SAT ist polynomiell verifizierbar.

Satz

SAT$\in \mathcal{NP}$.

Beweis

Algorithmus Polynomieller Verifizierer

EINGABE: $(\phi(x_1, \ldots, x_n), \mathbf{c})$, wobei $\mathbf{c} = (c_1, \ldots, c_n) \in \{0,1\}^n$.

- Falls $\phi(c_1, \ldots, c_n) = 1$, akzeptiere. Sonst lehne ab.

Korrektheit:

- $\phi(x_1, \ldots, x_n) \in$ SAT $\Leftrightarrow \exists$ Belegung $\mathbf{c} \in \{0,1\}^n : \phi(\mathbf{c}) = 1$

Laufzeit:

- Belegung von ϕ mit $\mathbf{c}$: $\mathcal{O}(|\phi|)$ auf RAM.
- Auswertung von ϕ auf $\mathbf{c}$: $\mathcal{O}(|\phi|^2)$ auf RAM.

Konjunktive Normalform

Definition Konjunktive Normalform (KNF)

Seien $x_1, \ldots, x_n$ Boolesche Variablen und ϕ eine Boolesche Formel.

- Wir bezeichnen die Ausdrücke x_i und $\neg x_i$ als *Literale*.
- *Klauseln* sind disjunktive Verknüpfungen von Literalen.
- ϕ ist in KNF, falls ϕ eine Konjunktion von Klauseln ist.
- Eine KNF Formel ϕ ist in 3-KNF, falls jede Klausel genau 3 Literale enthält.

Bsp:

- $\neg x_1 \vee x_2$ und x_3 sind Klauseln.
- $(\neg x_1 \vee x_2) \wedge x_3$ ist in KNF.
- $(\neg x_1 \vee x_2 \vee x_2) \wedge (x_3 \vee x_3 \vee x_3)$ ist in 3-KNF.

Die Sprache 3-SAT

Definition 3SAT

3SAT$:= \{\phi \mid \phi$ ist eine erfüllbare 3-KNF Boolesche Formel.$\}$

- Offenbar gilt 3SAT $\subset$ SAT.

Satz

3SAT$\in \mathcal{NP}$.

Beweis

Algorithmus NTM für 3SAT

Eingabe: $\phi(x_1, \ldots, x_n) \in$ 3-KNF

1. Rate nicht-deterministisch eine Belegung $(c_1, \ldots, c_n) \in \{0,1\}^n$.
2. Falls $\phi(c_1, \ldots, c_n) = 1$, akzeptiere. Sonst lehne ab.

- Laufzeit Schritt 1: $\mathcal{O}(n) = \mathcal{O}(|\phi|)$, Schritt 2: $\mathcal{O}(|\phi|)$.
- D.h. die Laufzeit ist polynomiell in der Eingabelänge $|\phi|$.

Simulation von NTMs durch DTMs

Satz Simulation von NTM durch DTM

Sei N eine NTM, die die Sprache L in Laufzeit $t(n)$ entscheidet. Dann gibt es eine DTM M, die L in Zeit $\mathcal{O}(2^{t(n)})$ entscheidet.

Sei $B(w) = (V, E)$ der Berechnungsbaum von N bei Eingabe w.

Algorithmus DTM M für L

1. Führe Tiefensuche auf $B(w)$ aus.
2. Falls akzeptierender Berechnungspfad gefunden wird, akzeptiere.
3. Sonst lehne ab.

- Tiefensuche auf $B(w)$ benötigt Laufzeit $\mathcal{O}(|V| + |E|) = \mathcal{O}(|V|)$.
- Berechnungspfade in $B(w)$ besitzen höchstens Länge $t(n)$.
- D.h. $B(w)$ besitzt höchstens $2^{t(n)}$ Blätter.
- Damit besitzt $B(w)$ höchstens $|V| \leq 2 \cdot 2^{t(n)} - 1$ viele Knoten.
- D.h. die Gesamtlaufzeit ist $\mathcal{O}(2^{t(n)})$.

