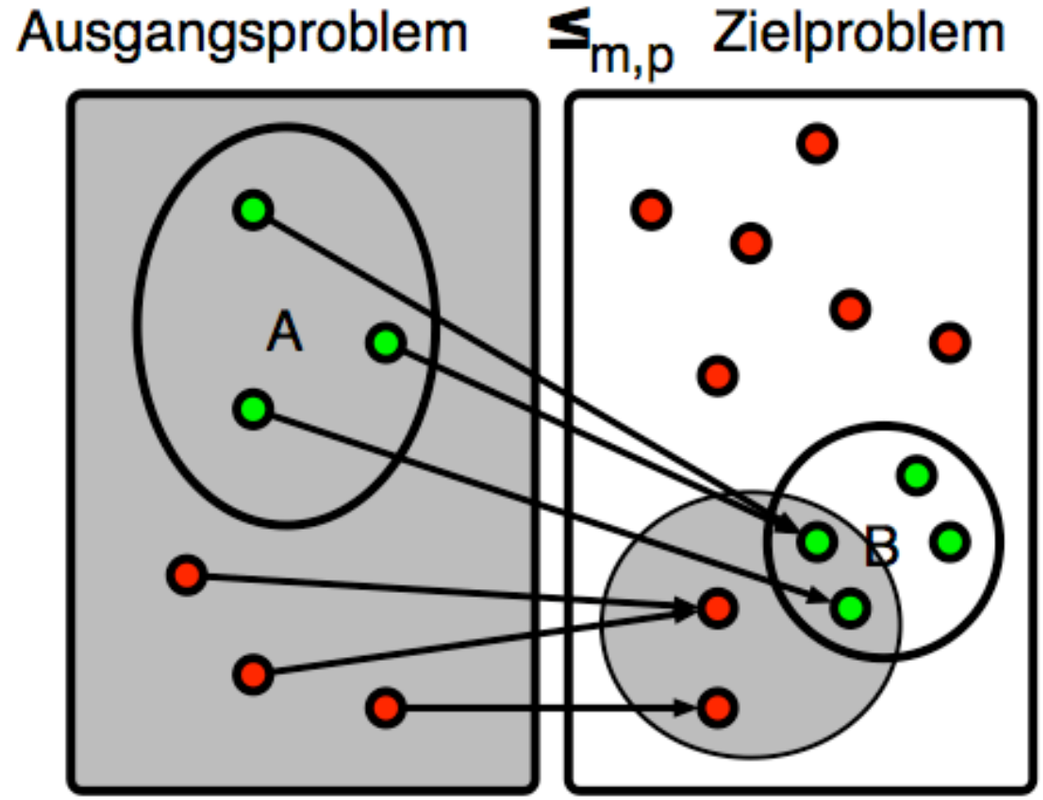
Komplexitätstheorie

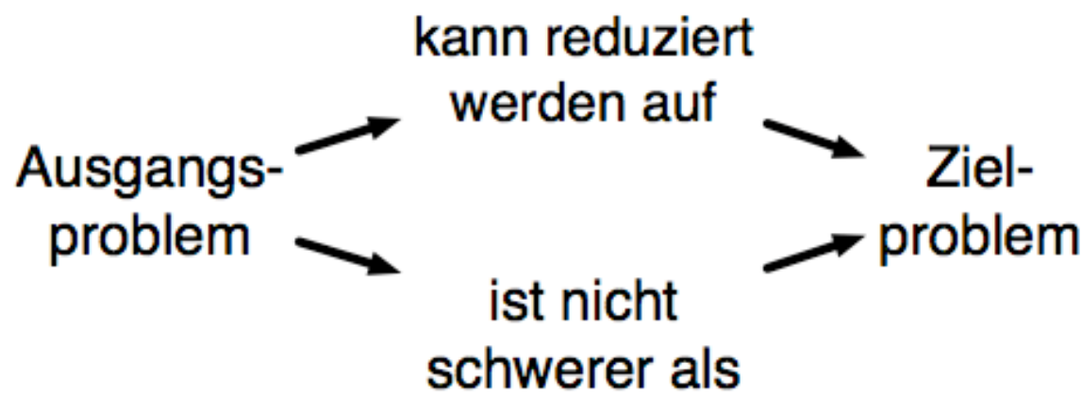
Polynom-Zeit-Reduktion

21

Die Polynom-Zeit-Abbildungsreduktion

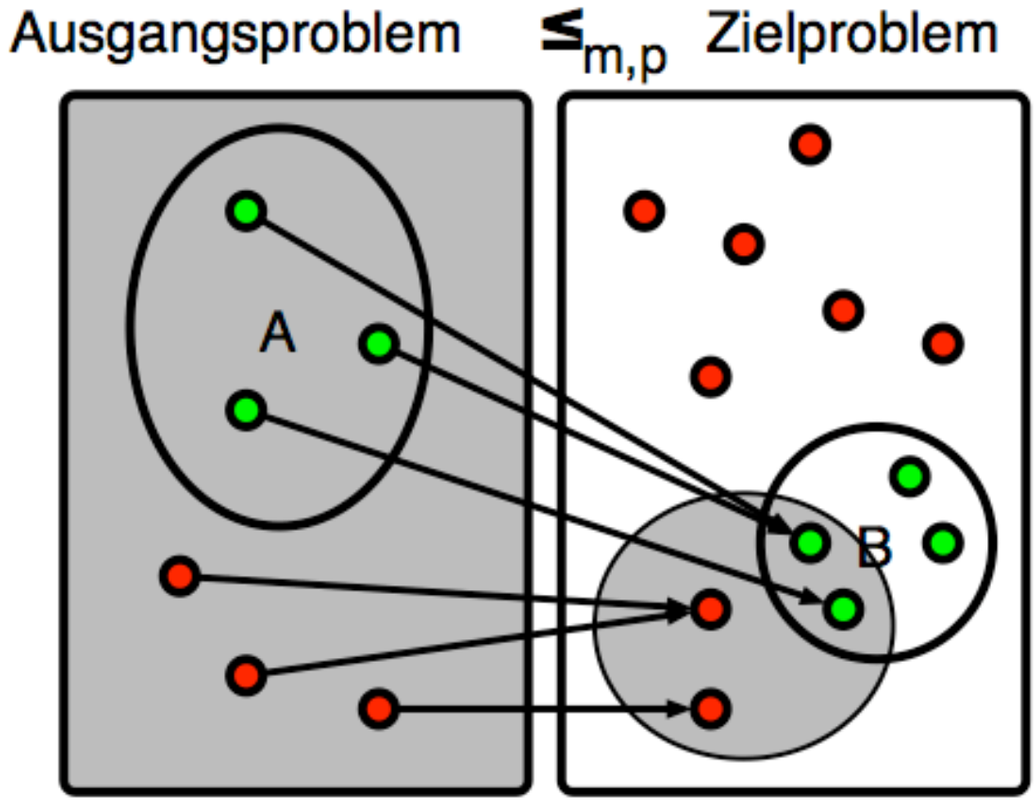
- **Definition (Abbildungsreduktion, Polynomial Time Mapping Reduction, Many-one)**
  - Eine Sprache A kann durch Abbildung auf eine Sprache B in Polynom-Zeit reduziert werden: $A \leq_{m,p} B$,
    - falls es eine in Polynom-Zeit berechenbare Funktion $f: \Sigma^* \rightarrow \Sigma^*$ gibt,
    - so dass für alle w: $w \in A \Leftrightarrow f(w) \in B$
  - Die Funktion f heißt die Reduktion von A auf B.

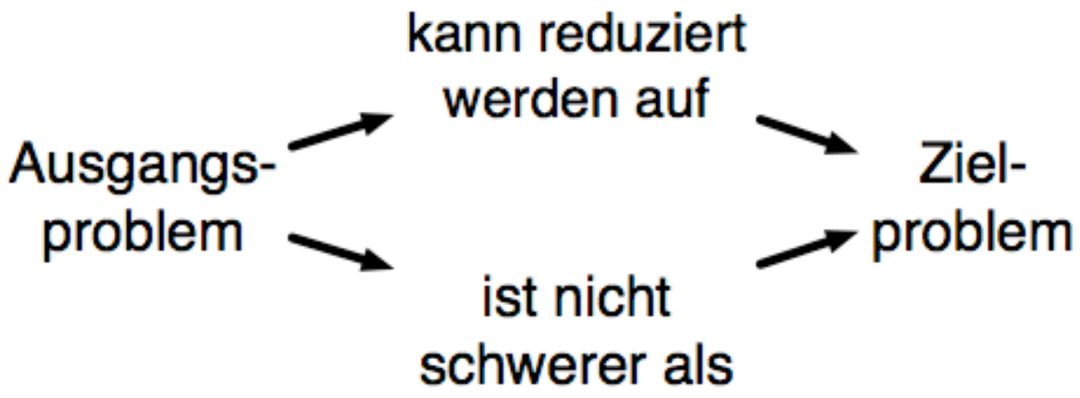




Polynom-Zeit-Abbildungsreduktion und P

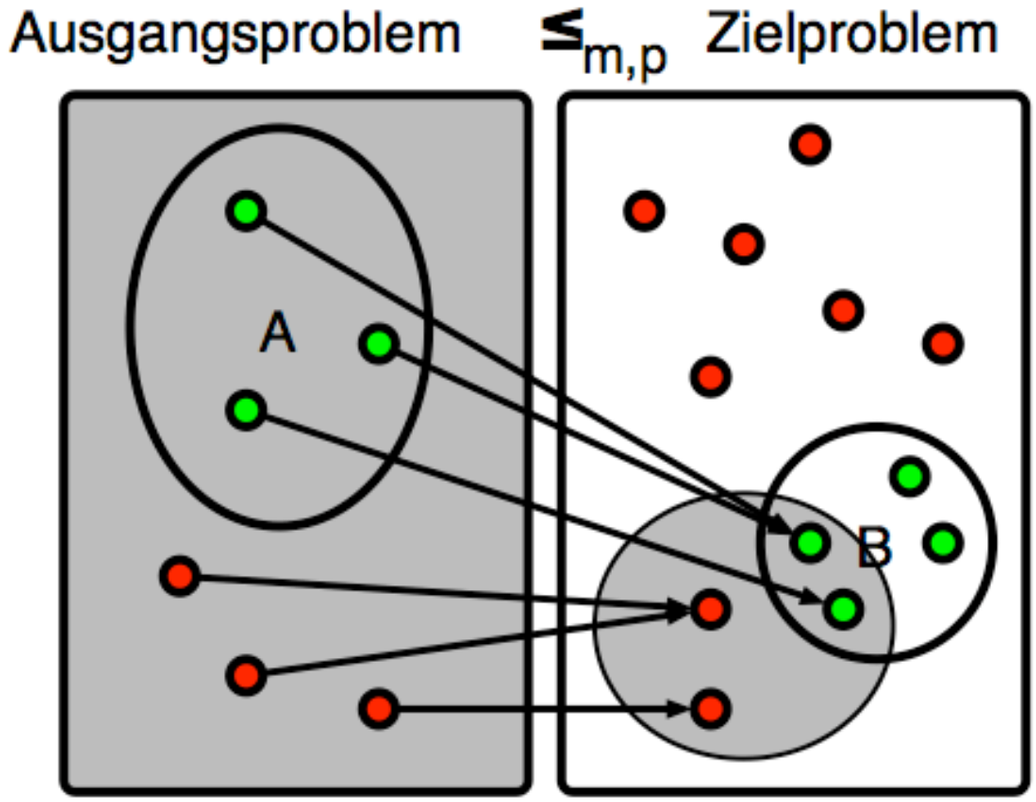
- **Theorem**
  - Falls $A \leq_{m,P} B$ und B ist in P, dann ist A auch in P.
- **Beweis**
  - Sei M, eine Turing-Maschine, die B in Polynom-Zeit $O(n^c)$ entscheidet.
  - Betrachte die Polynom-Zeit-TM:
  - N = “Auf Eingabe w:
    - Berechne f(w) in Zeit $O(n^k)$
    - Führe die Berechnung von M auf Eingabe f(w) durch
    - N gibt aus, was M ausgibt”
  - N entscheidet A
  - N berechnet A in Polynom-Zeit:
    - f(w) kann in Polynom-Zeit berechnet werden
    - f(w) hat Länge $O(|w|^k)$
    - M rechnet auf Eingabe f(w) in Zeit $O(|f(w)|^c) = O(|w|^{k\,c})$

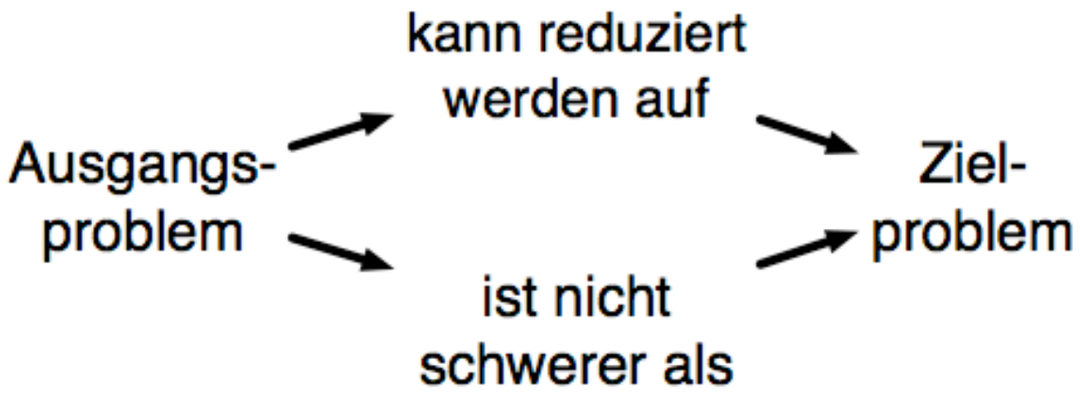




Polynom-Zeit-Abbildungsreduktion und NP

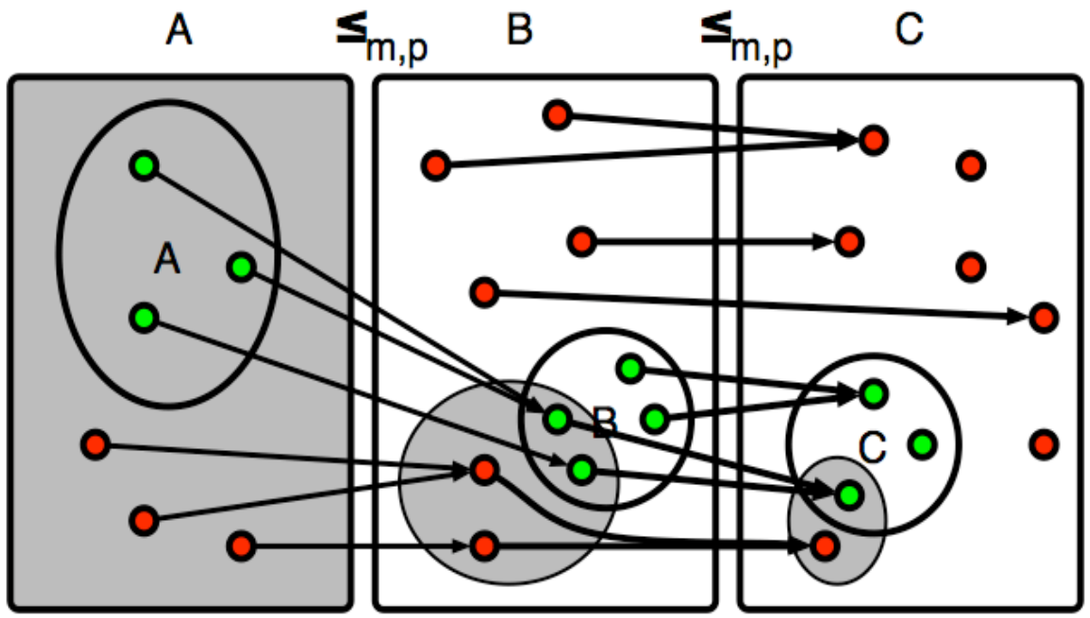
- **Theorem**
  - Falls $A \leq_{m,p} B$ und B ist in NP, dann ist A auch in NP.
- **Beweis**
  - Sei M, eine NTM, die B in Polynom-Zeit $O(n^c)$ entscheidet.
  - Betrachte die Polynom-Zeit-NTM:
  - N = “Auf Eingabe w:
    - Berechne f(w) in Zeit $O(n^k)$
    - Führe die (nicht-deterministische) Berechnung von M auf Eingabe f(w) durch
    - N gibt aus, was M ausgibt”
  - N entscheidet A
    - da $w \in A \Leftrightarrow f(w) \in B$
  - N berechnet A in Polynom-Zeit:
    - s.o.





