

# Diskrete Mathematik II

Alexander May

Fakultät für Mathematik  
Ruhr-Universität Bochum

Sommersemester 2014

# Organisatorisches

- Vorlesung: **Di 9-12** in HZO 100 (3+1 SWS, 6.75 CP)
- Übung: **Di 12:30-14:00** in ND 3/99, **Mi 12:30-14:00** in NC 2/99
  - ▶ Assistent: **Stefan Hoffmann**, Korrektor: **Manuel Fersch**
  - ▶ Präsenzübung ist **zweiwöchentlich**: 08.04., 15.04., 29.04., ...
  - ▶ Vorrechenübung ist **zweiwöchentlich**: 22.04., 06.05., 20.05., ...  
Abgabe der Übungen am selben Tag **vor** der Vorlesung.
  - ▶ Gruppenabgaben bis 3 Personen
  - ▶ Bonussystem:  
1/3-Notenstufe für 50%, 2/3-Notenstufe für 75%
- Klausur: August/September(?)

# Themengebiete

## 1 Komplexitätstheorie

- ▶ Klassen P und NP
- ▶ Reduktionen
- ▶ Anwendung: Sicherheitsbeweise in der Kryptographie

## 2 Algorithmische Zahlentheorie

- ▶ Quadratische Reste
- ▶ Beispiel Anwendungen: Zufallszahlengenerator, Identity-Based Encryption

## 3 Kodierungstheorie

- ▶ Komprimierende Codes
- ▶ Beispiel Anwendungen: Kommunikation (Mobilfunk, Internet), Speicher (MP3)
- ▶ Fehlererkennende Codes
- ▶ Ausfalltolerante Codes
- ▶ Beispiel Anwendungen: Mobilfunk, Internet, CD, Secret Sharing, Kryptosystem

# Weiterführende Referenzen

**Ziel:** Einfaches aber mächtiges Rechnermodell.

- Michael R. Garey, David S. Johnson, “Computers and Intractability”, Freeman, 2000
- J. Blömer, “Einführung in Algorithmen und Komplexität”, Vorlesungsskript Universität Paderborn, 2002
- N. Koblitz, “A Course in Number Theory and Cryptography”, Springer Verlag, 1994
- Steven Roman, “Introduction to Coding and Information Theory”, Springer Verlag, 1996

# Einführung in die NP-Vollständigkeitstheorie

## Notationen

- Alphabet  $A = \{a_1, \dots, a_m\}$  aus Buchstaben  $a_i$
- Worte der Länge  $n$  sind Elemente aus  $A^n = \{a_{i_1} \dots a_{i_n} \mid a_{i_j} \in A\}$ .
- $A^0 = \{\epsilon\}$ , wobei  $\epsilon$  das leere Wort ist.
- $A^* = \bigcup_{n=0}^{\infty} A^n$ ,  $A^+ = A^* \setminus \{\epsilon\}$ ,  $A^{\leq m} = \bigcup_{n=0}^m A^n$
- Länge  $|a_1 \dots a_n| = n$ .  $\text{bin}(a_1)$  ist Binärkodierung von  $a_1$ .

## Definition Sprache $L$

Sei  $A$  ein Alphabet. Eine Menge  $L \subseteq A^*$  heißt *Sprache* über dem Alphabet  $A$ . Das *Komplement* von  $L$  über  $A$  ist definiert als  $\bar{L} = A^* \setminus L$ .

# Turingmaschine (informal)

Turingmaschine besteht aus:

- Einseitig unendlichem Band mit Zellen (Speicher),
- Kontrolle und einem Lesekopf, der auf einer Zelle steht.

Arbeitsweise einer Turingmaschine

- Bandsymbol  $\triangleright$  steht in der Zelle am linken Bandende.
- Kontrolle besitzt Zustände einer endlichen Zustandsmenge.
- Abhängig vom Zelleninhalt und Zustand schreibt die Kontrolle ein Zeichen und bewegt den Lesekopf nach links oder rechts.
- Zu Beginn der Berechnung gilt:
  - ▶ Lesekopf befindet sich auf dem linken Bandende  $\triangleright$ .
  - ▶ Band enthält  $\triangleright a_1 \dots a_n \sqcup \sqcup \dots$ , wobei  $a_1 \dots a_n$  die Eingabe ist.
- Turingmaschine  $M$  hält gdw Kontrolle im Zustand  $q_a$  oder  $q_r$ .
  - ▶ Falls  $M$  in  $q_a$  hält, so akzeptiert  $M$  die Eingabe  $a_1 \dots a_n$ .
  - ▶ Falls  $M$  in  $q_r$  hält, so verwirft  $M$  die Eingabe  $a_1 \dots a_n$ .
  - ▶ Falls  $M$  nie in die Zustände  $q_a, q_r$  kommt:  $M$  läuft unendlich.

# Turingmaschine (formal)

## Definition Deterministische Turingmaschine (Turing 1936)

Eine deterministische Turingmaschine DTM ist ein 4-Tupel  $(Q, \Sigma, \Gamma, \delta)$  bestehend aus

- 1 Zustandmenge  $Q$ : Enthält Zustände  $q_a, q_r, s$ .
- 2 Bandalphabet  $\Gamma$  mit  $\sqcup, \triangleright \in \Gamma$
- 3 Eingabealphabet  $\Sigma \subseteq \Gamma \setminus \{\sqcup, \triangleright\}$ .
- 4 Übergangsfunktion  $\delta : Q \setminus \{q_a, q_r\} \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ 
  - ▶ Es gilt stets  $\delta(q, \triangleright) = (q', \triangleright, R)$  (am linken Bandende).
  - ▶ Es gilt nie  $\delta(q, a) = (q', \triangleright, L/R)$  (nicht am linken Bandende).

## Beispiel DTM $M_1$

**Bsp:**  $a^n, n \geq 1$

- $Q = \{q_0, q_1, q_a, q_r\}$  mit  $s = q_0$
- $\Sigma = \{a\}$  und  $\Gamma = \{\sqcup, \triangleright, a\}$
- Übergangsfunktion

$\delta$	$a$	$\sqcup$	$\triangleright$
$q_0$	$(q_1, a, R)$	$(q_r, \sqcup, R)$	$(q_0, \triangleright, R)$
$q_1$	$(q_1, a, R)$	$(q_a, \sqcup, R)$	$(q_1, \triangleright, R)$

**Notation der Konfigurationen bei Eingabe  $a^2$ :**

$q_0 \triangleright aa$   
 $\vdash \triangleright q_0 aa$   
 $\vdash \triangleright a q_1 a$   
 $\vdash \triangleright a a q_1 \sqcup$   
 $\vdash \triangleright a a \sqcup q_a \sqcup$



# Nachfolgekongfigurationen

## Notation Nachfolgekongfiguration

- Direkte Nachfolgekongfiguration:  $aqb \vdash a'q'b'$
- $i$ -te Nachfolgekongfiguration:  $aqb \vdash^i a'q'b'$
- Indirekte Nachfolgekongfiguration  $aqb \vdash^* a'b'q'$ , d.h.  
 $\exists i \in \mathbb{N} : aqb \vdash^i a'q'b'$ .

## Akzeptanz und Ablehnen von Eingaben

- DTM  $M$  erhalte Eingabe  $w \in \Sigma^*$ .
  - ▶  $M$  akzeptiert  $w \Leftrightarrow \exists a, b \in \Gamma^*$  mit  $s \triangleright w \vdash^* aq_a b$
  - ▶  $M$  lehnt  $w$  ab  $\Leftrightarrow \exists a, b \in \Gamma^*$  mit  $s \triangleright w \vdash^* aq_r b$

# Akzeptierte Sprache, $L$ rekursiv aufzählbar

## Definition Akzeptierte Sprache, Rekursive Aufzählbarkeit

Sei  $M$  eine DTM. Dann ist die von  $M$  *akzeptierte Sprache*

$$L(M) = \{w \in \Sigma^* \mid M \text{ akzeptiert Eingabe } w\}.$$

Eine Sprache  $L$  heißt *rekursiv aufzählbar* gdw eine DTM  $M$  existiert mit  $L = L(M)$ .

- Unsere Beispiel-DTM  $M_1$  akzeptiert die Sprache  $L(M_1) = \{a\}^+$ .
- D.h.  $L = \{a\}^+$  ist rekursiv aufzählbar, da für  $M_1$  gilt  $L = L(M_1)$ .
- Aus der obigen Definition folgt:  
 $L$  ist nicht rekursiv aufzählbar  $\Leftrightarrow \nexists$  DTM  $M$  mit  $L = L(M)$ .
- Es gibt Sprachen, die nicht rekursiv aufzählbar sind, z.B.  
 $\bar{H} = \{\langle M, x \rangle \mid \text{DTM } M \text{ hält bei Eingabe } x \text{ nicht.}\}$ . (ohne Beweis)

# Entscheidbarkeit und rekursive Sprachen

## Definition Entscheidbarkeit

Sei  $M$  eine DTM, die die Sprache  $L(M)$  akzeptiert.  $M$  *entscheidet* die Sprache  $L(M)$  gdw  $M$  alle Eingaben  $w \notin L(M)$  ablehnt. D.h. insbesondere  $M$  hält auf allen Eingaben.

Eine Sprache  $L$  heißt *entscheidbar* gdw eine DTM  $M$  existiert, die  $L$  entscheidet.

- Unsere Beispiel-DTM  $M_1$  entscheidet die Sprache  $L(M_1) = \{a\}^+$ .
- $L = \{a\}^+$  ist entscheidbar, da  $M_1$  die Sprache  $L$  entscheidet.

## Korollar Entscheidbarkeit impliziert rekursive Aufzählbarkeit

Sei  $L$  eine entscheidbare Sprache. Dann ist  $L$  rekursiv aufzählbar.

- Die Rückrichtung stimmt nicht:  
Es gibt rekursiv aufzählbare  $L$ , die nicht entscheidbar sind, z.B.  
 $H = \{\langle M, x \rangle \mid \text{DTM } M \text{ hält auf Eingabe } x.\}$ . (ohne Beweis)

# Entscheiden versus Berechnen

## Definition Berechnung von Funktionen

Eine DTM  $M$  berechnet die Funktion  $f : \mathbb{N}^n \rightarrow \mathbb{N}$ , falls  $M$  für jedes  $(a_1, \dots, a_n)$  bei Eingabe  $\text{bin}(a_1)\# \dots \# \text{bin}(a_n)$  den Bandinhalt  $\text{bin}(f(a_1, \dots, a_n))$  berechnet und in  $q_a$  hält.

- Werden der Einfachheit halber Sprachen entscheiden, nicht Funktionen berechnen.

# Laufzeit einer DTM, Klasse DTIME

## Definition Laufzeit einer DTM

Sei  $M$  eine DTM mit Eingabealphabet  $\Sigma$ , die bei jeder Eingabe hält. Sei  $T_M(w)$  die Anzahl der Rechenschritte – d.h. Bewegungen des Lesekopfes von  $M$  – bei Eingabe  $w$ . Dann bezeichnen wir die Funktion

$T_M(n) : \mathbb{N} \rightarrow \mathbb{N}$  mit  $T_M(n) = \max\{T_M(w) \mid w \in \Sigma^{\leq n}\}$   
als *Zeitkomplexität* bzw. *Laufzeit* der DTM  $M$ .

- Die Laufzeit wächst monoton in  $n$ .
- Unsere Beispiel-DTM  $M_1$  mit  $L(M_1) = \{a\}^*$  besitzt Laufzeit  $\mathcal{O}(n)$ .

## Definition DTIME

Sei  $t : \mathbb{N} \rightarrow \mathbb{N}$  eine monoton wachsende Funktion. Die Klasse DTIME ist definiert als

$DTIME(t(n)) := \{L \mid L \text{ wird von DTM mit Laufzeit } \mathcal{O}(t(n)) \text{ entschieden.}\}$ .

- Es gilt  $L(M_1) \in DTIME(n)$ .

# Registermaschine RAM

Registermaschine RAM besteht aus den folgenden Komponenten:

- Eingabe-/ und Ausgabe-Register
- Speicherregister
- Programm
- Befehlszähler
- Akkumulator

Funktionsweise einer RAM:

- Liest Eingabe aus Eingaberegister und lässt Programm auf Eingabe laufen.
- Führt Arithmetik im Akkumulator aus.
- Ergebnisse können im Speicherregister gespeichert werden.
- Befehlszähler realisiert Sprünge, Schleifen und bedingte Anweisungen im Programm.
- Ausgabe erfolgt im Ausgaberegister.

# DTMs versus RAMs, Churchsche These

## Fakt Polynomielle Äquivalenz von DTMs und RAMs

Sei  $t : \mathbb{N} \rightarrow \mathbb{N}$  eine monoton wachsende Funktion mit  $t(n) \geq n$ . Jede RAM mit Laufzeit  $t(n)$  kann durch eine DTM  $M$  mit Laufzeit  $\mathcal{O}(t(n)^3)$  simuliert werden.

## Churchsche These (1936)

”Die im intuitiven Sinne berechenbaren Funktionen sind genau die durch Turingmaschinen berechenbaren Funktionen.”

- These ist nicht beweisbar oder widerlegbar.
- Alle bekannten Berechenbarkeitsbegriffe führen zu DTM-berechenbaren Funktionen.

# Die Klasse $\mathcal{P}$

## Definition Klasse $\mathcal{P}$

Die Klasse  $\mathcal{P}$  ist definiert als

$$\mathcal{P} = \bigcup_{k \in \mathbb{N}} \text{DTIME}(n^k).$$

- $L \in \mathcal{P}$  gdw eine DTM existiert, die  $L$  in Laufzeit  $\mathcal{O}(n^k)$  entscheidet.
- $\mathcal{P}$  ist die Klasse aller in Polynomialzeit entscheidbaren Sprachen. (auf DTMs, RAMs, etc.)
- Hintereinanderausführung/Verzahnung von DTMs mit polynomieller Laufzeit liefert polynomielle Gesamtlaufzeit.
- $\mathcal{P}$  beinhaltet praktische und theoretisch interessante Probleme.
- Probleme ausserhalb von  $\mathcal{P}$  sind in der Praxis oft nur für kleine Instanzen oder approximativ lösbar.



# Kodierung der Eingabe

- Erinnerung: Zeitkomplexität  $T_M(n)$  ist eine Funktion in  $|w| = n$ .
- Benötigen geeignete Kodierung der Eingabe  $w$ .
- Kodierung einer Zahl  $n \in \mathbb{N}$ 
  - ▶ Verwenden Binärkodierung  $\text{bin}(n)$  mit Eingabelänge  $\Theta(\log n)$ .
- Kodierung eines Graphen  $G = (V, E)$ 
  - ▶ Kodieren Knotenanzahl  $n$  unär, d.h.  $|V| = n$ .
  - ▶  $m$  Kanten mit Adjazenzliste  $|E| = m$  oder Adjazenzmatrix  $|E| = \Theta(n^2)$ .

## Bsp:

- PFAD :=  $\{(G, s, t) \mid G \text{ ist Graph mit Pfad von } s \text{ nach } t.\} \in \mathcal{P}$ .
  - ▶ Starte Breitensuche in  $s$ .
  - ▶ Falls  $t$  erreicht wird, akzeptiere. Sonst lehne ab.
  - ▶ Laufzeit  $\mathcal{O}(|V| + |E|)$ , d.h. linear in der Eingabelänge von  $G$ .
- TEILERFREMD :=  $\{(x, y) \mid \text{gcd}(x, y) = 1\} \in \mathcal{P}$ .
  - ▶ Berechne mittels Euklidischem Algorithmus  $d = \text{gcd}(x, y)$ .
  - ▶ Falls  $d = 1$ , akzeptiere. Sonst lehne ab.
  - ▶  $\mathcal{O}(\log^2(\max\{x, y\}))$ , quadratisch in  $|x| = \Theta(\log x)$ ,  $|y| = \Theta(\log y)$ .

# Optimierungsvariante vs Entscheidungsvariante

RUCKSACK<sub>opt</sub>

- Gegeben:  $n$  Gegenstände mit Gewichten  $W = \{w_1, \dots, w_n\} \subset \mathbb{N}$  und Profiten  $P = \{p_1, \dots, p_n\} \subset \mathbb{N}$ , Kapazität  $b \in \mathbb{N}$ .
- Gesucht:  $I \subseteq [n] : \sum_{i \in I} w_i \leq B$ , so dass  $\sum_{i \in I} p_i$  maximiert wird.

Sprache RUCKSACK:

RUCKSACK :=  $\{(W, P, b, k) \mid \exists I \subseteq [n] : \sum_{i \in I} w_i \leq B \text{ und } \sum_{i \in I} p_i \geq k\}$ .

## Naiver Algorithmus zum Entscheiden von RUCKSACK

- 1 Für alle  $I \subseteq [n]$ :
    - 1 Falls  $\sum_{i \in I} w_i \leq B$  und  $\sum_{i \in I} p_i \geq k$ , akzeptiere.
  - 2 Lehne ab.
- Prüfung von  $2^n$  vielen Untermengen in Schritt 1.
  - Eingabegrößen:  $\log w_i, \log p_i, n, \log b, \log k$ .
  - D.h. die Gesamtlaufzeit ist exponentiell im Eingabeparameter  $n$ .
  - Prüfung *einzelner potentieller Lösungen* in Schritt 1.1 ist effizient.

# Polynomielle Verifizierer und NP

## Definition Polynomieller Verifizierer

Sei  $L \subseteq \Sigma^*$  eine Sprache. Eine DTM  $V$  heißt *Verifizierer für  $L$* , falls  $V$  für alle Eingaben  $w \in \Sigma^*$  hält und folgendes gilt:

$$w \in L \Leftrightarrow \exists c \in \Sigma^* : V \text{ akzeptiert Eingabe } (w, c).$$

Das Wort  $c$  nennt man einen *Zeugen oder Zertifikat für  $w$* .

$V$  heißt *polynomieller Verifizierer für  $L$* , falls  $V$  für alle  $w \in \Sigma^*$  in Laufzeit polynomiell in  $|w|$  hält und folgendes gilt:

$$w \in L \Leftrightarrow \exists c \in \Sigma^*, |c| \leq |w|^k, k \in \mathbb{N} : V \text{ akzeptiert Eingabe } (w, c).$$

$L$  ist *polynomiell verifizierbar*  $\Leftrightarrow \exists$  polynomieller Verifizierer für  $L$ .

## Definition Klasse $\mathcal{NP}$

$$\mathcal{NP} := \{L \mid L \text{ ist polynomiell verifizierbar.}\}$$

# Polynomieller Verifizierer für RUCKSACK

## Satz

RUCKSACK  $\in \mathcal{NP}$ .

## Beweis:

## Algorithmus Polynomieller Verifizierer für RUCKSACK

Eingabe:  $(W, P, b, k, c)$  mit Zeuge  $c = I \subseteq [n]$

- 1 Falls  $\sum_{i \in I} w_i \leq b$  und  $\sum_{i \in I} p_i \geq k$ , akzeptiere.
- 2 Lehne ab.

## Laufzeit:

- Eingabegrößen:  $\log w_i, \log p_i, \log b, \log k, n$
- Laufzeit:  $\mathcal{O}(n \cdot \log(\max_i \{w_i, p_i, b, k\}))$  auf RAM.
- D.h. die Laufzeit ist polynomiell in den Eingabegrößen.

# Optimaler Wert einer Lösung mittels Entscheidung

RUCKSACK<sub>wert</sub>

- Gegeben:  $W = \{w_1, \dots, w_n\}$ ,  $P = \{p_1, \dots, p_n\}$  und  $B$ .
- Gesucht:  $\max_{I \subseteq [n]} \{ \sum_{i \in I} p_i \mid \sum_{i \in I} w_i \leq b \}$

Sei  $M$  eine DTM, die RUCKSACK in Laufzeit  $T(M)$  entscheide.

## Algorithmus OPTIMUM

Eingabe:  $W, P, B$

- 1  $l \leftarrow 0, r \leftarrow \sum_{i=1}^n p_i$
- 2 WHILE ( $l \neq r$ )
  - 1 Falls  $M$  bei Eingabe  $(W, P, b, \lceil \frac{l+r}{2} \rceil)$  akzeptiert,  $l \leftarrow \lceil \frac{l+r}{2} \rceil$ .
  - 2 Sonst  $r \leftarrow \lceil \frac{l+r}{2} \rceil - 1$ .

Ausgabe:  $l$

- Korrektheit: Binäre Suche nach Optimum auf Intervall  $[0, \sum_{i=1}^n p_i]$ .
- Laufzeit:  $\mathcal{O}(\log(\sum_{i=1}^n p_i)) \cdot T(M)$ .
- Insbesondere: Laufzeit ist polynomiell, falls  $T(M)$  polynomiell ist.

# Optimale Lösung mittels optimalem Wert

**Ziel:** Bestimme Lösung  $I \subseteq [n]$  mit optimalem Wert.

## Algorithmus Optimale Lösung

Eingabe:  $W, P, b$

- 1  $opt \leftarrow \text{OPTIMUM}(W, P, b), I \leftarrow \emptyset$
- 2 For  $i \leftarrow 1$  to  $n$ 
  - 1 Falls  $(\text{OPTIMUM}(W \setminus \{w_i\}, P \setminus \{p_i\}, b) = opt$ ,  
setze  $W \leftarrow W \setminus \{w_i\}, P \leftarrow P \setminus \{p_i\}$ .
  - 2 Sonst  $I \leftarrow I \cup \{i\}$ .

Ausgabe:  $I$

## Korrektheit:

- Invariante vor  $i$ -tem Durchlauf:  $\exists J \subseteq \{1, \dots, i-1\}: I \cup J$  ist optimal.
- $i$  wird nur dann in  $I$  aufgenommen, falls  $I$  zu optimaler Teilmenge erweitert werden kann.
- **Laufzeit:**  $\mathcal{O}(n \cdot T(\text{OPTIMUM})) = \mathcal{O}(n \cdot \log(\sum_{i=1}^n p_i) \cdot T(M))$ .
- D.h. Laufzeit ist polynomiell, falls  $T(M)$  polynomiell ist.

# Sprache Zusammengesetzt

ZUSAMMENGESSETZT :=  $\{N \in \mathbb{N} \mid N = pq \text{ mit } p, q \geq 2\}$

## Satz

ZUSAMMENGESSETZT  $\in \mathcal{NP}$ .

## Beweis:

## Algorithmus Polynomieller Verifizierer für ZUSAMMENGESSETZT

Eingabe:  $(N, c)$  mit  $c = (p, q) \in \{2, \dots, N-1\}^2$

- 1 Berechne  $p \cdot q$ . Falls  $p \cdot q = N$ , akzeptiere. Sonst lehne ab.

## Laufzeit:

- Eingabelänge:  $|N| = \Theta(\log N)$
- Laufzeit:  $\mathcal{O}(\log^2 N)$ , d.h. polynomiell in der Eingabelänge.

# $\mathcal{P}$ versus $\mathcal{NP}$

## Satz

$$\mathcal{P} \subseteq \mathcal{NP}.$$

- $L \in \mathcal{P} \Rightarrow \exists$  DTM  $M$ , die  $L$  in polynomieller Laufzeit entscheidet.
- $\Rightarrow \exists$  DTM  $M$ , die stets hält und genau die Eingaben  $w \in L$  in Laufzeit polynomiell in  $|w|$  akzeptiert.
- $\Rightarrow \exists$  DTM  $V$ , die stets hält und genau die Eingaben  $(w, c)$  mit  $w \in L, c = \epsilon$  in Laufzeit polynomiell in  $|w|$  akzeptiert.  
Dabei ignoriert  $V$  die Eingabe  $c$  und wendet  $M$  auf  $w$  an.
- $\Rightarrow L \in \mathcal{NP}$ .

- **Großes offenes Problem:** Gilt  $\mathcal{P} = \mathcal{NP}$  oder  $\mathcal{P} \subset \mathcal{NP}$ ?



# Nichtdeterministische Turingmaschinen

Wir bezeichnen mit  $\mathcal{P}(S)$  die Potenzmenge einer Menge  $S$ .

## Definition Nichtdeterministische Turingmaschine

Eine *nicht-deterministische Turingmaschine (NTM)* ist ein Tupel  $(Q, \Sigma, \Gamma, \delta)$ , wobei

- $Q, \Sigma, \Gamma$  sind wie bei DTM definiert.
- $\delta : Q \setminus \{q_a, q_r\} \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$
  
- Bsp:  $\delta(q, a) = \{(q_1, a_1, L), (q_2, a_2, R)\}$ .
- NTM besitzt Wahlmöglichkeiten für den Zustandsübergang.
- Beschränken uns oBdA auf NTMs mit  $\leq 2$  Wahlmöglichkeiten.

# Berechnungsbaum

- Seien die Konfigurationen einer NTM Knoten in einem Berechnungsbaum.
  - ▶ Die Startkonfiguration bildet den Wurzelknoten.
  - ▶ Mögliche Nachfolgekongfigurationen bilden Kinderknoten.
- Pfade heißen Berechnungspfade der NTM.
- Betrachten nur NTMs mit Berechnungspfaden endlicher Länge.
- Ein Berechnungspfad heißt akzeptierend, falls er in  $q_a$  endet.

## Definition Akzeptierte Sprache einer NTM

Sei  $N$  eine NTM.

- $N$  akzeptiert Eingabe  $w \Leftrightarrow \exists$  akzeptierenden Berechnungspfad im Berechnungsbaum von  $N$  bei Eingabe  $w$ .
- Die von  $N$  akzeptierte Sprache  $L(N)$  ist definiert als
$$L(N) = \{w \in \Sigma^* \mid N \text{ akzeptiert die Eingabe } w.\}$$

# Die Laufzeit einer NTM

## Definition Laufzeit einer NTM

Sei  $N$  eine DTM mit Eingabe  $w$ .

- $T_N(w) :=$  **maximale** Anzahl Rechenschritte von  $N$  auf  $w$ , d.h.  $T_N(w)$  ist die Länge eines längsten Berechnungspfades.
- $T_N : \mathbb{N} \rightarrow \mathbb{N}$ ,  $T_N(n) := \max\{T_N(w) \mid w \in \Sigma^{\leq n}\}$  heißt *Laufzeit* oder *Zeitkomplexität* von  $N$ .
- Wir definieren die Klasse NTIME für NTMs analog zur Klasse DTIME für DTMs.

## Definition NTIME

Sei  $t : \mathbb{N} \rightarrow \mathbb{N}$  eine monoton wachsende Funktion.

$\text{NTIME}(t(n)) := \{L \mid L \text{ wird von NTM in Laufzeit } \mathcal{O}(t(n)) \text{ entschieden.}\}$

# NTM, die RUCKSACK entscheidet

## Algorithmus NTM für RUCKSACK

Eingabe:  $W, P, b, k$

- 1 Erzeuge nichtdeterministisch einen Zeugen  $I \subseteq [n]$ .
  - 2 Falls  $\sum_{i \in I} w_i \leq b$  und  $\sum_{i \in I} p_i \geq k$ , akzeptiere.
  - 3 Sonst lehne ab.
- D.h. NTM erzeugt sich im Gegensatz zum Verifizierer ihren Zeugen  $I$  selbst.
  - Laufzeit: Schritt 1:  $\mathcal{O}(n)$ , Schritt 2:  $\mathcal{O}(n \cdot \log(\max\{w_i, p_i, b, k\}))$ .
  - D.h. die Laufzeit ist polynomiell in der Eingabelänge.

# $\mathcal{NP}$ mittels NTMs

## Satz

$\mathcal{NP}$  ist die Klasse aller Sprachen, die von einer NTM in polynomieller Laufzeit entschieden wird, d.h.

$$\mathcal{NP} = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k).$$

Zeigen:

- $\exists$  polynomieller Verifizierer für  $L$
- $\Leftrightarrow \exists$  NTM  $N$ , die  $L$  in polynomieller Laufzeit entscheidet.

# Verifizierer $\Rightarrow$ NTM

" $\Rightarrow$ ": Sei  $V$  ein Verifizierer für  $L$  mit Laufzeit  $\mathcal{O}(n^k)$  für ein festes  $k$ .

## Algorithmus NTM $N$ für $L$

Eingabe:  $w$  mit  $|w| = n$ .

