# Grundlagen der Theoretischen Informatik

**Komplexitätstheorie (I)**

22.07.2015 und 23.07.2015

Viorica Sofronie-Stokkermans

e-mail: sofronie@uni-koblenz.de

# Übersicht

1. Motivation
2. Terminologie
3. Endliche Automaten und reguläre Sprachen
4. Kellerautomaten und kontextfreie Sprachen
5. Turingmaschinen und rekursiv aufzählbare Sprachen
6. Berechenbarkeit, (Un-)Entscheidbarkeit
7. Komplexitätsklassen P und NP

# Komplexitätstheorie

## Inhalt

- Definition der Komplexitätsklassen **P** und **NP**.
- Begriff der **Reduktion**:
  Ein Problem (eine Sprache) wird auf eine zweite reduziert.
  Das erste Problem ist dann höchstens so schwer wie das zweite.
- Den Begriff eines **NP-schweren** Problems.
- Einige Probleme der Graphentheorie: sie sind **NP-vollständig**.
- Die wichtigsten **Komplexitätsklassen** und ihre Struktur.

# Komplexitätstheorie

**Welche Arten von Komplexität gibt es?**

- **Zeit**
- **Speicher**

# DTIME und NTIME

**Definition [NTIME(T(n)), DTIME(T(n))]**

**Basismodell:** $k$-DTM $\mathcal{M}$ (ein Band für die Eingabe).

Wenn $\mathcal{M}$ mit jedem Eingabewort der Länge $n$ höchstens $T(n)$ Schritte macht, dann wird sie **$T(n)$-zeitbeschränkt** genannt.
Die von $\mathcal{M}$ akzeptierte Sprache hat **Zeitkomplexität** $T(n)$
(tatsächlich meinen wir $\max(n+1, \lceil T(n) \rceil)$).

- **DTIME**(T(n)) ist die Klasse der Sprachen, die von $T(n)$-zeitbeschränkten, DTMn akzeptiert werden.
- **NTIME**(T(n)) ist die Klasse der Sprachen, die von $T(n)$-zeitbeschränkten, NTMn akzeptiert werden.

# DSPACE und NSPACE

**Definition [NSPACEmit(S(n)), DSPACE(S(n))]**

**Basismodell:** $k$-DTM $\mathcal{M}$ davon ein spezielles Eingabeband (**offline DTM**).

Wenn $\mathcal{M}$ für jedes Eingabewort der Länge $n$ maximal $S(n)$ Zellen auf den Ablagebändern benutzt, dann heißt $\mathcal{M}$ $S(n)$-**speicherbeschränkt.**
Die von $\mathcal{M}$ akzeptierte Sprache hat **Speicherkomplexität** $S(n)$ (tatsächlich meinen wir $\max(1, \lceil S(n) \rceil)$)

- **DSPACE**(S(n)) ist die Klasse der Sprachen, die von $S(n)$-speicherbeschränkten, DTMn akzeptiert werden.
- **NSPACE**(S(n)) ist die Klasse der Sprachen, die von $S(n)$-speicherbeschränkten, NTMn akzeptiert werden.

# DSPACE und NSPACE

Wieso eine *offline*-Turing-Maschine?

**Bandbeschränkung von weniger als linearem Wachstum**.

**Beispiel** Zu welcher Zeit-/Speicherkomplexitätsklasse gehört

$$\mathcal{L}_{\mathrm{mirror}} := \{wcw^R : \ w \in (0+1)^*\},$$

also die Menge aller Wörter die um den mittleren Buchstaben $c$ gespiegelt werden können?

# DSPACE und NSPACE

**Beispiel (Forts.)**

**Zeit:** **DTIME**(n+1). Die Eingabe wird einfach rechts vom $c$ in umgekehrter Reihenfolge kopiert. Wenn das $c$ gefunden wird, wird einfach der übrige Teil (das $w$) mit der Kopie von $w$ auf dem Band verglichen.