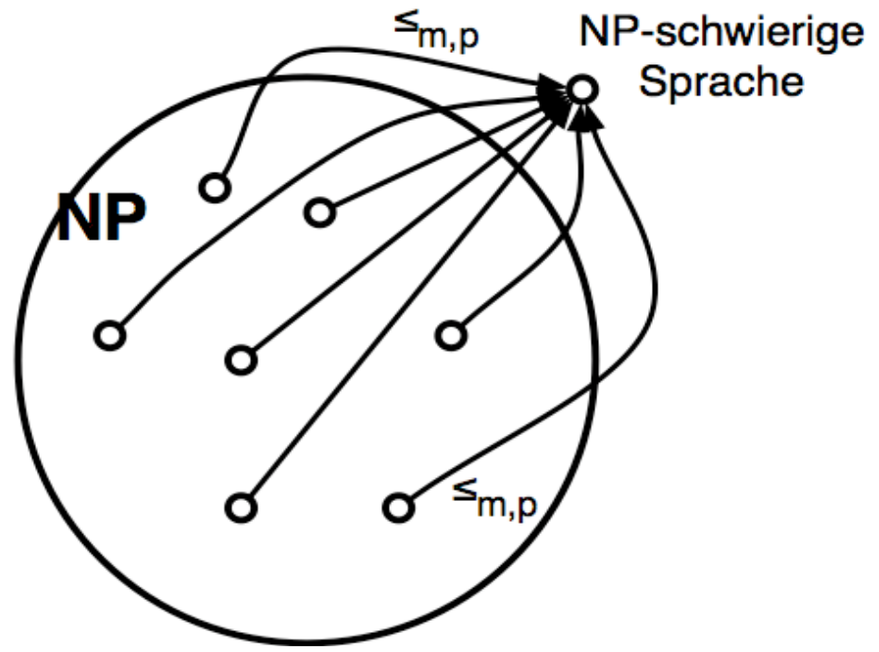
Transitivität von Abbildungsreduktion

- **Lemma:**
  - Aus $A \leq_{m,p} B$ und $B \leq_{m,p} C$ folgt $A \leq_{m,p} C$.
- **Beweis:**
  - Sei f die Reduktionsfunktion von A nach B
  - Sei g die Reduktionsfunktion von B nach C
  - Dann ist $g \circ f$ die Reduktionsfunktion von A nach C.
  - Korrektheit:
    - $x \in A \Leftrightarrow f(x) \in B \Leftrightarrow g(f(x)) \in C$
  - Laufzeit:
    - f(x) kann in Zeit $O(|x|^k)$ berechnet werden
    - g(y) kann in Zeit $O(|y|^{k'})$ berechnet werden
    - $|f(x)| \leq |x|^k$
    - Damit hat g(f(x)) die Laufzeit $O(|f(x)|^{k'}) = O(|x|^{k\,k'})$

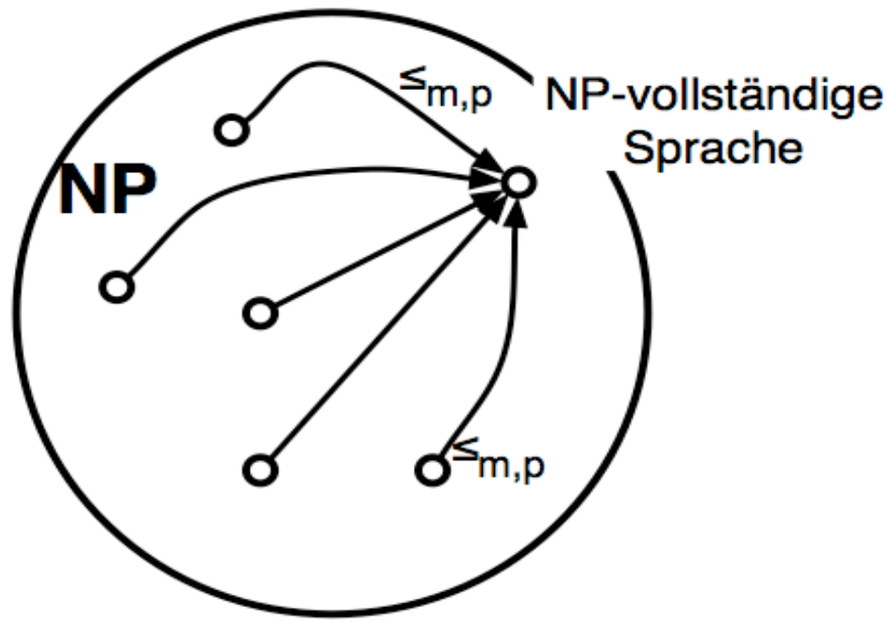


NP-Vollständigkeit

- **Definition:**
  - Eine Sprache S ist **NP-vollständig**, wenn:
    - S ∈ NP
    - S ist **NP-schwierig**, d.h. für alle L ∈ NP: $L \leq_{m,p} S$



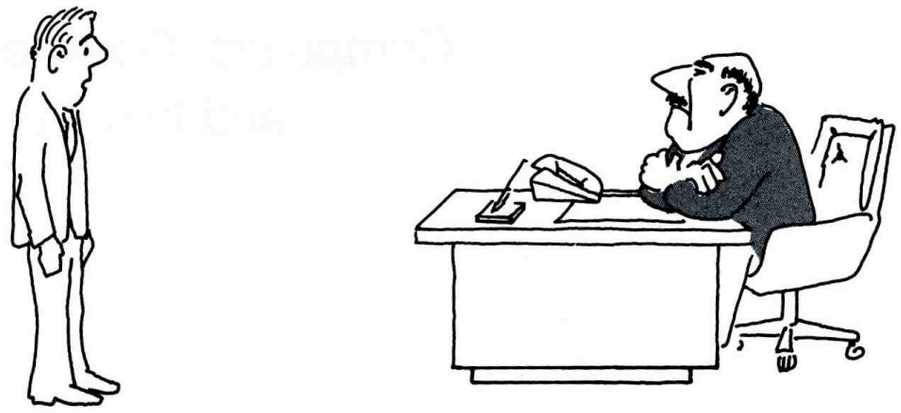
- **Lemma:**
  - Sei S eine NP-vollständige Sprache.
  - Dann ist eine Sprache T ist **NP-vollständig**, wenn:
    - T ∈ NP
    - $S \leq_{m,p} T$



NP-Vollständigkeit

- **Für ein unbekanntes NP-Problem X, sollte man**
  - nicht nur nach einem Algorithmus mit polynomieller Laufzeit forschen
    - $X \in P$
  - sondern auch nach einem NP-Vollständigskeitsbeweis
- **Beweisideen nicht unbedingt naheliegend …**
- **Beispiele für NP-Vollständigkeitsbeweise**
  - VERTEX-COVER ist NP-vollständig
  - (U)HAMPATH ist NP-vollständig
  - SUBSET-SUM ist NP-vollständig

specifications, and the bandersnatch department is already 13 components behind schedule. You certainly don't want to return to his office and report:



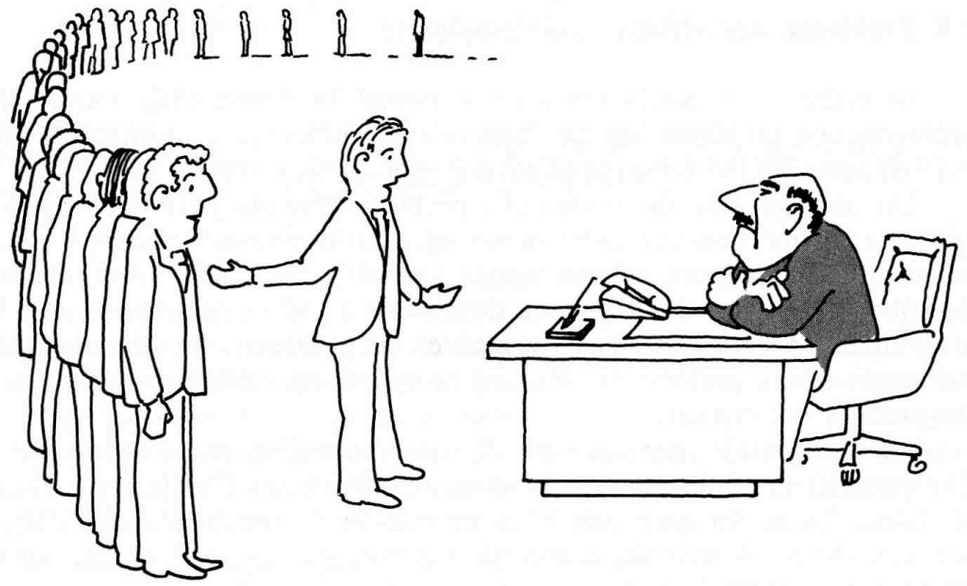
"I can't find an efficient algorithm, I guess I'm just too dumb."

To avoid serious damage to your position within the company, it would be much better if you could prove that the bandersnatch problem is *inherently* intractable, that no algorithm could possibly solve it quickly. You then could stride confidently into the boss's office and proclaim:

"I can't find an efficient algorithm, because no such algorithm is possible!"

Unfortunately, proving inherent intractability can be just as hard as finding efficient algorithms. Even the best theoreticians have been stymied in their attempts to obtain such proofs for commonly encountered hard problems. However, having read this book, you have discovered something

almost as good. The theory of NP-completeness provides many straightforward techniques for proving that a given problem is "just as hard" as a large number of other problems that are widely recognized as being difficult and that have been confounding the experts for years. Armed with these techniques, you might be able to prove that the bandersnatch problem is NP-complete and, hence, that it is equivalent to all these other hard problems. Then you could march into your boss's office and announce:



"I can't find an efficient algorithm, but neither can all these famous people."

At the very least, this would inform your boss that it would do no good to fire you and hire another expert on algorithms.

Of course, our own bosses would frown upon our writing this book if its sole purpose was to protect the jobs of algorithm designers. Indeed, discovering that a problem is NP-complete is usually just the beginning of work on that problem. The needs of the bandersnatch department won't disappear overnight simply because their problem is known to be NP-complete. However, the knowledge that it is NP-complete does provide valuable information about what lines of approach have the potential of being most productive. Certainly the search for an efficient, exact algorithm should be accorded low priority. It is now more appropriate to concentrate on other, less ambitious, approaches. For example, you might look for efficient algorithms that solve various special cases of the general problem. You might look for algorithms that, though not guaranteed to run quickly, seem likely to do so most of the time. Or you might even relax the problem somewhat, looking for a fast algorithm that merely finds designs that

Komplexitätstheorie

3-SAT und Clique

Wiederholung Boolesche Algebra

- **Eine Boolesche Variable ist entweder falsch (0) oder wahr (1)**
- **Als Operationen für Boolesche Variablen verwenden wir**
  - die Disjunktion (Oder): $x \vee y$
    - $0 \vee 0 = 0$
    - $0 \vee 1 = 1$
    - $1 \vee 0 = 1$
    - $1 \vee 1 = 1$
  - die Konjunktion (Und): $x \wedge y$
    - $0 \wedge 0 = 0$
    - $0 \wedge 1 = 0$
    - $1 \wedge 0 = 0$
    - $1 \wedge 1 = 1$
  - die Negation $\neg x = x$
    - $\neg 0 = 1$
    - $\neg 1 = 0$
- **Eine Boolesche Formel besteht aus der Kombination dieser Operationen**
  - $f(x,y) = (x \vee y) \wedge (\neg x \vee \neg y)$
- **Rechenregeln:**
  - Doppelte Negation
    - $\neg \neg x = x$
  - Kommuntativ-Gesetz
    - $x \wedge y = y \wedge x$
    - $x \vee y = y \vee x$
  - Distributiv-Gesetz
    - $(x \vee y) \wedge z = (x \wedge z) \vee (x \wedge z)$
    - $(x \wedge y) \vee z = (x \vee z) \wedge (x \vee z)$
  - De-Morgan-Regeln
    - $\neg (x \vee y) = \neg x \wedge \neg y$
    - $\neg (x \wedge y) = \neg x \vee \neg y$

Besondere Boolesche Funktionen

- **Literal:**
  - ist eine einfache oder negierte Variable
  - z.B.: x, y, z, ¬x, ¬y, ¬z
- **Klausel:**
  - ist eine Disjunktion von Literalen
  - z.B.:
    - x ∨ y
    - z ∨ ¬x ∨ ¬y
    - x ∨ ¬z
- **Konjunktive Normalform (CNF)**
  - Konjunktion von Klauseln
  - z.B.:
    - (x ∨ y) ∧ (z ∨ ¬x ∨ ¬y) ∧ (x ∨ ¬z)
- **k-konjunktive Normalform (k-CNF)**
  - Konjunktion von Klauseln aus k Literalen, z.B. 3-CNF
    - (x ∨ y ∨ y) ∧ (z ∨ ¬x ∨ ¬y) ∧ (x ∨ ¬z ∨ ¬z)
- **Disjunktive Normalform (DNF)**
  - Disjunktion aus Konjunktion von Literalen
    - (x ∧ y) ∨ (z ∧ ¬x ∧ ¬y) ∨ (x ∧ ¬z)

Das Erfüllbarkeitsproblem Boolescher Funktionen

- **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 (Satisfiability Problem of Boolean Formulas)**
  - Das Erfüllbarkeitsproblem der Booleschen Funktion ist definiert als:
  - SAT = { φ | φ ist eine erfüllbare Funktion}, d.h.
  - Gegeben:
    - Boolesche Funktion φ
  - Gesucht:
    - Gibt es $x_1,x_2,..,x_n$ so dass φ(x1,x2,..,xn) = 1
- **Spezialfall: 3-SAT**
  - 3-SAT = { φ | φ ist eine erfüllbare Funktion in 3-CNF}
    z.B.:

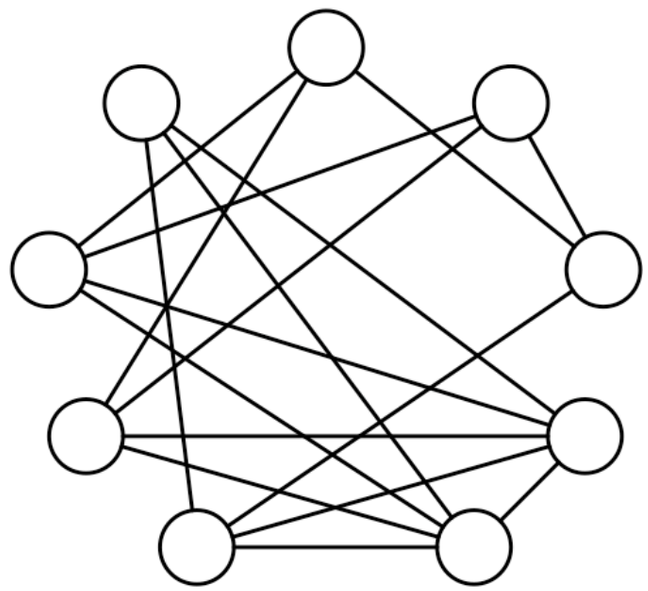
$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$

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?