- 1 Erzeuge nicht-deterministisch einen Zeugen  $c$  mit  $|c| = \mathcal{O}(n^k)$ .
- 2 Simuliere  $V$  mit Eingabe  $(w, c)$ .
- 3 Falls  $V$  akzeptiert, akzeptiere. Sonst lehne ab.

- Korrektheit:  
 $w \in L \Leftrightarrow \exists c \text{ mit } |c| = \mathcal{O}(n^k) : V \text{ akzeptiert } (w, c) \text{ in Zeit } \mathcal{O}(n^{k'})$ .  
 $\Leftrightarrow N \text{ akzeptiert die Eingabe } w \text{ in Laufzeit } \mathcal{O}(n^{\max\{k, k'\}})$ .
- Damit entscheidet  $N$  die Sprache  $L$  in polynomieller Laufzeit.

# NTM $\Rightarrow$ Verifizierer

" $\Leftarrow$ ": Sei  $N$  eine NTM, die  $L$  in Laufzeit  $\mathcal{O}(n^k)$  entscheidet.

## Algorithmus Verifizierer

Eingabe:  $w, c$

- 1  $c$  ist Kodierung eines Berechnungspfades von  $N$  bei Eingabe  $w$ .
- 2 Simuliere  $N$  auf Eingabe  $w$  auf dem Berechnungspfad  $c$ .
- 3 Falls  $N$  akzeptiert, akzeptiere. Sonst lehne ab.

Korrektheit:

$$\begin{aligned} w \in L &\Leftrightarrow \exists \text{ akzeptierender Berechnungspfad } c \text{ von } N \text{ f\u00fcr } w \\ &\Leftrightarrow V \text{ akzeptiert } (w, c). \end{aligned}$$

Laufzeit:

- L\u00e4ngster Berechnungspfad von  $N$  besitzt L\u00e4nge  $\mathcal{O}(n^k)$ .
- D.h. die Gesamtlaufzeit von  $V$  ist ebenfalls  $\mathcal{O}(n^k)$ .

# Boolesche Formeln

## Definition Boolesche Formel

- Eine Boolesche Variable  $x_i$  kann Werte aus  $\{0, 1\}$  annehmen, wobei  $0 \cong$  falsch und  $1 \cong$  wahr.
- Jede Boolesche Variable  $x_i$  ist eine Boolesche Formel.
- Sind  $\phi, \phi'$  Boolesche Formeln, so auch  $\neg\phi, \phi \wedge \phi', \phi \vee \phi', (\phi)$ .
- Wir ordnen die Operatoren nach absteigender Priorität:  $()$ ,  $\neg$ ,  $\wedge$ ,  $\vee$ .
- $\phi$  ist erfüllbar  $\Leftrightarrow \exists$  Belegung der Variablen in  $\phi$ , so dass  $\phi = 1$ .

## Bsp:

- $\phi = \neg(x_1 \vee x_2) \wedge x_3$  ist erfüllbar mit  $(x_1, x_2, x_3) = (0, 0, 1)$ .
- $\phi' = x_1 \wedge \neg x_1$  ist eine nicht-erfüllbare Boolesche Formel.



# Satisfiability SAT

## Definition SAT

$\text{SAT} := \{\phi \mid \phi \text{ ist eine erfüllbare Boolesche Formel.}\}$

Kodierung von  $\phi$ :

- Kodieren Variable  $x_i$  durch  $\text{bin}(i)$ .
- Kodieren  $\phi$  über dem Alphabet  $\{0, 1, (, ), \neg, \wedge, \vee\}$ .

# SAT ist polynomiell verifizierbar.

## Satz

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

## Beweis

### Algorithmus Polynomieller Verifizierer

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

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

Korrektheit:

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

Laufzeit:

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

# Konjunktive Normalform

## Definition Konjunktive Normalform (KNF)

Seien  $x_1, \dots, x_n$  Boolesche Variablen und  $\phi$  eine Boolesche Formel.

- *Literale* sind Ausdrücke der Form  $x_i$  und  $\neg x_i$ .
- *Klauseln* sind disjunktive Verknüpfungen von Literalen.
- $\phi$  ist in KNF, falls  $\phi$  eine Konjunktion von Klauseln ist.
- Eine KNF Formel  $\phi$  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 :=  $\{\phi \mid \phi \text{ ist eine erfüllbare 3-KNF Boolesche Formel.}\}$

- Offenbar gilt  $3\text{SAT} \subset \text{SAT}$ .

## Satz

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

## Beweis

### Algorithmus NTM für 3SAT

Eingabe:  $\phi(x_1, \dots, x_n) \in 3\text{-KNF}$

- 1 Rate nicht-deterministisch eine Belegung  $(c_1, \dots, c_n) \in \{0, 1\}^n$ .
- 2 Falls  $\phi(c_1, \dots, c_n) = 1$ , 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  $\Sigma$  ein Alphabet und  $f : \Sigma^* \rightarrow \Sigma^*$ . Eine 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.

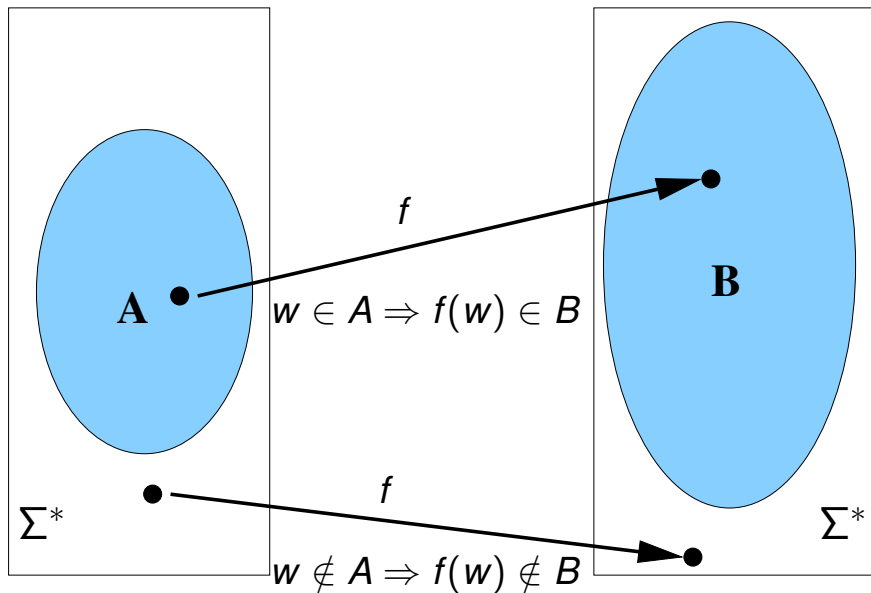
## Definition Polynomielle Reduktion

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

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

Wir schreiben  $A \leq_p B$  und bezeichnen  $f$  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:

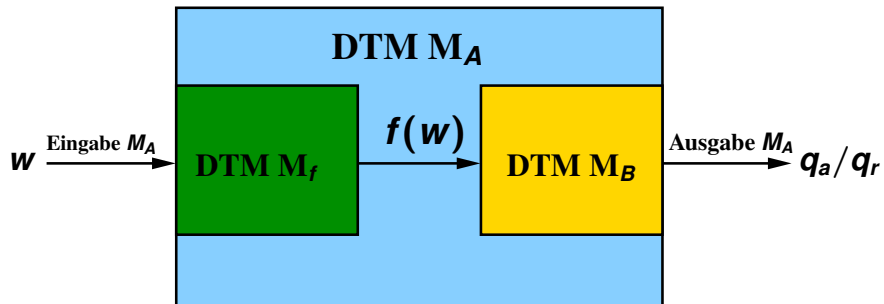
- $M_A$  akzeptiert  $w \Leftrightarrow M_B$  akzeptiert  $f(w) \Leftrightarrow f(w) \in B \Leftrightarrow w \in A$ .

### 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

- $\phi$  erfüllbar  $\Leftrightarrow \exists$  erfüllende Belegung  $B$  für  $\phi$ .  
 $\Leftrightarrow B$  setzt in jeder der  $n$  Klauseln mind. ein Literal wahr.  
 $\Leftrightarrow$  Wahre Literale entsprechen einer  $n$ -Clique in  $G$ .

# Die Reduktion $f$

## Algorithmus $M_f$ für $f$

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

- 1 Wahl der Knotenmenge  $V$  von  $G$ 
  - ▶ Definiere  $3n$  Knoten mit Labeln  $a_{i1}, a_{i2}, a_{i3}$  für  $i = 1, \dots, n$ .
- 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  $k = n = \frac{|V|}{3}$ , 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:  $\phi \in 3SAT \Rightarrow f(\phi) = (G, k) \in CLIQUE$

- Sei  $\phi \in 3SAT$ . Dann besitzt  $\phi$  eine erfüllende Belegung  $B$ .
- Damit setzt  $B$  in jeder Klausel  $(a_{i1} \vee a_{i2} \vee a_{i3})$ ,  $i = 1, \dots, n$  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.  $f(\phi) = (G, n) \in CLIQUE$

# Korrektheit von $f$ : Rückrichtung

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

- Sei  $f(\phi) = (G, n) \in \text{CLIQUE}$ . Dann besitzt  $G$  eine  $n$ -Clique  $v_1, \dots, v_n$ .
- Nach Konstruktion der Kantenmenge von  $E$  gilt:
  - 1  $v_1, \dots, v_n$  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  $v_1, \dots, v_n$  wahr setzt.
  - 1  $B$  setzt in jeder Klausel ein Literal  $v_i$  auf wahr.
  - 2  $B$  ist eine konsistente Belegung.
- Damit ist  $B$  eine erfüllende Belegung für  $\phi$ .
- D.h.  $\phi \in \text{3SAT}$ .

# $\mathcal{NP}$ -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.

# Satz von Cook-Levin (1971)

## Satz von Cook-Levin

SAT ist  $\mathcal{NP}$ -vollständig.

**Beweis:** Müssen zeigen

- 1 SAT  $\in \mathcal{NP}$  (bereits gezeigt)
- 2 Für alle  $L \in \mathcal{NP}$  existiert polynomiell berechenbare Reduktion  $f$ :

$$w \in L \Leftrightarrow f(w) \in \text{SAT}.$$

**Beweisidee:** Sei  $L \in \mathcal{NP}$  beliebig.

- $\exists$  NTM  $N$  mit polynomieller Laufzeit  $n^k$  mit
$$w \in L \Leftrightarrow N \text{ akzeptiert } w.$$
- Konstruieren aus  $(N, w)$  eine Formel  $\phi$  mit
  - 1  $N$  akzeptiert  $w \Leftrightarrow f(w) = \phi \in \text{SAT}$
  - 2  $f$  ist in Zeit polynomiell in  $|w| = n$  berechenbar.
- Betrachten dazu  $(n^k + 1) \times (n^k + 1)$  Berechnungstabelle von  $N$ .

## Berechnungstabelle $T$ von $N$ auf $w$

$q_0$	$\triangleright$	$w_1$	$\dots$	$w_n$	$\sqcup$	$\dots$	$\sqcup$
$\triangleright$	$q_i$	$w_1$	$\dots$	$w_n$	$\sqcup$	$\dots$	$\sqcup$
		$\vdots$				$\vdots$	

- Tabelle  $T$  entspricht einem Pfad im Berechnungsbaum.
- Erste Zeile enthält die Startkonfiguration.
- $(i + 1)$ -te Zeile ist mögliche Nachfolgekonfiguration der  $i$ -ten Zeile.
- In Laufzeit  $n^k$  können höchstens  $n^k$  Zellen besucht werden.
- $T$  akzeptierend  $\Leftrightarrow T$  enthält eine akzeptierende Konfiguration.
- Konstruieren  $\phi$  derart, dass  $\phi$  erfüllbar ist gdw.  $T$  akzeptierend ist.

# Struktur der Formel für $\phi$

- Sei  $T(i, j)$  der Eintrag in der  $i$ -ten Zeile und  $j$ -ten Spalte von  $T$ .
- $T(i, j) \in Q \cup \Gamma$  für alle  $i, j$ .
- Definieren  $\phi$  über den Booleschen Variablen  $x_{i,j,s}$  mit

$$x_{i,j,s} = 1 \Leftrightarrow T(i, j) = s \quad \text{für } s \in Q \cup \Gamma.$$

Formel für  $\phi$ :  $\phi = \phi_{Start} \wedge \phi_{accept} \wedge \phi_{Eintrag} \wedge \phi_{move}$  mit

$\phi_{Start}$ :  $T$  beginnt mit Startkonfiguration.

$\phi_{accept}$ :  $T$  muss Eintrag  $q_a$  besitzen.

$\phi_{Eintrag}$ :  $T$  enthält Einträge aus  $Q \cup \Gamma$ .

$\phi_{move}$ :  $T$  besitzt gültige Nachfolgekonfigurationen.

## Definition von $\phi_{Start}$ , $\phi_{accept}$ und $\phi_{Eintrag}$

$\phi_{Start}$ : Kodieren die Startkonfiguration  $q_0 \triangleright w_1 \dots w_n$

$$x_{1,1,q_0} \wedge x_{1,2,\triangleright} \wedge x_{1,3,w_1} \wedge \dots \wedge x_{1,n+2,w_n} \wedge x_{1,n+3,\sqcup} \wedge \dots \wedge x_{1,n^k+1,\sqcup}$$

$\phi_{accept}$ :  $\phi$  ist erfüllend gdw  $T$  eine erfüllende Konfiguration enthält

$$\phi_{accept} = \bigvee_{1 \leq i, j \leq n^k+1} x_{i,j,q_a}$$

$\phi_{Eintrag}$ :  $T(i, j) \in Q \cup \Gamma$ , d.h. es gibt ein  $s \in Q \cup \Gamma$  mit  $x_{i,j,s} = 1$ .

- $T(i, j)$  enthält mindestens einen Eintrag  $s \in Q \cup \Gamma$ :

$$\phi_{\geq 1} = \bigvee_{s \in Q \cup \Gamma} x_{i,j,s}$$

- $T(i, j)$  enthält höchstens einen Eintrag  $s \in Q \cup \Gamma$ :

$$\phi_{\leq 1} = \bigwedge_{s, t \in Q \cup \Gamma, s \neq t} \neg(x_{i,j,s} \wedge x_{i,j,t})$$

- Liefert insgesamt  $\phi_{Eintrag} = \bigwedge_{1 \leq i, j \leq n^k+1} (\phi_{\geq 1} \wedge \phi_{\leq 1})$ .

## Definition von $\phi_{move}$

**Ziel:** Zeile  $i + 1$  muss Nachfolgekonfiguration von Zeile  $i$  sein.

- Definieren Fenster  $F$  der Größe  $2 \times 3$ .
- $(i, j)$ -Fenster besitzt Einträge  $(i, j - 1)$ ,  $(i, j)$ ,  $(i, j + 1)$  und  $(i + 1, j - 1)$ ,  $(i + 1, j)$ ,  $(i + 1, j + 1)$ .
- Tabelle  $T$  besitzt  $(i, j)$ -Fenster für  $i = 1, \dots, n^k$ ,  $j = 2, \dots, n^k$ .
- Fenster  $F$  heißt legal gwd F's Einträge  $\delta$  nicht widersprechen.

# Beispiele für legale Fenster

Sei  $\delta$  wie folgt definiert

- $\delta(q_1, a) = \{(q_1, b, R)\}$ .
- $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$ .

a	$q_1$	b
$q_2$	a	c

**legal**

a	$q_1$	b
a	a	$q_2$

**legal**

a	b	b
a	a	b

**nicht legal**

a	a	$q_1$
a	a	b

**legal**

a	$q_1$	b
$q_1$	a	a

**nicht legal**

$\sqcup$	b	a
$\sqcup$	b	a

**legal**

a	$q_1$	b
$q_2$	b	$q_2$

**nicht legal**

a	b	a
a	b	$q_2$

**legal**

b	b	b
c	b	b

**legal**

# Korrektheit der Konstruktion

## Lemma Korrektheit Berechnungstabelle

Sei  $T$  eine Tabelle mit den folgenden Eigenschaften.

- 1 Die erste Zeile ist die Startkonfiguration von  $N$  auf  $w$ .
- 2 Jedes Fenster ist legal.

Dann ist  $T$  eine Berechnungstabelle von  $N$  auf Eingabe  $w$ .

### Beweis:

- $T(i, j) \neq T(i + 1, j)$  ist nur dann möglich, falls einer der Einträge  $T(i, j - 1)$ ,  $T(i, j)$  oder  $T(i, j + 1)$  einen Zustand enthält.
- Falls die obere Zeile einen Zustand ändert, muss sich die untere Zeile gemäß  $\delta$  ändern.
- D.h. jede Zeile ist eine Nachfolgekonfiguration der Vorgängerzeile.
- Damit ist  $T$  eine Berechnungstabelle.



# Konstruktion von $\phi_{move}$

- Informal gilt:  $\phi_{move} = \bigwedge_{1 \leq i \leq n^k, 2 \leq j \leq n^k}$  Fenster  $(i, j)$  ist legal.
- Die Anzahl legaler Fenster hängt nur von den möglichen Übergängen in  $N$  ab, nicht von der Eingabe  $w$ .
- D.h. es gibt eine Menge  $F$  von 6-Tupeln  $(f_1, \dots, f_6)$ , so dass  $F$  alle legalen Fenster beschreibt.
- Damit können wir das Prädikat [Fenster  $(i, j)$  ist legal] formalisieren

$$\bigvee_{(f_1, \dots, f_6) \in F} (x_{i, j-1, f_1} \wedge x_{i, j, f_2} \wedge x_{i, j+1, f_3} \wedge x_{i+1, j-1, f_4} \wedge x_{i+1, j, f_5} \wedge x_{i+1, j+1, f_6}).$$

# Reduktion ist polynomiell

## Lemma Länge von $\phi$

Sei  $N$  eine NTM mit Laufzeit  $n^k$  bei Eingabe  $w$ ,  $|w| = n$ . Dann besitzt die Formel  $\phi = \phi_{Start} \wedge \phi_{accept} \wedge \phi_{Eintrag} \wedge \phi_{move}$  Länge  $\mathcal{O}(n^{2k})$ , d.h. ihre Länge ist polynomiell in  $n$ .

Zudem ist  $\phi$  bei Eingabe  $(N, w)$  in Zeit  $\mathcal{O}(n^{2k})$  berechenbar.

- $\phi_{Start}$ :
  - Anzahl Literale:  $\mathcal{O}(n^k)$ , Berechnung direkt aus  $w$
- $\phi_{accept}$ :
  - Anzahl Literale:  $\mathcal{O}(n^{2k})$
- $\phi_{Eintrag}$ :
  - Anzahl Literale in  $\phi_{\geq 1}, \phi_{\leq 1}$ :  $\mathcal{O}(1)$ , unabhängig von  $w$ .
  - Anzahl Literale in  $\phi_{Eintrag}$ :  $\mathcal{O}(n^{2k})$ .
- $\phi_{move}$ :
  - Anzahl legaler Fenster  $|F|$ :  $\mathcal{O}(1)$ , unabhängig von  $w$ .
  - Anzahl Literale in  $\phi_{move}$ :  $\mathcal{O}(n^{2k})$ .

# Von SAT zu 3SAT

## Satz

3SAT ist  $\mathcal{NP}$ -vollständig.

- Modifizieren zunächst vorigen Beweis derart, dass  $\phi$  in KNF ist.
- $\phi_{start}$  und  $\phi_{accept}$  sind bereits in KNF.
- $\phi_{Eintrag} = \bigwedge_{i,j} (\phi_{\geq 1} \wedge \phi_{\leq 1}) = \bigwedge_{i,j} \phi_{\geq 1} \wedge \bigwedge_{i,j} \phi_{\leq 1}$ 
  - ▶  $\phi_{\geq 1}$  besteht aus einer Klausel.
  - ▶ Schreiben  $\phi_{\leq 1}$  als Konjunktion von Klauseln:

$$\phi_{\leq 1} = \bigwedge_{s \neq t} (\neg x_{i,j,s} \vee \neg x_{i,j,t}).$$

- $\phi_{move}$ : Wandle disjunktive Normalform des Prädikats für legale Fenster

$$\bigvee_{(f_1, \dots, f_6) \in F} (x_{i,j-1,f_1} \wedge x_{i,j,f_2} \wedge \dots \wedge x_{i+1,j+1,f_6}).$$

in KNF um. Umwandlung in  $\mathcal{O}(1)$ , da  $|F|$  unabhängig von  $|w| = n$ .

## Umwandlung von KNF in 3-KNF

Sei  $\phi = k_1 \wedge \dots \wedge k_m$  eine KNF-Formel, wobei  $k_j = a_1 \vee \dots \vee a_n$  eine Klausel mit  $n > 3$  Literalen ist.

- Führen neue Variablen  $z_1, \dots, z_{n-3}$  ein.
- Ersetzen Klausel  $k_j$  durch die 3-KNF Formel

$$k'_j = (a_1 \vee a_2 \vee z_1) \wedge (\neg z_1 \vee a_3 \vee z_2) \wedge (\neg z_2 \vee a_4 \vee z_3) \wedge \dots \wedge (\neg z_{n-3} \vee a_{n-1} \vee a_n)$$

- zu zeigen:  $k_j$  ist erfüllbar gdw.  $k'_j$  erfüllbar ist.
- $B$  ist eine erfüllende Belegung für  $k_j$  gdw ein Literal  $a_i$  wahr ist.
- Dann ist aber  $k'_j$  erfüllbar mit  $a_i = 1$  und

$$z_j = 1 \text{ für } j < i - 1 \quad \text{und} \quad z_j = 0 \text{ für } j \geq i - 1.$$

- Sei andererseits  $k'_j$  erfüllbar.
- Dann muss ein Literal  $a_i$  wahr sein, und damit ist  $k$  erfüllbar.
- Können  $\phi$  in KNF bzw. in 3-KNF in  $\mathcal{O}(|\phi|)$  Schritten umwandeln.

# $\mathcal{NP}$ -Vollständigkeit von CLIQUE

## Satz

CLIQUE ist  $\mathcal{NP}$ -vollständig.

**Beweis:** zu zeigen

- 1 CLIQUE  $\in \mathcal{NP}$ 
  - ▶ Übung
- 2  $\exists \mathcal{NP}$ -vollständige Sprache  $L$  mit  $L \leq_p$  CLIQUE
  - ▶ Bereits gezeigt: 3SAT ist  $\mathcal{NP}$ -vollständig.
  - ▶ Bereits gezeigt: 3SAT  $\leq_p$  CLIQUE.

# Knotenüberdeckung

## Definition $k$ -Knotenüberdeckung

Sei  $G = (V, E)$  ein ungerichteter Graph. Eine Knotenmenge  $U \subseteq V$ ,  $|U| = k$  heißt  $k$ -Knotenüberdeckung, falls

$$e \cap U \neq \emptyset \text{ für alle } e \in E.$$

Wir definieren die folgende Sprache.

KNOTENÜBERDECKUNG :=  $\{(G, k) \mid G \text{ besitzt eine } k\text{-Knotenüberdeckung.}\}$

## Satz

KNOTENÜBERDECKUNG ist  $\mathcal{NP}$ -vollständig.

**Beweis:** zu zeigen

- 1 KNOTENÜBERDECKUNG  $\in \mathcal{NP}$  (Übung)
- 2 3-SAT  $\leq_p$  KNOTENÜBERDECKUNG, d.h. es gibt berechenbares  $f$ :  
 $\phi \in 3\text{SAT} \Leftrightarrow f(\phi) = (G, k) \in \text{KNOTENÜBERDECKUNG}$

# Die Reduktion $f$

## Idee der Reduktion $f$ :

- Konstruieren für jedes Literal  $x_i$  Knotenpaar mit Labeln  $x_i$  und  $\neg x_i$ .
- Knotenlabel einer Überdeckung bilden erfüllende Belegung.

## Algorithmus $M_f$

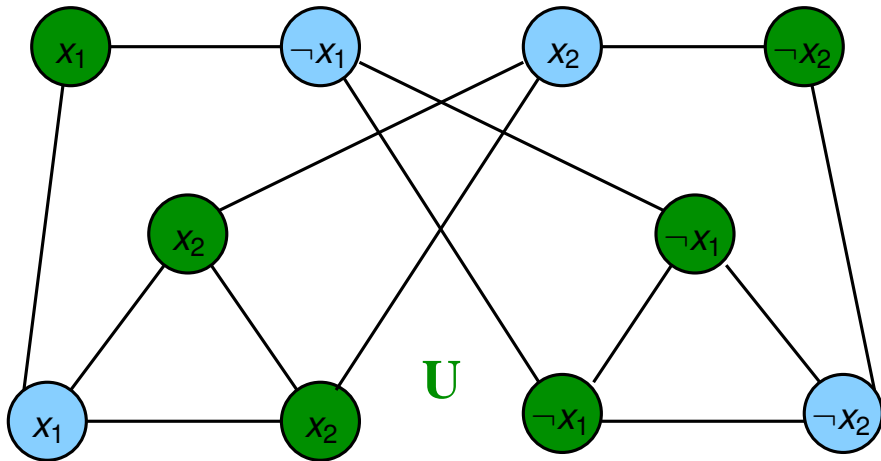
Eingabe:  $\phi(x_1, \dots, x_n) = K_1 \wedge \dots \wedge K_m$  mit  $K_j = \ell_{j1} \vee \ell_{j2} \vee \ell_{j3}$ .

- 1 Variablenknoten: Für  $i = 1 \dots n$ :
  - ▶ Konstruiere zwei verbundene Knoten mit Labeln  $x_i$  und  $\neg x_i$ .
- 2 Klauselknoten: Für  $j = 1 \dots m$ :
  - ▶ Konstruiere 3 paarweise verbundene Knoten mit Labeln  $\ell_{j1}, \ell_{j2}, \ell_{j3}$ .
- 3 Verbinde Variablen- und Klauselknoten mit denselben Labeln.
- 4 Setze  $k = n + 2m$ .

Ausgabe:  $(G, k)$

- Schritt 1:  $\mathcal{O}(n)$ , Schritt 2:  $\mathcal{O}(m)$ , Schritt 3:  $\mathcal{O}(m)$ , Schritt 4:  $\mathcal{O}(1)$ .
- $|\phi| = \mathcal{O}(n + m) = \mathcal{O}(m)$ , d.h. die Laufzeit ist polynomiell in  $|\phi|$ .

Reduktion für  $\phi = (x_1 \vee x_2 \vee x_2) \wedge (\neg x_1 \vee \neg x_1 \vee \neg x_2)$





## $\phi \in 3SAT \Rightarrow f(\phi) \in \text{KNOTENÜBERDECKUNG}$

Sei  $\phi(x_1, \dots, x_n) \in 3SAT$

- Dann gibt es eine erfüllende Belegung der Variablen  $x_1, \dots, x_n$ .
- In die Menge  $U$  werden die folgenden Knoten aufgenommen.
  - ▶  $n$  Variablenknoten:  
Falls  $x_i = 1$ , ist Knoten mit Label  $x_i$  in  $U$ . Sonst Knoten mit  $\neg x_i$ .
  - ▶  $2m$  Klauselknoten:  
Für jede Klausel ist mindestens ein Knoten mit einem Variablenknoten aus  $U$  verbunden. Die *anderen beiden Knoten* sind in  $U$ .
- $U$  ist eine  $n + 2m$ -Knotenüberdeckung:
  - ▶ Die Kanten zwischen Variablenknoten  $x_i, \neg x_i$  sind überdeckt durch einen Variablenknoten.
  - ▶ Kanten zwischen Klauselknoten  $l_{j1}, l_{j2}, l_{j3}$  sind überdeckt durch zwei Klauselknoten.
  - ▶ Kanten zwischen Variablen- und Klauselknoten sind überdeckt: Entweder der Variablenknoten überdeckt die Kante oder einer der beiden Klauselknoten.
- D.h.  $f(\phi) = (G, n + 2m) \in \text{KNOTENÜBERDECKUNG}$

# Korrektheit: Rückrichtung

Sei  $f(\phi) = (G, n + 2m) \in \text{KNOTENÜBERDECKUNG}$ :

- Dann gibt es eine  $(n + 2m)$ -Knotenüberdeckung  $U$  mit:
  - ▶ Mindestens ein Variablenknoten  $x_i$  oder  $\neg x_i$  ist in  $U$  für alle  $i$ .
  - ▶ Mindestens 2 von 3 Klauselknoten  $\ell_{j1}, \ell_{j2}, \ell_{j3}$  sind in  $U$  für alle  $j$ .
  - ▶ Da  $|U| = n + 2m$ :  
Jeweils *genau ein* Variablenknoten und *genau zwei* Klauselknoten.
- Sei  $B$  die Belegung, die die Variablenknoten aus  $U$  auf wahr setzt.
  - ▶  $B$  ist eine konsistente Belegung.
  - ▶ Für alle Klauseln  $K_j$  mit Knoten  $\ell_{j1}, \ell_{j2}, \ell_{j3}$  ist ein  $\ell_{jk}, k \in [3]$  nicht in  $U$ .
  - ▶ Die Kante vom Klausel- zum Variablenknoten mit demselben Label  $\ell_{jk}$  wird überdeckt.
  - ▶ D.h. der Variablenknoten  $\ell_{jk}$  ist in  $U$ . Damit erfüllt  $\ell_{jk}$  die Klausel  $K_j$ .
- D.h.  $B$  ist eine erfüllende Belegung für  $\phi$ .
- Damit gilt  $\phi \in 3\text{SAT}$ .