**Speicher:** Die gerade beschriebene Maschine liefert eine Schranke von **DSPACE**(n).

**Geht es noch besser?**

**Ja:** **DSPACE**(lg $n$). Wir benutzen zwei Bänder als Binär-Zähler. Die Eingabe auf das Auftreten von genau einem $c$ benötigt keinen Speicher (kann mit einigen Zustnden getan werden). Als zweites prüfen wir Zeichen für Zeichen auf der linken und auf der rechten Seite: dazu müssen die zu prüfenden Positionen gespeichert werden (sie werden auf den beiden Bändern kodiert).

Man kommt auch mit einem einzigen Zähler aus (und einem Band).

# Wichtige Fragen

## Wichtige Fragen (1)

**Zeit:** Wird jede Sprache aus **DTIME**(f(n)) von einer DTM entschieden?

**Speicher:** Wird jede Sprache aus **DSPACE**(f(n)) von einer DTM entschieden?

# Wichtige Fragen

## Wichtige Fragen (1)

**Zeit:** Wird jede Sprache aus **DTIME**(f(n)) von einer DTM entschieden?

**Speicher:** Wird jede Sprache aus **DSPACE**(f(n)) von einer DTM entschieden?

**Die Funktionen $f$ sind immer sehr einfache Funktionen, insbesondere sind sie alle berechenbar. In dieser Vorlesung betrachten wir nur Potenzen $f(n) = n^i$.**

# Wichtige Fragen

## Wichtige Fragen (2)

**Zeit/Speicher:** Wie sieht es mit **NTIME**(f(n)), **NSPACE**(f(n)) aus?

**Zeit vs. Speicher:** Welche Beziehungen gelten zwischen
**DTIME**(f(n)), **DSPACE**(f(n)), **NTIME**(f(n)), **NSPACE**(f(n))?

# Wichtige Fragen

**Halten, hängen nach $n$ Schritten.**

**Zeitbeschränkt:** Was bedeutet es, dass **eine DTM höchstens $n$ Schritte macht**?

**Halten?:** Streng genommen, müßte sie dann entweder **halten** oder **hängen**. Das bedeutet, im ersten Fall, dass die Eingabe **akzeptiert** wird.

**Hängen?:** DTM'n auf beidseitig unendlichem Band können aber nicht hängen.

# Wichtige Fragen

**Stoppen nach $n$ Schritten.**

**Stoppen:** Wir wollen unter **eine DTM macht höchstens $n$ Schritte** folgendes verstehen:

- Sie **hält** (und **akzeptiert** die Eingabe) innerhalb von $n$ Schritten.
- Sie **hängt** (und **akzeptiert** die Eingabe **nicht**) innerhalb von $n$ Schritten.
- Sie **stoppt** innerhalb von $n$ Schritten **ohne in den Haltezustand überzugehen**. Auch hier wird die Eingabe nicht akzeptiert.

# Antworten

**Antworten (informell)**

**Zeit:** Jede Sprache aus **DTIME**(f(n)) ist entscheidbar: Man warte einfach solange wie $f(n)$ angibt. Falls bis dahin kein "Ja" gekommen ist, ist die Antwort eben "Nein".

**Speicher:** Jede Sprache aus **DSPACE**(f(n)) ist entscheidbar. Denn es gibt nur **endlich viele verschiedene Konfigurationen**. Falls also die DTM nicht terminiert (was passiert), macht man folgendes: man schreibt alle Konfigurationen auf (die komplette Rechnung). Falls die DTM nicht terminiert, muss sie in eine Schleife laufen (eine Konfiguration erreichen, die sie schon einmal hatte). Das lässt sich feststellen.

# Antworten

## Antworten (informell)

**NTM vs. DTM:** Trivial sind

- $\mathbf{DTIME}(f(n)) \subseteq \mathbf{NTIME}(f(n))$ und
- $\mathbf{DSPACE}(f(n)) \subseteq \mathbf{NSPACE}(f(n))$.