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

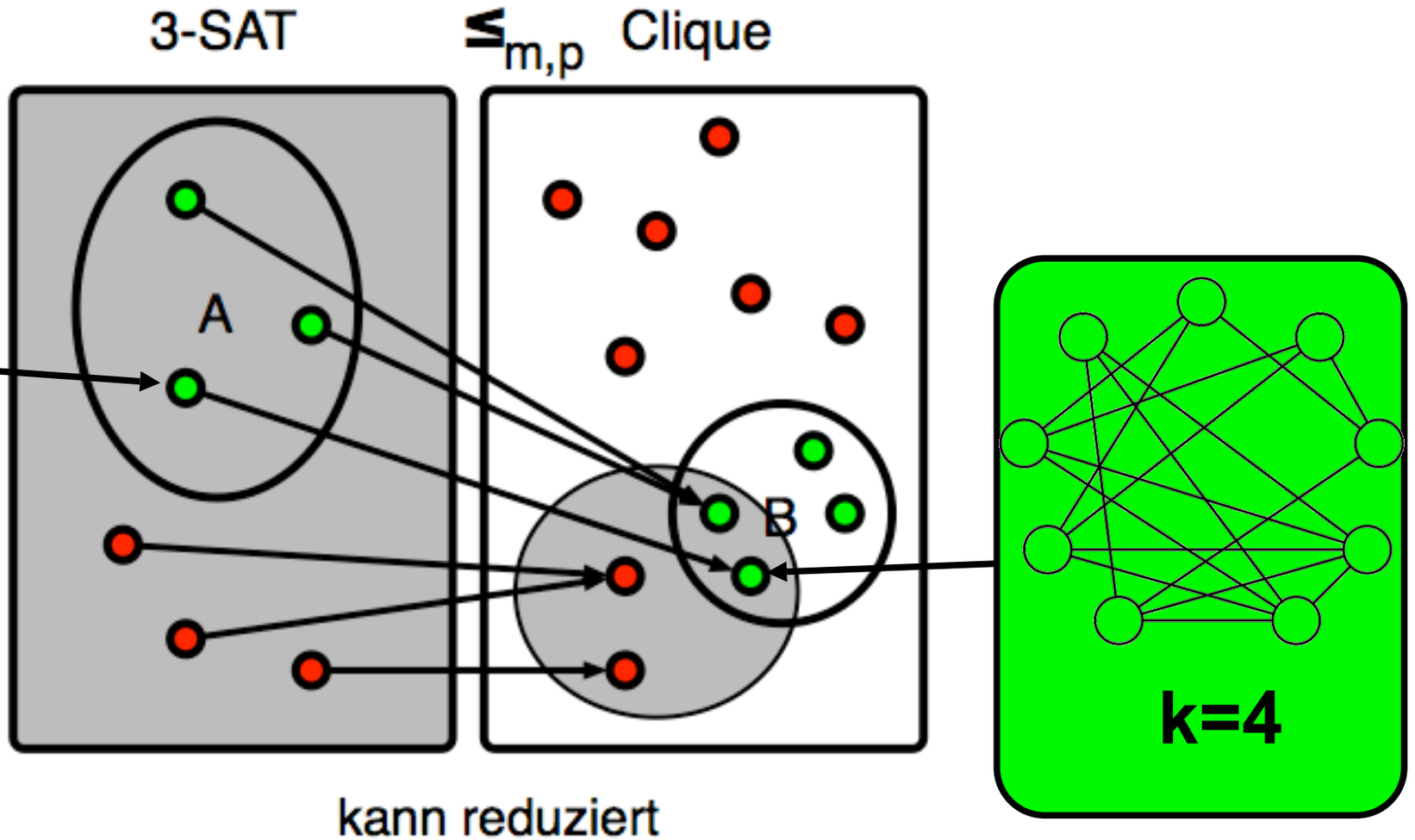
k=4

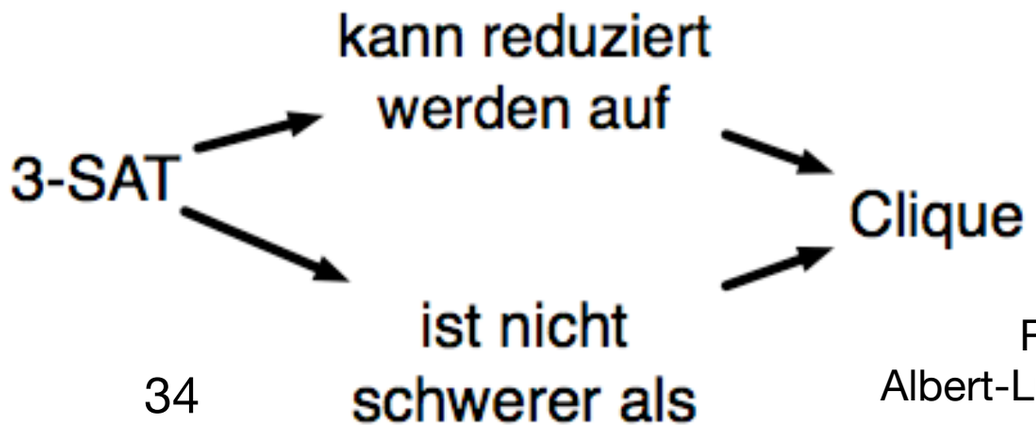


3-SAT läßt sich auf Clique reduzieren

- **Theorem: 3-SAT ≤m,p CLIQUE**

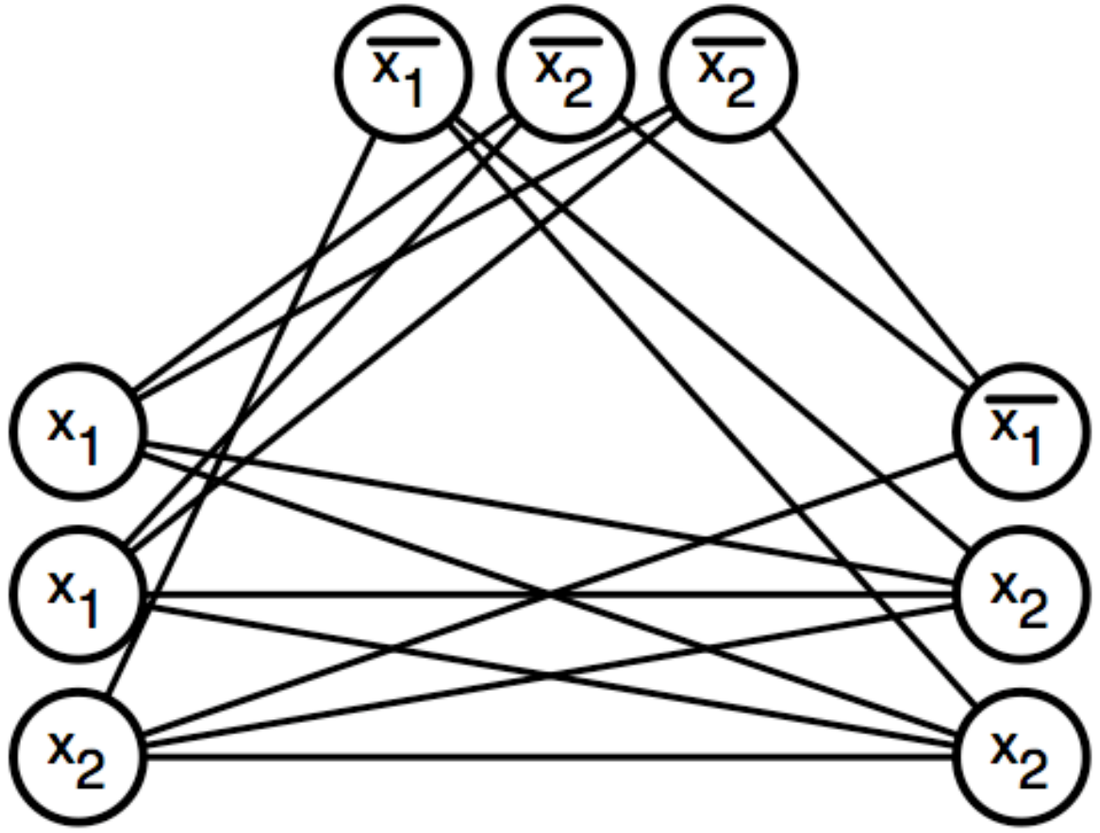
$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