Polynomielle Reduktion

Definition Polynomiell berechenbare Funktion

Sei Σ ein Alphabet und $f : \Sigma^* \to \Sigma^*$. Die Funktion f heißt polynomiell berechenbar gdw. eine DTM M existiert, die für jede Eingabe w in Zeit polynomiell in $|w|$ den Wert $f(w)$ berechnet.

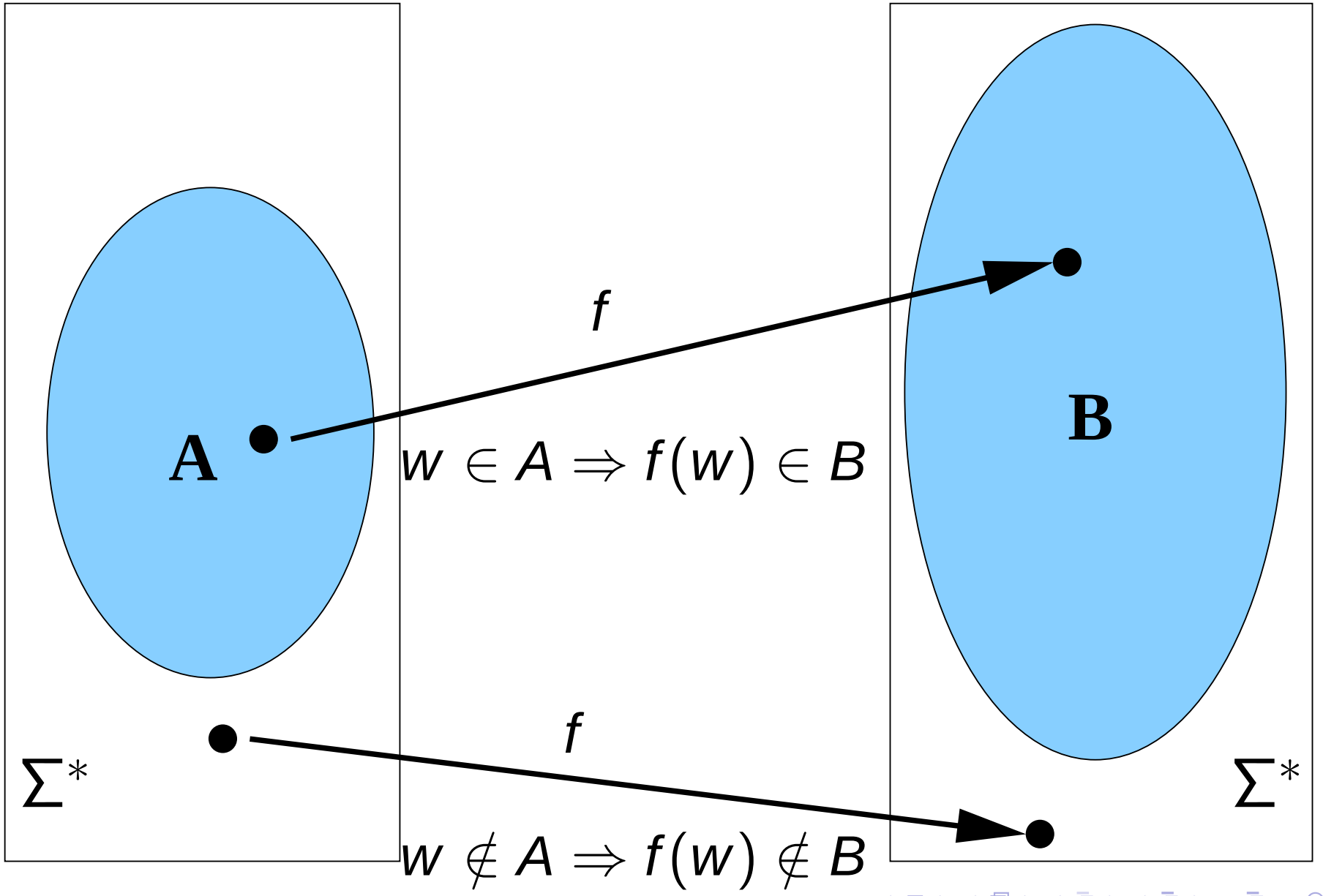
Definition Polynomielle Reduktion

Seien $A, B \subseteq \Sigma^*$ Sprachen. A heißt *polynomiell reduzierbar auf* B, falls eine polynomiell berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ existiert mit

$$w \in A \Leftrightarrow f(w) \in B \quad \text{für alle } w \in \Sigma^*.$$

Wir schreiben $A \leq_p B$ und bezeichen f als *polynomielle Reduktion*.