Versucht man aber eine NTM durch eine DTM zu simulieren, braucht man (vermutlich) exponentiell mehr Zeit.

Für die Speicherkomplexität kann man zeigen:

- $\mathbf{NSPACE}(f(n)) \subseteq \mathbf{DSPACE}(f^2(n))$.

# Antworten

## Antworten (informell)

**Zeit vs. Speicher:** Trivial sind

- **DTIME**$(f(n)) \subseteq$ **DSPACE**$(f(n))$ und
- **NTIME**$(f(n)) \subseteq$ **NSPACE**$(f(n))$.

Aber **DSPACE**(f(n)), **NSPACE**mit(f(n)) sind viel größer.

# Bandkompression/Zeitbeschleunigung

## Konstante Faktoren werden ignoriert

Nur **die funktionale Wachstumsrate einer Funktion in Komplexitätsklassen zählt**: Konstante Faktoren werden ignoriert.

# Bandkompression

**Theorem [Bandkompression]**
Für jedes $c \in \mathbb{R}^+$ und jede Speicherfunktion $S(n)$ gilt:

$$\mathbf{DSPACE}(S(n)) = \mathbf{DSPACE}(cS(n))$$
$$\mathbf{NSPACE}(S(n)) = \mathbf{NSPACE}(cS(n))$$

Beweis Eine Richtung ist trivial. Die andere geht, indem man der Inhalt einer festen Anzahl $r$ ($> \frac{2}{c}$) von benachbarten Zellen auf dem Band als **ein neues Symbol** darstellt. **Die Zustände der neuen Maschine simulieren die alten Kopfbewegungen als Zustandsübergänge** (innerhalb des neuen Symbols). D.h. für $r$ Zellen der alten Maschine werden nun nur maximal 2 benutzt: im ungünstigsten Fall, wenn man von einem Block in den benachbarten geht.

# Zeitbeschleunigun

**Theorem [Zeitbeschleunigung]**
Für jedes $c \in \mathbb{R}^+$ und jede Zeitfunktion $T(n)$ mit $\lim_{n\to\infty} \frac{T(n)}{n} = \infty$ gilt:

$$\mathbf{DTIME}(T(n)) = \mathbf{DTIME}(cT(n))$$

$$\mathbf{NTIME}(T(n)) = \mathbf{NTIME}(cT(n))$$

Beweis Eine Richtung ist trivial. Der Beweis der anderen läuft wieder über die Repräsentation einer festen Anzahl $r$ $(> \frac{4}{c})$ benachbarter Bandzellen durch **neue Symbole**. Im Zustand der neuen Maschine wird wieder kodiert, welches Zeichen und welche Kopfposition (der simulierten, alten Maschine) aktuell ist.

**Wenn die alte Maschine simuliert wird, muss die neue nur $4$ statt $r$ Schritte machen**: 2 um die neuen Felder zu drucken und weitere 2 um den Kopf auf das neue und wieder zurück auf das alte (im schlimmsten Fall) zu bewegen.

# Wachstumsrate von DTIME und DSPACE

### Wachstumsrate von DTIME, DSPACE

Es sei $T(n)$ eine Zeitfunktion mit $\lim_{n\to\infty} \frac{T(n)}{n} = \infty$, und es sei $S(n)$ eine Speicherfunktion.

**(a)** Es sei $f(n) = \mathbf{O}(T(n))$. Dann gilt: $\mathbf{DTIME}(f(n)) \subseteqq \mathbf{DTIME}(T(n))$.

**(b)** Es sei $g(n) = \mathbf{O}(S(n))$. Dann gilt: $\mathbf{DSPACE} g(n) \subseteqq \mathbf{DSPACE}(S(n))$.

# Wachstumsrate von DTIME und DSPACE

**Definition [P, NP, PSPACE]**

$$
\begin{aligned}
\mathbf{P} &:= \bigcup_{i \geqslant 1} \mathbf{DTIME}(n^i) \\
\mathbf{NP} &:= \bigcup_{i \geqslant 1} \mathbf{NTIME}(n^i) \\
\mathbf{PSPACE} &:= \bigcup_{i \geqslant 1} \mathbf{DSPACE}(n^i)
\end{aligned}
$$