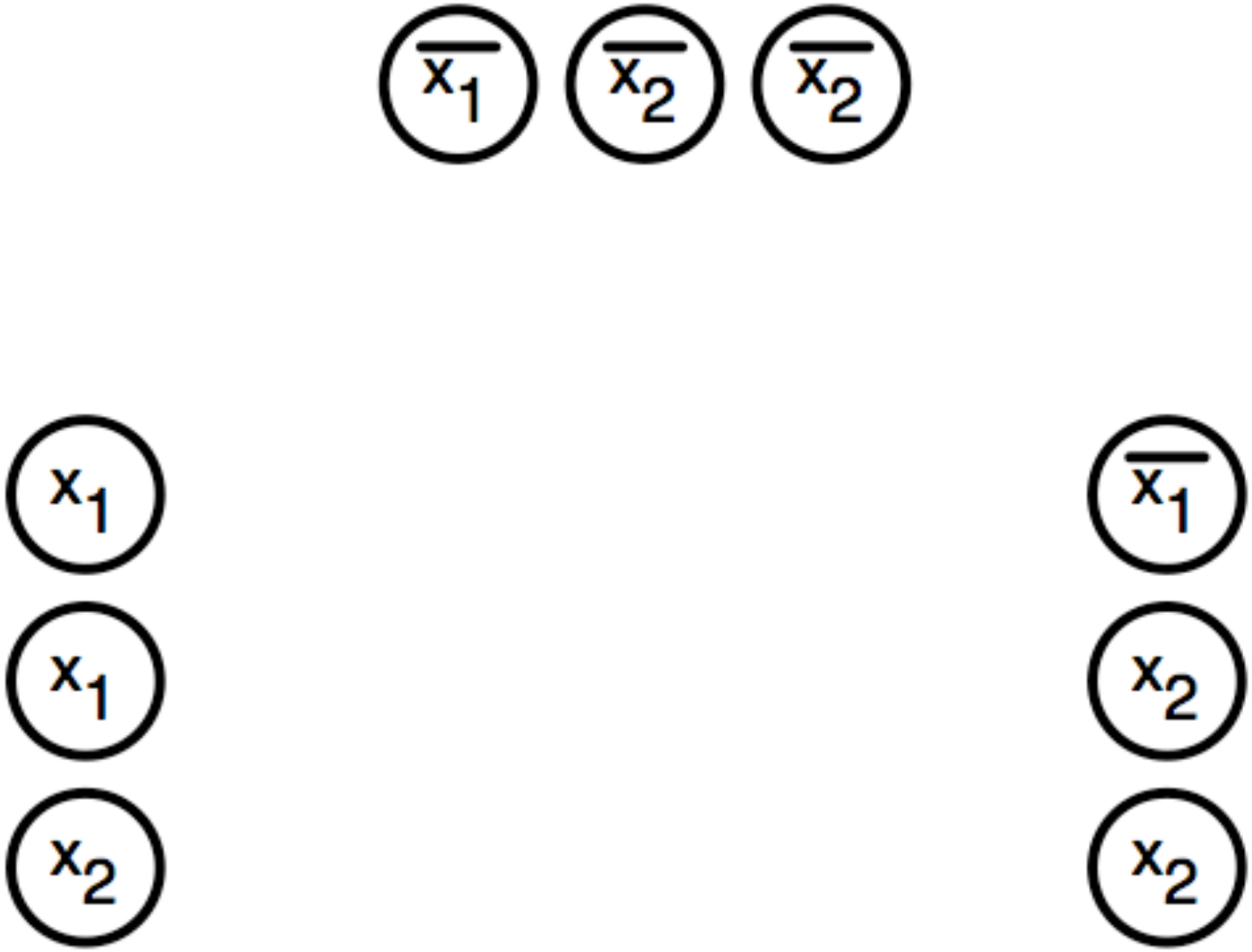


3-SAT läßt sich auf Clique reduzieren

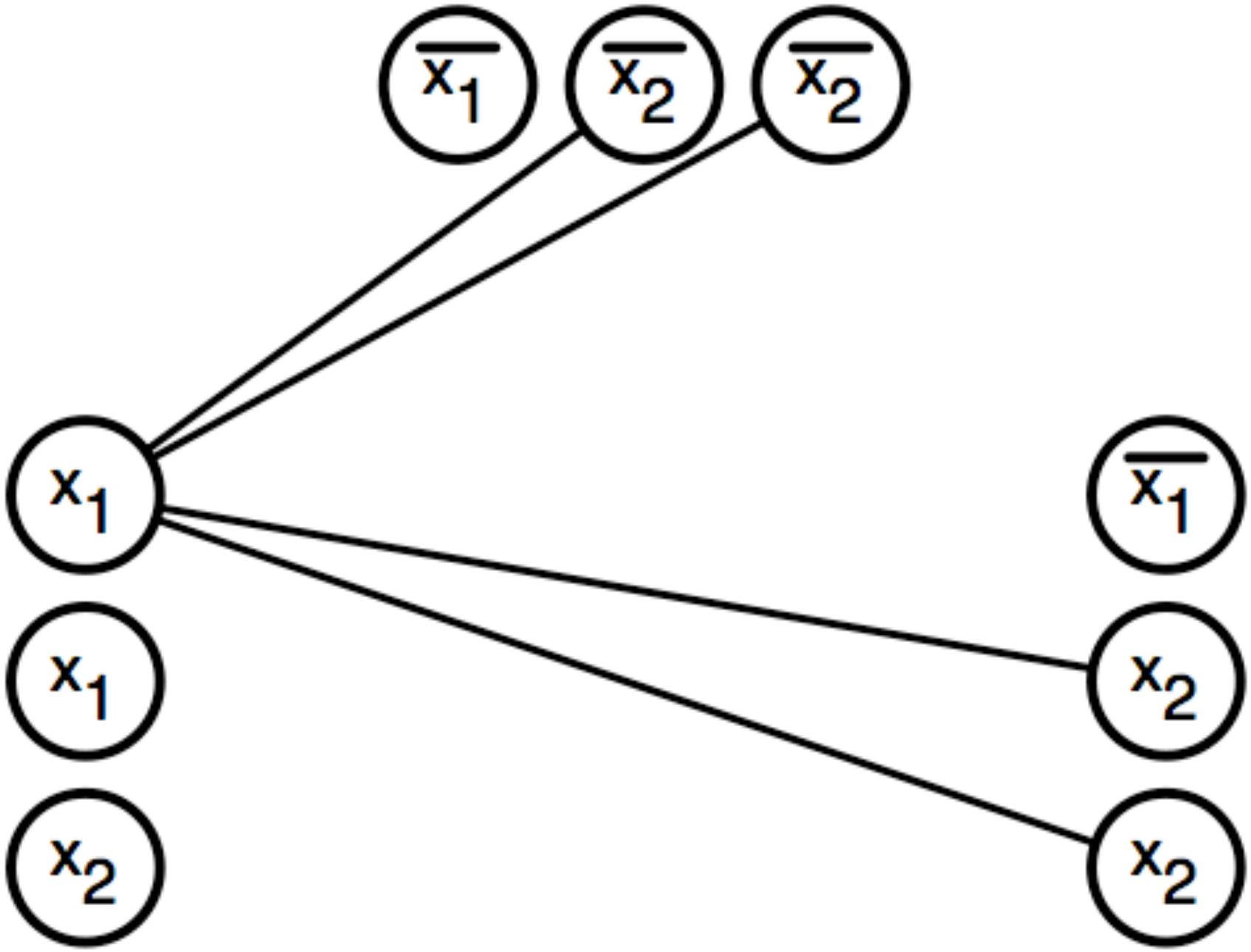
- **Theorem: 3-SAT ≤m,p CLIQUE**
- **Beweis**
  - Konstruiere Reduktionsfunktion f:
  - f(φ) = <G,k>
  - k = Anzahl der Klauseln
  - Für jede Klausel C in φ werden drei Knoten angelegt, die mit den Literalen der Klausel bezeichnet werden
  - Füge Kante zwischen zwei Knoten ein, gdw.
    - die beiden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



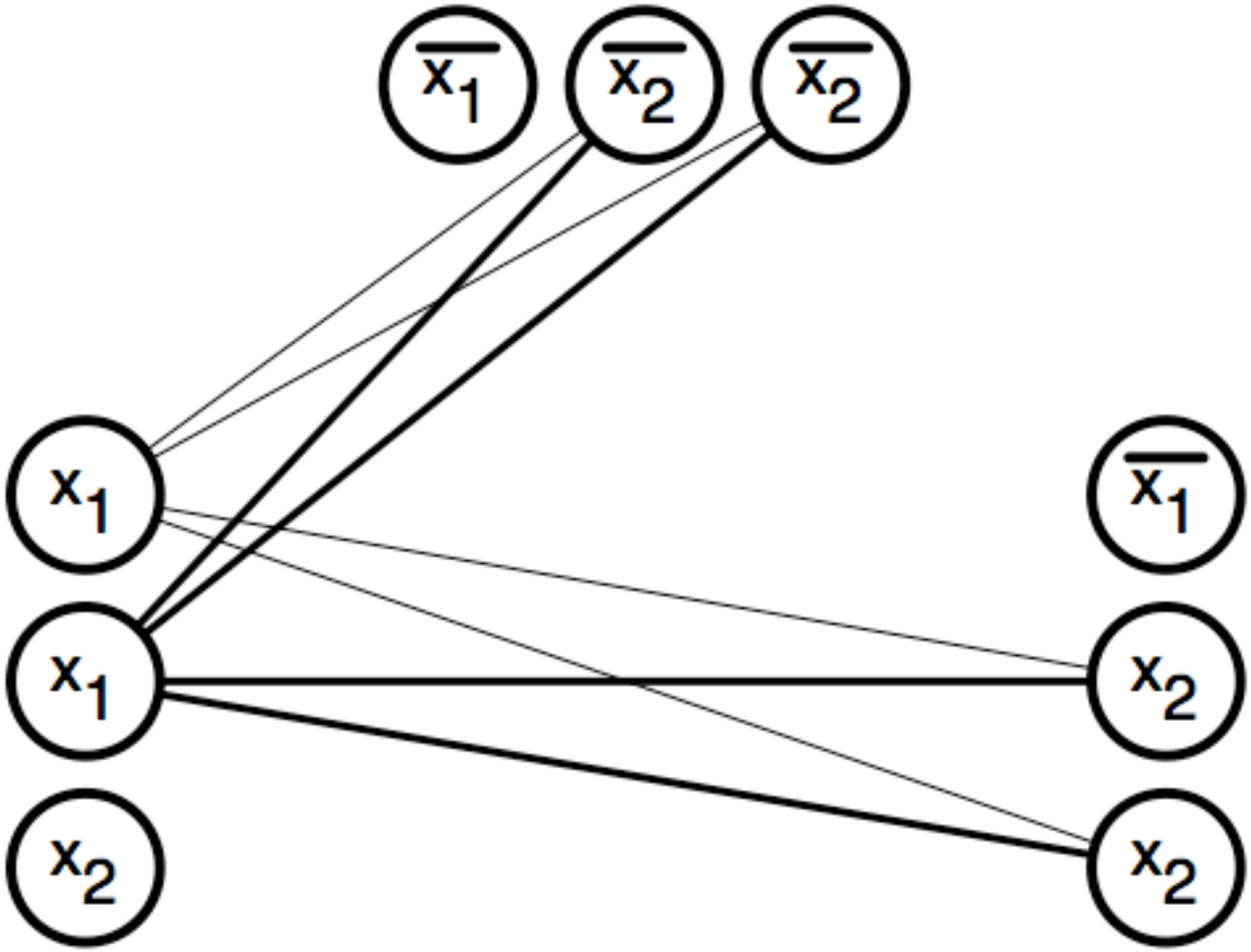
$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



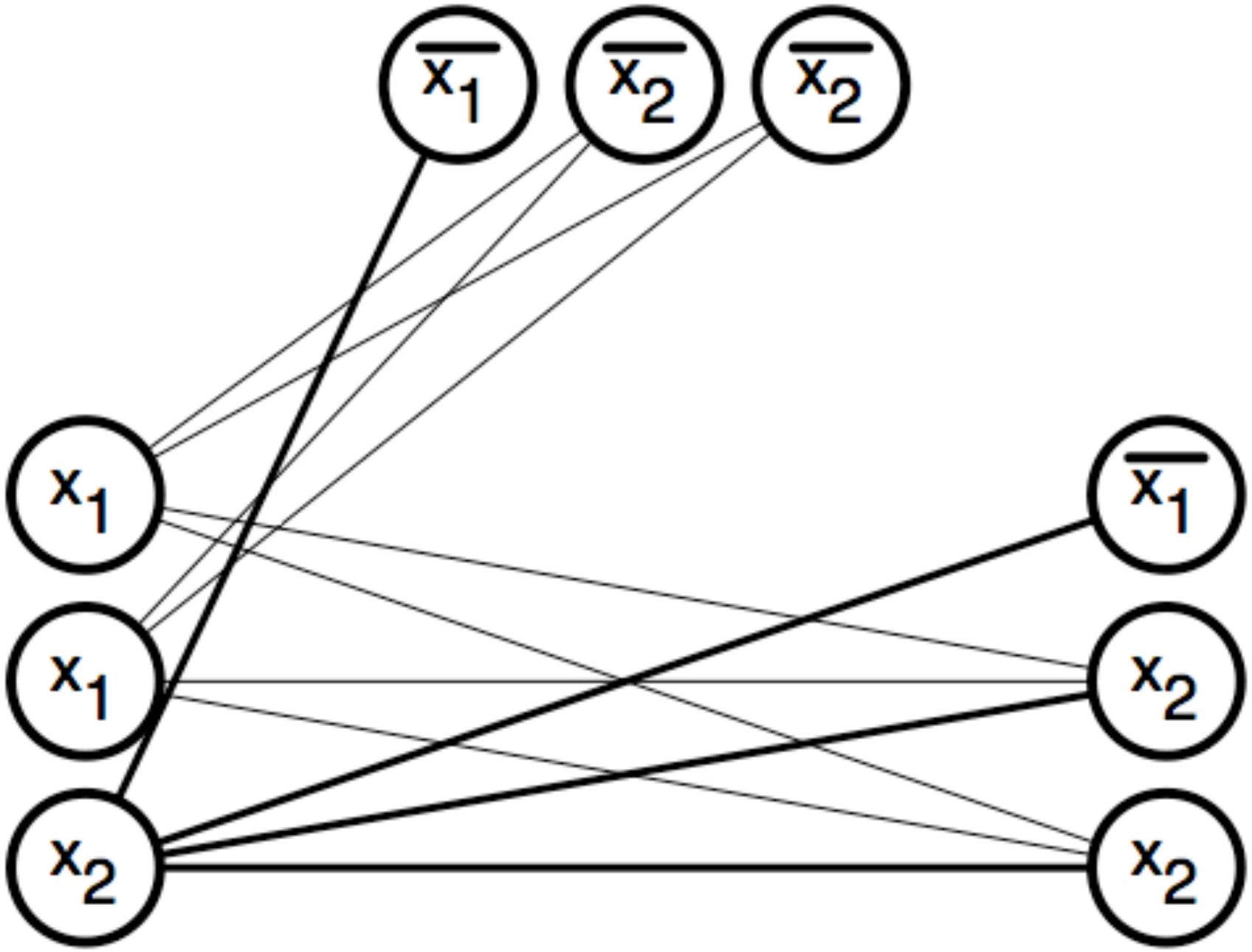
$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



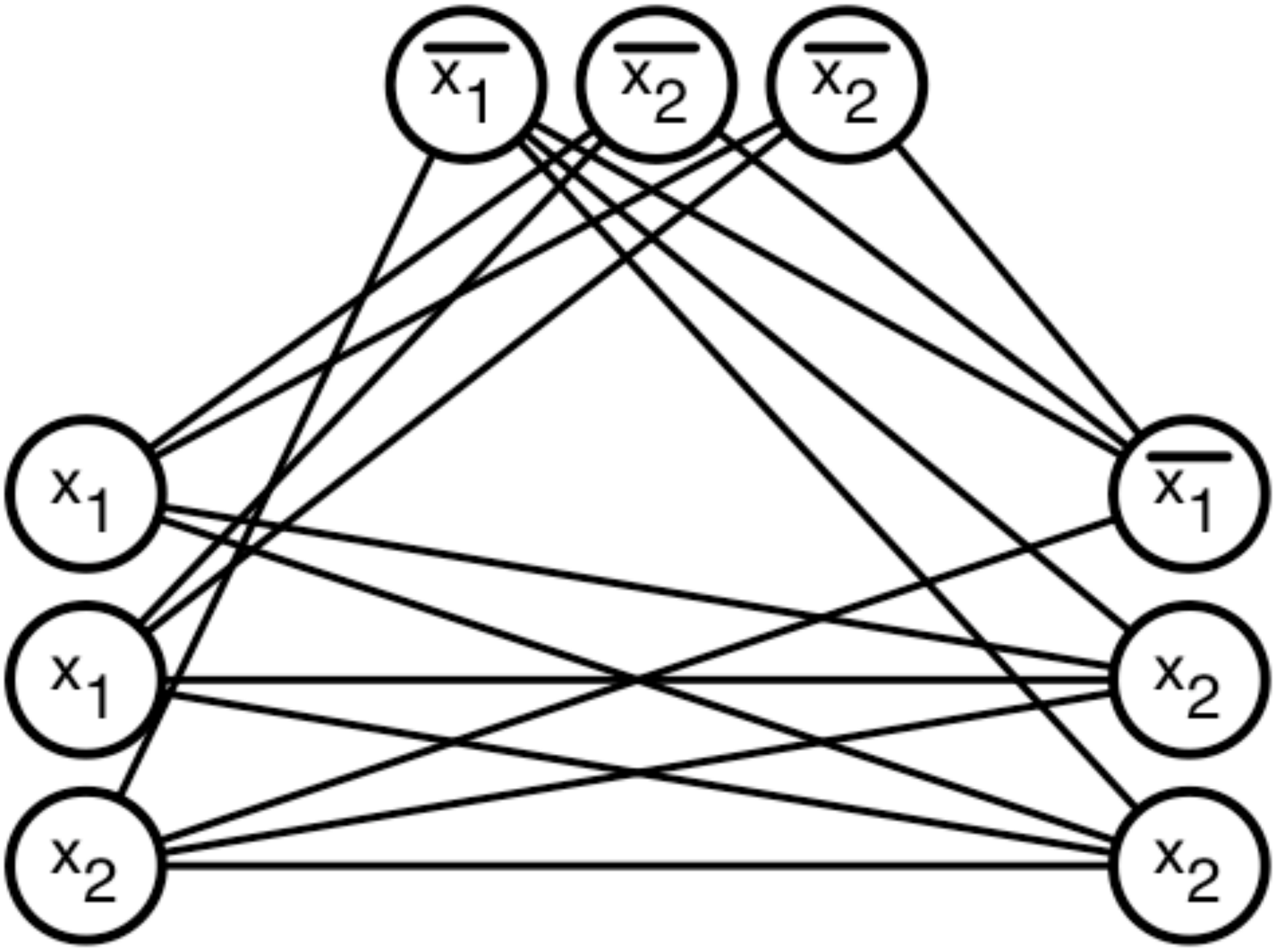
$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