Graphische Darstellung $w \in A \Leftrightarrow f(w) \in B$



A ist nicht schwerer als B.

Satz $\mathcal{P}$-Reduktionssatz

Sei $A \leq_p B$ und $B \in \mathcal{P}$. Dann gilt $A \in \mathcal{P}$.

- Wegen $B \in \mathcal{P}$ existiert DTM M_B, die B in polyn. Zeit entscheidet.
- Wegen $A \leq_p B$ existiert DTM M_f, die f in polyn. Zeit berechnet.

Algorithmus DTM M_A für A

Eingabe: w

1. Berechne $f(w)$ mittels M_f auf Eingabe w.
2. Falls M_B auf Eingabe $f(w)$ akzeptiert, akzeptiere. Sonst lehne ab.

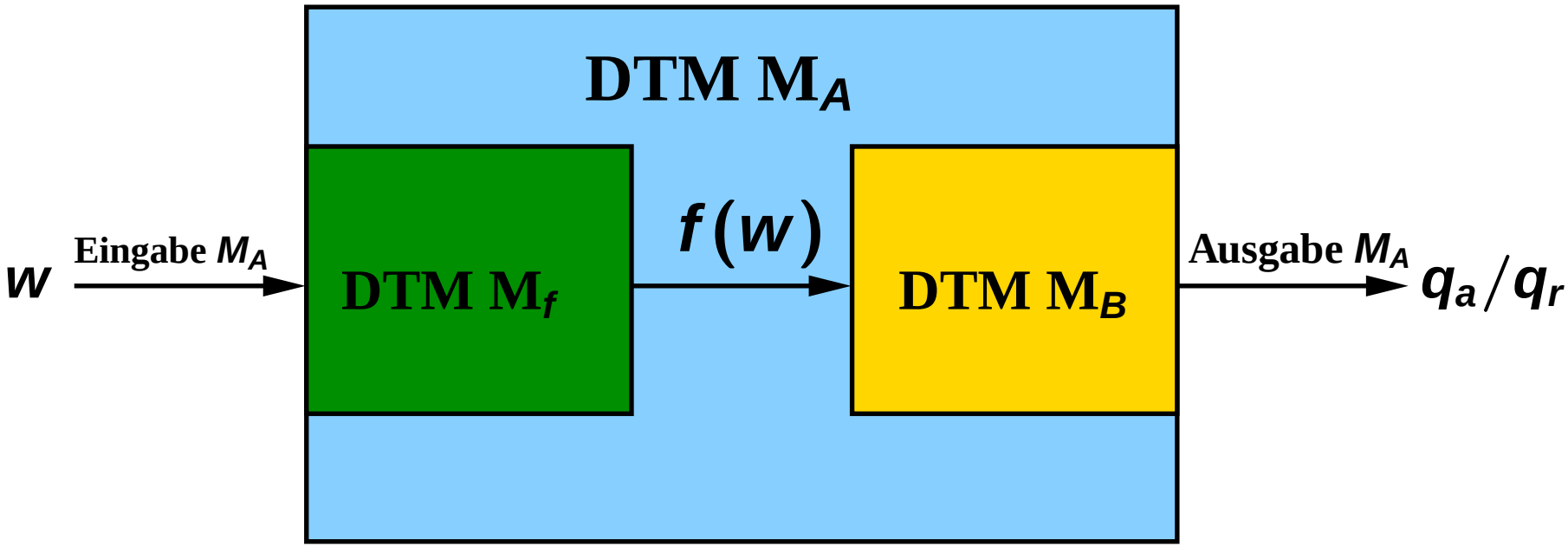
Korrektheit:

- M_A akzeptiert $w \Leftrightarrow M_B$ akzeptiert $f(w) \Leftrightarrow f(w) \in B \Leftrightarrow w \in A$.