# Subset Sum

## Definition Sprache SubsetSum

Sei  $M = \{m_1, \dots, m_n\} \subset \mathbb{N}$  und  $t \in \mathbb{N}$ . Wir definieren die Sprache

$$\text{SUBSETSUM} := \{(M, t) \mid \exists S \subseteq M : \sum_{s \in S} s = t\}.$$

## Satz

SUBSETSUM ist  $\mathcal{NP}$ -vollständig.

- 1 SUBSETSUM  $\in \mathcal{NP}$  (Übung)
- 2  $3\text{SAT} \leq_p \text{SUBSETSUM}$

Idee der Reduktion  $f(\phi(x_1, \dots, x_n)) = (S, t)$ : Konstruieren

- für jedes  $x_i$  Elemente  $y_i, z_i \in S$  für  $x_i = 1$  bzw.  $x_i = 0$ ,
- für jede Klausel  $K_j$  Variablen  $g_j, h_j \in S$  für nicht erfüllte Literale.
- Definieren Tabelle  $T$  mit Zeilen  $y_i, z_i, g_j, h_j$  und Zeile  $t$ . Die Spalten bestehen aus  $x_i$  und  $K_j$  für  $i \in [n], j \in [m]$ .
- Einträge in einer Zeile werden als Dezimaldarstellung interpretiert.

# Konstruktion der Reduktion $f$

## Algorithmus $M_f$

EINGABE:  $\phi(x_1, \dots, x_n) = K_1 \wedge \dots \wedge K_m$  mit  $K_j = \ell_{j1} \vee \ell_{j2} \vee \ell_{j3}$

- 1 Erstelle Tabelle  $T$  mit Spalten für  $x_1, \dots, x_n$  und  $K_1, \dots, K_m$ .
- 2 Erstelle  $2n$  Variablenzeilen für  $x_i, i = 1, \dots, n$ :
  - ▶  $y_i$ : Einsen in Spalte  $x_i$ . Für alle Spalten  $K_j$ : Anzahl Literale  $x_i$  in  $K_j$ .
  - ▶  $z_i$ : Einsen in Spalte  $x_i$ . Für alle Spalten  $K_j$ : Anzahl Literale  $\neg x_i$  in  $K_j$ .
- 3 Erstelle  $2m$  Klauselzeilen für  $K_j, j = 1, \dots, m$ :
  - ▶  $g_j, h_j$ : Einsen jeweils in Spalte  $K_j$ .
- 4 Erstelle Zeile  $t$ : Einsen in Spalten  $x_i$ , Dreien in Spalten  $K_j$ .
- 5 Fülle mit Nullen. Definiere  $y_1, z_1, \dots, y_n, z_n, g_1, h_1, \dots, g_m, h_m, t$  mittels des Dezimalwerts der betreffenden Zeile.

AUSGABE:  $(M, t)$  mit  $M = \{y_1, z_1, \dots, y_n, z_n, g_1, h_1, \dots, g_m, h_m\}$ .

## Laufzeit:

- Eingabelänge  $|\phi| \geq \max\{m, n\} = \Omega(m + n)$
- $T(M_f) = \mathcal{O}((n + m)^2)$ , d.h. polynomiell in der Eingabelänge.

Bsp für  $\phi = (x_1 \vee x_2 \vee x_2) \wedge (\neg x_1 \vee x_2 \vee \neg x_2)$

- Definieren Tabelle  $T$

	$x_1$	$x_2$	$K_1$	$K_2$
$y_1$	1	0	1	0
$z_1$	1	0	0	1
$y_2$	0	1	2	1
$z_2$	0	1	0	1
$g_1$	0	0	1	0
$h_1$	0	0	1	0
$g_2$	0	0	0	1
$h_2$	0	0	0	1
$t$	1	1	3	3

- Setze  $y_1 = 1010, z_1 = 1001, \dots, t = 1133$ .
- Belegung  $x_1, x_2 = 1$  erfüllt alle Literale in  $K_1$  und Literal  $x_2$  in  $K_2$ .
- Zahlen  $y_1, y_2$  summieren sich mit  $g_2, h_2$  für  $K_2$  zu  $t$ .

# Korrektheit: $\phi \in 3SAT \Rightarrow f(\phi) \in SUBSETSUM$

Sei  $\phi \in 3SAT$

- Dann besitzt  $\phi$  eine erfüllende Belegung  $B$ .
- Nimm  $y_i$  in  $S$  auf, falls  $x_i = 1$  in  $B$ . Sonst nimm  $z_i$  in  $S$  auf.
- Betrachten  $t' = \sum_{s \in S} s$ :
  - ▶  $B$  ist konsistente Belegung: Obere  $n$  Dezimalstellen von  $t'$  sind 1.
  - ▶  $B$  ist erfüllend: Untere  $m$  Dezimalstellen  $t_1, \dots, t_m$  sind aus  $\{1, 2, 3\}$ .
- Falls  $t_j = 1$ , nimm  $g_j$  und  $h_j$  in  $S$  auf. Falls  $t_j = 2$ , nimm  $g_j$  in  $S$  auf.
- Damit gilt  $\sum_{s \in S} s = t$ .
- D.h.  $f(\phi) = (M, t) \in SUBSETSUM$

# Korrektheit $f(\phi) \in \text{SUBSETSUM} \Rightarrow \phi \in \text{3SAT}$

Sei  $f(\phi) \in \text{SUBSETSUM}$

- Dann gibt es  $S \subseteq M$  mit  $\sum_{s \in S} s = t$ , wobei  $t = 1 \dots 13 \dots 3$ .
- Die oberen  $n$  Dezimalstellen von  $t$  sind 1.
  - ▶ Damit enthält  $S$  für jedes  $i$  genau eines der Elemente  $y_i, z_i$ .
  - ▶ Sei  $B$  die Belegung mit  $x_1 = 1$  für  $y_i \in S$  und  $x_1 = 0$  für  $z_i \in S$ .
- Die unteren  $m$  Dezimalstellen  $t_1, \dots, t_m$  von  $t$  sind 3.
  - ▶ D.h.  $t_j$  kann nicht allein als Summe von  $g_j$  und  $h_j$  dargestellt werden.
  - ▶ Für jedes  $t_j$  kommt mindestens ein Beitrag aus einer der Zeilen  $y_i$  bzw.  $z_i$ .
  - ▶ D.h. das Literal  $x_i$  bzw.  $\neg x_i$  erfüllt die Klausel  $K_j$ .
- Damit ist  $B$  eine erfüllende Belegung für  $\phi$ .
- D.h.  $\phi \in \text{3SAT}$ .

# Das Rucksackproblem

## Definition Sprache Rucksack

Gegeben sind  $n$  Gegenstände mit Gewichten  $W = \{w_1, \dots, w_n\} \subset \mathbb{N}$  und Profiten  $P = \{p_1, \dots, p_n\} \subset \mathbb{N}$ . Seien ferner  $b, k \in \mathbb{N}$ .

$\text{RUCKSACK} := \{(W, P, b, k) \mid \exists I \subseteq [n] : \sum_{i \in I} w_i \leq b \text{ und } \sum_{i \in I} p_i \geq k.\}$

## Satz

RUCKSACK ist  $\mathcal{NP}$ -vollständig.

**Beweis:** zu zeigen

- 1 RUCKSACK  $\in \mathcal{NP}$  (bereits gezeigt)
- 2 SUBSETSUM  $\leq_p$  RUCKSACK



Reduktion  $f(M, t) = (W, P, b, k)$

### Algorithmus $M_f$

EINGABE:  $M, t$

- 1 Setze  $b := t$  und  $k := t$ .
- 2 For  $i = 1$  to  $n$ : Setze  $w_i := m_i$  und  $p_i := m_i$

AUSGABE:  $W, P, b, k$

### Laufzeit:

- Eingabelänge:  $\log(t) + \sum_{i=1}^n \log(m_i)$
- Schritt 1:  $\mathcal{O}(\log t)$ , Schritt 2:  $\mathcal{O}(\sum_{i=1}^n \log(m_i))$
- D.h. Gesamtlaufzeit ist polynomiell in der Eingabelänge.

$(M, t) \in \text{SUBSETSUM} \Leftrightarrow f(M, t) \in \text{RUCKSACK}$

Sei  $(M, t) \in \text{SUBSETSUM}$

- Dann gibt es eine Menge  $I \subseteq [n]$  mit  $\sum_{i \in I} m_i = t$ .
- Damit gilt  $\sum_{i \in I} m_i \leq t$  und  $\sum_{i \in I} m_i \geq t$ .
- Es folgt  $\sum_{i \in I} w_i \leq b$  und  $\sum_{i \in I} p_i \geq k$ .
- Damit gilt  $f(M, t) = (W, P, b, k) \in \text{RUCKSACK}$

Sei  $(W, P, B, k) = f(M, t) \in \text{RUCKSACK}$

- Dann gibt es eine Menge  $I \subseteq [n]$  mit  $\sum_{i \in I} w_i \leq b$  und  $\sum_{i \in I} p_i \geq k$ .
- D.h. es gibt eine Menge  $I \subseteq [n]$  mit  $\sum_{i \in I} m_i \leq t$  und  $\sum_{i \in I} m_i \geq t$ .
- Setze  $S = \{m_i \in M \mid i \in I\}$ . Dann gilt  $S \subseteq M$  und  $\sum_{s \in S} s = t$ .
- Damit ist  $(M, t) \in \text{SUBSETSUM}$

# Exakte Überdeckung

## Definition Exakte Überdeckung

Sei  $U = \{u_1, \dots, u_n\}$  und  $F = \{S_1, \dots, S_m\} \subseteq \mathcal{P}(U)$ , d.h.  $S_i \subseteq U$ .  
Eine Menge  $C \subseteq F$  heißt *exakte Überdeckung* von  $U$  falls

- 1  $\bigcup_{S_i \in C} S_i = U$
- 2  $S_i \cap S_j = \emptyset$  für alle  $S_i, S_j \in C$  mit  $i \neq j$ .

COVER :=  $\{(U, F) \mid F \text{ enthält eine exakte Überdeckung von } U.\}$

## Bsp:

- $U = \{1, 2, 3, 4, 5\}$ ,  $F = \{\{2, 3\}, \{1, 3\}, \{4, 5\}, \{1\}\}$
- $C = \{\{2, 3\}, \{4, 5\}, \{1\}\}$  ist eine exakte Überdeckung von  $U$ .
- $F$  ist *keine* exakte Überdeckung von  $U$ .

# $\mathcal{NP}$ -Vollständigkeit der exakten Überdeckung

## Satz

COVER ist  $\mathcal{NP}$ -vollständig.

Zeigen

- 1 COVER  $\in \mathcal{NP}$  (Übung)
- 2  $3\text{SAT} \leq_p \text{COVER}$

## Idee der Reduktion

- $U$  enthält alle Variablen  $x_i$ , Klauseln  $K_j$  und Literale  $\ell_{jk}$ .
- $F$  enthält geeignete Mengen für Variablen, Klauseln und Literale.

# Reduktion $f(\phi) = (U, F)$

## Algorithmus $M_f$

EINGABE:  $\phi(x_1, \dots, x_n) = K_1 \wedge \dots \wedge K_m$  mit  $K_j = l_{j1} \vee l_{j2} \vee l_{j3}$

- 1 Setze  $U = \{x_1, \dots, x_n, K_1, \dots, K_m, l_{11}, l_{12}, l_{13}, \dots, l_{m1}, l_{m2}, l_{m3}\}$ .
- 2 Definition von  $F$  als Vereinigung der Mengen
  - ▶ Variablen:  $V_{i0} = \{x_i\} \cup \{l_{jk} \mid l_{jk} = x_i\}$  und  
 $V_{i1} = \{x_i\} \cup \{l_{jk} \mid l_{jk} = \neg x_i\}$  für alle  $i, j, k$ .
  - ▶ Klauseln:  $K_{jk} = \{K_j, l_{jk}\}$  für alle  $j \in [m], k \in [3]$ .
  - ▶ Literale:  $L_{jk} = \{l_{jk}\}$  für alle  $j \in [m], k \in [3]$ .

AUSGABE:  $U, F$

## Laufzeit:

- Eingabelänge von  $\phi$  ist  $|\phi| = \Omega(m + n)$
- Schritt 1:  $\mathcal{O}(n + m + |\phi|)$
- Schritt 2: Variablen  $\mathcal{O}(n + |\phi|)$ , Klauseln  $\mathcal{O}(m)$ , Literale  $\mathcal{O}(|\phi|)$ .
- D.h. die Laufzeit ist linear in der Eingabelänge.

## Bsp.: $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$

- $U = \{x_1, x_2, x_3, K_1, K_2, l_{11}, l_{12}, l_{13}, l_{21}, l_{22}, l_{23}\}$
- $V_{i0} : V_{10} = \{x_1, l_{11}\}, V_{20} = \{x_2, l_{12}, l_{22}\}, V_{30} = \{x_3, l_{33}\}$
- $V_{i1} : V_{11} = \{x_1, l_{21}\}, V_{21} = \{x_2\}, V_{31} = \{x_3, l_{13}\}$
- $K_{1k} : K_{11} = \{K_1, l_{11}\}, K_{12} = \{K_1, l_{12}\}, K_{13} = \{K_1, l_{13}\}$
- $K_{2k} : K_{21} = \{K_2, l_{21}\}, K_{22} = \{K_2, l_{22}\}, K_{23} = \{K_2, l_{23}\}$
- $L_{1k} : L_{11} = \{l_{11}\}, L_{12} = \{l_{12}\}, L_{13} = \{l_{13}\}$
- $L_{2k} : L_{21} = \{l_{21}\}, L_{22} = \{l_{22}\}, L_{23} = \{l_{23}\}$
- Erfüllende Belegung von  $\phi: x_1 = 0, x_2 = 1, x_3 = 1$ .

# Korrektheit: $\phi \in 3SAT \Rightarrow f(\phi) = (U, F) \in COVER$

Sei  $\phi(x_1, \dots, x_n) \in 3SAT$

- Dann gibt es eine erfüllende Belegung  $B$  der Variablen  $x_1, \dots, x_n$ .
- $B$  setzt in jeder Klausel  $K_j$  mindestens ein Literal  $\ell_{jk}$  auf wahr.
- Definiere Menge  $C \subseteq F$  mittels  $B$ :
  - ▶ Variablen: Falls  $x_i = 0$ , nimm  $V_{i0}$  in  $C$  auf. Sonst  $V_{i1}$ .
  - ▶ Klauseln: Nimm Menge  $K_{jk}$ , die  $\ell_{jk}$  enthält, in  $C$  auf.
  - ▶ Literale: Für alle nicht von  $C$  abgedeckten  $\ell_{jk'}$ , nimm  $L_{jk'}$  in  $C$  auf.
- $C$  ist eine exakte Überdeckung, denn
  - ▶ Variablen  $x_i$ : Werden durch  $V_{i0}$  oder  $V_{i1}$  abgedeckt.
  - ▶ Klauseln  $K_j$ : Werden durch  $K_{jk}$  abgedeckt.  
Die paarweisen Schnitte der Mengen  $V_{i0}, V_{i1}, K_{jk}$  sind *leer*.
  - ▶ Literale  $\ell_{jk'}$ : Werden durch weitere erfüllte Literale aus  $L_{jk'}$  abgedeckt.
- Damit ist  $(U, F) \in COVER$

# Korrektheit: $f(\phi) = (U, F) \in \text{COVER} \Rightarrow \phi \in \text{3SAT}$

Sei  $f(\phi) = (U, F) \in \text{COVER}$

- Dann gibt es eine Menge  $C \subseteq F$  mit
  - ▶ Die Vereinigung der Mengen in  $C$  deckt  $U$  ab.
  - ▶ Der paarweise Schnitt von Mengen in  $C$  ist leer.
- Damit gilt für  $C$ 
  - ▶ Variablen  $x_i$ : Entweder ist  $V_{i0}$  oder  $V_{i1}$  in  $C$ .
  - ▶ Klauseln  $K_j$ : Genau eine Klauselmenge  $K_{jk}$  ist in  $C$ .
- Definieren Variablen in  $B$ :  $x_i = 0$  falls  $V_{i0} \in C$ , sonst  $x_i = 1$ .
  - ▶ Die von den  $V_{i0}, V_{i1}$  abgedeckten Literale sind auf falsch gesetzt.
  - ▶ Jede Klauselmenge  $K_{jk}$  muss ein wahres Literal  $\ell_{jk}$  enthalten.
- D.h.  $B$  ist eine erfüllende Belegung.
- Damit gilt  $\phi \in \text{3SAT}$ .



# Hamiltonscher Kreis

## Definition Hamiltonscher Kreis

Sei  $G$  ein Graph. Ein Kreis in  $G$ , der jeden Knoten genau einmal enthält, heißt *Hamiltonscher Kreis*.

Für gerichtete Graphen definieren wir die Sprache

$$\text{GH-KREIS} := \{G \mid G \text{ gerichtet, } G \text{ besitzt einen Hamiltonschen Kreis.}\}$$

Für ungerichtete Graphen definieren wir analog

$$\text{UH-KREIS} := \{G \mid G \text{ ungerichtet, } G \text{ besitzt Hamiltonschen Kreis.}\}$$

## Satz

GH-KREIS ist  $\mathcal{NP}$ -vollständig.

- Beweis kann mittels  $\text{COVER} \leq_p \text{GH-KREIS}$  geführt werden.
- Wir verzichten hier auf den nicht-trivialen Beweis.

# NP-Vollständigkeit von Hamiltonkreis

## Satz

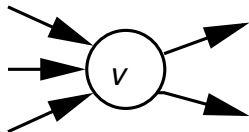
UH-KREIS ist  $\mathcal{NP}$ -vollständig.

Zeigen

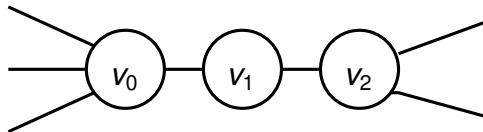
- 1 UH-KREIS  $\in \mathcal{NP}$  (Übung)
- 2 GH-KREIS  $\leq_p$  UH-KREIS

Idee der Reduktion f:

**Ersetze**



**durch**



# Reduktion $f(G) = G'$

## Algorithmus $M_f$

EINGABE:  $G = (V, E)$  gerichteter Graph mit  $V = [n]$ ,  $E = [m]$

- 1 Konstruktion der Knotenmenge  $V'$ :
  - ▶ Für jeden Knoten  $v \in V$  konstruiere  $v_0, v_1, v_2$
- 2 Konstruktion der Kantenmenge  $E'$ :
  - ▶  $E' = \{\{u_2, v_0\}, \{v_0, v_1\}, \{v_1, v_2\} \mid (u, v) \in E\}$ .

AUSGABE:  $G' = (V', E')$  ungerichteter Graph

## Laufzeit:

- Eingabelänge  $|G| = \Omega(n + m)$
- Schritt 1:  $\mathcal{O}(n)$ , Schritt 2:  $\mathcal{O}(n + m)$
- D.h. die Gesamtlaufzeit ist linear in der Eingabelänge.

# Korrektheit: $G \in \text{GH-KREIS} \Leftrightarrow f(G) = G' \in \text{UH-KREIS}$

Sei  $G \in \text{GH-KREIS}$

- Dann existiert eine Permutation  $\pi : [n] \rightarrow [n]$ , so dass  $G$  einen Hamiltonschen Kreis  $H = (\pi(1), \pi(2), \dots, \pi(n), \pi(1))$  enthält.
- $G'$  enthält den Hamiltonschen Kreis  $H' = (\pi(1)_0, \pi(1)_1, \pi(1)_2, \dots, \pi(n)_0, \pi(n)_1, \pi(n)_2, \pi(1)_0)$ .
- Damit ist  $G' \in \text{UH-KREIS}$

Sei  $G' \in \text{UH-KREIS}$

- $G'$  enthält einen Hamiltonschen Kreis  $H'$ .
  - ▶  $H'$  muss für alle  $v \in V'$  die Kanten  $\{v_0, v_1\}$  und  $\{v_1, v_2\}$  enthalten, sonst könnte  $v_1$  nicht in  $H'$  sein.
  - ▶  $H'$  ist oBdA von der Form  $(\pi(1)_0, \pi(1)_1, \pi(1)_2, \dots, \pi(n)_0, \pi(n)_1, \pi(n)_2, \pi(1)_0)$ .
- $G$  besitzt Hamiltonschen Kreis  $H = (\pi(1), \pi(2), \dots, \pi(n), \pi(1))$ .
- Damit ist  $G \in \text{GH-KREIS}$

# Übersicht unserer $\mathcal{NP}$ -vollständigen Probleme

## Vorlesung:

- SAT
- 3SAT
- CLIQUE
- KNOTENÜBERDECKUNG
- SUBSETSUM
- RUCKSACK
- COVER
- GH-KREIS
- UH-KREIS

## Übung:

- TEILGRAPH
- INDEPENDENT SET
- 0,1-PROGRAMMIERUNG
- LÄNGSTER PFAD
- HALF-CLIQUE

# Diffie-Hellman Schlüsselaustausch (1976)

## Öffentliche Parameter:

- Generator  $g$  einer multiplikativen Gruppe  $G$  mit primärer Ordnung  $q$ .
- Die Beschreibungslänge von Elementen in  $G$  ist  $\mathcal{O}(\log^2 q)$ .
- Gruppenoperationen in  $G$  sollen Laufzeit  $\mathcal{O}(\log^2 q)$  kosten.

## Protokoll Diffie-Hellman Schlüsselaustausch

EINGABE:  $p, g$

- 1 Alice wählt  $a \in_R \mathbb{Z}_q$  und schickt  $g^a$  an Bob.
- 2 Bob wählt  $b \in_R \mathbb{Z}_q$  und schickt  $g^b$  an Alice.
- 3 Alice berechnet  $(g^b)^a = g^{ab}$ , Bob analog  $(g^a)^b = g^{ab}$ .

Gemeinsamer geheimer DH-Schlüssel:  $g^{ab}$ .

# Sicherheit gegenüber passive Angreifer

- Angreifer Eve für DH-Schlüsselaustausch erhält  $g, g^a, g^b$ .
- **Sicherheit:** Eve kann  $g^{ab}$  nicht von  $g^z, z \in_R \mathbb{Z}_q$  unterscheiden.

## Definition Decisional Diffie-Hellman (DDH)

Sei  $g$  Generator einer multiplikativen Gruppe  $G$  mit Ordnung  $q$ . Wir definieren die Sprache

$$\text{DDH} := \{(q, g, g^a, g^b, g^z) \mid g^z = g^{ab}\}.$$

# Das ElGamal Kryptosystem (1984)

## Algorithmus ElGamal

- **Schlüsselerzeugung:** Sei  $g$  Generator einer multiplikativen Gruppe  $G$  mit primärer Ordnung  $q$ . Wähle  $x \in_R \mathbb{Z}_q$ . Setze  $h := g^x$ . Öffentlicher Schlüssel:  $q, g, h$ , geheimer Schlüssel:  $x$ ,
- **Verschlüsselung:** Für Nachrichten  $m \in G$  wähle  $y \in_R \mathbb{Z}_q$  und berechne

$$Enc(m) = c = (c_1, c_2) = (g^y, m \cdot (h)^y).$$

- **Entschlüsselung:** Für einen Chiffretext  $c = (c_1, c_2)$  berechne

$$Dec(c) = \frac{c_2}{c_1^x} = \frac{m \cdot g^{xy}}{g^{xy}} = m.$$

## Laufzeit:

- Verschlüsselung:  $\mathcal{O}(\log y \cdot \log^2 q) = \mathcal{O}(\log^3 q)$
- Entschlüsselung:  $\mathcal{O}(\log x \cdot \log^2 q) = \mathcal{O}(\log^3 q)$



# Sicherheit von ElGamal

**Intuitiv:** Eve soll  $c_2 = m \cdot g^{ab}$  nicht von  $c'_2 \in_R G$  unterscheiden können.

## Protokoll Unterscheider

EINGABE:  $q, g, g^x$

- 1 Eve wählt  $m \in G$  und schickt  $m$  an Alice.
- 2 Alice wählt  $b \in_R \{0, 1\}$ ,  $y \in_R \mathbb{Z}_q$ :
  - ▶ Falls  $b = 0$ : Sende  $Enc(m) = (g^y, m \cdot g^{xy})$  an Eve zurück.
  - ▶ Falls  $b = 1$ : Sende  $(g^y, c'_2) \in_R \mathbb{Z}_p^* \times \mathbb{Z}_p^*$  an Eve zurück.

Eves AUSGABE:  $b' \in \{0, 1\}$

- Eve gewinnt das Spiel gdw  $b' = b$ .
- D.h. Eve muss eine gültige Verschlüsselung  $c_2$  von einem zufälligen Gruppenelement  $c'_2$  unterscheiden.

## Definition Sprache ElGamal

$$\text{ELGAMAL} := \{(q, g, g^x, g^y, m, c_2) \mid c_2 = m \cdot g^{xy}\}.$$

# Sicherheitsbeweis per Reduktion

## Satz Sicherheit von ElGamal unter DDH

Das ElGamal Kryptosystem ist sicher gegen polynomielle Angreifer (mit Erfolgsws 1) unter der Annahme, dass DDH nicht effizient entscheidbar ist.

### Logik des Beweises:

- Zeigen:  $DDH \leq_p ELGAMAL$
- D.h. jeder polynomielle Algorithmus für ELGAMAL liefert einen polynomiellen Algorithmus für DDH. ( $\mathcal{P}$ -Reduktionssatz)
- **Ann.:** Es existiert ein polyn. Angreifer  $A$ , der Verschlüsselungen von zufälligen Gruppenelementen unterscheidet.
- Dann gibt es einen Algorithmus, der in polyn. Zeit DH-Schlüssel  $g^{ab}$  von zufälligen Gruppenelementen unterscheidet.
- **Widerspruch:** Nach Annahme gibt es keinen effizienten Algorithmus zum Entscheiden von DH-Schlüsseln  $g^{ab}$ .
- Daher kann es auch keinen polynomiellen Angreifer  $A$  geben.

# Reduktion $f$

## Algorithmus $M_f$

EINGABE:  $q, g, g^a, g^b, g^z$

1 Setze  $g^x := g^a$  und  $g^y := g^b$ .

2 Wähle  $m \in_R G$ .

3 Berechne  $c_2 = m \cdot g^z$ .

AUSGABE:  $q, g, g^x, g^y, m, c_2$

## Laufzeit:

- Eingabelänge:  $\Omega(\log q)$
- Gesamtlaufzeit:  $\mathcal{O}(\log^2(q))$

## Korrektheit Reduktion: $w \in \text{DDH} \leq_p f(w) \in \text{ELGAMAL}$

Sei  $(q, g, g^a, g^b, g^z) \in \text{DDH}$ .