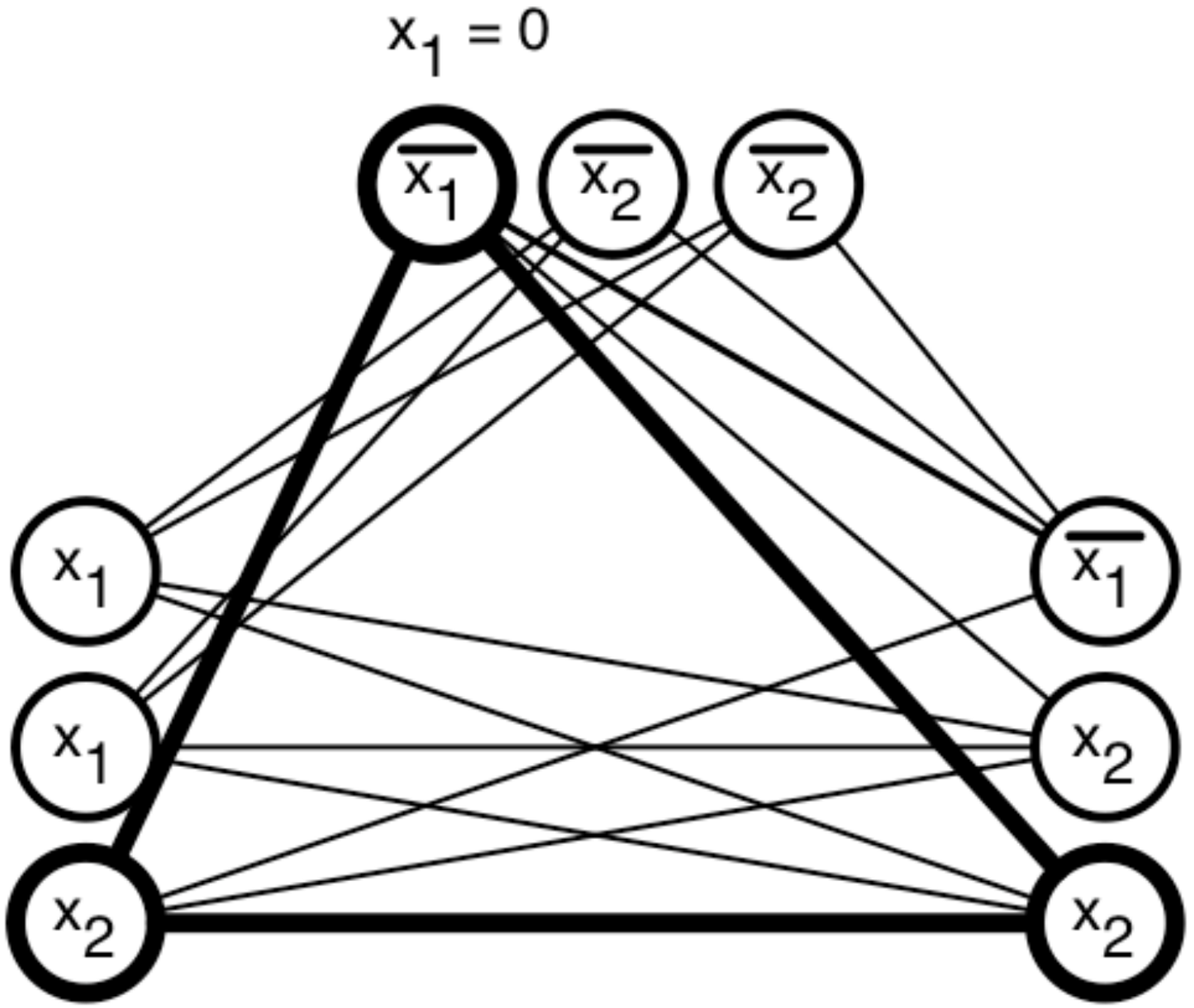
$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



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



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



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

0	0	**1**	**1**	0	0	**1**	**1**	**1**

Beweis der Korrektheit der Reduktionsfunktion

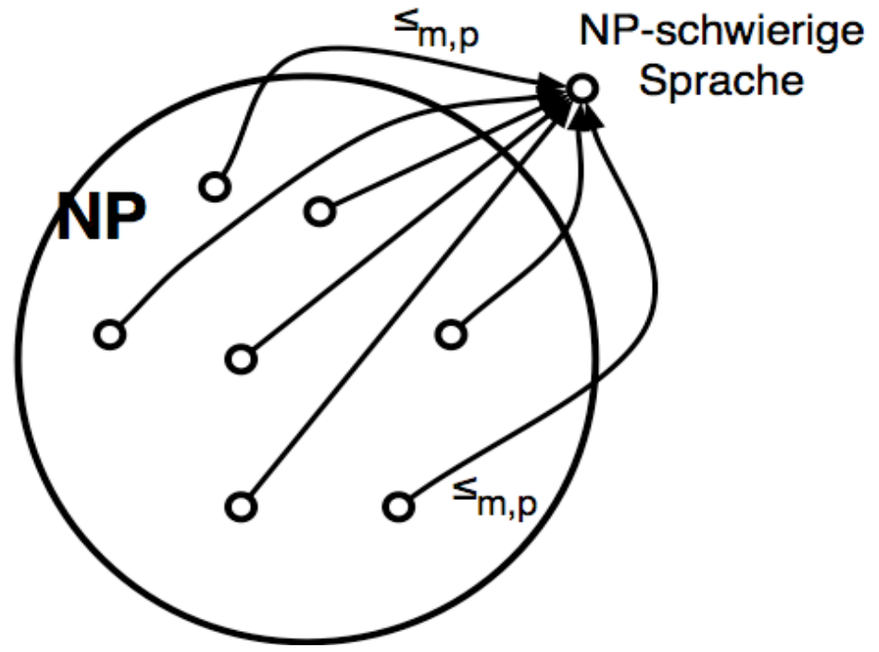
- **Die Reduktionsfunktion ist korrekt:**
- **Behauptung;**
  - Eine erfüllende Belegung in φ existert gdw. eine k-Clique in G existiert
- **1. Fall: eine erfüllende Belegung existiert in φ**
  - Dann liefert die Belegung in jeder Klausel mindestens ein Literal mit Wert 1
  - Wähle aus der Knotenmenge 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.

Komplexitätstheorie

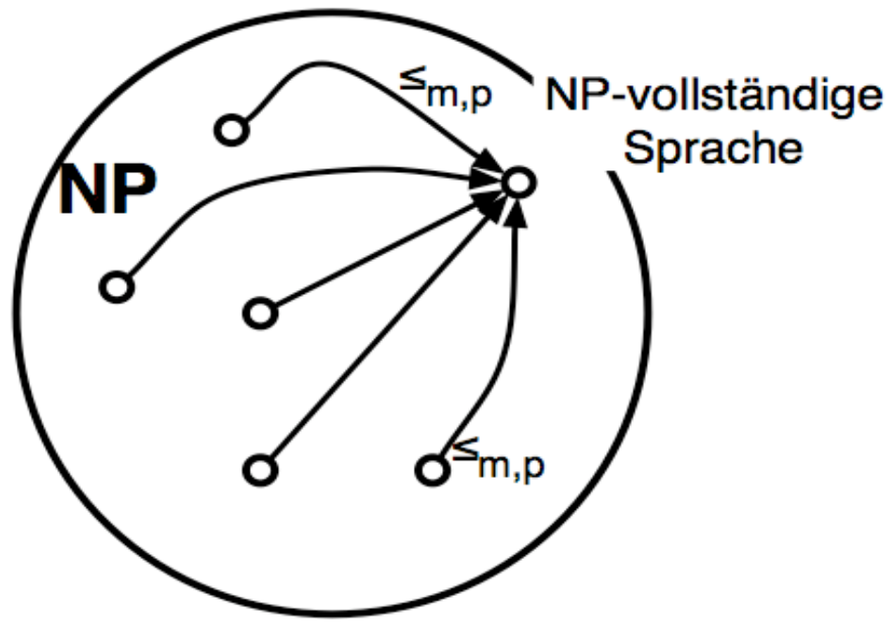
SAT ist NP-vollständig

NP-Vollständigkeit

- **Definition:**
  - Eine Sprache S ist **NP-vollständig**, wenn:
  - $S \in NP$
  - S ist **NP-schwierig**, d.h. für alle $L \in NP$: $L \leq_{m,p} S$

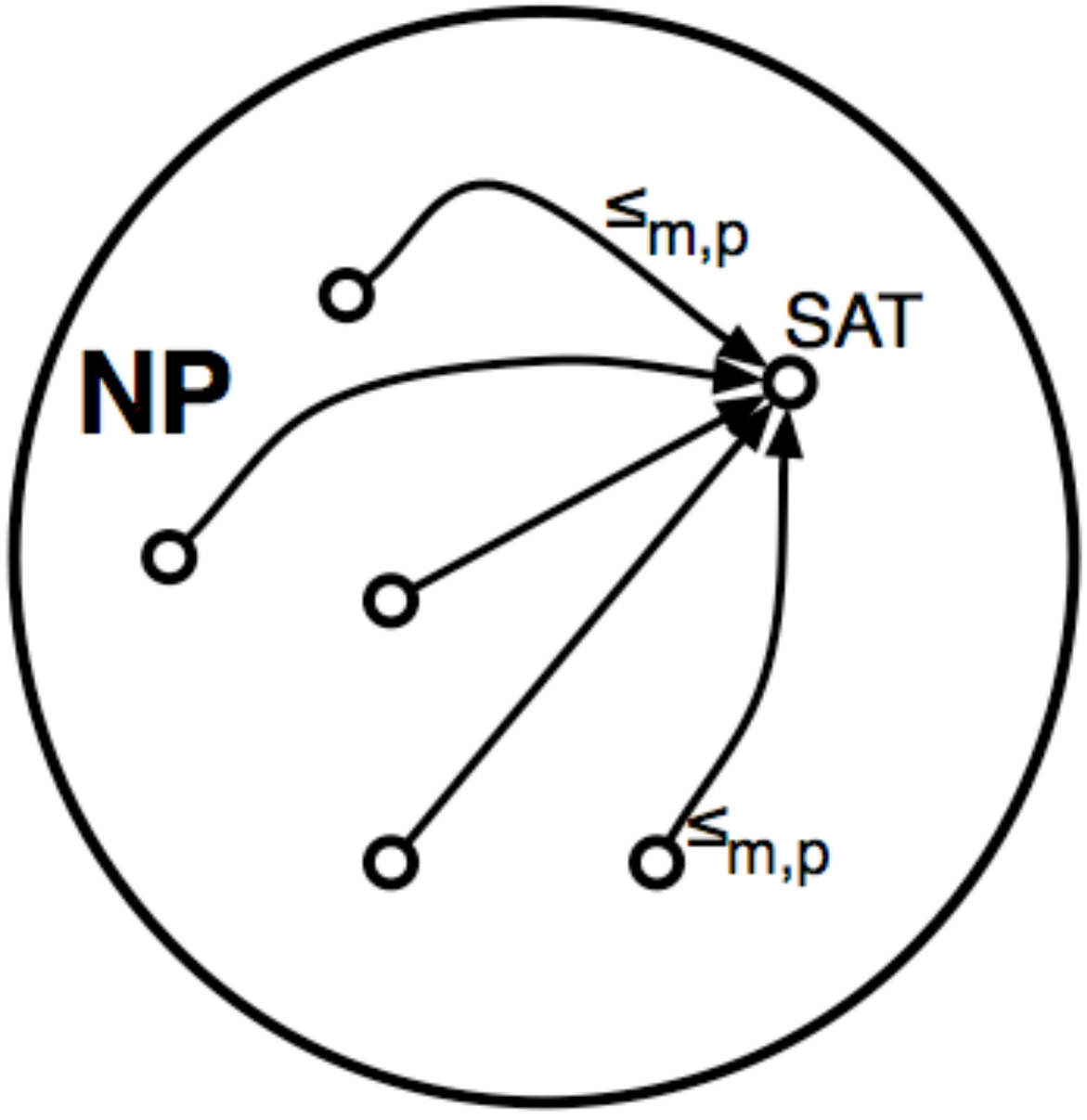


- **Lemma:**
  - Sei S eine NP-vollständige Sprache.
  - Dann ist eine Sprache T ist **NP-vollständig**, wenn:
    - $T \in NP$
    - $S \leq_{m,p} T$



Der Satz von Cook und Levin

- **Theorem (Cook/Levin)**
  - SAT ist NP-vollständig



Konsequenzen aus dem Satz von Cook und Levin

- **1. Fall: SAT $\in$ P:**
  - SAT ist NP-vollständig:
    - Für alle L $\in$ NP: L $\leq$m,p SAT
  - Daraus folgt für alle L $\in$ NP: L $\in$ P
  - Damit ist P=NP
- **2. Fall: SAT $\notin$ P:**
  - Damit existiert eine Sprache in NP, die nicht in P ist
  - Damit ist P≠NP
- **Also folgt aus dem Satz von Cook und Levin:**
  - SAT $\in$ P $\Leftrightarrow$ P = NP

Beweis-Strategie

- **Definition:**
  - Eine Sprache S ist NP-vollständig, wenn:
    - S $\in$ NP
    - S ist NP-schwierig
      - d.h. für alle L $\in$NP: L $\leq_{m,p}$ S
- **Theorem (Cook/Levin)**
  - SAT ist NP-vollständig

- **Beweis-Idee:**
  - Zuerst zeigen wir, dass SAT $\in$ NP
  - Sei L $\in$ NP. Wir beweisen folgende Reduktionen:
    - Es gibt ein k, so dass $L \leq_{m,p} A_{NTIME(n^k)}$
    - Für alle k: $A_{NTIME(n^k)} \leq_{m,p} A_{NTIME(n)}$
    - $A_{NTIME(n)} \leq_{m,p}$ PARKETT
    - PARKETT $\leq_{m,p}$ FinPred
    - FinPred $\leq_{m,p}$ SAT
  - Daraus folgt für alle L $\in$ NP: $L \leq_{m,p}$ SAT

SAT ∈ NP

- **Folgende NTM löst das SAT-Problem:**
  - "Auf Eingabe der Funktion $\Phi(x_1,x_2,...,x_n)$
  - Rate nichtdeterministisch eine Belegungen aller Variablen $x_1,x_2,...,x_n$
  - Setze diese Belegung ein
  - Falls $\Phi(x_1,x_2,...,x_n)$ =1 akzeptiere
  - Sonst, verwerfe"
- **Laufzeit:**
  - Die Laufzeit der NTM ist polynomiell, da Boolesche Funktion in Linearzeit ausgewertet werden können
- **Korrektheit**
  - Falls die Formel erfüllbar ist, wird die Belegung durch "Nichtdeterministisches Raten" gefunden
  - Ist die Formel nicht erfüllbar, wird niemals akzeptiert
- **Damit ist SAT als erstes NP-vollständiges Problem identifiziert worden.**

Es gibt ein k, so dass $L \leq_{m,p} A_{NTIME(n^k)}$

- **Definition**
  - Das Wortproblem von NTIME(n^k):
  - $A_{NTIME(n^k)}$ = { <M,x> | NTM M akzeptiert $x \in \Sigma^n$ in Laufzeit n^k }
- **Lemma**
  - Für alle $L \in NP$ gibt es ein $k \in \mathbf{N}$, so dass $L \leq_{m,p} A_{NTIME(n^k)}$
- **Beweis**
  - Sei $L \in NTIME(n^k)$, dann gibt es eine NTM M die in Laufzeit n^k jede Eingabe $x \in \Sigma^n$ akzeptiert.
  - Definiere Reduktionsfunktion: f(x) = <M,x>, für diese NTM M
  - Korrektheit:
    - $x \in L \Leftrightarrow$ M akzeptiert in Laufzeit n^k die Eingabe x $\Leftrightarrow <M,x> \in A_{NTIME(n^k)}$
  - Reduktionsfunktion hat Laufzeit: $|x| + O(1)$
    - Da die Eingabe mit der (konstanten) Maschine M kombiniert werden muss.