Laufzeit:

- $T(M_A) = \mathcal{O}(T(M_f) + T(M_B))$, d.h. polynomiell in $|w|$.

Graphische Darstellung des Reduktionsbeweises



Transitivität polynomieller Reduktionen

Satz Transitivität von $\leq_p$

Seien $A, B, C \subseteq \Sigma^*$ Sprachen mit $A \leq_p B$ und $B \leq_p C$. Dann gilt $A \leq_p C$.

- Sei f die polynomielle Reduktion von A auf B, d.h.
$$w \in A \Leftrightarrow f(w) \in B \quad \text{für alle } w \in \Sigma^*.$$
- Sei g die polynomielle Reduktion von B auf C, d.h.
$$v \in B \Leftrightarrow g(v) \in C \quad \text{für alle } v \in \Sigma^*.$$
- Dann gilt insbesondere $w \in A \Leftrightarrow f(w) \in B \Leftrightarrow g(f(w)) \in C$.
- Damit ist die Komposition $g \circ f$ eine Reduktion von A auf C.
- $g \circ f$ kann in polynomieller Zeit berechnet werden durch Hintereinanderschaltung der polynomiellen DTMs für f und g.

Clique

Definition Clique

Sei $G = (V, E)$ ein ungerichteter Graph. $C \subseteq V$, $|C| = k$ heißt k-Clique in G, falls je zwei Knoten in C durch eine Kante verbunden sind.

$$\text{CLIQUE} := \{(G, k) \mid G \text{ enthält eine } k\text{-Clique.}\}$$

Satz

$3\text{SAT} \leq_p \text{CLIQUE}$

Zu zeigen: Es gibt eine Reduktion f mit

1. f ist eine polynomiell berechenbare Funktion
2. $\phi \in 3\text{SAT} \Leftrightarrow f(\phi) = (G, k) \in \text{CLIQUE}$

Idee für die Reduktion: Konstruiere (G, k) derart, dass

ϕ erfüllbar $\Leftrightarrow$ $\exists$ erfüllende Belegung B für ϕ.
$\Leftrightarrow$ B setzt in jeder Klausel mind. ein Literal auf wahr.
$\Leftrightarrow$ Wahre Literale entsprechen einer k-Clique in G.

Die Reduktion f

Algorithmus M_f für f

Eingabe: $\phi = (a_{11} \vee a_{12} \vee a_{13}) \wedge \ldots \wedge (a_{n1} \vee a_{n2} \vee a_{n3})$

1. Wahl der Knotenmenge V von G
   - Definiere $3n$ Knoten mit Labeln a_{i1}, a_{i2}, a_{i3} für $i = 1, \ldots, n$.
2. Wahl der Kantenmenge E: Setze Kante $(u, v) \in E$ **außer** wenn
   - u, v entsprechen Literalen derselben Klausel, denn die Clique soll aus Literalen verschiedener Klauseln bestehen.
   - Label von u ist Literal x und Label von v ist $\neg x$, denn x soll nicht gleichzeitig auf wahr und falsch gesetzt werden (Konsistenz).
3. Wahl von k.
   - Setze $k = n$, denn alle Klauseln sollen erfüllt werden.

Ausgabe: (G, k)

zu zeigen: f ist polynomiell berechenbar.

- Laufzeit Schritt 1: $\mathcal{O}(n)$, Schritt 2: $\mathcal{O}(n^2)$, Schritt 3: $\mathcal{O}(1)$.
- Gesamtlaufzeit $\mathcal{O}(n^2)$ ist polynomiell in der Eingabelänge.