# Wachstumsrate von DTIME und DSPACE

**Definition [P, NP, PSPACE]**

$$
\begin{aligned}
\mathbf{P} &:= \bigcup_{i \geqslant 1} \mathbf{DTIME}(n^i) \\
\mathbf{NP} &:= \bigcup_{i \geqslant 1} \mathbf{NTIME}(n^i) \\
\mathbf{PSPACE} &:= \bigcup_{i \geqslant 1} \mathbf{DSPACE}(n^i)
\end{aligned}
$$

**Intuitiv**

- Probleme in **P** sind effizient lösbar, jene aus **NP** können in exponentieller Zeit gelöst werden.
- **PSPACE** ist eine sehr große Klasse, weit größer als **P** oder **NP** .

# Komplexitätsklassen für Funktionen

**Komplexitätsklassen für Funktionen**

Eine Funktion $f : \mathbb{N} \to \mathbb{N}$ ist in **P**, falls es eine DTM $\mathcal{M}$ und ein Polynom $p(n)$ gibt, so dass für jedes $n$ der Funktionswert $f(n)$ in höchstens $p(\text{länge}(n))$ Schritten von $\mathcal{M}$ berechnet wird.

Dabei gilt $\text{länge}(n) = \lg n$, denn man braucht $\lg n$ Zeichen um die Zahl $n$ binär darzustellen.

Analog funktioniert dies für alle anderen Komplexitätsklassen.

# Beziehungen zwischen den Komplexitätsklassen

**Frage:**

Was sind die genauen Beziehungen zwischen den Komplexitätsklassen P, NP, PSPACE?

# Beziehungen zwischen den Komplexitätsklassen

**Frage:**

Was sind die genauen Beziehungen zwischen den Komplexitätsklassen P, NP, PSPACE?

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE}$$

# Beziehungen zwischen den Komplexitätsklassen

**Frage:**

Wie zeigen wir, dass ein gegebenes Problem in einer bestimmten Klasse ist?

**Antwort**

**Reduktion auf ein bekanntes!**

Wir brauchen eines, mit dem wir anfangen können: SAT

# Komplexitätsklassen

**Frage:**

Können wir in **NP** Probleme finden, die **die schwierigsten in NP** sind?

# Komplexitätsklassen

**Frage:**

Können wir in **NP** Probleme finden, die **die schwierigsten in NP** sind?

**Antwort**

Es gibt mehrere Wege, ein schwerstes Problem zu definieren. Sie hängen davon ab, welchen **Begriff von Reduzierbarkeit** wir benutzen.

Für einen gegebenen Begriff von Reduzierbarkeit ist die Antwort: **Ja**.

Solche Probleme werden **vollständig in der gegebenen Klasse** bezüglich des Begriffs der Reduzierbarkeit genannt.

# Reduktion

**Definition (Polynomial-Zeit-Reduzibilität)**
Seien $L_1, L_2$ Sprachen.
$L_1$ ist Polynomial-Zeit reduzibel auf $L_2$, bezeichnet mit $L_1 \preceq_{\text{pol}} L_2$,
wenn es eine **Polynomial-Zeit beschränkte DTM** gibt,
die für jede Eingabe $w$ eine Ausgabe $f(w)$ erzeugt, so dass

$$w \in L_1 \ \underline{\text{gdw}} \ f(w) \in L_2$$

# Reduktion

**Lemma [Polynomial-Zeit-Reduktionen]**

1. Sei $L_1$ Polynomial-Zeit-reduzibel auf $L_2$ ($L_1 \preceq_{\text{pol}} L_2$). Dann gilt

   | | | |
   |---|---|---|
   | Wenn $L_2$ in **NP** ist | dann ist auch | $L_1$ in **NP** |
   | Wenn $L_2$ in **P** ist | dann ist auch | $L_1$ in **P** |