- Dann gilt  $g^z = g^{ab} = g^{xy}$ .
- Damit ist  $c_2 = m \cdot g^z = m \cdot g^{xy}$  korrekte Verschlüsselung von  $m$ .
- D.h.  $(q, g, g^x, g^y, m, \delta) \in \text{ELGAMAL}$

Sei  $f(q, g, g^a, g^b, g^z) = (q, g, g^x, g^y, m, c_2) \in \text{ELGAMAL}$ .

- Dann ist  $c_2 = m \cdot g^z$  eine korrekte Verschlüsselung von  $m$ .
- D.h.  $\text{Dec}(c) = \frac{m \cdot g^z}{g^{ab}} = m$  und damit  $g^z = g^{xy} = g^{ab}$ .
- Dann ist  $(q, g, g^a, g^b, g^z) \in \text{DDH}$ .

# Brechen von ElGamal ist nicht schwerer als DDH

## Satz

ELGAMAL  $\leq_p$  DDH

**Beweis:** Wir definieren die folgende Reduktion  $f$ .

## Algorithmus $M_f$

EINGABE:  $q, g, g^x, g^y, m, c_2$

- 1 Setze  $g^a := g^x$  und  $g^b := g^y$ .
- 2 Berechne  $g^z = \frac{c_2}{m}$ .

AUSGABE:  $q, g, g^a, g^b, g^z$

## Laufzeit:

- Eingabelänge:  $\Omega(\log q)$
- Laufzeit:  $\mathcal{O}(\log^2 q)$

# Korrektheit von $f: w \in \text{ELGAMAL} \Leftrightarrow f(w) \in \text{DDH}$

Sei  $(q, g, g^x, g^y, m, c_2) \in \text{ELGAMAL}$ .

- Dann ist  $c_2 = m \cdot g^{xy}$  korrekte Verschlüsselung von  $m$ .
- Damit gilt  $\frac{c_2}{m} = g^{xy} = g^{ab} = g^z$ .
- D.h.  $(q, g, g^a, g^a, g^z) \in \text{DDH}$ .

Sei  $f(q, g, g^x, g^y, m, c_2) = (q, g, g^a, g^b, g^z) \in \text{DDH}$ .

- Dann gilt  $g^z = g^{ab} = g^{xy}$ .
- Damit folgt  $c_2 = m \cdot g^z = m \cdot g^{xy}$  ist Verschlüsselung von  $m$ .
- D.h.  $(q, g, g^x, g^y, m, c_2) \in \text{ELGAMAL}$ .

# Quadratische Reste

## Definition Quadratischer Rest

Sei  $n \in \mathbb{N}$ . Ein Element  $a \in \mathbb{Z}_n$  heißt *quadratischer Rest* in  $\mathbb{Z}_n$ , falls es ein  $b \in \mathbb{Z}_n$  gibt mit  $b^2 = a \pmod n$ . Wir definieren

$$QR_n = \{a \in \mathbb{Z}_n^* \mid a \text{ ist ein quadratischer Rest}\} \text{ und } QNR_n = \mathbb{Z}_n^* \setminus QR_n.$$

## Lemma Anzahl quadratischer Reste in primen Restklassen

Sei  $p > 2$  prim. Dann gilt  $|QR_p| = \frac{|\mathbb{Z}_p^*|}{2} = \frac{p-1}{2}$ .

- Sei  $a \in QR_p$ . Dann gilt  $a = b^2 = (-b)^2$ .
- D.h. jeder quadratische Rest  $a$  besitzt  $\geq 2$  Quadratwurzeln.
- Da  $\mathbb{F}_p$  ein Körper ist, besitzt das Polynom  $p(x) = x^2 - a$  höchstens zwei Nullstellen in  $\mathbb{F}_p$ . D.h.  $a$  hat  $\leq 2$  Quadratwurzeln.
- Damit bildet  $f : \mathbb{Z}_p^* \rightarrow QR_p, x \mapsto x^2 \pmod p$  jeweils genau zwei Elemente  $\pm b$  auf einen quadratischen Rest  $a \in QR_p$  ab.
- D.h. genau die Hälfte der Elemente in  $\mathbb{Z}_p^*$  ist in  $QR_p$ .

# Das Legendre Symbol

## Definition Legendre Symbol

Sei  $p > 2$  prim und  $a \in \mathbb{N}$ . Das *Legendre Symbol* ist definiert als

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{falls } p|a \\ 1 & \text{falls } (a \bmod p) \in QR_p \\ -1 & \text{falls } (a \bmod p) \in QNR_p. \end{cases}$$



# Berechnung des Legendre Symbols

## Satz

$$\left(\frac{a}{p}\right) = a^{\frac{p-1}{2}} \pmod{p}.$$

- Für  $p|a$  sind beide Seiten Null. Gelte also  $p \nmid a$ .
- Da  $a^{p-1} = 1 \pmod{p}$ , folgt  $a^{\frac{p-1}{2}} = \pm 1$ .
- Sei  $g$  Generator von  $\mathbb{Z}_p^*$  und  $a = g^j$  für ein  $j \in \mathbb{Z}_{p-1}$ .
- Es gilt für die linke Seite  $a \in QR_p$  gdw.  $j$  gerade ist.
- Für die rechte Seite gilt

$$a^{\frac{p-1}{2}} = g^{\frac{j(p-1)}{2}} = 1 \text{ gdw } p-1 \text{ teilt } \frac{j(p-1)}{2}.$$

- Damit ist die rechte Seite ebenfalls 1 gdw  $j$  gerade ist.

Das Legendresymbol lässt sich in Zeit  $\mathcal{O}(\log a \log^2 p)$  berechnen.

# Eigenschaften des Legendre Symbols

## Lemma Eigenschaften Quadratischer Reste

1 Multiplikativität:  $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right) \left(\frac{b}{p}\right)$

2  $(QR, \cdot)$  ist eine multiplikative Gruppe.

3  $\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}} = \begin{cases} 1 & \text{für } p = \pm 1 \pmod{8} \\ -1 & \text{für } p = \pm 3 \pmod{8}. \end{cases}$

1  $\left(\frac{ab}{p}\right) = (ab)^{\frac{p-1}{2}} \pmod{p} = \left(a^{\frac{p-1}{2}} \pmod{p}\right) \cdot \left(b^{\frac{p-1}{2}} \pmod{p}\right) = \left(\frac{a}{p}\right) \left(\frac{b}{p}\right).$

2 Übungsaufgabe

3 ohne Beweis (nicht-trivial)

# Das Quadratische Reziprozitätsgesetz

## Satz Quadratisches Reziprozitätsgesetz (Gauß)

Seien  $p, q > 2$  prim. Dann gilt

$$\left(\frac{q}{p}\right) = (-1)^{\frac{(p-1)(q-1)}{4}} \left(\frac{p}{q}\right) = \begin{cases} -\left(\frac{p}{q}\right) & \text{für } p = q = 3 \pmod{4} \\ \left(\frac{p}{q}\right) & \text{sonst.} \end{cases}$$

ohne Beweis (nicht-trivial)

- Liefert alternativen Algorithmus zur Berechnung des Legendre Symbols.

- **Bsp:** 
$$\begin{aligned} \left(\frac{6}{11}\right) &= \left(\frac{3}{11}\right) \cdot \left(\frac{2}{11}\right) = -\left(\frac{11}{3}\right) \cdot (-1) \\ &= -\left(\frac{2}{3}\right) \cdot (-1) = -(-1) \cdot (-1) = (-1). \end{aligned}$$

- D.h. 6 ist quadratischer Nichtrest in  $\mathbb{Z}_{11}^*$ .
- Benötigen Primfaktorzerlegung, um das QR-Gesetz anzuwenden.

# Das Jacobi Symbol

## Definition Jacobi Symbol

Sei  $n = p_1^{e_1} \cdot \dots \cdot p_k^{e_k} \in \mathbb{N}$  ungerade und  $a \in \mathbb{N}$ . Dann ist das *Jacobi Symbol* definiert als

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{e_1} \cdot \dots \cdot \left(\frac{a}{p_k}\right)^{e_k}.$$

- **Warnung:**  $\left(\frac{a}{n}\right) = 1$  impliziert nicht, dass  $a \in QR_n$  ist.
- Bsp:  $\left(\frac{2}{15}\right) = \left(\frac{2}{3}\right) \cdot \left(\frac{2}{5}\right) = (-1)(-1) = 1$ .
- D.h.  $2 \in QNR_3$  und  $2 \in QNR_5$ . Damit besitzt  $x^2 = 2$  weder Lösungen modulo 3 noch modulo 5.
- Nach CRT besitzt  $x^2 = 2 \pmod{15}$  ebenfalls keine Lösung.

# Verallgemeinerungen für das Jacobi Symbol

## Satz

Für alle ungeraden  $m, n$  gilt

$$1 \quad \left(\frac{2}{n}\right) = (-1)^{\frac{n^2-1}{8}}.$$

$$2 \quad \left(\frac{m}{n}\right) = (-1)^{\frac{(m-1)(n-1)}{4}} \left(\frac{n}{m}\right) = \begin{cases} -\left(\frac{n}{m}\right) & \text{für } m = n = 3 \pmod{4} \\ \text{sonst.} & \end{cases}.$$

Wir beweisen hier nur das Analog des Reziprozitätsgesetzes.

- Falls  $\text{ggT}(m, n) > 1$ , sind beide Seiten 0. Sei also  $\text{ggT}(m, n) = 1$ .
- Schreiben Primfaktorzerlegung  $m = p_1 \dots p_r$  und  $n = q_1 \dots q_s$ . ( $p_i$ 's und  $q_j$ 's können dabei jeweils mehrmals auftreten)
- Wandel  $\left(\frac{m}{n}\right) = \prod_{i,j} \left(\frac{p_i}{q_j}\right)$  zu  $\left(\frac{n}{m}\right) = \prod_{i,j} \left(\frac{q_j}{p_i}\right)$  durch  $rs$ -malige Anwendung des Reziprozitätsgesetzes.
- Anzahl  $(-1)$  entspricht Anzahl Paare  $(i, j)$  mit  $p_i = q_j = 3 \pmod{4}$ .
- D.h.  $\left(\frac{m}{n}\right) = -\left(\frac{n}{m}\right)$  gdw. ungerade viele  $p_i, q_j$  kongruent  $3 \pmod{4}$ .
- Es gibt ungerade viele  $p_i, q_j = 3 \pmod{4}$  gdw.  $m = n = 3 \pmod{4}$  ist.

# Rekursive Berechnung des Jacobi Symbols

**Idee:** Für ungerades  $n$  gilt

$$\left(\frac{m}{n}\right) = \left(\frac{2}{n}\right)^k \cdot \left(\frac{m'}{n}\right) = \left(\frac{2}{n}\right)^k \cdot (-1)^{\frac{(m'-1)(n-1)}{4}} \left(\frac{n \bmod m'}{m'}\right).$$

## Algorithmus Jacobi-Symbol

EINGABE:  $m, n$

- 1 Falls  $ggT(m, n) > 1$ , Ausgabe 0.
- 2 Falls  $m = 1$ , Ausgabe 1.
- 3 Sei  $m = 2^k m'$  mit  $m'$  ungerade.
- 4 Ausgabe  $(-1)^{\frac{k(n^2-1)}{8}} \cdot (-1)^{\frac{(m'-1)(n-1)}{4}} \cdot \text{Jacobi-Symbol}(n \bmod m', m')$

AUSGABE:  $\left(\frac{m}{n}\right)$

**Bsp:**  $\left(\frac{14}{15}\right) = \left(\frac{2}{15}\right) \cdot \left(\frac{7}{15}\right) = (-1) \cdot \left(\frac{15 \bmod 7}{7}\right) = (-1).$

- **Laufzeit:** Analog zum Euklidischen Algorithmus:  
 $\mathcal{O}(\log \max\{m, n\})$  rekursive Aufrufe.
- Jeder Aufruf kostet  $\mathcal{O}(\log^2 \max\{m, n\})$ .

# Das Quadratische Reste Problem

## Definition Pseudoquadrate

Sei  $N = pq$  mit  $p, q$  prim. Eine Zahl  $a$  heißt *Pseudoquadrat* bezüglich  $N$ , falls

$$\left(\frac{a}{N}\right) = 1 \text{ und } a \notin QR_N.$$

Wir definieren die Sprache

$$\text{QUADRAT} := \{a \in \mathbb{Z}_N^* \mid \left(\frac{a}{N}\right) = 1 \text{ und } a \in QR_N\}.$$

- Für alle Pseudoquadrate  $a$  gilt:  $\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = (-1)$ .
- D.h. die Sprache QUADRAT kann effizient entschieden werden, falls  $p, q$  bekannt sind. Im Allgemeinen ist nur  $N$  bekannt.

## Quadratische Reduzibilitätsannahme (QR-Annahme)

Es gibt keinen polynomiellen Algorithmus, der QUADRAT entscheidet.

# Quadratwurzeln in $\mathbb{Z}_N^*$

## Lemma

Sei  $N = pq$  mit  $p, q$  prim und  $p = q = 3 \pmod{4}$  (sogenannte Blum-Zahl). Dann besitzt jedes  $a = x^2 \in QR_N$  genau eine Quadratwurzel in  $QR_N$ , die sogenannte Hauptwurzel.

## Beweis:

- Die Lösungen des Gleichungssystems  $\left| \begin{array}{l} y = \pm x \pmod{p} \\ y = \pm x \pmod{q} \end{array} \right|$  liefern mittels Chinesischem Restsatz 4 Lösungen in  $\mathbb{Z}_N^*$ .
- Eine Lösung ist in  $QR_N$  gdw sie in  $QR_p \times QR_q$  ist.
- Betrachten Lösung modulo  $p$  (analog mod  $q$ ):

$$\left(\frac{x}{p}\right) = \left(\frac{(-1)(-x)}{p}\right) = \left(\frac{-1}{p}\right) \cdot \left(\frac{-x}{p}\right).$$

- Für  $p = 3 \pmod{4}$  gilt  $\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}} = (-1)$ .
- D.h.  $\left(\frac{x}{p}\right) = -\left(\frac{-x}{p}\right)$  und entweder  $x$  oder  $-x$  ist in  $QR_p$ .
- Damit ist genau eine der 4 Lösungen in  $QR_N$ .



# Der Blum-Blum-Shub (BBS) Pseudozufallsgenerator

## Korollar

Die Abb.  $f : QR_N \rightarrow QR_N, x \mapsto x^2 \bmod N$  ist eine Bijektion auf  $QR_N$ .

- $(k, \ell)$ -Pseudozufallsgeneratoren generieren aus  $k$  Zufallsbits eine Sequenz von  $\ell > k$  Zufallsbits.
- Der  $(k, \ell)$ -BBS Generator verwendet obige Bijektion.

## Algorithmus BBS Pseudozufallsgenerator (1986)

EINGABE:  $N = pq$  Blumzahl der Bitlänge  $|N| = k$ ,  
 $1^\ell$  mit  $\ell \in \mathbb{N}$  und  $\ell > k$ ,  $r \in \mathbb{Z}_N^*$

- 1 Setze  $s_0 = r^2 \bmod N$ .
- 2 For  $i = 1$  to  $\ell$ 
  - 1 Setze  $s_i \leftarrow s_{i-1}^2 \bmod N$ . Gib  $z_i = s_i \bmod 2$  aus.

AUSGABE:  $(z_1, \dots, z_\ell) \in \{0, 1\}^\ell$ .

**Laufzeit:**  $\mathcal{O}(\ell \log^2 N)$ , d.h. polynomiell in der Eingabelänge.

# Die Sicherheit des BBS Generators

**Sicherheit:** Man kann die Verteilung der  $(z_1, \dots, z_\ell)$  nicht von der Gleichverteilung auf  $\{0, 1\}^\ell$  unterscheiden.

Man kann folgendes zeigen:

- Sei  $A$  ein polynomieller Unterscheider für  $(z_1, \dots, z_\ell)$ .
- Dann gibt es einen polyn. Algorithmus  $B$ , der  $s_0 \bmod 2$  berechnet.

## Satz Sicherheit des BBS Generators

Die Ausgabe des BBS Generators ist von der Gleichverteilung in polynomieller Zeit ununterscheidbar unter der QR-Annahme.

- Annahme:  $\exists$  polyn. Unterscheider  $A$  für den BBS Generator.
- Sei  $B$  ein Algorithmus, der  $s_0 \bmod 2$  berechnet.
- Zeigen, dass dann ein polyn. Algorithmus für QUADRAT existiert. (Widerspruch zur Quadratischen Residuositätsannahme)

# Entscheiden der Sprache QUADRAT

## Algorithmus für QUADRAT

EINGABE:  $N, a \in \mathbb{Z}_N^*$  mit  $\left(\frac{a}{N}\right) = 1, 1^\ell$

- 1 Setze  $s_0 \leftarrow a \bmod N$ .
- 2 Berechne  $(z_1, \dots, z_\ell)$  mittels BBS Generator.
- 3 Berechne  $z_0 \leftarrow B(z_1, \dots, z_\ell)$ .
- 4 Falls  $z_0 = (a \bmod 2)$ , Ausgabe " $x \in QR_N$ ".  
Sonst Ausgabe " $x \notin QR_N$ ".

**Laufzeit:**  $\mathcal{O}(\ell \cdot \log^2 N + T(B))$

**Korrektheit:**

- Wegen  $\left(\frac{a}{N}\right) = 1$  ist entweder  $a$  oder  $(-a) = N - a$  in  $QR_N$ .
- D.h.  $a$  oder  $(-a)$  ist eine Hauptwurzel von  $s_1 = a^2 \bmod N$ .
- Genau eine der beiden Zahlen  $a, (-a)$  ist gerade.
- $z_0$  ist das unterste Bit der Hauptwurzel von  $s_1 = a^2 \bmod N$ .
- D.h.  $a$  ist eine Hauptwurzel gdw  $z_0$  und  $a \bmod 2$  übereinstimmen.

## Algorithmus Goldwasser-Micali Kryptosystem (1984)

- 1 Schlüsselgenerierung: Wähle Blumzahl  $N = pq$ . Wähle  $z \in_R \mathbb{Z}_N^*$ , so dass  $z$  ein Pseudoquadrat ist. Setze den öffentlichen Schlüssel  $pk = (N, z)$  und den privaten Schlüssel  $sk = (p, q)$ .
- 2 Verschlüsselung: Für  $m \in \{0, 1\}$  wähle  $x \in_R \mathbb{Z}_N^*$  und berechne
$$c \leftarrow z^m x^2 \pmod{N}.$$
- 3 Entschlüsselung: Für einen Chiffretext  $c$  berechne

$$m = \begin{cases} 0 & \text{falls } c \in QR_N, \text{ d.h. falls } \left(\frac{c}{p}\right) = 1. \\ 1 & \text{falls } c \notin QR_N, \text{ d.h. falls } \left(\frac{c}{p}\right) = (-1). \end{cases}$$

# Sicherheit des Goldwasser-Micali Kryptosystems

## Korrektheit

- Falls  $m = 0$  ist  $c = x^2$  ein zufälliger quadratischer Rest in  $\mathbb{Z}_N^*$ .
- Falls  $m = 1$  ist  $c = z \cdot x^2$  ein zufälliges Pseudoquadrat.
- Es gilt  $\left(\frac{c}{N}\right) = \left(\frac{z^m x^2}{N}\right) = \left(\frac{z}{N}\right)^m \cdot \left(\frac{x^2}{N}\right) = 1$ .
- D.h. entweder  $\left(\frac{c}{p}\right) = \left(\frac{c}{q}\right) = 1$  oder  $\left(\frac{c}{p}\right) = \left(\frac{c}{q}\right) = (-1)$ .
- Im ersten Fall ist  $c \in QR_N$ , im zweiten Fall gilt  $c \notin QR_N$ .

## Laufzeit:

- Verschlüsselung:  $\mathcal{O}(\log^2 N)$
- Entschlüsselung:  $\mathcal{O}(\log^2 N)$

## Satz Sicherheit des Goldwasser-Micali Kryptosystems

Das GM Kryptosystem ist sicher unter der QR-Annahme.

## Beweisidee:

- Unterscheiden von Verschlüsselungen von 0 und 1 ist äquivalent zum Entscheiden der Sprache QUADRAT.

# Bit Commitments

## Szenario informal:

### 1 Commitment-Phase:

- ▶ Alice platziert ein Bit  $b \in \{0, 1\}$  in einem Safe, der in Bob's Zimmer steht. Bob besitzt keinen Safeschlüssel.
- ▶ Bob kann den Safe nicht einsehen, lernt also nichts über  $b$ .  
**(Concealing Eigenschaft)**

### 2 Revealing-Phase:

- ▶ Alice öffnet den Safe und zeigt Bob das Bit  $b$ .
- ▶ Alice kann ihr Bit dabei nicht ändern.  
**(Binding Eigenschaft)**

## Mathematische Modellierung

- Commitment mittels  $f : \{0, 1\} \times X \rightarrow Y$  für endliche Mengen  $X, Y$ .
- Commitment (sog. Blob): Wähle  $x \in X$  und sende  $f(b, x)$  an Bob.
- Öffnen des Commitments: Sende  $b$  und  $x$  an Bob.

# Bit Commitment via Goldwasser-Micali Kryptosystem

## Öffentliche Parameter:

- Blumzahl  $N$ , Pseudoquadrat  $z \in \mathbb{Z}_N^*$
- $X = Y = \mathbb{Z}_N^*$

## Algorithmus Goldwasser-Micali Bit Commitment

### 1 Commitment-Phase

- ▶ Wähle  $x \in_R \mathbb{Z}_N^*$ .
- ▶ Sende Blob  $f(b, x) = z^b x^2 \bmod N$  an Bob.

### 2 Revealing-Phase

- ▶ Sende  $b, x$  an Bob.
- ▶ Bob überprüft die Korrektheit von  $f(b, x) = z^b x^2 \bmod N$ .

## Concealing Eigenschaft:

- Unter der QR-Annahme lernt Bob nichts über das Bit  $b \in \{0, 1\}$ .

# Binding Eigenschaft

## Satz

Goldwasser-Micali Commitments besitzen die Binding Eigenschaft.

### Beweis:

- **Annahme:** Alice kann Blob  $f(b, x)$  für  $b = 0$  und  $b = 1$  öffnen.
- D.h. Alice kann  $x_1, x_2 \in \mathbb{Z}_N^*$  berechnen mit

$$f(b, x) = z^0 x_1^2 = z^1 x_2^2 \pmod{N}.$$

- Daraus folgt  $z = \left(\frac{x_1}{x_2}\right)^2 \pmod{N}$ , d.h.  $\frac{x_1}{x_2}$  ist Quadratwurzel von  $z$ .  
(Widerspruch:  $z$  ist ein Pseudoquadrat in  $\mathbb{Z}_N^*$ .)



# Münzwurf über das Telefon

- Bit Commitments haben zahlreiche Anwendungen in kryptographischen Protokollen.
- Exemplarisch hier ein Protokoll für einen fairen Münzwurf.

## Algorithmus Münzwurf via Internet

- 1 Alice sendet Bob Commitment für Bit  $b \in \{0, 1\}$ .
  - 2 Bob rät ein Bit  $b' \in \{0, 1\}$ .
  - 3 Alice öffnet ihr Bit. Bob gewinnt gdw  $b' = b$ .
- Concealing-Eigenschaft verhindert, dass Bob etwas über  $b$  lernt.
  - Binding-Eigenschaft verhindert, dass Alice  $b$  in  $1 - b'$  ändert.

# Berechnen von Quadratwurzeln modulo $p$

## Satz Quadratwurzeln mod $p$

Sei  $p$  prim,  $p = 3 \pmod 4$  und  $a \in QR_p$ . Dann sind die beiden Quadratwurzeln von  $a$  von der Form

$$x = \pm a^{\frac{p+1}{4}} \pmod p, \text{ wobei } a^{\frac{p+1}{4}} \in QR_p.$$

- Es gilt

$$x^2 = a^{\frac{p+1}{2}} = a^{\frac{p-1}{2}} \cdot a = \left(\frac{a}{p}\right) \cdot a = a \pmod p.$$

- Ferner gilt  $a^{\frac{p+1}{4}} \pmod p \in QR_p$  wegen

$$\left(\frac{a^{\frac{p+1}{4}}}{p}\right) = \left(\frac{a}{p}\right)^{\frac{p+1}{4}} = 1.$$

D.h. Quadratwurzeln können in Zeit  $\mathcal{O}(\log^3 p)$  berechnet werden.

# Das Blum-Goldwasser Kryptosystem

## Algorithmus Blum-Goldwasser Kryptosystem (1985)

- 1 Schlüsselgenerierung: Wähle Blumzahl  $N = pq$ .  
Setze  $pk = N$  und  $sk = (p, q)$ .
- 2 Verschlüsselung: Für  $m = (m_1, \dots, m_\ell) \in \{0, 1\}^\ell$ :
  - ▶ Wähle  $r \in_R \mathbb{Z}_N^*$ .
  - ▶  $(z_1, \dots, z_\ell) \leftarrow$  BBS Generator auf  $s_0 = r^2 \bmod N$ .
  - ▶ For  $i = 1$  to  $\ell$ : Berechne  $c_i = m_i + z_i \bmod 2$ .
  - ▶ Berechne  $s_{\ell+1} = s_0^{2^{\ell+1}} \bmod N$ .
  - ▶ AUSGABE: Chiffretext  $c = (c_1, \dots, c_\ell, s_{\ell+1}) \in \{0, 1\}^\ell \times \mathbb{Z}_N^*$ .
- 3 Entschlüsselung von  $c$  mittels  $sk = (p, q)$ :

- ▶ Berechne  $s_0 \in \mathbb{Z}_N^*$  als Lösung von 
$$\left| \begin{array}{l} s_0 = s_{\ell+1}^{\left(\frac{p+1}{4}\right)^{\ell+1}} \bmod p \\ s_0 = s_{\ell+1}^{\left(\frac{q+1}{4}\right)^{\ell+1}} \bmod q \end{array} \right|.$$
- ▶  $(z_1, \dots, z_\ell) \leftarrow$  BBS Generator auf  $s_0 = r^2 \bmod N$ .
- ▶ For  $i = 1$  to  $\ell$ : Berechne  $m_i = c_i + z_i \bmod 2$ .

# Laufzeit und Korrektheit

## Korrektheit:

- $(z_1, \dots, z_\ell)$  wird als One-Time Pad für  $m$  verwendet.
- Entschlüsselung berechnet  $\ell + 1$ -malig die Hauptwurzel von  $s_{\ell+1}$ .
- Dies rekonstruiert die Saat  $s_0$  des BBS Generators.

## Laufzeit:

- Verschlüsselung:  $\mathcal{O}(\ell \cdot \log^2 N)$
- Entschlüsselung:  $\mathcal{O}(\log^3 N + \ell \cdot \log^2 N)$ .

## Satz Sicherheit des BG-Kryptosystems

Das Blum Goldwasser Kryptosystem ist sicher unter der Annahme, dass Blumzahlen  $N = pq$  schwer zu faktorisieren sind.

(ohne Beweis)

# Elliptische Kurven

## Definition Elliptische Kurve

Sei  $p \neq 2, 3$  prim,  $f(x) = x^3 + ax + b \in \mathbb{Z}_p[x]$ ,  $4a^3 + 27b^2 \neq 0 \pmod{p}$ .  
Wir definieren für  $f(x)$  eine *elliptische Kurve*  $E$  als

$$\{(x, y) \in \mathbb{Z}_p \mid y^2 = x^3 + ax + b\} \cup \{\mathbf{O}\},$$

wobei  $\mathbf{O}$  der Punkt im Unendlichen heißt.

## Anmerkungen:

- Die Bedingung  $4a^3 + 27b^2 \neq 0$  ist äquivalent zu der Forderung, dass  $f(x)$  in  $\mathbb{Z}_p^*$  keine mehrfachen Nullstellen besitzt. (Übung)
- Für jeden Punkt  $P = (x, y)$  auf  $E$  liegt auch  $(x, -y)$  auf  $E$ .
- Wir definieren  $-P = (x, -y)$ .
- Für  $P = \mathbf{O}$  definieren wir  $-P = \mathbf{O}$  und  $P + Q = Q$  für alle  $Q$  auf  $E$ .