Für alle k≥1 : $A_{NTIME(n^k)} \leq_{m,p} A_{NTIME(n)}$

- **Definition**
  - Das Wortproblem von NTIME(n^k):
  - $A_{NTIME(n^k)}$ = { <M,x> | NTM M akzeptiert $x \in \Sigma^n$ in Laufzeit n^k }
- **Lemma**
  - Für alle k ≥1 : $A_{NTIME(n^k)} \leq_{m,p} A_{NTIME(n)}$
- **Beweis**
  - Betrachte h(M) = M':
    - modifiziert NTM M zu einer NTM M' mit gleicher Laufzeit und Akzeptanzverhalten nur dass das Zeichen "#" wie "_" behandelt wird
  - Reduktionsfunktion:

    f(<M,x>) = <h(M),x $\#^{n^k-n}$>

- **Laufzeit auf Eingabe der Länge n ist $O(n + n^k) = O(n^k)$**
- **Korrektheit:**
  - <M,x> $\in A_{NTIME(n^k)}$

    $\Leftrightarrow$ M akzeptiert in Laufzeitn^k die Eingabe x

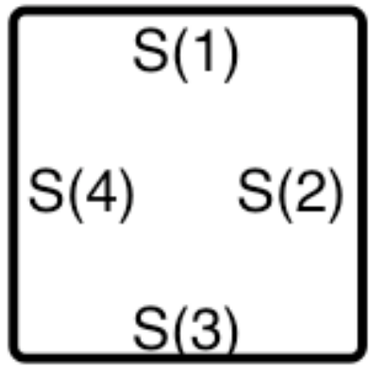
    $\Leftrightarrow$ M' akzeptiert in Laufzeit n^k die Eingabe x $\#^{n^k-n}$

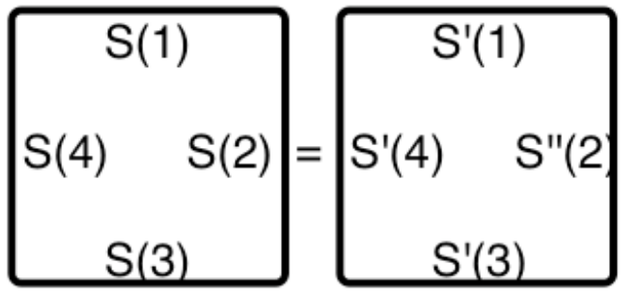
    $\Leftrightarrow$ M' akzeptiert in Laufzeit $n' = n^k$ die Eingabe x $\#^{n^k-n}$ mit Länge n'

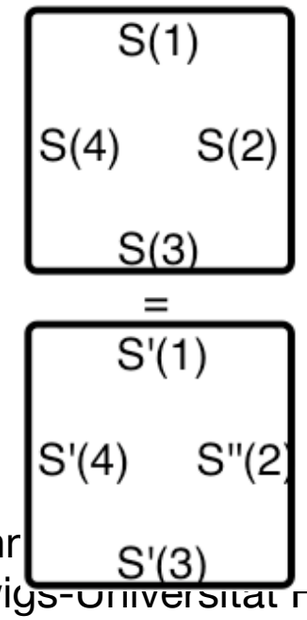
    $\Leftrightarrow$ <M' x $\#^{n^k-n}$> $\in A_{NTIME(n)}$

Das Parkett-Problem

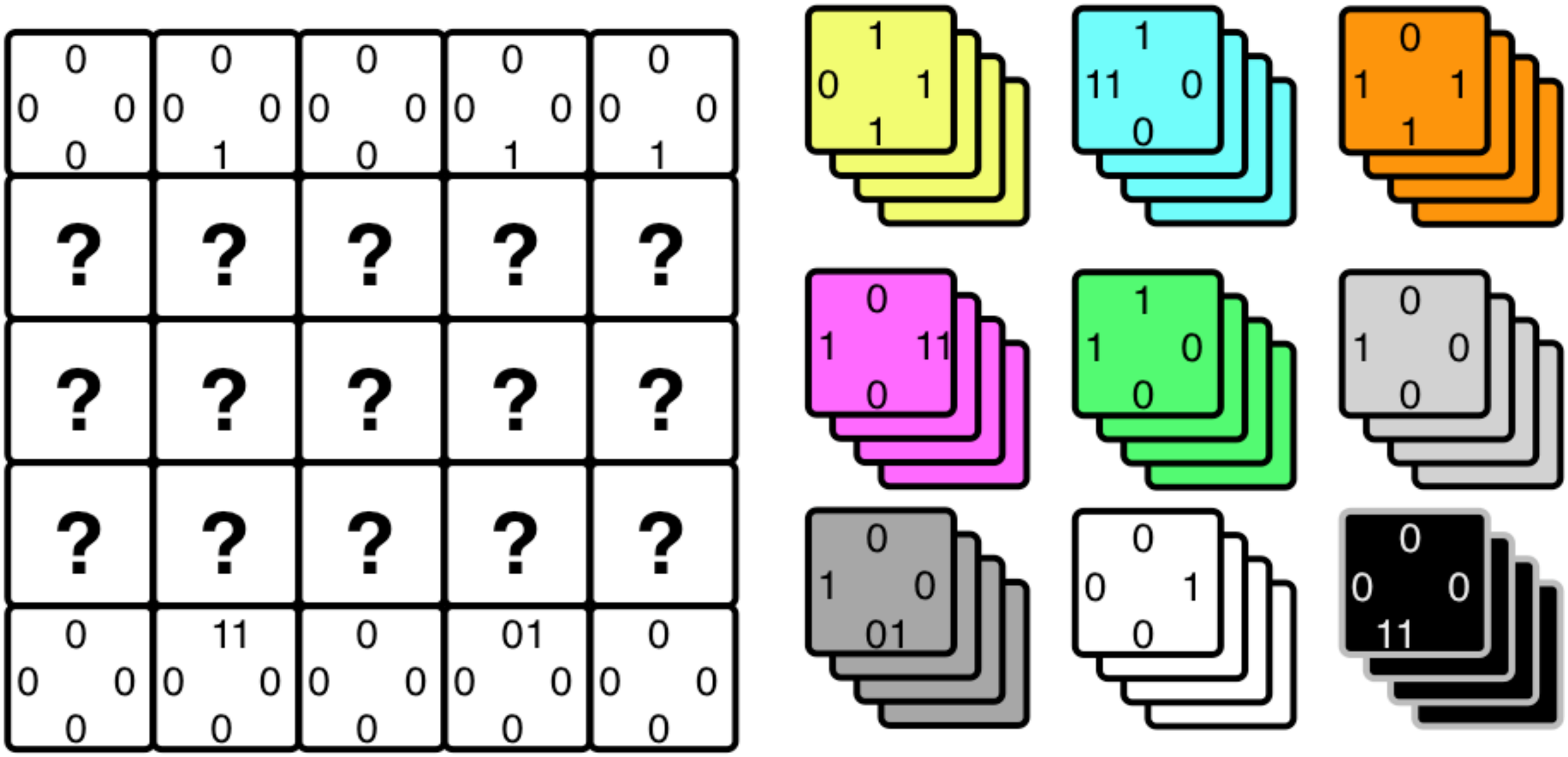
- **Definition**
  - Das Parkettproblem ist das Problem, eine gegebene quadratische Fläche mit Parkettstücken verschiedener Form lückenlos abzudecken
  - Auch bekannt als zweidimensionales Domino-Problem, also:
    - Parkett-Teil S: 4-Tupel (S(1), S(2), S(3), S(4)) mit S(i) ∈ {0,1}*
    - H(S,S') := S und S' passen horizontal, d.h. S(2)=S'(4)
    - V(S,S') := S und S' passen vertikal, d.h. S(3)=S'(1)
  - Gegeben:
    - Menge von Bauteilen M = $\{B_1,B_2,B_3,..., B_k\}$
    - Oberer Rand $S_{1,1}$, ..., $S_{1,m}$
    - Unterer Rand $S_{m,1}$, ..., $S_{m,m}$
  - Gesucht:
    - Gibt es eine Menge von Bauteilen $S_{i,j}$, für i ∈ {1,..,m} , j ∈ {2,..,m-1} die passen, d.h.
      - $H(S_{i,j} ,S_{i,j+1})$ für alle i ∈ {1,..,m}, j ∈ {1,..,m-1} und
      - $V(S_{i,j} ,S_{i+1,j})$ für alle i ∈ {1,..,m-1}, j ∈ {1,..,m}.





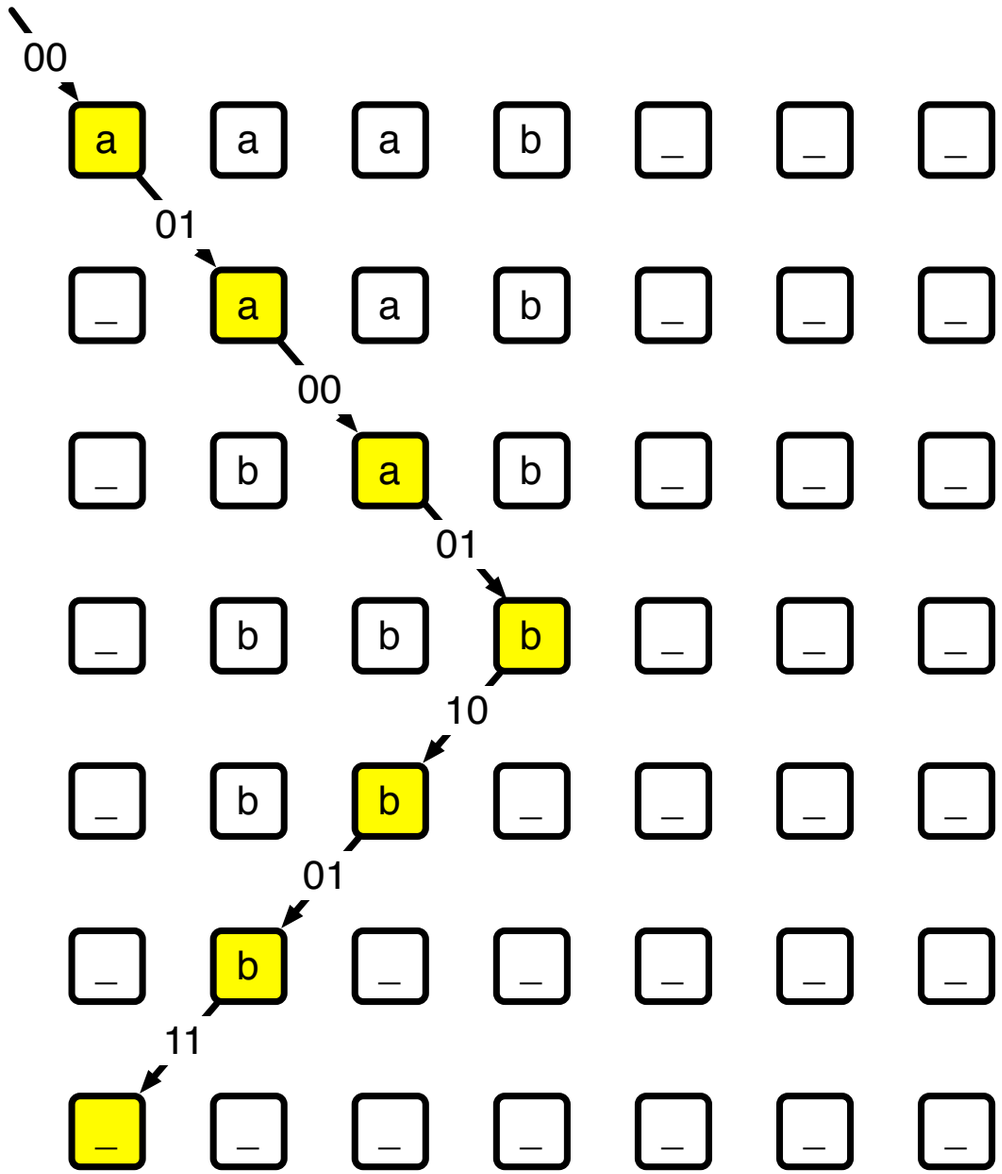


Das Parkett-Problem

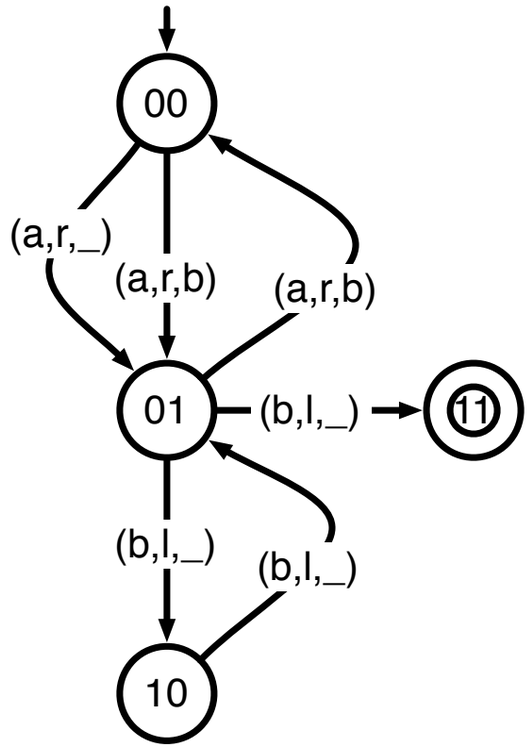


$A_{NTIME(n)} \leq_{m,p}$ PARKETT

- **Betrachte NTM M**
  - die nach der Berechnung das gesamte Band mit _ beschreibt
  - mit eindeutigen akzeptierenden Zustand q_{akz}
  - und die am linken Rand der Eingabe akzeptiert