2. Die Komposition zweier Polynomial-Zeit-Reduktionen ist wieder eine Polynomial-Zeit-Reduktion.

# NP

**Theorem.**
Eine Sprache $L$ is in **NP** genau dann wenn es eine Sprache $L'$ in **P** und ein $k \geq 0$ gibt, so das für alle $w \in \Sigma$ gilt:

$$w \in L \quad \text{gdw.} \quad \text{es gibt ein } c : \langle w, c \rangle \in L' \text{ und } |c| < |w|^k.$$

$c$ wird **Zeuge** (*witness* oder Zertifikat/*certificate*) von $w$ in $L$ genannt.

Eine DTM, die die Sprache $L'$ akzeptiert, wird **Prüfer** (*verifier*) von $L$ genannt.

**Wichtig:**
Ein Entscheidungsproblem ist in **NP** genau dann wenn **jede Ja-Instanz ein kurzes Zertifikat** hat (d.h. seine Länge polynomial in der Länge der Eingabe ist), welche in polynomial-Zeit verifiziert werden kann.

# Vollständige und harte Probleme

**Definition [NP-vollständig, NP-hart]**

- Eine Sprache $L$ heißt **NP-hart (NP-schwer)** wenn jede Sprache $L' \in \mathbf{NP}$ polynomial-zeit-reduzibel auf $L$ ist.
- Eine Sprache $L$ heißt **NP-vollständig** wenn sie
  1. in **NP** ist ($L \in \mathbf{NP}$), und
  2. **NP**-hart ist

# Vollständige und harte Probleme

**Definition [NP-vollständig, NP-hart]**

- Eine Sprache $L$ heißt **NP-hart (NP-schwer)** wenn jede Sprache $L' \in \mathbf{NP}$ polynomial-zeit-reduzibel auf $L$ ist.
- Eine Sprache $L$ heißt **NP-vollständig** wenn sie
  1. in **NP** ist ($L \in \mathbf{NP}$), und
  2. **NP**-hart ist

**Definition [PSPACE-vollständig, PSPACE-hart]**

- Eine Sprache $L$ heißt **PSPACE-hart (PSPACE-schwer)** wenn jede Sprache $L' \in \mathbf{PSPACE}$ polynomial-zeit-reduzibel auf $L$ ist.
- Eine Sprache $L$ heißt **PSPACE-vollständig** wenn sie
  1. in **PSPACE** ist ($L \in \mathbf{PSPACE}$) und
  2. **PSPACE**-hart ist

# Vollständige und harte Probleme

**Bemerkenswert**

- Wenn gezeigt werden kann, dass auch nur ein einziges **NP**-hartes Problem in **P** liegt, dann ist $\mathbf{P} = \mathbf{NP}$.
- Wenn $\mathbf{P} \neq \mathbf{NP}$ gilt, dann ist kein einziges **NP**-vollständiges Problem in polynomieller Zeit lösbar.

# Vollständige und harte Probleme

**Bemerkenswert**

- Wenn gezeigt werden kann, dass auch nur ein einziges **NP**-hartes Problem in **P** liegt, dann ist $\mathbf{P} = \mathbf{NP}$.
- Wenn $\mathbf{P} \neq \mathbf{NP}$ gilt, dann ist kein einziges **NP**-vollständiges Problem in polynomieller Zeit lösbar.

**Eine Million Euro für den, der das „$\mathbf{P} = \mathbf{NP}$“-Problem löst!**
(Millenium Probleme)

# Vollständige und harte Problem

### Wie zeigt man NP-Vollständigkeit?

Um zu zeigen, dass eine Sprache $L$ **NP**-vollständig ist:

- Finde bekanntermaßen **NP**-vollständige Sprache $L'$ und
- reduziere sie auf $L$:

$$L' \preceq L$$

Das genügt, da jede Sprache aus **NP** auf $L'$ reduzierbar ist und wegen $L' \preceq L$ dann auch auf $L$.