# Addition von Punkten

## Algorithmus Addition von Punkten auf $E$

EINGABE:  $P = (x_1, y_1)$ ,  $Q = (x_2, y_2)$  auf  $E$  mit  $P, Q \neq \mathbf{O}$

1 Falls  $x_1 = x_2$  und  $y_1 = -y_2$ , Ausgabe  $\mathbf{O}$ .

2 Setze  $\alpha := \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{für } x_1 \neq x_2 \\ \frac{3x_1^2 + a}{2y_1} & \text{für } x_1 = x_2 \end{cases}$ . Setze  $\beta = y_1 - \alpha x_1$ .

3 Berechne  $x_3 = \alpha^2 - x_1 - x_2$  und  $y_3 = -(\alpha x_3 + \beta)$ .

AUSGABE:  $P + Q = (x_3, y_3)$

### Anmerkungen:

- Sei  $P \neq Q$ . Wir betrachten die Gerade  $G$  durch  $P, Q$ .
- Falls  $Q = -P$ , so liegt  $G$  parallel zur  $y$ -Achse. Wir definieren

$$P + (-P) = \mathbf{O}.$$

- Sonst ist  $G$  definiert durch  $y = \alpha x + \beta$  mit Steigung  $\alpha = \frac{y_2 - y_1}{x_2 - x_1}$ .
- Für  $P = Q$  besitzt die Tangente im Punkt  $P$  Steigung  $\alpha = \frac{3x_1^2 + a}{2y_1}$ .

# Addition von Punkten

## Lemma Addition von Punkten auf $E$

Seien  $P, Q$  auf  $E$  mit  $P \neq -Q$ . Dann schneidet die Gerade durch  $P, Q$  die Kurve  $E$  in einem dritten Punkt  $R$  mit  $R := -(P + Q)$ .

### Beweis:

- Wir zeigen nur  $P \neq Q$ . Der Beweis für  $P = Q$  folgt analog.
- Wie zuvor setzen wir  $P = (x_1, y_1)$ ,  $Q = (x_2, y_2)$  und  $R = (x_3, y_3)$ .
- Sei  $G$  die Gerade  $y = \alpha x + \beta$  durch  $P, Q$ . Dann gilt für  $i = 1, 2$ 
$$(\alpha x_i + \beta)^2 = x_i^3 + ax_i + b.$$
- $x_1, x_2$  sind damit Nullstellen des Polynoms  $g(x) = x^3 - \alpha^2 x^2 + \dots$
- Dann muss  $g(x)$  3 Nullstellen besitzen
$$g(x) = (x - x_1)(x - x_2)(x - x_3) = x^3 - (x_1 + x_2 + x_3)x^2 + \dots$$
- Durch Koeffizientenvergleich folgt  $x_1 + x_2 + x_3 = \alpha^2$ .
- Wir erhalten  $y_3 = \alpha x_3 + \beta$  und damit  $-R = (x_3, -y_3)$ .

# Eigenschaften der Addition auf $E$

## Korollar Effizienz der Addition

Sei  $E$  eine elliptische Kurve mit Punkten  $P, Q$ . Dann kann  $P + Q$  in Laufzeit  $\mathcal{O}(\log^2 p)$  berechnet werden.

- Wir benötigen nur Addition, Multiplikation und Division in  $\mathbb{Z}_p$ .

## Satz von Mordell

Jede elliptische Kurve  $E$  bildet mit der definierten Addition eine abelsche Gruppe.

### Beweis:

- Abgeschlossenheit:  $P + Q$  liefert wieder einen Punkt auf  $E$ .
- Neutrales Element ist der Punkt  $\mathbf{O}$ .
- Inverses von  $P \neq \mathbf{O}$  ist  $-P$  und  $-\mathbf{O} = \mathbf{O}$ .
- Abelsch: Berechnung von  $G$  unabhängig von Reihenfolge  $P, Q$ .
- Assoziativität kann durch Nachrechnen gezeigt werden.



# Gruppenordnung einer elliptischen Kurve

## Satz von Hasse

Sei  $E$  eine elliptische Kurve über  $\mathbb{F}_p$ . Dann gilt

$$|E| = p + 1 + t \text{ mit } |t| \leq 2\sqrt{p}.$$

**Anmerkungen:** (ohne Beweis)

- Sei  $x \in \mathbb{Z}_p$  und  $f(x) = x^3 + ax + b$ .
- Falls  $f(x)$  ein quadratischer Rest modulo  $p$  ist, dann existieren genau zwei Lösungen  $\pm y$  der Gleichung  $y^2 = f(x) \pmod{p}$ , d.h.  $(x, y)$  und  $(x, -y)$  liegen auf  $E$ .
- Falls  $f(x)$  ein Nichtrest ist, besitzt  $E$  keinen Punkt der Form  $(x, \cdot)$ .
- Genau die Hälfte aller Elemente in  $\mathbb{Z}_p^*$  ist ein quadratischer Rest.
- Falls  $x \mapsto f(x)$  sich zufällig verhält auf  $\mathbb{Z}_p$ , erwarten wir  $\frac{p}{2} \cdot 2 = p$  Punkte. Hinzu kommt der Punkt  $\mathbf{O}$ , d.h.  $|E| \approx p + 1$ .
- Satz von Hasse:  $x \mapsto f(x)$  ist fast zufällig mit Fehler  $|t| \leq 2\sqrt{p}$ .

# Unser Modell

- Shannon 1948: Informationstheorie und Mathematik der Kommunikation
- Hamming 1950: Erste Arbeit über fehlerkorrigierende Codes

## Modell:

Sender  $\rightarrow$  Kodierer  $\rightarrow$  Kanal  $\rightarrow$  Dekodierer  $\rightarrow$  Empfänger

- Kanal ist bandbreitenbeschränkt (Kompression)
- Kanal ist fehleranfällig (Fehlerkorrektur)
  - ▶ Bits können ausfallen:  $0 \rightarrow \epsilon, 1 \rightarrow \epsilon$  (Ausfallkanal)
  - ▶ Bits können kippen:  $0 \rightarrow 1, 1 \rightarrow 0$  (Symmetrischer Kanal)

# Motivierendes Bsp: Datenkompression

## Szenario:

- Kanal ist **fehlerfrei**.
- Übertragen gescannte Nachricht:  
Wahrscheinlichkeiten: 99% weißer, 1% schwarzer Punkt.
- Weiße Punkte erhalten Wert 0, schwarze Wert 1.

## Kodierer:

- Splitten Nachricht in Blocks der Größe 10.
- Wenn Block  $x=0000000000$ , kodiere mit 0, sonst mit 1x.
- 1 dient als Trennzeichen beim Dekodieren.

## Dekodierer:

- Lese den Code von links nach rechts.
- Falls 0, dekodiere 0000000000.
- Falls 1, übernehme die folgenden 10 Symbole.

# Erwartete Codelänge

Sei  $q := \text{Ws}[\text{Block ist } 0000000000] = (0.99)^{10} \geq 0.9$ .

Sei  $Y$  Zufallsvariable für die Codewortlänge eines 10-Bit Blocks:

$$E[Y] = \sum_{y \in \{0,1\}^x} |y| \cdot \text{Ws}(Y = |y|) = 1 \cdot q + 11 \cdot (1 - q) = 11 - 10q.$$

- D.h. die erwartete Bitlänge der Kodierung eines 10-Bit Blocks ist
$$11 - 10q \leq 2.$$
- Datenkompression der Nachricht auf 20%.
- Können wir noch stärker komprimieren?
- Entropie wird uns Schranke für Komprimierbarkeit liefern.

# Ausblick: fehlerkorrigierende Codes

## Szenario: Binärer symmetrischer Kanal

- Bits 0,1 kippen mit Ws  $p, p < \frac{1}{2}$  zu 1,0. (Warum  $< \frac{1}{2}$ ?)
- Korrekte Übertragung  $0 \mapsto 0, 1 \mapsto 1$  mit Ws  $1 - p$ .
- In unserem Beispiel  $p = 0.1$ .

## Kodierer:

- Verdreifache jedes Symbol, d.h.  $0 \mapsto 000, 1 \mapsto 111$
- Repetitionscode der Länge 3.

## Dekodierer:

- Lese den Code in 3er-Blöcken.
- Falls mindestens zwei Symbole 0 sind, dekodiere zu 0.
- Sonst dekodiere zu 1.

# Ws Dekodierfehler

Symbol wird falsch dekodiert, falls mind. zwei der drei Bits kippen.

$$\begin{aligned} & W_s(\text{Bit wird falsch dekodiert}) \\ &= W_s(\text{genau 2 Bits kippen}) + W_s(\text{genau 3 Bits kippen}) \\ &= 3 * p^2 * (1 - p) + p^3 = 3 * 10^{-2} * (1 - 10^{-1}) + 10^{-3} \end{aligned}$$

- Ohne Kodierung Fehlerws von 0.1.
- Mit Repetitionscode Fehlerws von  $\approx 0.03$ .
- Nachteil: Kodierung ist dreimal so lang wie Nachricht.
- **Ziel:**  
Finde guten Tradeoff zwischen Fehlerws und Codewortlänge.

# Ausblick: fehlertolerante Codes

## Szenario: Binärer Ausfallkanal

- Bits 0,1 gehen mit Ws  $p, p < \frac{1}{2}$  verloren, d.h.  $0 \mapsto \epsilon$  bzw.  $1 \mapsto \epsilon$ .
- Korrekte Übertragung  $0 \mapsto 0, 1 \mapsto 1$  mit Ws  $1 - p$ .
- In unserem Beispiel  $p = 0.1$ .

**Kodierer:** Repetitionscode der Länge 3.

## Dekodierer:

- Lese den Code in 3er-Blöcken.
- Falls 3er-Block Zeichen  $x \in \{0, 1\}$  enthält, Ausgabe  $x$ .

**Fehler beim Dekodieren:** Alle drei Symbole gehen verloren.

- $Ws(\text{Bit kann nicht dekodiert werden}) = p^3 = 0.001$ .
- Fehlerws kleiner beim Ausfallkanal als beim sym. Kanal.

# Definition Code

## Bezeichnungen:

- Alphabet  $A = \{a_1, \dots, a_n\}$ , Menge von Symbolen  $a_i$
- Nachricht sind Elemente  $m \in A^*$ .

## Definition Code

Sei  $A$  ein Alphabet. Eine (binäre) *Codierung*  $C$  des Alphabets  $A$  ist eine injektive Abbildung

$$C : \quad A \rightarrow \{0, 1\}^* \\ a_i \mapsto C(a_i).$$

Die *Codierung einer Nachricht*  $m = a_{i_1} \dots a_{i_\ell} \in A^*$  definieren wir als

$$C(m) = C(a_{i_1}) \dots C(a_{i_\ell}) \quad (\text{Erweiterung von } C \text{ auf } A^*).$$

Die Abbildung  $C$  heißt *Code*.



# Bezeichnungen Code

- Die Elemente  $c_i := C(a_i)$  bezeichnen wir als *Codeworte*.
- Wir bezeichnen sowohl die Abbildung von Nachrichten auf Codeworte als auch die *Menge der Codeworte* mit dem Buchstaben  $C$ .
- Falls  $C \subseteq \{0, 1\}^n$  spricht man von einem *Blockcode* der Länge  $n$ . In einem Blockcode haben alle Codeworte die gleiche Länge.

# Entschlüsselbarkeit von Codes

**Szenario:** Datenkompression in fehlerfreiem Kanal

## Definition eindeutig entschlüsselbar

Ein Code heißt eindeutig entschlüsselbar, falls jedes Element aus  $\{0, 1\}^*$  Bild höchstens einer Nachricht ist. D.h. die Erweiterung der Abbildung  $C$  auf  $A^*$  muss injektiv sein.

## Definition Präfixcode

Ein Code  $C = \{c_1, \dots, c_n\}$  heißt Präfixcode, falls es keine zwei Codeworte  $c_i \neq c_j$  gibt mit

$c_i$  ist Präfix (Wortanfang) von  $c_j$ .

# Beispiel

	$a_1$	$a_2$	$a_3$
$C_1$	0	0	1
$C_2$	0	1	00
$C_3$	0	01	011
$C_4$	0	10	11

- $C_1$  ist kein Code, da  $C_1 : A \rightarrow \{0, 1\}^*$  nicht injektiv.
- $C_2$  ist nicht eindeutig entschlüsselbar, da  $C_2 : A^* \rightarrow \{0, 1\}^*$  nicht injektiv.
- $C_3$  ist eindeutig entschlüsselbar, aber kein Präfixcode.
- $C_4$  ist ein Präfixcode.

# Präfixcodes sind eindeutig entschlüsselbar.

## Satz Präfixcode eindeutig entschlüsselbar

Sei  $C = \{c_1, \dots, c_n\}$  ein Präfixcode. Dann kann jede kodierte Nachricht  $C(m)$  in Zeit  $\mathcal{O}(|C(m)|)$  eindeutig zu  $m$  decodiert werden.

### Beweis:

- Zeichne binären Baum
  - ▶ Kanten erhalten Label 0 für linkes Kind, 1 für rechtes Kind.
  - ▶ Codewort  $c_i = c_{i_1} \dots c_{i_k}$  ist Label des Endknoten eines Pfads von der Wurzel mit den Kantenlabeln  $c_{i_1}, \dots, c_{i_n}$
- **Präfixeigenschaft:** Kein einfacher Pfad von der Wurzel enthält zwei Knoten, die mit Codeworten gelabelt sind.
- Codewort  $c_j$  ist Blatt in Tiefe  $|c_j|$

# Algorithmus Dekodierung Präfix

## Algorithmus Dekodierung Präfix

- 1 Lese  $C(m)$  von links nach rechts.
- 2 Starte bei der Wurzel. Falls 0, gehe nach links. Falls 1, gehe nach rechts.
- 3 Falls Blatt mit Codewort  $c_i = C(a_i)$  erreicht, gib  $a_i$  aus und iteriere.

**Laufzeit:**  $\mathcal{O}(|C(m)|)$

# Woher kommen die Nachrichtensymbole?

## Modell

- *Quelle*  $Q$  liefert Strom von Symbolen aus  $A$ .
- Quellwahrscheinlichkeit:  $Ws[ \text{Quelle liefert } a_j ] = p_j$
- $Ws p_j$  ist unabhängig von der Zeit und vom bisher produzierten Strom (erinnerungslose Quelle)
- $X_i$ : Zufallsvariable für das Quellsymbol an der  $i$ -ten Position im Strom, d.h.

$$Ws[X_i = a_j] = p_j \quad \text{für } j = 1, \dots, n \text{ und alle } i.$$

**Ziel:** Kodiere  $a_j$  mit großer  $Ws p_j$  mittels kleiner Codewortlänge.

# Kompakte Codes

## Definition Erwartete Codewortlänge

Sei  $Q$  eine Quelle mit Alphabet  $A = \{a_1, \dots, a_n\}$  und Quellwahrscheinlichkeiten  $p_1, \dots, p_n$ . Die Größe

$$E(C) := \sum_{i=1}^n p_i |C(a_i)|$$

bezeichne die erwartete Codewortlänge.

## Definition Kompakter Code

Ein Code  $C$  heißt kompakt bezüglich einer Quelle  $Q$ , falls er *minimale erwartete Codewortlänge* besitzt.

# Wann sind Codes eindeutig entschlüsselbar?

## Definition Suffix

Sei  $C$  ein Code. Ein String  $s \in \{0, 1\}^*$  heißt Suffix in  $C$  falls

- 1  $\exists c_i, c_j \in C : c_i = c_j s$  oder
- 2  $\exists c \in C$  und einen Suffix  $s'$  in  $C : s' = cs$  oder
- 3  $\exists c \in C$  und einen Suffix  $s'$  in  $C : c = s's$ .

- Bedingung 1: Codewort  $c_j$  lässt sich zu Codewort  $c_i$  erweitern.
- Bedingung 2: Codewort  $c$  lässt sich zu Suffix  $s'$  erweitern.
- Bedingung 3: Suffix  $s'$  lässt sich zu Codewort  $c$  erweitern.



# Effiziente Berechnung von Suffixen

## Algorithmus Berechnung Suffix

EINGABE:  $C = \{c_1, \dots, c_n\}$

- 1 Setze  $S := \emptyset, T := \emptyset$ .
- 2 Für alle  $c_i, c_j \in C \times C$ : Falls es ein  $s \in \{0, 1\}^*$  gibt mit  $c_i = c_j s$ , füge  $s$  in  $S$  und  $T$  ein.
- 3 Solange  $T \neq \emptyset$ 
  - 1 Entferne ein beliebiges  $s'$  aus  $T$ .
  - 2 Für alle  $c \in C$ : Falls es ein  $s \in \{0, 1\}^* \setminus S$  gibt mit  $s' = cs$  oder  $c = s's$ , füge  $s$  zu  $S$  und  $T$  hinzu.

AUSGABE: Menge  $S$  der Suffixe von  $C$

# Laufzeit Suffixberechnung

## Laufzeit:

- Schritt 2:  $\mathcal{O}(n^2)$  Codewortpaare
- Suffixlänge ist durch  $\max_i\{|c_i|\}$  beschränkt.
- Es kann höchstens  $n \cdot \max_i\{|c_i|\}$  Suffixe geben. (Warum?)
- Schritt 3:  $\mathcal{O}(n^2 \cdot \max_i\{|c_i|\})$
- **Polynomiell in der Eingabelänge:  $n, \max_i\{|c_i|\}$**

# Beispiele Suffixberechnung

- Code  $C_2 = \{0, 1, 00\}$ 
  - ▶ Suffix  $s_1 = 0$ , denn  $c_3 = c_1 0$ .
- Code  $C_3 = \{0, 01, 011\}$ 
  - ▶ Suffix  $s_1 = 1$ , denn  $c_2 = c_1 1$ .
  - ▶ Suffix  $s_2 = 11$ , denn  $c_3 = c_1 11$ .
- Code  $C_4 = \{0, 10, 110\}$ 
  - ▶ Keine Suffixe, da Präfixcode.
- Code  $C_5 = \{1, 110, 101\}$ 
  - ▶ Suffix  $s_1 = 10$ , denn  $c_2 = c_1 10$ .
  - ▶ Suffix  $s_2 = 01$ , denn  $c_3 = c_1 01$ .
  - ▶ Suffix  $s_3 = 0$ , denn  $s_3 = c_1 0$ .
  - ▶ Suffix  $s_4 = 1$ , denn  $c_3 = s_1 1$ .

# Kriterium für eindeutig entschlüsselbar

## Satz Eindeutig entschlüsselbar

$C$  ist ein eindeutig entschlüsselbarer Code  $\Leftrightarrow$  Kein Suffix ist Codewort in  $C$ .

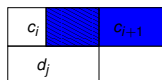
z.z.:  $C$  nicht eindeutig entschlüsselbar  $\Rightarrow$  Suffix ist Codewort

- Sei  $C$  nicht eindeutig entschlüsselbar.
- Dann existiert ein String  $s \in \{0, 1\}^*$ , der sich auf zwei Arten als Codewortfolge darstellen lässt.
- Seien  $c_1 \dots c_n$  und  $d_1 \dots d_m$  diese Codewortfolgen.
- Wir konstruieren sukzessive Suffixe für diese Folgen.
- Die konstruierten Suffixe beginnen jeweils mit Codewortpräfixen.
- Der letzte Suffix ist identisch mit dem letzten Codewort.

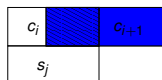


# Suffix ist Codewort

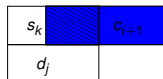
- Fall 1: Codewort  $c_i$  lässt sich zu  $d_j$  erweitern



- Fall 2: Codewort  $c_i$  lässt sich zu Suffix  $s_j$  erweitern



- Fall 3: Suffix  $s_k$  lässt sich zu Codewort  $d_j$  erweitern



# Rückrichtung

z.z.: Suffix  $s$  ist ein Codewort  $\Rightarrow C$  ist nicht eindeutig entschlüsselbar

- Suffix  $s$  ist aus Anwendungen der drei Regeln entstanden.
- Berechne die Kette zurück, aus der  $s$  entstanden ist.
  - ▶ Setze String  $c^* \leftarrow s$ . Iteriere:
    - ▶ 1. Fall  $c_i = c_j s$ :  $c^* \leftarrow c_j c^*$ , terminiere.
    - ▶ 2. Fall  $s' = c s$ :  $c^* \leftarrow c c^*$ ,  $s \leftarrow s'$ .
    - ▶ 3. Fall  $c = s' s$ :  $c^* \leftarrow s' c^*$ ,  $s \leftarrow s'$ .
- Kette muss mit 1. Fall  $c_i = c_j s'$  terminieren.
- Zwei verschiedene Entschlüsselungen:  
Eine beginnt mit  $c_i$ , die andere mit  $c_j$ .
- Beide sind gültig, da der letzte Suffix ein Codewort ist.

**Beispiel:** Für  $C = \{1, 110, 101\}$  erhalten wir für den Suffix 1 den String  $c^* = 1101$  mit gültigen Dekodierungen  $1|101$  und  $110|1$ .

# Sätze von Kraft und McMillan

## Satz von Kraft

Ein Präfixcode  $C$  für das Alphabet  $A = \{a_1, \dots, a_n\}$  mit Kodierungslängen  $|C(a_j)| = \ell_j$  existiert gdw

$$\sum_{j=1}^n 2^{-\ell_j} \leq 1.$$

## Satz von McMillan

Ein eindeutig entschlüsselbarer Code  $C$  für das Alphabet  $A = \{a_1, \dots, a_n\}$  mit Kodierungslängen  $|C(a_j)| = \ell_j$  existiert gdw

$$\sum_{j=1}^n 2^{-\ell_j} \leq 1.$$

# Präfixcodes genügen

## Korollar

Ein Präfixcode  $C$  existiert gdw es einen eindeutig entschlüsselbaren Code  $C$  mit denselben Kodierungslängen gibt.

## Beweis:

- Wir zeigen den Ringschluss für:  
 $\sum_{j=1}^n 2^{-\ell_j} \leq 1 \Rightarrow \text{Präfix} \Rightarrow \text{Eindeutig entschlüsselbar}$   
(Präfix  $\Rightarrow$  Eindeutig entschlüsselbar: letzte Vorlesung)
- Gegeben sind Kodierungslängen  $\ell_j$ .
- Gesucht ist ein Präfixcode mit  $\ell_j = |C(a_j)|$ .
- Definiere  $\ell := \max\{\ell_1, \dots, \ell_n\}$ ,  $n_i := \text{Anzahl } \ell_j \text{ mit } \ell_j = i$ .

$$\sum_{j=1}^n 2^{-\ell_j} = \sum_{i=1}^{\ell} n_i 2^{-i} \leq 1.$$



# Beweis: $\sum_{i=1}^{\ell} n_i 2^{-i} \leq 1 \Rightarrow$ Präfix

## Induktion über $\ell$ :

- **IA**  $\ell = 1$ : Aus  $\sum_{i=1}^{\ell} n_i 2^{-i}$  folgt  $n_1 \leq 2$ .
- Können Präfixcode  $C \subseteq \{0, 1\}^{\ell}$  für max. 2 Codeworte konstruieren.
- **IS**  $\ell - 1 \rightarrow \ell$ : Es gilt  $n_{\ell} \leq 2^{\ell} - n_1 2^{\ell-1} - n_2 2^{\ell-2} - \dots - n_{\ell-1} 2$ .
- **IV**: Präfixcode  $C'$  mit  $n_i$  Worten der Länge  $i$ ,  $i = 1, \dots, \ell - 1$ .
- Anzahl der Worte der Länge  $\ell$ :  $2^{\ell}$
- Wir zählen die durch  $C'$  ausgeschlossenen Worte der Länge  $\ell$ .
- Sei  $c_i \in C'$  mit Länge  $\ell_i$ . Dann enthalten alle  $c_i s \in \{0, 1\}^{\ell}$  mit beliebigem  $s \in \{0, 1\}^{\ell - \ell_i}$  den Präfix  $c_i$ .
- Durch Präfixe der Länge 1 ausgeschlossene Worte:  $n_1 \cdot 2^{\ell-1}$ .
- Durch Präfixe der Länge 2 ausgeschlossene Worte:  $n_2 \cdot 2^{\ell-2}$ .
- $\vdots$
- Durch Präfixe der Länge  $\ell - 1$  ausgeschlossene Worte:  $n_{\ell-1} \cdot 2$ .
- D.h. wir kodieren die  $n_{\ell}$  Worte mit den verbleibenden  $2^{\ell} - (n_1 2^{\ell-1} + \dots + n_{\ell-1} 2) \geq n_{\ell}$  Worten der Länge  $\ell$ .
- Der resultierende Code ist ein Präfixcode.

# Eindeutig entschlüsselbar $\Rightarrow \sum_{j=1}^n 2^{-\ell_j} \leq 1$

- Sei  $C$  eindeutig entschlüsselbar mit  $C(a_j) = \ell_j$ ,  $\ell = \max_j \{\ell_j\}$ .
- Wählen  $r \in \mathbb{N}$  beliebig. Betrachten

$$\left( \sum_{j=1}^n 2^{-\ell_j} \right)^r = \sum_{i=1}^{r\ell} n_i 2^{-i} \text{ für } n_i \in \mathbb{N}.$$

- Interpretation der  $n_i$ : Anzahl Strings aus  $\{0, 1\}^i$ , die sich als Folge von  $r$  Codeworten schreiben lässt.
- $C$  eindeutig entschlüsselbar: Jeder String aus  $\{0, 1\}^i$  lässt sich als höchstens eine Folge von Codeworten schreiben, d.h.  $n_i \leq 2^i$ .
- Damit gilt  $\sum_{i=1}^{r\ell} n_i 2^{-i} \leq r\ell \Rightarrow \sum_{j=1}^n 2^{-\ell_j} \leq (r\ell)^{\frac{1}{r}}$
- Für  $r \rightarrow \infty$  folgt  $\sum_{j=1}^n 2^{-\ell_j} \leq 1$ .

# Huffman Kodierung

**Szenario:** Quelle  $Q$  mit Symbole  $\{a_1, \dots, a_n\}$

- $a_i$  sortiert nach absteigenden Quellws.  $p_1 \geq p_2 \geq \dots \geq p_n$ .

## Algorithmus Huffman-Kodierung

**Eingabe:** Symbole  $a_i$  mit absteigend sortierten  $p_i$ ,  $i = 1, \dots, n$ .

- 1 IF ( $n=2$ ), Ausgabe  $C(a_1) = 0$ ,  $C(a_2) = 1$ .
- 2 ELSE
  - 1 Bestimme  $k \in \mathbb{Z}_{n-1}$  mit  $p_k \geq p_{n-1} + p_n \geq p_{k+1}$ .
  - 2  $(p_1, \dots, p_k, p_{k+1}, p_{k+2}, \dots, p_{n-1}) \leftarrow$   
 $(p_1, \dots, p_k, p_{n-1} + p_n, p_{k+1}, \dots, p_{n-2})$
  - 3  $(C(a_1), \dots, C(a_{k-1}), C(a_{k+1}), \dots, C(a_{n-2}), C(a_k)0, C(a_k)1) \leftarrow$   
Huffmann-Kodierung( $a_1, \dots, a_{n-1}, p_1, \dots, p_{n-1}$ )

**Ausgabe:** kompakter Präfixcode für  $Q$

**Laufzeit:**  $\mathcal{O}(n^2)$

( $\mathcal{O}(n \log n)$  mit Hilfe von Heap-Datenstruktur)

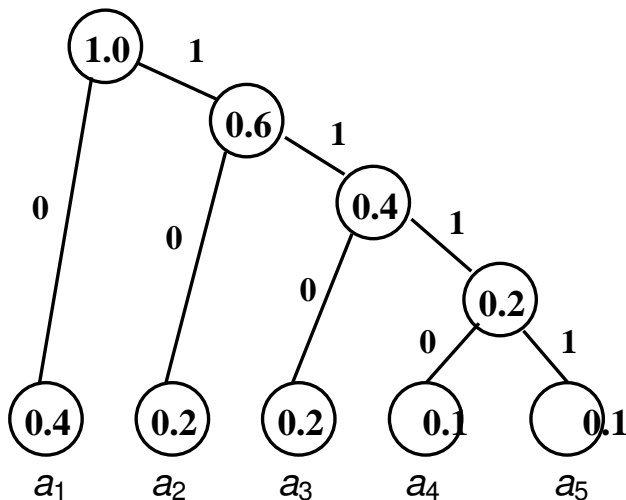
# Beispiel Huffman-Kodierung

**Beispiel:**  $p_1 = 0.4$ ,  $p_2 = p_3 = 0.2$ ,  $p_4 = p_5 = 0.1$

$a_i$	$p_i$	$C(a_i)$	$p_i$	$C(a_i)$	$p_i$	$C(a_i)$	$p_i$	$C(a_i)$
$a_1$	0.4	00	0.4	00	<b>0.4</b>	1	<b>0.6</b>	0
$a_2$	0.2	01	0.2	01	0.4	00	0.4	1
$a_3$	0.2	11	<b>0.2</b>	10	0.2	01		
$a_4$	0.1	100	0.2	11				
$a_5$	0.1	101						

- **Fett** gedruckt: Stelle  $k$  Man beachte:  $k$  ist nicht eindeutig, d.h.  $C$  ist nicht eindeutig.
- $E(C) = (0.4 + 0.2 + 0.2) * 2 + 2 * 0.1 * 3 = 2.2$
- Huffman-Tabelle: Spalten 1 und 3. Mittels Huffman-Tabelle kann jeder String  $m \in A^*$  in Zeit  $\mathcal{O}(|C(m)|)$  kodiert werden.