Akzeptiert auf leerer Eingabe



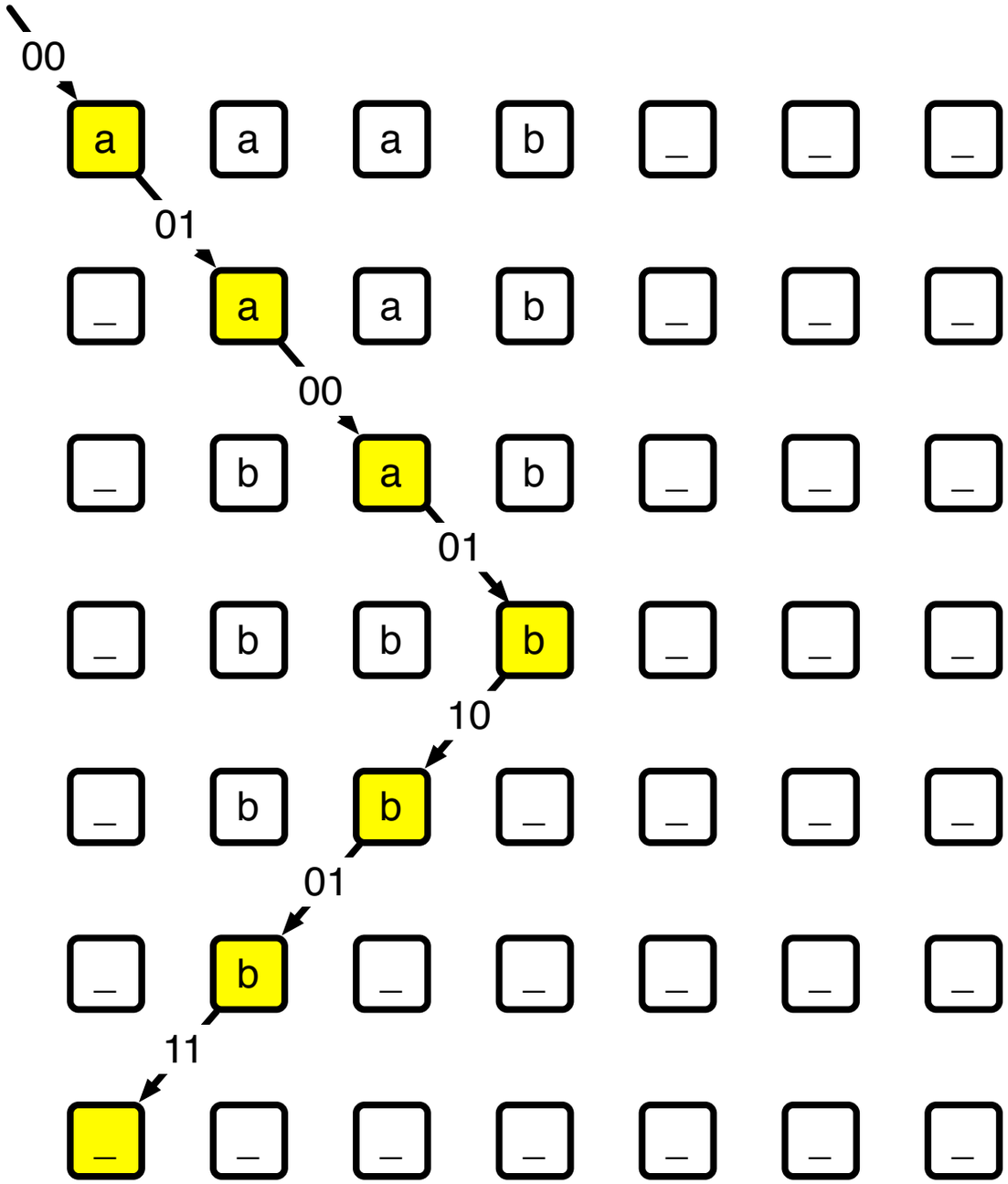
$A_{NTIME(n)} \leq_{m,p}$ PARKETT

- **Sei bin(x) eine Kodierung jeweils der endlichen Zustände und Eingabesymbole als eindeutige Binärzahl mit der festen Länge 1+ log max{|Σ|,|Q|}**

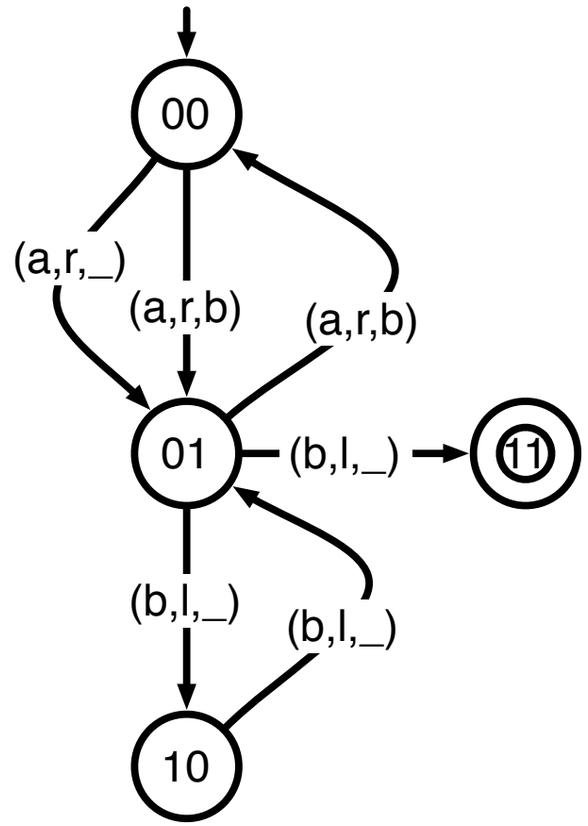
- **Kodiere die Linearzeit-Berechnung wie folgt:**
  - Die Bandzelle j im Schritt i wird als Bauteil $S_{i,j}$ kodiert
  - Ist der Kopf der NTM auf der Bandzelle wird in die Bandzelle zusätzlich der Zustand kodiert

- Für die obere Zellen ergeben sich folgende Teile
  - $S_{1,1}$ = (0, 0, 1 bin(x_1) bin(q_0), 0)
  - $S_{1,i}$ = (0, 0, 0 bin(x_i), 0) für i ≥ 2
- Für die untere Zeile ergeben sich die Teile
  - $S_{m,1}$ = (1 bin("_") bin(q_{akz}),0,0,0)
  - $S_{m,i}$ = (1 bin("_"),0,0,0) für i ≥ 2

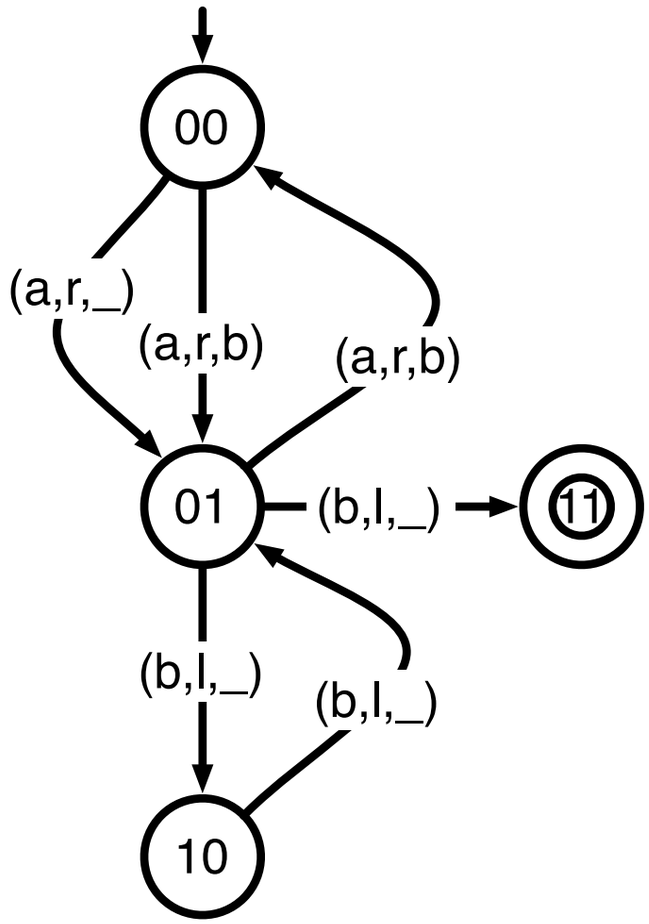
Berechnung einer 1-Band-NTM



Akzeptiert auf leerer Eingabe



0 0 0 1a00	0 0 0 0a	0 0 0 0a	0 0 0 0b	0 0 0 0_	0 0 0 0_	0 0 0 0_	0 0 0 0_
1a00 0 101 0_	0a 101 0 1a01	0a 0 0 0a	0b 0 0 0b	0_ 0 0 0_	0_ 0 0 0_	0_ 0 0 0_	0_ 0 0 0_
0_ 0 0 0_	1a01 0 100 0b	0a 100 0 1a00	0b 0 0 0b	0_ 0 0 0_	0_ 0 0 0_	0_ 0 0 0_	0_ 0 0 0_
0_ 0 0 0_	0b 0 0 0b	1a00 0 101 0b	0b 101 0 1b01	0 0 0 0_	0 0 0 0_	0 0 0 0_	0 0 0 0_
0_ 0 0 0_	0b 0 0 0b	0b 0 110 1b10	1b01 110 0 0_	0_ 0 0 0_	0_ 0 0 0_	0_ 0 0 0_	0_ 0 0 0_
0_ 0 0 0_	0b 0 101 1b01	1b10 101 0 0_	0_ 0 0 0_	0_ 0 0 0_	0_ 0 0 0_	0_ 0 0 0_	0_ 0 0 0_
0_ 0 111 1_11	1b01 111 0 0_	0_ 0 0 0_	0_ 0 0 0_	0_ 0 0 0_	0_ 0 0 0_	0_ 0 0 0_	0_ 0 0 0_
1_11 0 0 0	0_ 0 0 0_	0_ 0 0 0_	0_ 0 0 0_	0_ 0 0 0_	0_ 0 0 0_	0_ 0 0 0_	0_ 0 0 0_



Parkettierung gelungen

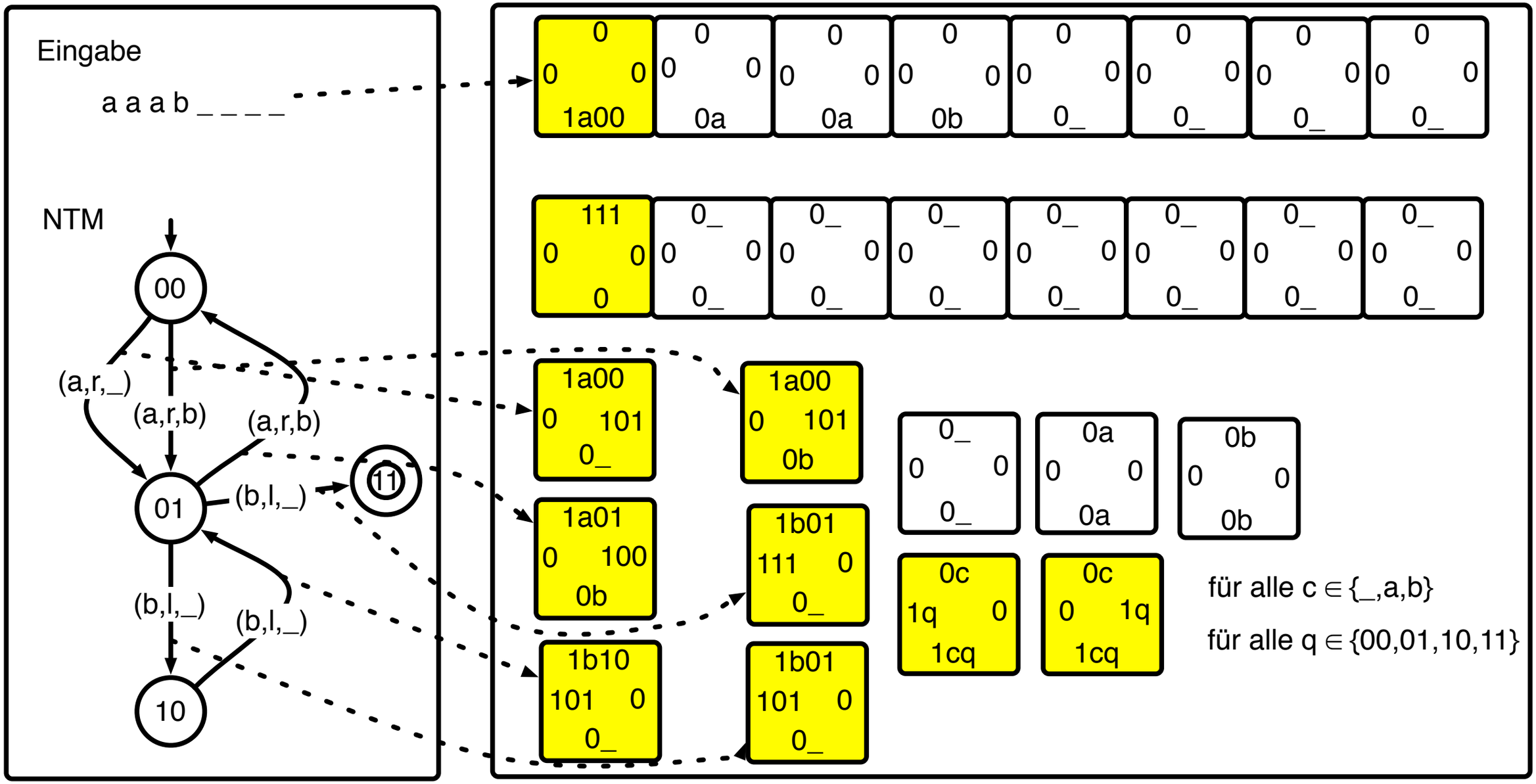
$A_{NTIME(n)} \leq_{m,p}$ PARKETT

- **Kodiere die Berechnungsschritte wie folgt:**
  - Die Bandzelle j im Schritt i wird als Bauteil $S_{i,j}$ kodiert
  - Ist der Kopf der NTM auf der Bandzelle wird in die Bandzelle zusätzlich der Zustand kodiert
  - Für die inneren Zellen gibt es folgende Fälle:
    - Der Kopf der NTM ist vor und nach einem Berechnungsschritt i nicht in der Zelle j, sei z das Zeichen auf dem Band und bin(z) die Binärdarstellung
      - entspricht Baustein (0 bin(z), 0, 0 bin(z), 0)
    - Der Kopf der NTM mit Zustand q bewegt sich von links in die Zelle mit Zustand
      - entspricht Baustein (0 bin(z), 0, 1 bin(z) bin(q), 1 bin(q))
    - Analog von rechts:
      - entspricht Baustein (0 bin(z), 1 bin(q) , 1 bin(z) bin(q), 0 , 0)
    - Der Kopf war im Schritt zuvor in der Zelle.
    - Die Berechnung ergibt aus Zustand q mit Symbol z, Nachfolgezustand q' mit Symbol z' und Bewegungsrichtung links
      - entspricht Baustein (1 bin(z) bin(q) , 0, 0 bin(z'), 1 bin(q'))
    - Entsprechend rechts
      - entspricht Baustein (1 bin(z) bin(q) , 1 bin(q'), 0 bin(z'), 0)