**Hierfür häufig verwendet:**
SAT-Problem, d.h.

$$L' = L_{\text{sat}} = \{w \mid w \text{ ist eine erfüllbare aussagenlogische Formel}\}$$

# Abgschlossenheit der Komplexitätsklassen

## P, PSPACE sind abgeschlossen unter Komplement

Alle Komplexitätsklassen, die mittels **deterministischer Turing-Maschinen** definiert sind, sind **abgeschlossen unter Komplement-Bildung**

Denn:
Wenn eine Sprache $L$ dazu gehört, dann auch ihr Komplement
(einfach die alte Maschine ausführen und die Ausgabe invertieren).

# Abgschlossenheit der Komplexitätsklassen

## Abgeschlossenheit von NP unter Komplement

**Frage:**

Ist **NP** abgeschlossen unter Komplementbildung?

# Abgschlossenheit der Komplexitätsklassen

## Abgeschlossenheit von NP unter Komplement

**Frage:**
Ist **NP** abgeschlossen unter Komplementbildung?

**Antwort:**
Keiner weiß es!

# Die Kompexitätsklasse co-NP

**Definition [co-NP]**

**co-NP** ist die Klasse dr Sprachen deren Komplemente in **NP** liegen:

$$\textbf{co-NP} = \{L \mid \overline{L} \in \textbf{NP}\}$$

# Beziehungen zwischen Komplexitätsklassen

**Die folgenden Beziehungen sind momentan noch unbekannt**

1. **P** $=?$ **NP**.
2. **NP** $=?$ **co-NP**.
3. **P** $=?$ **PSPACE**.
4. **NP** $=?$ **PSPACE**.

# Beispiele

# NP-vollständige Probleme

- Ist ein (un-) gerichteter Graph hamiltonsch? (**Hamiltonian circle**)
- Ist eine logische Formel erfüllbar? (**Satisfiability**)
- Gibt es in einem Graphen eine Clique der Größe $k$? (**Clique of size** $k$)
- Ist ein Graph mit drei Farben zu färben? (3-**colorability**)
- Gibt es in einer Menge von ganzen Zahlen eine Teilmenge mit der Gesamtsumme $x$? (**Subset Sum**)

# NP-vollständige Probleme

**Definition [Hamilton Circle]**

**Hamilton-Kreis:** Weg entlang der Kanten in einem Graphen, der jeden Knoten genau einmal besucht und wieder zum Ausgangspunkt zurückkehrt.

$L_{\mathbf{Ham}_{\mathrm{undir}}}$: Die Sprache, die aus allen ungerichteten Graphen besteht, in denen es einen Hamilton-Kreis gibt.

$L_{\mathbf{Ham}_{\mathrm{dir}}}$: Die Sprache, die aus allen gerichteten Graphen besteht, in denen es einen Hamilton-Kreis gibt.

# NP-vollständige Probleme

**Definition[Maximale Clique: $L_{\textbf{Clique}_k}$]**

Eine **Clique** in einem Graphen ist ein **vollständiger Teilgraph von** $G$.

Für $k \in \mathbb{N}$:

$L_{\textbf{Clique}_k}$ Die Sprache, die aus allen ungerichteten Graphen besteht, die eine Clique der Größe $k$ enthalten.

# NP-vollständige Probleme

**Definition[$k$-colorability: $L_{\mathbf{Color}_{\leq k}}$]**

Ein (ungerichteter) Graph heißt **$k$-färbbar**, falls jeder Knoten mit einer von $k$ Farben so gefärbt werden kann, dass benachbarte Knoten verschiedene Farben haben.

Für $k \in \mathbb{N}$:

$L_{\mathbf{Color}_{\leq k}}$ Die Sprache, die aus allen ungerichteten, mit höchstens $k$ Farben färbbaren Graphen besteht.