## Wahl eines anderen $k$



$$E(C') = 0.4 * 1 + 0.2 * (2 + 3) + 0.1 * 2 * 4 = 2.2$$

# Eigenschaften kompakter Codes

Sei  $l_i := |C(a_i)|$ .

## Lemma Eigenschaften kompakter Codes

Sei  $C$  ein kompakter Code, oBdA ist  $C$  ein Präfixcode.

- 1 Falls  $p_i > p_j$ , dann ist  $l_i \leq l_j$
- 2 Es gibt mindestens zwei Codeworte in  $C$  mit maximaler Länge.
- 3 Unter den Worten mit maximaler Länge existieren zwei Worte, die sich nur in der letzten Stelle unterscheiden.

# Beweis der Eigenschaften

## Beweis:

- 1 Sei  $l_i > l_j$ . Dann gilt

$$\begin{aligned} p_i l_i + p_j l_j &= p_i(l_i - l_j + l_j) + p_j(l_j - l_i + l_i) \\ &= p_i l_j + p_j l_i + (l_i - l_j)(p_i - p_j) > p_i l_j + p_j l_i \end{aligned}$$

D.h. vertauschen der Kodierungen von  $a_i$  und  $a_j$  verkürzt den Code.

- 2 Sei  $c = c_1 \dots c_n \in C$  das einzige Codewort mit maximaler Länge. Streichen von  $c_n$  führt zu einem Präfixcode mit kürzerer erwarteter Codewortlänge.
- 3 Annahme: Alle Paare von Codeworten maximaler Länge unterscheiden sich nicht nur in der letzten Komponente.
  - ▶ Entferne die letzte Komponente eines beliebigen Codewortes maximaler Länge.
  - ▶ Wir erhalten einen Präfixcode mit kürzerer Länge.

# Optimalität der Huffman-Kodierung

## Satz

Die Huffman-Kodierung liefert einen kompakten Code.

### Beweis per Induktion über $n$ .

- **IA:**  $n = 2$ : Für  $\{a_1, a_2\}$  ist die Codierung  $\{0, 1\}$  kompakt.
- **IS:**  $n - 1 \rightarrow n$ : Sei  $C'$  ein kompakter Code für  $\{a_1, \dots, a_n\}$ .
  - ▶ Lemma,2+3:  $C'$  enthält zwei Codeworte maximaler Länge, die sich nur in der letzten Stelle unterscheiden.
  - ▶ Seien dies die Codeworte  $c_0, c_1$  für ein  $c \in \{0, 1\}^*$ .
  - ▶ Lemma,1: Die beiden Symbole  $a_{n-1}, a_n$  mit kleinster Quellws besitzen maximale Codewortlänge.
  - ▶ Vertausche die Kodierungen dieser Symbole mit  $c_0, c_1$ .
  - ▶  $a_{n-1}$  oder  $a_n$  tauchen mit Ws  $p_{n-1} + p_n$  auf.
  - ▶ Ersetze  $a_{n-1}$  und  $a_n$  durch  $a'$ .
  - ▶ **IV:** Huffman-Kodierung liefert kompakten Präfixcode  $C$  für  $a_1, \dots, a_{n-2}, a'$  mit Quellws  $p_1, \dots, p_{n-2}, p_{n-1} + p_n$
  - ▶ D.h.  $C(a_1), \dots, C(a_{n-2}), C(a')0 = c_0, C(a')1 = c_1$  ist Präfixcode mit erwarteter Codewortlänge  $E(C')$ .
  - ▶ Damit liefert die Huffman-Kodierung einen kompakten Präfixcode.



# Informationsgehalt einer Nachricht

Betrachten folgendes Spiel

- Gegeben: Quelle  $Q$  mit unbekanntem Symbolen  $\{a_1, a_2\}$  und  $p_1 = 0.9, p_2 = 0.1$ .
- Zwei Spieler erhalten rundenweise je ein Symbol.
- Gewinner ist, wer zuerst beide Symbole erhält.

Szenario:

- Spieler 1 erhält in der ersten Runde  $a_1$  und Spieler 2 erhält  $a_2$ .
- **Frage:** Wer gewinnt mit höherer  $Ws$ ? Offenbar Spieler 2.

**Intuitiv:** Je kleiner die Quellws, desto höher der Informationsgehalt.

# Eigenschaft von Information

Forderungen für eine Informationsfunktion

- 1  $I(p) \geq 0$ : Der Informationsgehalt soll positiv sein.
- 2  $I(p)$  ist stetig in  $p$ : Kleine Änderungen in der Ws  $p$  sollen nur kleine Änderungen von  $I(p)$  bewirken.
- 3  $I(p_i) + I(p_j) = I(p_i p_j)$ :
  - ▶  $X$  = Ereignis, dass  $a_i$  und  $a_j$  nacheinander übertragen werden.
  - ▶ Informationsgehalt von  $X$ :  $I(p_i) + I(p_j)$ ,  $W_s(X) = p_i p_j$

## Satz zur Struktur von $I(p)$

Jede Funktion  $I(p)$  für  $0 < p \leq 1$ , die obige drei Bedingungen erfüllt, ist von der Form

$$I(p) = C \log_2 \frac{1}{p}$$

für eine positive Konstante  $C$ .

## Beweis: Form von $I(p)$

- Forderung 3 liefert  $I(p^2) = I(p) + I(p) = 2I(p)$ .
- Induktiv folgt:  $I(p^n) = nI(p)$  für alle  $n \in \mathbb{N}$  und alle  $0 < p \leq 1$ .
- Substitution  $p \rightarrow p^{\frac{1}{n}}$  liefert:  $I(p) = nI(p^{\frac{1}{n}})$  bzw.  $I(p^{\frac{1}{n}}) = \frac{1}{n}I(p)$
- Damit gilt für alle  $q \in \mathbb{Q}$ :  $I(p^q) = qI(p)$ .
- Für jedes  $r \in \mathbb{R}$  gibt es eine Sequenz  $q_i$  mit  $\lim_{n \rightarrow \infty} q_n = r$ . Aus der Stetigkeit von  $I(p)$  folgt

$$I(p^r) = I(\lim_{n \rightarrow \infty} p^{q_n}) = \lim_{n \rightarrow \infty} I(p^{q_n}) = \lim_{n \rightarrow \infty} q_n I(p) = rI(p)$$

- Fixiere  $0 < q < 1$ . Für jedes  $0 < p \leq 1$  gilt

$$\begin{aligned} I(p) &= I(q^{\log_q p}) = I(q) \log_q p = -I(q) \log_q \left(\frac{1}{p}\right) = -I(q) \frac{\log_2 \frac{1}{p}}{\log_2 q} \\ &= C \log_2 \frac{1}{p} \quad \text{mit } C = -I(q) \cdot \frac{1}{\log_2(q)} > 0. \end{aligned}$$

## Definition Information $I(p)$

### Definition $I(p)$

Die Information  $I(p)$  eines Symbols mit Quellws  $p > 0$  ist definiert als

$$I(p) = \log \frac{1}{p}.$$

Die Einheit der Information bezeichnet man als Bit.

# Beispiele für Information

- $Q = \{0, 1\}$  mit  $p_1 = p_2 = \frac{1}{2}$ . Dann ist  $I(\frac{1}{2}) = 1$ , d.h. für jedes gesendete Symbol erhält der Empfänger 1 Bit an Information.
- $Q = \{0, 1\}$  mit  $p_1 = 1, p_2 = 0$ . Dann ist  $I(1) = 0$ , d.h. der Empfänger enthält 0 Bit an Information pro gesendetem Zeichen.
- Beamer-Bild SXGA: Auflösung  $1280 * 1024$ , 256 Farben
  - ▶  $2^{1280*1024*8}$  mögliche Bilder. Annahme: Jedes gleich wahrscheinlich.
  - ▶ Information in Bit:  $I(2^{-1280*1024*8}) = 1280 * 1024 * 8 = 10.485.760$
- Meine Erklärung dieser Folie:  
 $\leq 1000$  Worte,  $\leq 10.000$  Worte Vokabular
  - ▶ Information meiner Erklärung:  $I(10.000^{-1000}) < 13.288$
  - ▶ Beispiel für "Ein Bild sagt mehr als 1000 Worte!"

# Entropie einer Quelle

## Definition Entropie einer Quelle

Sei  $Q$  eine Quelle mit Quellws  $P = \{p_1, \dots, p_n\}$ . Die *Entropie von  $Q$*  ist definiert als

$$H(Q) = \sum_{i=1}^n p_i I(p_i) = \sum_{i=1}^n p_i \log \frac{1}{p_i} = - \sum_{i=1}^n p_i \log p_i.$$

- D.h. Entropie ist die erwartete Information pro Quellsymbol.
- Für  $p_i = 0$  definieren wir  $p_i \log \frac{1}{p_i} = 0$ .
- $P = \{\frac{1}{n}, \frac{1}{n}, \dots, \frac{1}{n}\} : H(Q) = \sum_{i=1}^n \frac{1}{n} \log n = \log n$
- $P = \{\frac{1}{n}, \frac{1}{n}, \dots, \frac{1}{n}, 0\} : H(Q) = \sum_{i=1}^n \frac{1}{n} \log n = \log n$
- $P = \{1, 0, 0, \dots, 0\} : H(Q) = 1 * \log 1 = 0$

Wollen zeigen:  $0 \leq H(Q) \leq \log n$ .

# Wechsel zu anderer Ws-Verteilung

## Lemma Wechsel Ws-Verteilung

Sei  $P = \{p_1, \dots, p_n\}$  eine Ws-Verteilung und  $Q = \{q_1, \dots, q_n\}$  mit  $\sum_{i=1}^n q_i \leq 1$ . Dann gilt

$$\sum_{i=1}^n p_i l(p_i) \leq \sum_{i=1}^n p_i l(q_i).$$

Gleichheit gilt genau dann, wenn  $p_i = q_i$  für alle  $i = 1, \dots, n$ .

### Beweis:

Nützliche Ungleichung für das Rechnen mit logs:

$$x - 1 \geq \ln x = \log x \cdot \ln 2 \quad \text{für alle } x > 0$$

Gleichheit gilt gdw  $x = 1$ .

# Beweis des Lemmas

$$\begin{aligned}\sum_{i=1}^n p_i l(p_i) - \sum_{i=1}^n p_i l(q_i) &= \sum_{i=1}^n p_i \left( \log \frac{1}{p_i} - \log \frac{1}{q_i} \right) \\ &= \sum_{i=1}^n p_i \log \frac{q_i}{p_i} \\ &\leq \frac{1}{\ln 2} \sum_{i=1}^n p_i \left( \frac{q_i}{p_i} - 1 \right) \\ &= \frac{1}{\ln 2} \left( \sum_{i=1}^n q_i - \sum_{i=1}^n p_i \right) \\ &= \frac{1}{\ln 2} \left( \sum_{i=1}^n q_i - 1 \right) \leq 0.\end{aligned}$$

Gleichheit gilt gdw  $\frac{q_i}{p_i} = 1$  für alle  $i = 1, \dots, n$ .



# Untere und obere Schranken für $H(P)$

## Satz Schranken für $H(P)$

Sei  $Q$  eine Quelle mit Ws-Verteilung  $P = \{p_1, \dots, p_n\}$ . Dann gilt

$$0 \leq H(Q) \leq \log n.$$

Weiterhin gilt  $H(Q) = \log n$  gdw alle  $p_i = \frac{1}{n}$  für  $i = 1, \dots, n$  und  $H(Q) = 0$  gdw  $p_i = 1$  für ein  $i \in [n]$ .

### Beweis:

- Sei  $P' = \{\frac{1}{n}, \dots, \frac{1}{n}\}$  die Gleichverteilung.
- Nach Lemma zum Wechsel von Ws-Verteilungen gilt

$$H(Q) = \sum_{i=1}^n p_i \log \frac{1}{p_i} \leq \sum_{i=1}^n p_i \log \frac{1}{p'_i} = \log n \sum_{i=1}^n p_i = \log n.$$

- Gleichheit gilt gdw  $p_i = p'_i = \frac{1}{n}$  für alle  $i$ .

## Untere Schranke für $H(P)$

Verwenden Ungleichung  $\log x \geq 0$  für  $x \geq 1$ . Gleichheit gilt gdw  $x = 1$ .

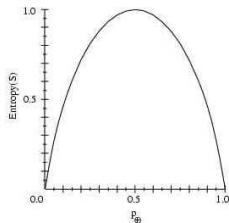
$$H(Q) = \sum_{i=1}^n p_i \log \frac{1}{p_i} \geq 0,$$

mit Gleichheit gdw  $\frac{1}{p_i} = 1$  für ein  $i \in [n]$ . □

### Bsp: binäre Entropiefunktion

- Binäre Quelle  $Q = \{a_1, a_2\}$  mit  $P = \{p, 1 - p\}$

$$H(Q) = p \log \frac{1}{p} + (1 - p) \log \frac{1}{1 - p}.$$



# Kodieren einer binären Quelle

**Szenario:** Binäre Quelle  $Q$  mit  $P = \{\frac{1}{4}, \frac{3}{4}\}$  mit

$$H(Q) = \frac{1}{4} \cdot \log 4 + \frac{3}{4} \cdot \log \frac{4}{3} \approx 0.811.$$

- Huffman-Kodierung von  $Q$ :

$$C(a_1) = 0, C(a_2) = 1 \text{ mit } E(C) = 1.$$

- **Problem:** Wie können wir  $a_2$  mit kurzem Codewort kodieren?
- **Idee:** Kodieren Zweierblöcke von Quellsymbolen.

## Quellerweiterungen von $Q$

- Betrachten  $Q^2 = \{a_1 a_1, a_1 a_2, a_2 a_1, a_2 a_2\}$  mit Quellws

$$p_1 = \frac{1}{16}, p_2 = p_3 = \frac{3}{16}, p_4 = \frac{9}{16}.$$

- Huffman-Kodierung von  $Q^2$  liefert

$$C(a_1 a_1) = 101, C(a_2 a_1) = 100, C(a_1 a_2) = 11, C(a_2 a_2) = 0$$

$$\text{mit } E(C) = 3 \cdot \left(\frac{1}{16} + \frac{3}{16}\right) + 2 \cdot \frac{3}{16} + 1 \cdot \frac{9}{16} = \frac{27}{16}.$$

- Jedes Codewort kodiert zwei Quellsymbole, d.h. die durchschnittliche Codewortlänge pro Quellsymbol ist

$$E(C)/2 = \frac{27}{32} = 0.844.$$

- *Übung:* Für  $Q^3$  erhält man 0.823.

# $k$ -te Quellerweiterung $Q^k$

## Definition $k$ -te Quellerweiterung

Sei  $Q$  eine Quelle mit Alphabet  $A = \{a_1, \dots, a_n\}$  und Ws-Verteilung  $P = \{p_1, \dots, p_n\}$ . Die  $k$ -te Quellerweiterung  $Q^k$  von  $Q$  ist definiert über dem Alphabet  $A^k$ , wobei  $a = a_{i_1} \dots a_{i_k} \in A^k$  die Quellws  $p_{i_1} \cdot p_{i_2} \cdot \dots \cdot p_{i_k}$  besitzt.

## Satz Entropie von $Q^k$

Sei  $Q$  eine Quelle mit  $k$ -ter Quellerweiterung  $Q^k$ . Dann gilt

$$H(Q^k) = k \cdot H(Q).$$

# Beweis für $H(Q^k)$

$$\begin{aligned} H(Q^k) &= \sum_{(i_1, \dots, i_k) \in [n]^k} p_{i_1} \dots p_{i_k} \log \frac{1}{p_{i_1} \dots p_{i_k}} \\ &= \sum_{(i_1, \dots, i_k) \in [n]^k} p_{i_1} \dots p_{i_k} \log \frac{1}{p_{i_1}} + \dots + \sum_{(i_1, \dots, i_k) \in [n]^k} p_{i_1} \dots p_{i_k} \log \frac{1}{p_{i_k}} \end{aligned}$$

- Betrachten ersten Summanden

$$\begin{aligned} \sum_{(i_1, \dots, i_k) \in [n]^k} p_{i_1} \dots p_{i_k} \log \frac{1}{p_{i_1}} &= \sum_{i_1 \in [n]} p_{i_1} \log \frac{1}{p_{i_1}} \cdot \sum_{i_2 \in [n]} p_{i_2} \dots \sum_{i_k \in [n]} p_{i_k} \\ &= \sum_{i_1 \in [n]} p_{i_1} \log \frac{1}{p_{i_1}} \cdot 1 \dots 1 = H(Q). \end{aligned}$$

- Analog liefern die anderen  $k - 1$  Summanden jeweils  $H(Q)$ .

# Kodierungstheorem von Shannon

## Kodierungstheorem von Shannon (1948)

Sei  $Q$  eine Quelle für  $\{a_1, \dots, a_n\}$  mit Ws-Verteilung  $P = \{p_1, \dots, p_n\}$ .  
Sei  $C$  ein kompakter Code für  $Q$ . Dann gilt für die erwartete Codewortlänge

$$H(Q) \leq E(C) < H(Q) + 1.$$

**Beweis:**  $H(Q) \leq E(C)$

- Bezeichnen Codewortlängen  $\ell_i := |C(a_i)|$  und  $q_i := 2^{-\ell_i}$ .
- Nach Satz von McMillan gilt:  $\sum_{i=1}^n q_i = \sum_{i=1}^n 2^{-\ell_i} \leq 1$ .
- Lemma Wechsel Ws-Verteilung liefert

$$\begin{aligned} H(Q) &= \sum_{i=1}^n p_i \log \frac{1}{p_i} \leq \sum_{i=1}^n p_i \log \frac{1}{q_i} \\ &= \sum_{i=1}^n p_i \log 2^{\ell_i} = \sum_{i=1}^n p_i \ell_i = E(C). \end{aligned}$$

# $E(C) \leq H(Q) + 1$

- Wir konstruieren aus  $p_1, \dots, p_n$  einen Code  $C'$ .
- Die Codewortlängen  $\ell_i$  von  $C'$  werden wie folgt gewählt

$$\log \frac{1}{p_i} \leq \ell_i < \log \frac{1}{p_i} + 1.$$

Ein Code  $C'$  mit dieser Eigenschaft heißt *Shannon-Fano Code*.

- Damit gilt

$$\sum_{i=1}^n 2^{-\ell_i} \leq \sum_{i=1}^n 2^{-\log \frac{1}{p_i}} = \sum_{i=1}^n p_i = 1.$$

- Nach dem Satz von McMillan existiert ein eindeutig entschlüsselbarer Code  $C'$  mit diesen Codewortlängen  $\ell_i$ .
- Für jeden kompakten Code  $C$  gilt andererseits

$$\begin{aligned} E(C) \leq E(C') &= \sum_{i=1}^n p_i \ell_i < \sum_{i=1}^n p_i \left( \log \frac{1}{p_i} + 1 \right) \\ &= \sum_{i=1}^n p_i \log \frac{1}{p_i} + \sum_{i=1}^n p_i = H(Q) + 1 \end{aligned}$$



# Anwendung auf Quellerweiterungen

## Korollar zu Shannons Kodierungstheorem

Sei  $Q$  eine Quelle mit  $k$ -ter Quellerweiterung  $Q^k$ . Sei  $C$  ein kompakter Code für  $Q^k$ . Dann gilt

$$H(Q) \leq \frac{E(C)}{k} < H(Q) + \frac{1}{k}.$$

### Beweis:

- Anwendung von Shannon's Kodierungstheorem auf  $Q^k$  liefert

$$H(Q^k) \leq E(C) < H(Q^k) + 1.$$

- Anwenden von  $H(Q^k) = kH(Q)$  und teilen durch  $k$  liefert die Behauptung.

# Szenario für fehlerkorrigierende Codes

## Definition $(n, M)$ -Code

Sei  $C \subseteq \{0, 1\}^n$  ein binärer Blockcode der Länge  $n$  mit  $|C| = M$  Codeworten. Dann bezeichnen wir  $C$  als  $(n, M)$ -Code.

## Erinnerung: Binärer symmetrischer Kanal

- Bits 0,1 kippen mit Ws  $p, p < \frac{1}{2}$  zu 1,0.
- Korrekte Übertragung  $0 \mapsto 0, 1 \mapsto 1$  mit Ws  $1 - p$ .
- Kanal ist erinnerungslos, d.h. die Ws sind unabhängig von vorigen Ereignissen.
- Vorwärts-Kanalws:  $Ws[\mathbf{x} \text{ empfangen} \mid \mathbf{c} \text{ gesendet}]$ .
- Rückwärts-Kanalws:  $Ws[\mathbf{c} \text{ gesendet} \mid \mathbf{x} \text{ empfangen}]$ .

# Dekodieren

## Definition Dekodier-Kriterium

Sei  $C \subseteq \{0, 1\}^n$  ein  $(n, M)$ -Code. Ein Dekodier-Kriterium  $f$  ist eine Funktion  $f : \{0, 1\}^n \rightarrow C \cup \{\perp\}$ .

- Sei  $\mathbf{x} \in \{0, 1\}^n$ . Ein Dekodier-Kriterium dekodiert  $\mathbf{x}$  zu  $f(\mathbf{x}) \in C$  oder gibt Dekodierfehler  $f(\mathbf{x}) = \perp$  aus.
- **Ziel:** Konstruktion eines Dekodier-Kriteriums  $f$ , dass die Ws des korrekten Dekodierens maximiert.

$Ws[\text{korrekte Dekodierung} \mid \mathbf{x} \text{ empfangen}] = Ws[f(\mathbf{x}) \text{ gesendet} \mid \mathbf{x} \text{ empfangen}]$

- Summieren über alle möglichen empfangenen  $\mathbf{x}$  liefert

$$Ws[\text{korr. Dekodierung}] = \sum_{\mathbf{x} \in \{0,1\}^n} Ws[f(\mathbf{x}) \text{ gesendet} \mid \mathbf{x} \text{ empfangen}] \cdot Ws[\mathbf{x} \text{ empfangen}]$$

- Wir maximieren die Rückwärts-Kanalws

$$Ws[f(\mathbf{x}) \text{ gesendet} \mid \mathbf{x} \text{ empfangen}].$$

- D.h. wir dekodieren zu demjenigen Codewort  $\mathbf{c} = f(\mathbf{x})$ , das mit höchster Ws gesendet wurde.

# Maximum Likelihood Dekodierung

## Definition Maximum Likelihood Dekodierung

Ein Dekodierkriterium  $f$  heißt *Maximum-Likelihood Kriterium*, falls es die Vorwärts- $W_s$  für alle Codeworte maximiert, d.h.

$$W_s[\mathbf{x} \text{ empfangen} | f(\mathbf{x}) \text{ gesendet}] = \max_{c \in C} W_s[\mathbf{x} \text{ empfangen} | c \text{ gesendet}].$$

Eine Anwendung von  $f$  heißt Maximum-Likelihood Dekodierung.

# Warum Maximum Likelihood?

## Satz Maximum Likelihood optimal für gleichverteilte Codeworte

Sei  $C$  ein  $(n, M)$ -Code und  $W_s[\mathbf{c} \text{ gesendet}] = \frac{1}{M}$  für alle  $\mathbf{c} \in C$ . Dann maximiert die Maximum-Likelihood Dekodierung die  $W_s$  des korrekten Dekodierens.

### Beweis:

$W_s[\mathbf{c} \text{ gesendet} \mid \mathbf{x} \text{ empfangen}]$

$$\begin{aligned} &= \frac{W_s[\mathbf{x} \text{ empfangen} \mid \mathbf{c} \text{ gesendet}] W_s[\mathbf{c} \text{ gesendet}]}{\sum_{i=1}^M W_s[\mathbf{x} \text{ empfangen} \mid \mathbf{c}_i \text{ gesendet}] W_s[\mathbf{c}_i \text{ gesendet}]} \\ &= \frac{W_s[\mathbf{x} \text{ empfangen} \mid \mathbf{c} \text{ gesendet}]}{\sum_{i=1}^M W_s[\mathbf{x} \text{ empfangen} \mid \mathbf{c}_i \text{ gesendet}]} \end{aligned}$$

- Nenner ist konstant für jeden Kanal.
- Maximum-Likelihood maximiert den Zähler und damit den Term.

# Dekodieren zum Nachbarn minimaler Distanz

## Definition Hamming-Distanz

Seien  $\mathbf{x}, \mathbf{y} \in \{0, 1\}^n$ . Die Hamming-Distanz  $d(\mathbf{x}, \mathbf{y})$  ist definiert als die Anzahl der Stellen, an denen sich  $\mathbf{x}$  und  $\mathbf{y}$  unterscheiden.

## Satz

In jedem binären symmetrischen Kanal ist das Dekodier-Kriterium, das ein  $\mathbf{x}$  zum Codewort minimaler Hamming-Distanz dekodiert ein Maximum-Likelihood Kriterium.

## Beweis:

- Ws von genau  $k$  Fehlern an festen Stellen beim Senden von  $\mathbf{c}$

$$\text{Ws}[\mathbf{x} \text{ empfangen} | \mathbf{c} \text{ gesendet}] = p^k (1 - p)^{n-k}.$$

- Wegen  $p < \frac{1}{2}$  gilt  $p < 1 - p$ . Ein Dekodierkriterium  $f$ , das ein Codewort  $\mathbf{c}$  mit minimaler Distanz  $d(\mathbf{x}, \mathbf{c})$  wählt, minimiert  $k$ .
- Damit maximiert  $f$  die Vorwärts-Kanalws und ist somit ein Maximum-Likelihood Kriterium.

# Die Hamming-Distanz definiert eine Metrik.

## Satz Metrik Hamming-Distanz

Die Hamming-Distanz ist eine Metrik auf  $\{0, 1\}^n$ , d.h. für alle  $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \{0, 1\}^n$  gilt:

- 1 Positivität:  $d(\mathbf{x}, \mathbf{y}) \geq 0$ , Gleichheit gdw  $\mathbf{x} = \mathbf{y}$ .
- 2 Symmetrie:  $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$ .
- 3 Dreiecksungleichung:  $d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z})$ .

Beweis für 3:

**Ann.:**  $d(\mathbf{x}, \mathbf{z}) > d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z})$

- Verändern erst  $\mathbf{x}$  zu  $\mathbf{y}$ , dann  $\mathbf{y}$  zu  $\mathbf{z}$ .
- Müssen dazu  $d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z}) < d(\mathbf{x}, \mathbf{z})$  Stellen ändern.  
Widerspruch:  $\mathbf{x}$  und  $\mathbf{z}$  unterscheiden sich an  $d(\mathbf{x}, \mathbf{z})$  Stellen.