$A_{NTIME(n)} \leq_{m,p}$ PARKETT

- **Die Reduktionsfunktion:**
  - Auf Eingabe 1-Band NTM M mit Zustandsmenge Q, Startzustand q_0 und akzeptierenden Zustand q_{akz} welche in Linearzeit rechnet und die Eingabe löscht
  - und Eingabe $x \in \Sigma^*$
- **1. Erzeuge oberen und unteren Rand durch die Teile**
  - $S_{1,1}$ = (0, 0, 1 bin(x_1) bin(q_0), 0), $S_{1,i}$ = (0, 0, 0 bin(x_i), 0) für $i \geq 2$
  - $S_{m,1}$ = (1 bin("_") bin(q_{akz}) 0, 0, 0, 0), $S_{m,i}$ = (0 bin("_"), 0, 0 , 0) für $i \geq 2$
- **2. Erzeuge die folgenden Parkett-Teile**
  - Standardzellen: (0z,0,0z,0) für alle $z \in \Sigma$
  - Schiebezellen:
    - (0 bin(z), 0, 1 bin(z) bin(q), 1 bin(q)) für alle $z \in \Sigma$, $q \in Q$
    - (0 bin(z), 1 bin(q), 1 bin(z) bin(q), 0) für alle $z \in \Sigma$, $q \in Q$
  - Rechenzellen: Falls $(q,z) \rightarrow (q',z,l)$ in der Übergangsfunktion von M:
    - (1 bin(z) bin(q) , 0, 0 bin(z'), 1 bin(q'))
  - Falls $(q,z) \rightarrow (q',z,r)$ in der Übergangsfunktion von M
    - (1 bin(z) bin(q) , 1 bin(q'), 0 bin(z'), 0)
- **Laufzeit: O(n)** (Reduktion besteht hauptsächlich aus Kopieren)
- **Korrektheit:**
  - Jede akzeptierende Berechnung ergibt eine vollständige Parkettierung
  - Jede vollständige Parkettierung kann einer akzeptierenden Berechnung zugeordnet werden

Beispielreduktion



Definition FinPred

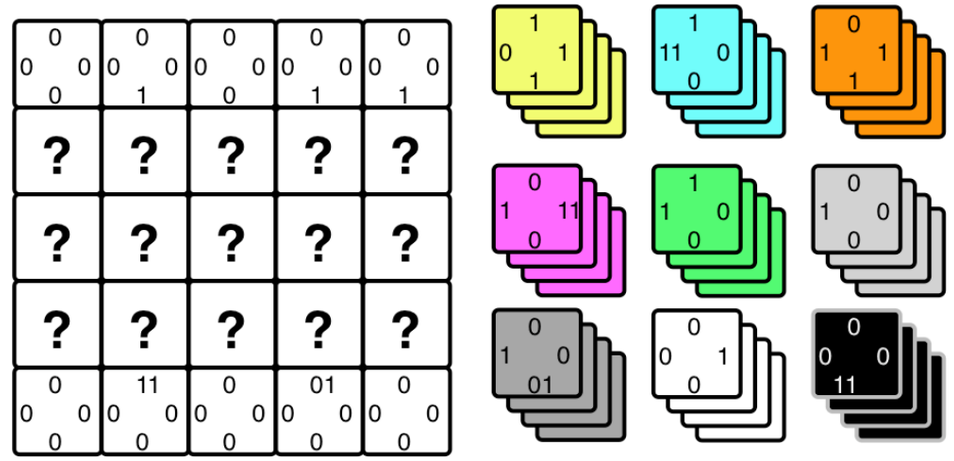
- **Endliches Prädikat**
  - Ist eine Funktion $P(x_1,...,x_k)$ die für Variablen $x_1,...,x_k \in \{0,1\}^p$ die Werte 0 (falsch) oder 1 (wahr) liefert
  - Wir beschränken uns auf die folgenden Prädikate auf Konstanten oder Variablen
    - Element: Für $x,y_1,...,y_k \in \{0,1\}^p$: für Konstanten $y_1,...,y_k$ und Variable x
      - $Element(x, y_1,...,y_k) := 1$ gdw $x \in \{y_1,...,y_k\}$
    - Teile sind gleich: Für Variablen $x,y \in \{0,1\}^p$ mit $x= x_1,...,x_p$ und $y= y_1,...,y_p$
      - $Teilgleich_{a,b,c}(x,y) = (x_a = y_b) \wedge (x_{a+1} = y_{b+1}) \wedge ... \wedge (x_{b+c-1} = y_{b+c-1})$
- **Definition FinPred (Endliche Prädikate)**
  - Gegeben:
    - Eine Menge von endlichen Prädikaten
  - Gesucht:
    - Gibt es eine Belegung der Variablen $x_1,...,x_k \in \{0,1\}^p$ so dass alle Prädikate erfüllt werden

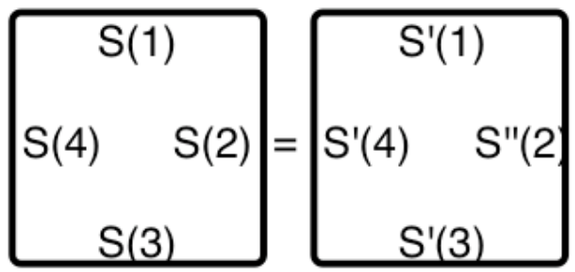
Beispiel für FinPred

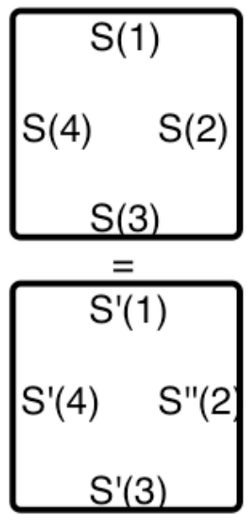
- **Beispiel:**
  - Element(x,010,100,111)
  - Element(y,110,000)
  - Teilgleich$_{2,3,1}$ (x,y)
  - Element(z,110,011,101)
  - Teilgleich$_{1,2,2}$ (x,z)
- **Lösung:**
  - x = 100
  - y = 110
  - z = 110
- **Teilgleich$_{2,3,1}$ (x,y)**
  - x= 100, y=110
- **Teilgleich$_{1,2,2}$ (x,z)**
  - x= 100, z=110

PARKETT $\leq_{m,p}$ FinPred

- **Lemma**
  - PARKETT ≤m,p FinPred
- **Reduktion**
  - PARKETT ist gegeben als
    - Menge von Bauteilen M = $\{B_1,B_2,B_3,..., B_k\}$
    - Ränder $S_{1,1}$, ..., $S_{1,m}$ und $S_{m,1}$, ..., $S_{m,m}$
  - Verwende für alle Ränder Zeichenketten gleicher Länge p
  - Kodiere ein Bauteil S als Zeichenkette S= S(1)S(2)S(3)S(4)
  - Seien $V_{i,j}$ Variablen aus $\{0,1\}^{4p}$
  - Füge Prädikate ein:
    - **`Element`**($V_{1,i}$,$S_{1,i}$) für alle i ∈ {1,..,m}
    - **`Element`**($V_{m,i}$,$S_{m,i}$) für alle i ∈ {1,..,m}
    - **`Element`**($V_{i,i}$, $B_1,B_2,B_3,..., B_k$) für alle i ∈ {2,..,m-1}, j ∈ {1,..,m}
    - **`Teilgleich`**$_{p+1,3p+1,p}$($V_{i,i}$,$V_{i,i+1}$) für alle i ∈ {1,..,m}, j ∈ {1,..,m-1}
      - d.h. horizontal passend S(2)= S'(4)
    - **`Teilgleich`**$_{2p+1,1,p}$($V_{i,i}$,$V_{i+1,i}$) für alle i ∈ {1,..,m-1}, j ∈ {1,..,m}
      - d.h. vertikal passend: S(3)=S'(1)
- **Korrektheit folgt aus der einfachen aussagenlogischen Umformung**
- **Laufzeit O(n^2), hauptsächlich für O(m^2) Prädikate `Teilgleich`**







FinPred $\leq_{m,p}$ SAT

- **Definition SAT**
  - Gegeben
    - Eine Boolesche Funktion ϕ mit Variablen $x_1,..., x_n$
  - Gibt es eine Belegung von $x_1,..., x_n$, für welche $\phi(x_1,..., x_n)=1$
- **Lemma: FinPred $\leq_{m,p}$ SAT**
- **Beweis: Reduktion:**
  - Forme jedes Prädikat als Boolesche Funktion um
    - Ersetze **`Element`**$(x, y_1,...,y_k)$ für $x \in \{0,1\}^p$ durch
      `Teilgleich`$_{1,1,p}(x, y_1)$ v
      `Teilgleich`$_{1,1,p}(x, y_2)$ v ... v
      `Teilgleich`$_{1,1,p}(x, y_k)$
    - Ersetze **`Teilgleich`**$_{a,b,c}(x, y)$ für $x= x_1,...,x_p$ und $y= y_1,...,y_p$ durch $[x_a = y_b] \wedge [x_{a+1} = y_{b+1}] \wedge ... \wedge [x_{a+c-1} = y_{b+c-1}]$
    - Ersetze $[x_i = y_i]$ durch $(x_i \wedge y_i) \vee (\neg x_i \wedge \neg y_i)$
  - Ver-"unde" alle entstehenden Booleschen Prädikate $T_1, T_2, ..., T_r$:
    - $T_1 \wedge T_2 \wedge ... \wedge T_r$
  - **Laufzeit: O(n)**
  - **Korrektheit: folgt aus der Äquivalenz der Umformungen**

Komplexitätstheorie

3-SAT ist NP-vollständig

SAT $\leq_{m,p}$ 3-SAT

- **Reduktionsfunktion**
  - Füge für jede Konjunktion oder Disjunktion zweier Teilterme eine neue Variable ein
  - Stelle für alle die Gleichungen auf
    - $x = y \vee z$ oder $x = y \wedge z$, wobei x,y,z Literale sind
  - Stelle alle Gleichungen als Klauseln dar mit Literalen x,y,z:
    - $x = y \vee z$
      - äquivalent zu $(x \wedge (y \vee z)) \vee (\neg x \wedge \neg(y \vee z))$
      - wird dargestellt als Klauseln: $(x \vee x \vee \neg y) \wedge (x \vee x \vee \neg z) \wedge (\neg x \vee y \vee z)$
    - $x = y \wedge z$
      - äquivalent zu $(x \wedge (y \wedge z)) \vee (\neg x \wedge \neg(y \wedge z))$
      - wird dargestellt als Klauseln: $(x \vee \neg y \vee \neg z) \wedge (\neg x \vee \neg x \vee y) \wedge (\neg x \vee \neg x \vee z)$

$$\exists x_1, x_2, x_3, x_4 \in \{0,1\} \; : \overline{(((x_1 \wedge \overline{x_2}) \vee \overline{x_3 \vee \overline{x_4}}) \wedge (\overline{x_1} \wedge \overline{x_3})}$$

$$\exists x_1, \ldots, x_4, h_1, \ldots h_5 \in \{0,1\}$$

$$h_4 = x_1 \wedge \overline{x_2} \qquad \overline{h_5} = x_3 \wedge \overline{x_4} \qquad h_3 = \overline{x_1} \wedge \overline{x_3}$$

$$h_2 = h_4 \vee h_5$$

$$\overline{h_1} = h_2 \wedge h_3$$

SAT $\leq_{m,p}$ 3-SAT

$$\exists x_1, x_2, x_3, x_4 \in \{0,1\} \ : \overline{(((x_1 \wedge \overline{x_2}) \vee \overline{x_3 \vee \overline{x_4}}) \wedge (\overline{x_1} \wedge \overline{x_3})}$$

$$\exists x_1, \ldots, x_4, h_1, \ldots h_5 \in \{0,1\}$$

$h_4 = x_1 \wedge \overline{x_2}$ $\quad \overline{h_5} = x_3 \wedge \overline{x_4}$ $\quad h_3 = \overline{x_1} \wedge \overline{x_3}$

$h_2 = h_4 \vee h_5$

$\overline{h_1} = h_2 \wedge h_3$

$$(h_1 \vee h_1 \vee h_2) \wedge (h_2 \vee h_2 \vee h_3) \wedge (\overline{h_1} \vee \overline{h_2} \vee \overline{h_3})$$

$$(h_2 \vee h_2 \vee \overline{h_4}) \wedge (h_2 \vee h_2 \vee \overline{h_5}) \wedge (\overline{h_2} \vee h_4 \vee h_4)$$

SAT $\leq_{m,p}$ 3-SAT

- **Reduktionsfunktion**
  - Füge für jede Konjunktion oder Disjunktion zweier Teilterme eine neue Variable ein
  - Stelle für alle die Gleichungen auf
    - $x = y \vee z$ oder $x = y \wedge z$, wobei x,y,z Literale sind
  - Stelle alle Gleichungen als Klauseln dar mit Literalen x,y,z:
    - $x = y \vee z$
      - wird dargestellt als Klauseln: $(x \vee x \vee \neg y) \wedge (x \vee x \vee \neg z) \wedge (\neg x \vee y \vee z)$
    - $x = y \wedge z$
      - wird dargestellt als Klauseln: $(x \vee \neg y \vee \neg z) \wedge (\neg x \vee \neg x \vee y) \wedge (\neg x \vee \neg x \vee z)$
- **Korrektheit:**
  - folgt aus der Äquivalenz der Teilformeln
- **Zeitaufwand zur Berechnung der Reduktion:**
  - Zerlegen der Teilformeln in Gleichungssysteme: $O(n^2)$
  - Umwandeln der Gleichungen in 3-KNF: $O(n)$

Theorem: 3-SAT ist NP-vollständig

- **Beweis:**
  - SAT $\leq_{m,p}$ 3-SAT
    - impliziert dass jedes NP-Problem auf 3-SAT reduziert werden kann
    - daher ist 3-SAT NP-schwierig
  - 3-SAT $\leq_{m,p}$ SAT
    - ist trivial: jede 3-KNF ist eine Boolesche Funktion
    - Da SAT $\in$ NP folgt: 3-SAT $\in$ NP
  - Damit ist 3-SAT NP-vollständig

Komplexitätstheorie

Parkett und Clique sind NP-vollständig

Theorem: PARKETT ist NP-vollständig

- **Beweis:**
  - Wir haben bewiesen:
    - Für alle $L \in NP$ gilt: $L \leq_{m,p} PARKETT$
    - Damit ist PARKETT NP-schwierig (nach Definition)
  - Andererseits:
    - $PARKETT \leq_{m,p} SAT$
    - Da $SAT \in NP$, folgt $PARKETT \in NP$
- Damit ist PARKETT NP-vollständig

Theorem: CLIQUE ist NP-vollständig

- **Definition CLIQUE:**
  - Gegeben:
    - Ein ungerichteter Graph G
    - Eine Zahl k
  - Gesucht:
    - Hat der Graph G eine Clique der Größe k?
- **Beweis:**
  - Wir haben bereits bewiesen: 3-SAT $\leq_{m,p}$ CLIQUE
    - 3-SAT ist NP-schwierig
    - Dann ist CLIQUE auch NP-schwierig
  - Zu beweisen: CLIQUE ist in NP

- Konstruiere Verifizierer:
  - Sei $v_1,v_2,..,v_k$ eine beliebige Knotenmenge der Größe k
  - Sei <G,k> die Instanz von CLIQUE
  - Verifiziere, ob die Knoten $v_1,v_2,..,v_k$ verschieden sind und ob alle Kanten zwischen den Knoten existieren
- Laufzeit: $O(n^2)$

4.2 NP

Ende