# NP-vollständige Probleme

**Definition[SAT, $k$-CNF, $k$-DNF]**

**DNF:** Eine Formel ist in **disjunktiver Normalform**, wenn sie von folgender Form ist:

$$(l_{11} \wedge \ldots \wedge l_{1n_1}) \vee \ldots \vee (l_{m1} \wedge \ldots \wedge l_{mn_m})$$

**CNF:** Eine Formel ist in **konjunktiver Normalform**, wenn sie von folgender Form ist:

$$(l_{11} \vee \ldots \vee l_{1n_1}) \wedge \ldots \wedge (l_{m1} \vee \ldots \vee l_{mn_m})$$

.....

# NP-vollständige Probleme

**Definition[SAT, $k$-CNF, $k$-DNF]** (Fortsetzung)

**$k$-DNF:** Eine Formel ist in $k$-*DNF* wenn sie in DNF ist und jede ihrer Konjuntionen genau $k$ Literale hat.

**$k$-CNF:** Eine Formel ist in $k$-*CNF* wenn sie in CNF ist und jede ihrer Disjunktion genau $k$ Literale hat.

# NP-vollständige Probleme

**Theorem [NP-vollständige Probleme]**

Die folgenden Probleme liegen in **NP** und sind **NP**-vollständig:

- $L_{\mathsf{sat}}$
- CNF
- $k$-CNF für $k \geq 3$

# NP-vollständige Probleme

**Theorem [NP-vollständige Probleme]**

Die folgenden Probleme liegen in **NP** und sind **NP**-vollständig:

- $L_{\mathsf{sat}}$
- CNF
- $k$-CNF für $k \geq 3$

**Theorem [Probleme in P]**

Die folgenden Probleme liegen in **P**:

- DNF
- $k$-DNF für alle $k$
- 2-CNF

# NP-vollständige Probleme

## Einige Beispiel-Reduktionen

- $L_{\text{CNF-SAT}} \preceq_{\text{pol}} L_{\text{Clique}_{\leq k}}$,
- $L_{\text{Ham}_{\text{dir}}} \preceq_{\text{pol}} L_{\text{Ham}_{\text{undir}}}$,
- $L_{\text{Ham}_{\text{undir}}} \preceq_{\text{pol}} L_{\text{Ham}_{cost\leq k}}, L_{\text{Clique}_k}$,
- $L_{\text{SAT}} \preceq_{\text{pol}} L_{\text{3-CNF}}$,
- $L_{\text{SAT}} \preceq_{\text{pol}} L_{\text{CNF-SAT}}$,
- $L_{\text{3-CNF}} \preceq_{\text{pol}} L_{\text{Color}_{\leq k}}$.

# NP-vollständige Probleme

**Beispiel:** CNF-SAT $\preceq_{\text{pol}}$ $\text{Clique}_{\leq k}$

Gegeben eine Instanz von CNF
(eine Konjunktion von Klauseln $C_1 \wedge C_2 \wedge \ldots \wedge C_k$)

Wir konstruieren daraus einen Graphen:

**Knoten:** die Paare $(x, i)$, so dass $x$ ein Literal ist, dass in der Klausel $C_i$ vorkommt.

**Kanten:** Es gibt eine Kante zwischen $(x, i)$ und $(y, j)$ falls:
(1) $i \neq j$, und
(2) $x, y$ sind nicht komplementär.

Es gilt dann:
**Die CNF-Formel ist erfüllbar genau dann,**
**wenn der zugeordnete Graph eine Clique der Größe $k$ hat.**

# NP-vollständige Probleme

- Ist ein (un-) gerichteter Graph hamiltonsch? (**Hamiltonian circle**)
- Ist eine logische Formel erfüllbar? (**Satisfiability**)
- Gibt es in einem Graphen eine Clique der Größe $k$? (**Clique of size** $k$)
- Ist ein Graph mit drei Farben zu färben? (3-**colorability**)
- Gibt es in einer Menge von ganzen Zahlen eine Teilmenge mit der Gesamtsumme $x$? (**Subset Sum**)