Korrektheit der Reduktion

Zeigen zunächst: $\phi \in 3SAT \Rightarrow f(\phi) = (G, k) \in \text{CLIQUE}$

- Sei $\phi \in 3SAT$. Dann besitzt ϕ eine erfüllende Belegung B.
- Damit setzt B in jeder Klausel $(a_{i1} \vee a_{i2} \vee a_{i3})$, $i = 1, \ldots, n$ mindestens ein Literal $a_{i\ell_i}, \ell_i \in [3]$ auf wahr.
- Die n Knoten mit Label $a_{i\ell_i}$ in G sind paarweise verbunden, da
  - die Literale $a_{i\ell_i}$ aus verschiedenen Klauseln stammen.
  - B ist eine konsistente Belegung, d.h. dass die Literale $a_{i\ell_i}$ von B alle konsistent auf wahr gesetzt werden.
- Die n Knoten mit Label $a_{i\ell_i}$ bilden eine n-Clique in G.
- D.h. $f(\phi) = (G, k) \in \text{CLIQUE}$

Korrektheit von f: Rückrichtung

Zeigen: $f(\phi) = (G, k) \in \text{CLIQUE} \Rightarrow \phi \in 3\text{SAT}$

- Sei $f(\phi) = (G, k) \in \text{CLIQUE}$. Dann besitzt G eine n-Clique $v_1, \ldots, v_n$.
- Nach Konstruktion der Kantenmenge von E gilt:
  1. $v_1, \ldots, v_n$ korrespondieren zu Variablen in verschiedenen Klauseln.
  2. $\nexists v_i, v_j$ mit Labeln x und $\neg x$.
- Sei B diejenige Belegung, die die Label von $v_1, \ldots, v_n$ wahr setzt.
  1. B setzt in jeder Klausel ein Literal v_i auf wahr.
  2. B ist eine konsistente Belegung.
- Damit ist B ist eine erfüllende Belegung für ϕ.
- D.h. $\phi \in 3\text{SAT}$.

$\mathcal{NP}$-Vollständigkeit

Definition $\mathcal{NP}$-vollständig

Sei L eine Sprache. Wir bezeichnen L als $\mathcal{NP}$-vollständig, falls

1. $L \in \mathcal{NP}$
2. Für **jede** Sprache $A \in \mathcal{NP}$ gilt: $A \leq_p L$.

Separation oder Gleichheit von $\mathcal{P}$ und $\mathcal{NP}$

Satz

Sei L eine $\mathcal{NP}$-vollständige Sprache und $L \in \mathcal{P}$. Dann gilt $\mathcal{P} = \mathcal{NP}$.

Beweis:

- Wir zeigen für ein beliebiges $A \in \mathcal{NP}$, dass $A \in \mathcal{P}$.
- Da $A \in \mathcal{NP}$ und L $\mathcal{NP}$-vollständig ist, gilt $A \leq_p L$.
- Nach Voraussetzung gilt $L \in \mathcal{P}$.
- $\mathcal{P}$-Reduktionssatz: Aus $A \leq_p L$, $L \in \mathcal{P}$ folgt $A \in \mathcal{P}$.
- Da dies für ein beliebiges $A \in \mathcal{NP}$ gilt, folgt $\mathcal{NP} \subseteq \mathcal{P}$.
- Wegen $\mathcal{P} \subseteq \mathcal{NP}$ gilt schließlich $\mathcal{P} = \mathcal{NP}$.

$\mathcal{NP}$ Vollständigkeits-Beweise

Satz $\mathcal{NP}$-Reduktionssatz

Seien B, L Sprachen. Sei L $\mathcal{NP}$-vollständig, $B \in \mathcal{NP}$ und $L \leq_p B$. Dann ist auch B $\mathcal{NP}$-vollständig.

Beweis: Müssen zeigen, dass $A \leq_p B$ für alle $A \in \mathcal{NP}$.

- Da L $\mathcal{NP}$-vollständig ist, gilt $A \leq_p L$ für beliebiges $A \in \mathcal{NP}$.
- Ferner gilt nach Voraussetzung $L \leq_p B$.
- Aus der Transitivität von $\leq_p$ folgt: $A \leq_p B$.
- Damit ist B ebenfalls $\mathcal{NP}$-vollständig.

Problem: Wir benötigen ein *erstes* $\mathcal{NP}$-vollständiges Problem.