## Definition $u$ -fehlererkennend

Sei  $C$  ein Code und  $u \in \mathbb{N}$ .  $C$  ist  $u$ -fehlererkennend, falls für alle Codeworte  $\mathbf{c}, \mathbf{c}' \in C$  gilt:  $d(\mathbf{c}, \mathbf{c}') \geq u + 1$ . Ein Code ist *genau*  $u$ -fehlererkennend, falls er  $u$ -fehlererkennend ist, aber nicht  $(u + 1)$ -fehlererkennend.

## Bsp:

- Repetitionscode  $R(3) = \{000, 111\}$  ist genau 2-fehlererkennend.
- $R(n) = \{0^n, 1^n\}$  ist genau  $(n - 1)$ -fehlererkennend.
- $C = \{000000, 000111, 111111\}$  ist genau 2-fehlererkennend.



# Fehlerkorrektur

## Definition $v$ -fehlerkorrigierend

Sei  $C$  ein Code und  $v \in \mathbb{N}$ .  $C$  ist  $v$ -fehlerkorrigierend, falls für alle  $c \in C$  gilt: Bis zu  $v$  können mittels Dekodierung zum eindeutigen Codewort minimaler Hammingdistanz korrigiert werden.

Ein Code ist *genau*  $v$ -fehlerkorrigierend, falls er  $v$ -fehlerkorrigierend aber nicht  $(v + 1)$ -fehlerkorrigierend ist.

**Anmerkung:** Existieren zwei verschiedene Codeworte mit minimaler Hammingdistanz, so wird eine Dekodierfehlermeldung  $\perp$  ausgegeben.

## Bsp:

- $R(3) = \{000, 111\}$  ist genau 1-fehlerkorrigierend.
- $R(4)$  ist genau 1-fehlerkorrigierend.
- $R(n)$  ist genau  $\lfloor \frac{n-1}{2} \rfloor$ -fehlerkorrigierend.
- $C = \{0^9, 0^4 1^5, 1^9\}$  ist genau 1-fehlerkorrigierend.

# Minimaldistanz eines Codes

## Definition Minimaldistanz

Sei  $C$  ein Code mit  $|C| \geq 2$ . Die *Minimaldistanz*  $d(C)$  eines Codes ist definiert als

$$d(C) = \min_{\mathbf{c} \neq \mathbf{c}' \in C} \{d(\mathbf{c}, \mathbf{c}')\}$$

D.h.  $d(C)$  ist die minimale Distanz zweier verschiedener Codeworte.

## Bsp:

- $R(n)$  besitzt Minimaldistanz  $d(R(n)) = n$ .
- $C = \{0001, 0010, 0101\}$  besitzt  $d(C) = 1$ .
- $C = \{0^9, 0^4 1^5, 1^9\}$  besitzt  $d(C) = 4$ .

## Korollar Fehlererkennung

Ein Code  $C$  ist  $u$ -fehlererkennend gdw  $d(C) \geq u + 1$ .

# Fehlerkorrektur vs Minimaldistanz

## Satz Fehlerkorrektur vs Minimaldistanz

Ein Code  $C$  ist  $\nu$ -fehlerkorrigierend gdw  $d(C) \geq 2\nu + 1$ .

### Beweis:

$\Leftarrow$ :

- **Ann.:**  $C$  ist nicht  $\nu$ -fehlerkorrigierend.
- D.h. bei Übertragung von  $\mathbf{c}$  entsteht  $\mathbf{x}$  mit  $d(\mathbf{c}, \mathbf{x}) \leq \nu$  und  $\exists \mathbf{c}' \neq \mathbf{c} : d(\mathbf{x}, \mathbf{c}') \leq \nu$
- Dreiecksungleichung:  $d(\mathbf{c}, \mathbf{c}') \leq d(\mathbf{c}, \mathbf{x}) + d(\mathbf{x}, \mathbf{c}') \leq 2\nu$   
(Widerspruch:  $d(C) \geq 2\nu + 1$ )

# Beweis der Hinrichtung " $\Rightarrow$ "

**Ann.:** Es gibt  $\mathbf{c} \neq \mathbf{c}' \in C$  mit  $d(\mathbf{c}, \mathbf{c}') = d(C) \leq 2v$ .

- 1. Fall:  $d(\mathbf{c}, \mathbf{c}') \leq v$ .  $\mathbf{c}$  kann durch Ändern von höchstens  $v$  Stellen in  $\mathbf{x} = \mathbf{c}'$  überführt werden.  $\mathbf{x}$  wird fälschlich zu  $\mathbf{c}'$  dekodiert (Widerspruch:  $C$  ist  $v$ -fehlerkorrigierend)
- 2. Fall:  $v + 1 \leq d(\mathbf{c}, \mathbf{c}') \leq 2v$ .
- OBdA unterscheiden sich in  $\mathbf{c}, \mathbf{c}'$  in den ersten  $d(C)$  Positionen. (Anderfalls sortiere die Koordinaten um.)
- Betrachten  $\mathbf{x}$ , das durch  $v$  Fehler in den ersten Koordinaten von  $\mathbf{c}$  entsteht, so dass
  - ▶  $\mathbf{x}$  stimmt mit  $\mathbf{c}'$  auf den ersten  $v$  Koordinaten überein.
  - ▶  $\mathbf{x}$  stimmt mit  $\mathbf{c}$  auf den folgenden  $d(C)$  Koordinaten überein.
  - ▶  $\mathbf{x}$  stimmt mit  $\mathbf{c}, \mathbf{c}'$  auf den restlichen Koordinaten überein.
- Es gilt  $d(\mathbf{c}, \mathbf{x}) = v \geq d(C) - v = d(\mathbf{c}', \mathbf{x})$ .
- D.h. entweder wird  $\mathbf{x}$  fälschlich zu  $\mathbf{c}'$  dekodiert, oder es entsteht ein Dekodierfehler. (Widerspruch:  $C$  ist  $v$ -fehlerkorrigierend)

# $(n, M, d)$ -Code

## Definition $(n, M, d)$ -Code

Sei  $C \subseteq \{0, 1\}^n$  mit  $|C| = M$  und Distanz  $d(C) = d$ . Dann bezeichnet man  $C$  als  $(n, M, d)$ -Code. Man nennt  $(n, M, d)$  die *Parameter des Codes*.

### Bsp:

- $R(n)$  ist ein  $(n, 2, n)$ -Code.
- $C = \{0000, 0011\}$  ist ein  $(4, 2, 2)$ -Code.
- $C = \{00, 01, 10, 11\}$  ist ein  $(2, 4, 1)$ -Code.

## Korollar

Sei  $C$  ein  $(n, M, d)$ -Code.

- 1  $C$  ist genau  $v$ -fehlerkorrigierend gdw  $d = 2v + 1$  oder  $d = 2v + 2$ .
- 2  $C$  ist genau  $\left\lfloor \frac{d-1}{2} \right\rfloor$ -fehlerkorrigierend.

**(Fehlerkorrektur-Schranke)**

# Maximale Codes

## Definition Maximale Code

Ein  $(n, M, d)$ -Code  $C$  ist maximal, falls kein  $(n, M + 1, d)$ -Code  $C'$  existiert mit  $C \subset C'$ .

### Bsp:

- $C_0 = \{0000, 1111\}$  ist maximal.
- $C_1 = \{0000, 0011, 1111\}$  ist nicht maximal.
- $C_2 = \{0000, 0011, 1111, 1100\}$  ist nicht maximal.
- $C_3 = \{0000, 0011, 1111, 1100, 1001, 0110, 1010, 0101\}$  ist maximal.

# Erweiterung nicht-maximaler Codes

## Satz Erweiterung von Codes

Sei  $C \subseteq \{0, 1\}^n$  ein  $(n, M, d)$ -Code.  $C$  ist maximal gdw für alle  $\mathbf{x} \in \{0, 1\}^n$  gilt: Es gibt ein  $\mathbf{c} \in C$  mit  $d(\mathbf{x}, \mathbf{c}) < d$ .

“ $\Rightarrow$ ”

- **Ann.:** Sei  $\mathbf{x} \in \{0, 1\}^n$ , so dass für alle  $\mathbf{c} \in C : d(\mathbf{x}, \mathbf{c}) \geq d$
- Dann ist  $C \cup \{\mathbf{x}\}$  ein  $(n, M + 1, d)$ -Code. (Widerspruch:  $C$  ist maximal.)

“ $\Leftarrow$ ”

- **Ann.:** Sei  $C$  nicht maximal.
- D.h.  $\exists \mathbf{x} \in \{0, 1\}^n : C \cup \{\mathbf{x}\}$  ist ein  $(n, M + 1, d)$ -Code
- Dann gilt  $d(\mathbf{x}, \mathbf{c}) \geq d$  für alle  $\mathbf{c} \in C$ .

# Ws für Dekodierfehler bei maximalen Codes

## Satz Dekodierfehler bei maximalen Codes

Sei  $C$  ein maximaler  $(n, M, d)$ -Code für einen binären symmetrischen Kanal. Für die Fehlerws beim Dekodieren zum Codewort mit minimalem Hammingabstand gilt

$$\sum_{k=d}^n \binom{n}{k} p^k (1-p)^{n-k} \leq W_s(\text{Dekodierfehler}) \leq 1 - \sum_{k=0}^{\lfloor \frac{d-1}{2} \rfloor} \binom{n}{k} p^k (1-p)^{n-k}$$

### Beweis:

- Korrekte Dekodierung bei  $\leq \lfloor \frac{d-1}{2} \rfloor$  Fehlern, d.h. mit Ws mind.

$$\sum_{k=0}^{\lfloor \frac{d-1}{2} \rfloor} \binom{n}{k} p^k (1-p)^{n-k}.$$



# Ws für Dekodierfehler bei maximalen Codes

## Beweis: Fortsetzung

- Sei  $\mathbf{x}$  das bei Übertragung von  $\mathbf{c} \in C$  empfangene Wort mit
$$d(\mathbf{x}, \mathbf{c}) \geq d.$$
- Da  $C$  maximal ist, existiert ein  $\mathbf{c}' \in C$  mit  $d(\mathbf{x}, \mathbf{c}') < d \leq d(\mathbf{x}, \mathbf{c})$ .
- D.h.  $\mathbf{x}$  wird zu  $\mathbf{c}'$  dekodiert anstatt zu  $\mathbf{c}$ .
- Damit erhalten wir bei  $\geq d$  Fehlern stets inkorrekte Dekodierung.
- Dies geschieht mit Ws

$$\sum_{k=d}^n \binom{n}{k} p^k (1-p)^{n-k}.$$

# Hammingkugel

## Definition Hammingkugel

Sei  $\mathbf{x} \in \{0, 1\}^n$  und  $r \geq 0$ . Wir definieren die  $n$ -dimensionale Hammingkugel mit Mittelpunkt  $\mathbf{x}$  und Radius  $r$  als

$$B^n(\mathbf{x}, r) = \{\mathbf{y} \in \{0, 1\}^n \mid d(\mathbf{x}, \mathbf{y}) \leq r\}.$$

**Bsp:**

- $B^3(001, 1) = \{001, 101, 011, 000\}$ .

## Satz Volumen von $B^n(\mathbf{x}, r)$

Das Volumen der Hammingkugel  $B^n(\mathbf{x}, r)$  ist  $V^n(r) = \sum_{i=0}^r \binom{n}{i}$ .

**Beweis:**

- Es gibt  $\binom{n}{i}$  String mit Distanz  $i$  von  $x$ .

# Packradius eines Codes

## Definition Packradius eines Codes

Sei  $C$  ein  $(n, M, d)$ -Code. Der Packradius  $pr(C) \in \mathbb{N}$  von  $C$  ist die größte Zahl, so dass die Hammingkugeln  $B^n(\mathbf{c}, pr(C))$  für alle  $\mathbf{c} \in C$  disjunkt sind.

## Korollar

Sei  $C$  ein  $(n, M, d)$ -Code.

- 1 Der Packradius von  $C$  ist  $pr(C) = \lfloor \frac{d-1}{2} \rfloor$ .
- 2  $C$  ist genau  $v$ -fehlerkorrigierend gdw  $pr(C) = v$ .

# Perfekte Codes

## Definition Perfekter Code

Sei  $C \subseteq \{0, 1\}^n$  ein  $(n, M, d)$ -Code.  $C$  heißt *perfekt*, falls

$$M \cdot V^n \left( \left\lfloor \frac{d-1}{2} \right\rfloor \right) = 2^n.$$

D.h. die maximalen disjunkten Hammingkugeln um die Codeworte partitionieren  $\{0, 1\}^n$ .

- Nicht für alle  $(n, M, d)$ , die obige Bedingung erfüllen, gibt es auch einen Code.
- $\{0, 1\}^n$  ist ein perfekter  $(n, 2^n, 1)$ -Code
  - ▶ Packradius ist 0, Hammingkugeln bestehen nur aus Codewort selbst.
  - ▶ Perfekter Code, aber nutzlos für Fehlerkorrektur.
- $R(n)$  ist für ungerade  $n$  ein perfekter  $(n, 2, n)$ -Code.
  - ▶  $2 \cdot \sum_{i=0}^{\frac{n-1}{2}} \binom{n}{i} = 2 \cdot \frac{2^n}{2} = 2^n$
  - ▶ Code ist nutzlos, da er nur zwei Codeworte enthält.

# Beispiele für Codes

**Hamming Code:**  $\mathcal{H}(h)$  ist ein  $(2^h - 1, 2^{n-h}, 3)$ -Code.

$\mathcal{H}(h)$  ist perfekt, denn

$$2^{n-h} (1 + 2^h - 1) = 2^n.$$

**Golay Codes:**  $\mathcal{G}_{23}$  ist ein  $(23, 2^{12}, 7)$ -Code.

$\mathcal{G}_{24}$  ist ein  $(24, 2^{12}, 8)$ -Code.

Einsatz: Voyager für Bilder von Jupiter und Saturn.

Der Golay Code  $(23, 2^{12}, 7)$  ist perfekt, denn

$$2^{12} \cdot \sum_{i=0}^3 \binom{23}{i} = 2^{12} \cdot 2^{11} = 2^{23}.$$

**Reed-Muller Code:**  $RM(r, m)$  ist ein  $(2^m, 2^{1+\binom{m}{1}+\dots+\binom{m}{r}}, 2^{m-r})$ -Code.

$RM(1, m) = (2^m, 2^{m+1}, 2^{m-1})$ .

Einsatz: Mariner 9 für Bilder vom Mars.

Die einzigen perfekten, binären  $v$ -fehlerkorrigierenden Codes mit  $v \geq 2$  sind Repetitionscodes und der obige Golay Code  $\mathcal{G}_{23}$ .

# Die Rate eines Codes

## Definition Rate eines Codes

Sei  $C$  ein  $(n, M, d)$ -Code.

- 1 Die *Übertragungsrate* ist definiert als  $\mathcal{R}(C) = \frac{\log_2(M)}{n}$ .
- 2 Die *Fehlerrate* ist definiert als  $\delta(C) = \frac{\lfloor \frac{d-1}{2} \rfloor}{n}$ .

## Bsp:

- $C = \{0^n\}$  hat Übertragungsrate 0, aber perfekte Fehlerkorrektur.
- $C = \{0, 1\}^n$  hat Übertragungsrate 1, aber keine Fehlerkorrektur.
- $\mathcal{R}(R(n)) = \frac{1}{n}$  und  $\delta(R(n)) = \frac{\lfloor \frac{n-1}{2} \rfloor}{n}$ .
  - ▶ Übertragungsrate konvergiert gegen 0, Fehlerrate gegen  $\frac{1}{2}$ .
- $\mathcal{R}(\mathcal{H}(h)) = \frac{n-h}{n} = 1 - \frac{h}{n}$  und  $\delta(\mathcal{H}(h)) = \frac{1}{n}$ .
  - ▶ Übertragungsrate konvergiert gegen 1, Fehlerrate gegen 0.

# Die Größe $A(n, d)$ und optimale Codes

## Definition Optimaler Code

Wir definieren

$$A(n, d) = \max\{M \mid \exists \text{ binärer } (n, M, d) - \text{Code}\}$$

Ein  $(n, M, d)$ -Code heißt optimal, falls  $M = A(n, d)$ .

- Bestimmung von  $A(n, d)$  ist offenes Problem.
- Zeigen hier obere und untere Schranken für  $A(n, d)$ .
- Für kleine Werte von  $n, d$  bestimmen wir  $A(n, d)$  wie folgt:
  - ▶ Zeigen  $A(n, d) \leq M$ .
  - ▶ Konstruieren  $(n, M, d)$ -Code.
- $A(n, d) \leq 2^n$  für  $d \in [n]$ : höchstens  $2^n$  Codeworte der Länge  $n$ .
- $A(n, 1) = 2^n$ :  $C = \{0, 1\}^n$ .
- $A(n, n) = 2$ :  $R(n)$ .
- $A(n, d) \leq A(n, d')$  für  $d, d' \in [n]$  mit  $d' \leq d$  (Übung)

# Singleton-Schranke

## Satz Singleton-Schranke

$$A(n, d) \leq 2^{n-d+1}$$

### Beweis:

- Sei  $C$  ein optimaler  $(n, M, d)$ -Code, d.h.  $M = A(n, d)$ .
- Wir entfernen die letzten  $d - 1$  Stellen aller  $M$  Codeworte.
- Die resultierenden  $M$  Worte sind alle verschieden, da sich alle Codeworte in mindestens  $d$  Stellen unterscheiden.
- Es gibt  $M$  viele unterschiedliche Worte der Länge  $n - (d - 1)$ , d.h.

$$M \leq 2^{n-d+1}.$$



# Vereinfachte Plotkin-Schranke

## Satz Vereinfachte Plotkin-Schranke

Sei  $n < 2d$ , dann gilt

$$A(n, d) \leq \frac{2d}{2d - n}.$$

- Sei  $C$  ein optimaler  $(n, M, d)$  – Code und  $S = \sum_{i < j} d(\mathbf{c}_i, \mathbf{c}_j)$ .
- Je zwei Codeworte besitzen Distanz mindestens  $d$ , d.h.  $S \geq d \binom{M}{2}$ .
- Betrachten erste Stelle in allen Codeworten:
  - ▶ Sei  $k$  die Anzahl der Nullen und  $(M - k)$  die Anzahl der Einsen.
  - ▶ Erste Stelle liefert Beitrag von  $k(M - k)$  zu  $S$ .
  - ▶  $k(M - k)$  ist maximal für  $k = \frac{M}{2}$ , d.h.  $k(M - k) \leq \frac{M^2}{4}$ .
  - ▶ Analog für jede der  $n$  Stellen, d.h.  $S \leq \frac{nM^2}{4}$ .
- Kombination beider Schranken und Auflösen nach  $M$  liefert

$$M \leq \frac{2d}{2d - n}.$$

# Vergleich der oberen Schranken

n	7	8	9	10	11	12	13
$A(n, 7)$	2	2	2	2	4	4	8
Singleton	2	4	8	16	32	64	128
Plotkin	2	2	2	3	4	7	14

## Kodierungstheorem von Shannon für fehlerbehaftete Kanäle

Gegeben sei ein binärer symmetrischer Kanal  $Q$  mit Fehlerws  $p$ . Für alle  $R < 1 + p \log_2 p + (1 - p) \log_2(1 - p) = 1 - H(Q)$  und alle  $\epsilon > 0$  gibt es für hinreichend große  $n$  einen  $(n, M)$ -Code  $C$  mit Übertragungsrate  $\mathcal{R}(C) \geq R$  und  $W_s(\text{Dekodierfehler}) \leq \epsilon$ .

- Beweis komplex, nicht-konstruktiv.
- Resultat gilt nur asymptotisch für genügend große Blocklänge.

# Erinnerung: Der Vektorraum $\mathbb{F}_2^n$

## Definition Vektorraum $\mathbb{F}_2^n$

$\mathbb{F}_2^n = (\{0, 1\}^n, +, \cdot)$  mit Addition modulo 2,  $+ : \mathbb{F}_2^n \times \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$  und skalarer Multiplikation  $\cdot : \mathbb{F}_2 \times \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$  definiert einen Vektorraum, d.h.

- 1 Assoziativität:  $\mathbf{x} + (\mathbf{y} + \mathbf{z}) = (\mathbf{x} + \mathbf{y}) + \mathbf{z}$
- 2 Kommutativität:  $\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}$
- 3  $\exists$  neutrales Element  $\mathbf{0}^n : \mathbf{0}^n + \mathbf{x} = \mathbf{x} + \mathbf{0}^n = \mathbf{x}$
- 4 Selbstinverse:  $\forall \mathbf{x} : \mathbf{x} = -\mathbf{x}$ , d.h.  $\mathbf{x} + \mathbf{x} = \mathbf{0}^n$ .
- 5 Skalare Multiplikation:  $\alpha(\mathbf{x} + \mathbf{y}) = \alpha\mathbf{x} + \alpha\mathbf{y}$ .

## Definition Unterraum des $\mathbb{F}_2^n$

$S \subseteq \mathbb{F}_2^n$  ist ein Unterraum des  $\mathbb{F}_2^n$  gdw

$$\mathbf{0}^n \in S \text{ und } \forall \mathbf{x}, \mathbf{y} \in S : \mathbf{x} - \mathbf{y} \in S.$$

**Bsp:** Code  $C = \{000, 100, 010, 110\}$  ist Unterraum des  $\mathbb{F}_2^3$ .

# Erzeugendensystem und Basis

## Definition Erzeugendensystem und Basis eines Unterraums

Sei  $S \subseteq \mathbb{F}_2^n$  ein Unterraum. Eine Menge  $G = \{\mathbf{g}_1, \dots, \mathbf{g}_k\} \subseteq S$  heißt *Erzeugendensystem* von  $S$ , falls jedes  $\mathbf{x} \in S$  als Linearkombination

$$\mathbf{x} = \alpha_1 \mathbf{g}_1 + \dots + \alpha_k \mathbf{g}_k \quad \text{mit } \alpha_i \in \mathbb{F}_2$$

geschrieben werden kann. Notation:  $S = \langle \mathbf{g}_1, \dots, \mathbf{g}_k \rangle$ .

Eine *Basis*  $B$  ist ein minimales Erzeugendensystem, d.h. keine echte Teilmenge von  $B$  erzeugt  $S$ .

### Bsp:

- $C = \{000, 100, 010, 110\}$  wird von  $G = \{000, 100, 010\}$  erzeugt.
- $B = \{100, 010\}$  ist eine Basis von  $C$ .
- $B' = \{100, 110\}$  ist ebenfalls eine Basis.

## Erinnerung Eigenschaften einer Basis

Sei  $S \subseteq \mathbb{F}_2^n$  ein Unterraum.

- 1 Jede linear unabhängige Teilmenge von  $S$  kann zu einer Basis ergänzt werden.
- 2 Jede Basis von  $S$  besitzt dieselbe Kardinalität, genannt die Dimension  $\dim(S)$ .
- 3 Jedes Erzeugendensystem  $G$  von  $S$  enthält eine Untermenge, die eine Basis von  $S$  ist.

## Definition Linearer Code

Sei  $C \subseteq \mathbb{F}_2^n$  ein Code. Wir bezeichnen  $C$  als *linearen Code*, falls  $C$  ein Unterraum ist. Sei  $k$  die Dimension des Unterraums und  $d$  die Distanz von  $C$ , dann bezeichnen wir  $C$  als  $[n, k, d]$ -Code.

## Bsp:

- $C = \{000, 100, 010, 110\}$  ist ein  $[3, 2, 1]$ -Code.
- $C = \langle 1011, 1110, 0101 \rangle$  ist ein  $[4, 2, 2]$ -Code.
- Jeder  $[n, k, d]$ -Code ist ein  $(n, 2^k, d)$ -Code.
- D.h. wir können  $M = 2^k$  Codeworte mittels einer Basis der Dimension  $k$  kompakt darstellen.
- Beispiele für lineare Codes:  
Hamming Codes, Golay Codes und Reed-Muller Codes.

# Generatormatrix eines linearen Codes

## Definition Generatormatrix

Sei  $C$  ein linearer  $[n, k, d]$ -Code mit Basis  $B = \{\mathbf{b}_1, \dots, \mathbf{b}_k\}$ . Die  $(k \times n)$ -Matrix

$$G = \begin{pmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_k \end{pmatrix} \in \mathbb{F}_2^{k \times n}$$

heißt Generatormatrix des Codes  $C$ .

## Definition Hamminggewicht

Sei  $\mathbf{c} \in \{0, 1\}^n$ . Das *Hamminggewicht* von  $\mathbf{c}$  ist definiert als

$$w(\mathbf{c}) = d(\mathbf{c}, \mathbf{0}).$$

D.h.  $w(\mathbf{c})$  ist die Anzahl der Einsen in  $\mathbf{c}$ .

# Distanz von linearen Codes

## Satz Distanz eines linearen Codes

Sei  $C$  ein linearer Code. Dann gilt

$$d(C) = \min_{\mathbf{c} \in C, \mathbf{c} \neq \mathbf{0}} \{w(\mathbf{c})\}.$$

“ $\leq$ ”:

- Sei  $\mathbf{c}_m = \min_{\mathbf{c} \in C, \mathbf{c} \neq \mathbf{0}} \{w(\mathbf{c})\}$ . Dann gilt

$$d(C) \leq d(\mathbf{c}_m, \mathbf{0}^n) = w(\mathbf{c}_m)$$

“ $\geq$ ”:

- Seien  $\mathbf{c}_i, \mathbf{c}_j$  Codeworte mit  $d(C) = d(\mathbf{c}_i, \mathbf{c}_j)$ .
- Aus der Linearität von  $C$  folgt  $\mathbf{c}_i + \mathbf{c}_j \in C$ . Daher gilt

$$d(C) = d(\mathbf{c}_i, \mathbf{c}_j) = d(\mathbf{c}_i + \mathbf{c}_j, \mathbf{0}) = w(\mathbf{c}_i + \mathbf{c}_j) \geq \min_{\mathbf{c} \in C, \mathbf{c} \neq \mathbf{0}} \{w(\mathbf{c})\}.$$

**Bsp:**  $G = \langle 110, 111 \rangle$  besitzt  $d(G) = w(001) = 1$ .



# Dekodierung mittels Standardarray

## Algorithmus Standardarray

Eingabe:  $C = \{\mathbf{c}_1, \dots, \mathbf{c}_M\}$  linearer  $[n, \log_2 M, d]$ -Code mit  $\mathbf{c}_1 = \mathbf{0}^n$ .

- 1  $S \leftarrow C$ . Schreibe  $C$  in erste Zeile einer Tabelle.
- 2 While  $S \neq \mathbb{F}_2^n$ 
  - 1 Wähle Fehlervektor  $\mathbf{f} \in \mathbb{F}_2^n \setminus S$  mit minimalem Gewicht.
  - 2 Schreibe  $\mathbf{c}_1 + \mathbf{f}, \dots, \mathbf{c}_m + \mathbf{f}$  in neue Tabellenzeile.
  - 3  $S \leftarrow S \cup \{\mathbf{c}_1 + \mathbf{f}, \dots, \mathbf{c}_m + \mathbf{f}\}$ .

**Beispiel:**  $C = \{0000, 1011, 0110, 1101\}$  besitzt Standardarray:

<b>0000</b>	<b>1011</b>	<b>0110</b>	<b>1101</b>
1000	0011	1110	0101
0100	1111	0010	1001
0001	1010	0111	1100

### Standardarray-Dekodierung:

Dekodieren  $\mathbf{x} \in \{0, 1\}^n$  zum Codewort in derselben Spalte.

# Korrektheit des Algorithmus

## Satz Dekodierung zum nächsten Nachbarn via Standardarray

Sei  $C$  ein linearer  $[n, k]$ -Code. Jeder String  $\mathbf{x}$  wird durch Standardarray-Dekodierung zu einem nächsten Nachbarn dekodiert.

- Sei  $\mathbf{c}_i$  die Standardarray-Dekodierung von  $\mathbf{x}$  mit  $\mathbf{x} = \mathbf{c}_i + \mathbf{f}_j$ . Es gilt

$$\begin{aligned} \min_{\mathbf{c} \in C} \{d(\mathbf{x}, \mathbf{c})\} &= \min_{\mathbf{c} \in C} \{w(\mathbf{x} - \mathbf{c})\} = \min_{\mathbf{c} \in C} \{w(\mathbf{f}_j + \mathbf{c}_i - \mathbf{c})\} \\ &= \min_{\mathbf{c} \in C} \{w(\mathbf{f}_j + \mathbf{c})\} \quad // \mathbf{c}_i - \mathbf{c} \text{ durchläuft alle Codeworte} \\ &\stackrel{2.1}{=} w(\mathbf{f}_j) = w(\mathbf{x} - \mathbf{c}_i) = d(\mathbf{x}, \mathbf{c}_i). \end{aligned}$$

## Satz Dekodierfehler perfekter linearer Codes

Sei  $C$  ein perfekter  $[n, k, d]$ -Code. Für einen binären symmetrischen Kanal mit Fehlerws  $p$  gilt für Standardarray-Dekodierung

$$W_s(\text{korrekte Dekodierung}) = \sum_{i=0}^{\lfloor \frac{d-1}{2} \rfloor} \binom{n}{i} p^i (1-p)^{n-i}. \quad (\text{Beweis: Übung})$$

# Inneres Produkt und Orthogonalität

## Fakt Eigenschaften des inneren Produkts

Seien  $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{F}_2^n$  und  $\alpha \in \mathbb{F}_2$ . Dann gilt für das innere Produkt  $\cdot : \mathbb{F}_2^n \times \mathbb{F}_2^n \rightarrow \mathbb{F}_2$  mit  $(x_1, \dots, x_n) \cdot (y_1, \dots, y_n) \mapsto x_1 y_1 + \dots + x_n y_n$

- 1 Kommutativität:  $\mathbf{x} \cdot \mathbf{y} = \mathbf{y} \cdot \mathbf{x}$
- 2 Distributivität:  $(\mathbf{x} + \mathbf{y}) \cdot \mathbf{z} = \mathbf{x} \cdot \mathbf{z} + \mathbf{y} \cdot \mathbf{z}$
- 3 Skalare Assoziativität:  $(\alpha \mathbf{x}) \cdot \mathbf{y} = \mathbf{x} \cdot (\alpha \mathbf{y}) = \alpha(\mathbf{x} \cdot \mathbf{y})$

## Definition Orthogonalität, orthogonales Komplement

Seien  $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^n$ . Wir bezeichnen  $\mathbf{x}, \mathbf{y}$  als orthogonal, falls  $\mathbf{x} \cdot \mathbf{y} = 0$ . Das *orthogonale Komplement*  $\{\mathbf{y}\}^\perp$  von  $\mathbf{y}$  ist definiert als die Menge

$$\{\mathbf{y}\}^\perp = \{\mathbf{x} \in \mathbb{F}_2^n \mid \mathbf{x} \cdot \mathbf{y} = 0\}.$$

# Lineare Codes mittels orthogonalem Komplement

## Satz Linearer Code $\{\mathbf{y}\}^\perp$

Sei  $\mathbf{y} \in \mathbb{F}_2^n$ . Dann ist  $\{\mathbf{y}\}^\perp$  ein linearer Code.

### Beweis:

- Zeigen, dass  $\{\mathbf{y}\}^\perp$  ein Unterraum des  $\mathbb{F}_2^n$  ist.
- Abgeschlossenheit: Seien  $\mathbf{x}, \mathbf{x}'$  im orthog. Komplement von  $\mathbf{y}$ .
- Dann ist auch  $\mathbf{x} - \mathbf{x}' \in \{\mathbf{y}\}^\perp$ , denn

$$(\mathbf{x} - \mathbf{x}') \cdot \mathbf{y} = \mathbf{x} \cdot \mathbf{y} - \mathbf{x}' \cdot \mathbf{y} = 0.$$

- $\mathbf{0} \in \{\mathbf{y}\}^\perp$ , denn  $\mathbf{0} \cdot \mathbf{y} = 0$ .

### Bsp:

- $\{\mathbf{1}\}^\perp = \{\mathbf{x} \in \mathbb{F}_2^n \mid x_1 + \dots + x_n = 0\} = \{\mathbf{x} \in \mathbb{F}_2^n \mid w(\mathbf{x}) \text{ gerade}\}$
- Wir nennen  $x_1 + \dots + x_n = 0$  die Parity Check Gleichung des Codes  $\{\mathbf{1}\}^\perp$ .

# Orthogonales Komplement erweitert auf Mengen

## Definition Orthogonales Komplement einer Menge

Sei  $C = \{\mathbf{c}_1, \dots, \mathbf{c}_M\} \subseteq \mathbb{F}_2^n$ . Das *orthogonale Komplement* von  $C$  ist definiert als

$$C^\perp = \{\mathbf{x} \in \mathbb{F}_2^n \mid c_i \cdot \mathbf{x} = 0 \text{ für } i = 1, \dots, M\}.$$

- Sei  $\mathbf{c}_i = c_{i1} c_{i2} \dots c_{in}$ . Für  $\mathbf{x} \in C^\perp$  gelten Parity Check Gleichungen

$$\begin{aligned} c_{11}x_1 + c_{12}x_2 + \dots + c_{1n}x_n &= 0 \\ &\vdots \\ c_{M1}x_1 + c_{M2}x_2 + \dots + c_{Mn}x_n &= 0 \end{aligned}$$

- Sei  $P = (c_{ij})_{1 \leq i \leq M, 1 \leq j \leq n} \in \mathbb{F}_2^{M \times n}$ , dann gilt  $P\mathbf{x}^t = \mathbf{0}^t$  bzw.

$$\mathbf{x}P^t = \mathbf{0}.$$

- Wir bezeichnen  $P$  als Parity Check Matrix von  $C^\perp$ .

# Dualer Code

## Satz Dualer Code

Sei  $C = \{\mathbf{c}_1, \dots, \mathbf{c}_M\} \subseteq \mathbb{F}_2^n$  ein Code. Das orthogonale Komplement  $C^\perp$  von  $C$  ist ein linearer Code, genannt der duale Code von  $C$ .

### Beweis:

- Abgeschlossenheit: Seien  $\mathbf{x}, \mathbf{x}' \in C^\perp$  und  $P = (c_{ij})_{1 \leq i \leq M, 1 \leq j \leq n}$ . Dann gilt

$$(\mathbf{x} - \mathbf{x}')P^t = \mathbf{x}P^t - \mathbf{x}'P^t = \mathbf{0}.$$

- $0^n \in C^\perp$ , denn  $0^n P^t = 0^M$ .

### Bsp

- Sei  $C^\perp = \{100, 111\}^\perp$ . Dann gelten die Parity Check Gleichungen

$$\begin{aligned}x_1 &= 0 \\x_1 + x_2 + x_3 &= 0.\end{aligned}$$

- Aus der 2. Gleichung folgt  $x_2 = x_3$  in  $\mathbb{F}_2$ , d.h.  $C^\perp = \{000, 011\}$ .

# Parity Check Matrix

## Definition Parity Check Matrix $P$

Sei  $C$  ein linearer  $[n, k]$ -Code. Jede Matrix  $P$  mit der Eigenschaft

$$C = \{\mathbf{x} \in \mathbb{F}_2^n \mid \mathbf{x}P^t = \mathbf{0}\}$$

heißt *Parity Check Matrix des Codes  $C$* .

- D.h.  $C$  wird sowohl durch eine Generatormatrix als auch durch eine Parity Check Matrix eindeutig definiert.
- Im Gegensatz zu Generatormatrizen setzen wir nicht voraus, dass die Zeilen von  $P$  linear unabhängig sind.
- **Bsp.:** Code  $C = \{011, 101\}^\perp$  besitzt die Parity Check Matrizen

$$P = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \text{ und } P' = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}.$$

# Eigenschaften dualer Codes

## Satz Eigenschaften dualer Codes

Seien  $C, D$  Codes mit  $C \subseteq D$ . Dann gilt  $D^\perp \subseteq C^\perp$ .

### Beweis:

- Sei  $\mathbf{x} \in D^\perp$ . Dann gilt  $\mathbf{x} \cdot \mathbf{d} = 0$  für alle  $\mathbf{d} \in D$ .
- Somit ist  $\mathbf{x} \cdot \mathbf{c} = 0$  für alle  $\mathbf{c} \in C \subseteq D$ , d.h.  $\mathbf{x} \in C^\perp$ .

## Satz Eigenschaften dualer Codes von linearen Codes

Sei  $C$  ein linearer  $[n, k, d]$ -Code mit Generatormatrix  $G$ . Dann gilt

- 1  $C^\perp = \{\mathbf{x} \in \mathbb{F}_2^n \mid \mathbf{x}G^t = \mathbf{0}\}$ , d.h.  $G$  ist Parity Check Matrix für  $C^\perp$ .
- 2  $\dim(C^\perp) = n - \dim(C)$ .
- 3  $C^{\perp\perp} = C$ .



## Beweis der Eigenschaften 1+2

- 1  $G$  besitze Zeilenvektoren  $\mathbf{g}_1, \dots, \mathbf{g}_k$ . Zeigen  $C^\perp = \{\mathbf{g}_1 \dots, \mathbf{g}_k\}^\perp$ .
- ▶ Mit vorigem Satz folgt:  $\{\mathbf{g}_1, \dots, \mathbf{g}_k\} \subseteq C \Rightarrow C^\perp \subseteq \{\mathbf{g}_1, \dots, \mathbf{g}_k\}^\perp$ .
  - ▶  $\{\mathbf{g}_1, \dots, \mathbf{g}_k\}^\perp \subseteq C^\perp$ : Sei  $\mathbf{x} \in \{\mathbf{g}_1, \dots, \mathbf{g}_k\}^\perp$ . Dann ist  $\mathbf{x}$  orthogonal zu jeder Linearkombination der  $\mathbf{g}_i$ , d.h.  $\mathbf{x}$  ist orthog. zu jedem  $\mathbf{c} \in C$ .
- 2 Mit 1. gelten die folgenden Parity Check Gleichungen für  $C^\perp$

$$g_{11}x_1 + g_{12}x_2 + \dots + g_{1n}x_n = 0$$

$$\vdots$$

$$g_{k1}x_1 + g_{k2}x_2 + \dots + g_{kn}x_n = 0$$

Umwandeln in linke Standardform liefert (eventuell nach Spaltenumbenennung)

$$x_1 + a_{1,k+1}x_{k+1} + \dots + a_{1,n}x_n = 0$$

$$\ddots$$
$$\vdots$$

$$x_k + a_{k,k+1}x_{k+1} + \dots + a_{k,n}x_n = 0$$

Variablen  $x_{k+1}, \dots, x_n$  frei wählbar. Daher gilt  $\dim(C^\perp) = n - k$ .

- 3 Zeigen  $C \subseteq C^{\perp\perp}$  und  $\dim(C) = \dim(C^{\perp\perp})$ . Damit gilt  $C = C^{\perp\perp}$ .

## Beweis $C = C^{\perp\perp}$

- Zeigen zunächst  $C \subseteq C^{\perp\perp}$ . Sei  $\mathbf{c} \in C$ .
- Es gilt  $C^\perp = \{\mathbf{x} \in \mathbb{F}_2^n \mid \mathbf{x} \cdot \mathbf{c}_i = \mathbf{0} \text{ für alle } \mathbf{c}_i \in C\}$ .
- Ferner  $C^{\perp\perp} = \{\mathbf{y} \in \mathbb{F}_2^n \mid \mathbf{y} \cdot \mathbf{x} = \mathbf{0} \text{ für alle } \mathbf{x} \in C^\perp\}$ , d.h.  $\mathbf{c} \in C^{\perp\perp}$ .
- Wegen 2. gilt:  
$$\dim(C^{\perp\perp}) = n - \dim(C^\perp) = n - (n - \dim(C)) = \dim(C).$$

### Korollar Existenz einer Parity Check Matrix

Sei  $C$  ein linearer Code. Jede Generatormatrix  $G$  von  $C^\perp$  ist eine Parity Check Matrix für  $C$ . D.h. insbesondere, dass jeder lineare Code  $C$  eine Parity Check Matrix besitzt.

### Beweis:

- $C^\perp$  ist linear, besitzt also eine Generatormatrix  $G$ .
- $G$  ist Parity Check Matrix für den Dualcode von  $C^\perp$ , d.h. für  $C^{\perp\perp} = C$ .

# Konstruktion eines dualen Codes

**Bsp:**  $C = \langle 1011, 0110 \rangle$ .

- Die Parity Check Gleichungen von  $C^\perp$  sind

$$\begin{array}{rcccc} x_1 & & +x_3 & +x_4 & = & 0 \\ & x_2 & +x_3 & & = & 0 \end{array}$$

- Wählen beliebige Werte für  $x_3, x_4$  und lösen nach  $x_1, x_2$  auf.
- $C^\perp = \{0000, 1001, 1110, 0111\} = \langle 1001, 1110 \rangle$
- $\dim(C^\perp) = 4 - \dim(C) = 2$

**Bsp:**  $C = \langle 1100, 0011 \rangle$

- Die Codeworte 1100 und 0011 sind orthogonal zueinander.
- Beide Codeworte 1100, 0011 sind orthogonal zu sich selbst.
- D.h.  $C \subseteq C^\perp$  und  $\dim(C) = 2 = \dim(C^\perp)$ .
- Damit ist  $C^\perp = C$ .  $C$  ist ein *selbst-dualer Code*.

# Präsentation eines Codes durch $G$ oder $P$

Vorteil der Präsentation durch Generatormatrix:

- Einfache Generierung aller Codeworte von  $C$

Vorteil der Präsentation durch Parity Check Matrix:

- Entscheidung, ob ein  $\mathbf{x}$  im Code  $C$  liegt.

## Satz Minimaldistanz via $P$

Sei  $C$  ein linearer  $[n, k, d]$ -Code mit Parity Check Matrix  $P$ . Für die Minimaldistanz von  $C$  gilt

$$d = \min\{r \in \mathbb{N} \mid \text{Es gibt } r \text{ linear abhängige Spalten in } P\}.$$

# Beweis zur Minimaldistanz via Spalten von $P$

## Beweis:

- Sei  $r$  die minimale Anzahl von linear abhängigen Spalten.
- Es gibt ein  $\mathbf{c} \in \mathbb{F}_2^n$  mit  $w(\mathbf{c}) = r$  und  $P \cdot \mathbf{c}^t = \mathbf{0}^t \Leftrightarrow \mathbf{c}P^t = \mathbf{0}$ .
- Damit gilt  $\mathbf{c} \in C$  und  $d \leq r$ .
  
- Annahme:  $d < r$ .
- Sei  $\mathbf{c}' \in C$  ein Codewort mit Gewicht  $d$ . Dann gilt  $P \cdot (\mathbf{c}')^t = \mathbf{0}^t$ .
- D.h. es gibt  $d < r$  linear abhängige Spalten in  $P$ .  
(Widerspruch zur Minimalität von  $r$ )

# Syndrome

## Definition Syndrom

Sei  $C \subseteq \mathbb{F}_2^n$  ein Code mit Parity Check Matrix  $P$  und  $\mathbf{x} \in \mathbb{F}_2^n$ . Das Syndrom von  $\mathbf{x}$  ist definiert als  $S(\mathbf{x}) = \mathbf{x}P^t$ .

## Satz Standardarrays und Syndrome

Sei  $C$  ein linearer Code mit Standardarray  $A$  und Parity Check Matrix  $P$ . Die Elemente  $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^n$  sind in derselben Zeile von  $A$  gdw  $S(\mathbf{x}) = S(\mathbf{y})$ .

### Beweis:

- Sei  $\mathbf{x} = \mathbf{f}_i + \mathbf{c}_j$  und  $\mathbf{y} = \mathbf{f}_k + \mathbf{c}_\ell$ .
- Es gilt  $S(\mathbf{x}) = S(\mathbf{f}_i + \mathbf{c}_j) = S(\mathbf{f}_i) + S(\mathbf{c}_j) = S(\mathbf{f}_i)$ .
- Analog folgt  $S(\mathbf{y}) = S(\mathbf{f}_k)$ . D.h.

$$\begin{aligned} S(\mathbf{y}) = S(\mathbf{x}) &\Leftrightarrow S(\mathbf{f}_i) = S(\mathbf{f}_k) \\ &\Leftrightarrow S(\mathbf{f}_i - \mathbf{f}_k) = \mathbf{0} \Leftrightarrow \mathbf{f}_i - \mathbf{f}_k \in C \Leftrightarrow i = k. \end{aligned}$$

# Syndromdekodierung mittels Syndromtabelle

- Dekodierung mittels Standardarray:  $\mathbf{x} = \mathbf{f}_i + \mathbf{c}_j$  mit Fehlervektor  $\mathbf{f}_i$ .
- Paarweise verschiedene Fehlervektoren bilden die erste Spalte eines Standardarrays.
- Berechne die folgende Syndromtabelle für  $C$

Fehlervektor	Syndrom
$\mathbf{0}$	$\mathbf{0}$
$\mathbf{f}_2$	$S(\mathbf{f}_2)$
$\mathbf{f}_3$	$S(\mathbf{f}_3)$
$\vdots$	$\vdots$
$\mathbf{f}_\ell$	$S(\mathbf{f}_\ell)$

## Algorithmus Syndromdekodierung

EINGABE:  $\mathbf{x} \in \mathbb{F}_2^n$

- 1 Berechne  $S(\mathbf{x})$  und vergleiche mit der Syndromspalte.
- 2 Falls  $S(\mathbf{x}) = S(\mathbf{f}_i)$ , Ausgabe  $\mathbf{c} = \mathbf{x} - \mathbf{f}_i$ .

# Äquivalente lineare Codes

## Definition Äquivalenz von linearen Codes

Sei  $C$  ein linearer Code mit Generatormatrix  $G$ . Ein Code  $C'$  mit Generatormatrix  $G'$  heißt zu  $C$  *äquivalenter Code*, falls  $G'$  eine Transformation aus  $G$  mittels folgender Operationen ist.

- 1 Vertauschen von zwei Zeilenvektoren
- 2 Vertauschen von zwei Spaltenvektoren
- 3 Addition eines Zeilenvektors zu einem anderen Zeilenvektor

## Fakt Systematische Codes

Sei  $C$  ein linearer  $[n, k]$ -Code mit Generatormatrix  $G$ . Dann gibt es einen zu  $C$  äquivalenten Code  $C'$  mit Generatormatrix in linker Standardform  $G' = [I_k | M_{k, n-k}]$ .  $C'$  nennt man *systematischen Code*.

- Für systematische  $C'$ :  $(x_1, \dots, x_k)G' = (x_1, \dots, x_k, y_1, \dots, y_{n-k})$ .
- $y_1, \dots, y_{n-k}$  nennt man die Redundanz der Nachricht.



# Umwandlung Generatormatrix in Parity Check Matrix

## Satz Konversion von Generatormatrix in Parity Check Matrix

Sei  $C$  ein linearer  $[n, k]$ -Code mit Generatormatrix  $G = [I_k | A]$ . Dann ist

$$P = [A^t | I_{n-k}]$$

eine Parity Check Matrix für  $C$ .

**Beweis:** Sei  $C'$  der Code mit Parity Check Matrix  $P$ :

1 Zeigen:  $C \subseteq C'$ .

▶ Für alle Zeilen  $\mathbf{g}_i$  von  $G$  gilt  $P\mathbf{g}_i^t = \mathbf{0}^t$ , denn  $j$ -ter Eintrag von  $P\mathbf{g}_i^t$ :

$$(a_{1j} \dots a_{kj} 0 \dots 1 \dots 0) \cdot (0 \dots 1 \dots 0 a_{i1} \dots a_{in-k}) = a_{ij} + a_{ij} = 0$$

▶ Aus  $P\mathbf{g}_i^t = \mathbf{0}^t$  folgt  $C \subseteq C'$ .

2 Zeigen:  $\dim(C) = \dim(C')$

▶  $P$  besitzt  $n - k$  linear unabhängige Zeilen.

▶ D.h. Dualcode  $(C')^\perp$  hat Generatormatrix  $P$  und Dimension  $n - k$ .

$$\dim(C') = n - \dim((C')^\perp) = n - (n - k) = k = \dim(C).$$

# Hamming-Matrix $H(h)$ und Hammingcode $\mathcal{H}(h)$

- Parametrisiert über die Zeilenanzahl  $h$ .
- Spaltenvektoren sind Binärdarstellung von  $1, 2, \dots, 2^h - 1$ .
- Bsp :

$$H(3) = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

- Hammingcode  $\mathcal{H}(h)$  besitzt die Parity Check Matrix  $H(h)$ .
- Hammingcodes unabhängig entdeckt von Golay (1949) und Hamming (1950).

## Satz Hammingcode

Der Hammingcode  $\mathcal{H}(h)$  mit Parity Check Matrix  $H(h)$  ist ein linearer  $[n, k, d]$ -Code mit den Parametern

$$n = 2^h - 1, k = n - h \text{ und } d = 3.$$

# $k$ und $d$ bei Hammingcodes

## Beweis:

- $H(h)$  enthält die  $h$  Einheits-Spaltenvektoren  $\mathbf{e}_1, \dots, \mathbf{e}_h$ .
  - Daraus folgt, die Zeilenvektoren von  $H(h)$  sind linear unabhängig.
  - D.h.  $H(h)$  ist eine Generatormatrix des dualen Codes  $\mathcal{H}(h)^\perp$ .
  - Damit ist  $\dim(\mathcal{H}(h)^\perp) = h$  und  $k = \dim(\mathcal{H}(h)) = n - h$ .
- 
- Je zwei Spalten in  $H(h)$  sind paarweise verschieden.
  - Die minimale Anzahl von linear abhängigen Spalten ist mindestens 3, d.h.  $d(\mathcal{H}(h)) \geq 3$ .
  - Die ersten drei Spalten sind stets linear abhängig, d.h.  $d(\mathcal{H}(h)) = 3$ .

# Dekodierung mit Hammingcodes

## Satz Korrigieren eines Fehlers

Sei  $\mathbf{c} \in \mathcal{H}(h)$  und  $\mathbf{x} = \mathbf{c} + \mathbf{e}_i$  für einen Einheitsvektor  $\mathbf{e}_i \in \mathbb{F}_2^{2^h-1}$ . Dann entspricht das Syndrom  $S(\mathbf{x})$  der Binärdarstellung von  $i$ .

### Beweis:

- Es gilt  $S(\mathbf{x}) = S(\mathbf{e}_i) = \mathbf{e}_i H(h)^t = (H(h)\mathbf{e}_i)^t$ .
- D.h.  $S(\mathbf{x})$  entspricht der  $i$ -ten Spalte von  $H(h)$ , die wiederum die Binärkodierung von  $i$  ist.

### Bsp:

- Verwenden  $\mathcal{H}(3)$  und erhalten  $\mathbf{x} = 1000001$ .

$$S(\mathbf{x}) = (1000001)H(3)^t = (110).$$

- Da 110 die Binärkodierung von 6 ist, kodieren wir zum nächsten Nachbarn 1000011.

# Simplex Code: Dualcode des Hammingcodes

## Satz Simplex Code

Der Dualcode des Hammingcodes  $\mathcal{H}(h)$  wird als Simplex Code  $\mathcal{S}(h)$  bezeichnet.  $\mathcal{S}(h)$  ist ein  $[2^h - 1, h, 2^{h-1}]$ -Code, bei dem für *alle* verschiedenen  $\mathbf{c}, \mathbf{c}' \in \mathcal{S}(h)$  gilt, dass  $d(\mathbf{c}, \mathbf{c}') = 2^{h-1}$ .

### Beweis:

- Hamming-Matrix  $H(h)$  ist Generatormatrix von  $\mathcal{S}(h) = \mathcal{H}(h)^\perp$ .
- Da  $\dim(\mathcal{S}(h)) = n - \dim(\mathcal{H}(h))$ , ist  $\mathcal{S}(h)$  ein  $[2^h - 1, h]$ -Code.
- Rekursive Definition: Es gilt

$$H(h+1) = \left( \begin{array}{ccc|c|ccc} 0 & \dots & 0 & 1 & 1 & \dots & 1 \\ \hline & & & 0 & & & \\ & & H(h) & \vdots & & H(h) & \\ & & & 0 & & & \end{array} \right).$$

- Sei  $\bar{\mathbf{c}}$  das Komplement von  $\mathbf{c}$  ist. Dann gilt

$$\mathcal{S}(h+1) = \{\mathbf{c}0\mathbf{c} \mid \mathbf{c} \in \mathcal{S}(h)\} \cup \{\mathbf{c}1\bar{\mathbf{c}} \mid \mathbf{c} \in \mathcal{S}(h)\}.$$

# Distanz $2^{h-1}$ zwischen zwei Worten im Simplex Code

Beweis von  $d(\mathbf{c}, \mathbf{c}') = 2^{h-1}$  per Induktion über  $h$ .

**IV**  $h = 1$ :

- $H(1) = (1)$ , d.h.  $\mathcal{S} = \{0, 1\}$  und damit  $d(0, 1) = 1 = 2^0$ .

**IS**  $h \rightarrow h + 1$ :

- Fall 1:  $d(\mathbf{c}0\mathbf{c}, \mathbf{c}'0\mathbf{c}') = 2 \cdot d(\mathbf{c}, \mathbf{c}') = 2 \cdot 2^{h-1} = 2^h$ .
- Fall 2:  $d(\mathbf{c}1\bar{\mathbf{c}}, \mathbf{c}'1\bar{\mathbf{c}}') = d(\mathbf{c}, \mathbf{c}') + d(\bar{\mathbf{c}}, \bar{\mathbf{c}}') = 2 \cdot d(\mathbf{c}, \mathbf{c}') = 2^h$ .
- Fall 3:

$$\begin{aligned}d(\mathbf{c}0\mathbf{c}, \mathbf{c}'1\bar{\mathbf{c}}') &= d(\mathbf{c}, \mathbf{c}') + 1 + d(\mathbf{c}, \bar{\mathbf{c}}') \\ &= d(\mathbf{c}, \mathbf{c}') + 1 + (2^h - 1 - d(\mathbf{c}, \mathbf{c}')) = 2^h.\end{aligned}$$

# Reed-Muller Codes

- Reed-Muller Code  $\mathcal{R}(r, m)$  ist definiert für  $m \in \mathbb{N}$ ,  $0 \leq r \leq m$ .
- Betrachten nur Reed-Muller Codes 1. Ordnung  $\mathcal{R}(1, m) = \mathcal{R}(m)$ .

## Definition Rekursive Darstellung von Reed-Muller Codes

- 1  $\mathcal{R}(1) = \mathbb{F}_2^2 = \{00, 01, 10, 11\}$ .
- 2 Für  $m \geq 1$ :  $\mathcal{R}(m+1) = \{\mathbf{c}\mathbf{c} \mid \mathbf{c} \in \mathcal{R}(m)\} \cup \{\mathbf{c}\bar{\mathbf{c}} \mid \mathbf{c} \in \mathcal{R}(m)\}$ .

- $R_1 = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$  ist eine Generatormatrix für  $\mathcal{R}(1)$ .
- $\mathcal{R}(2) = \{0000, 0011, 0101, 0110, 1010, 1001, 1111, 1100\}$  mit Generatormatrix

$$R_2 = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

# Parameter der Reed-Muller Codes

## Satz Reed-Muller Parameter

$\mathcal{R}(m)$  ist ein linearer  $(2^m, 2^{m+1}, 2^{m-1})$ -Code. Für alle  $\mathbf{c} \in \mathcal{R}(m) \setminus \{\mathbf{0}, \mathbf{1}\}$  gilt  $w(\mathbf{c}) = 2^{m-1}$ .

**IA:**  $m = 1$

- $\mathcal{R}(1)$  ist ein linearer  $(2^1, 2^2, 2^0)$ -Code. 01, 10 besitzen Gewicht  $2^0$ .

**IS:**  $m \rightarrow m + 1$

- $n = 2 \cdot 2^m = 2^{m+1}$ .
- $\{\mathbf{c}\mathbf{c} \mid \mathbf{c} \in \mathcal{R}(m)\}$  und  $\{\mathbf{c}\bar{\mathbf{c}} \mid \mathbf{c} \in \mathcal{R}(m)\}$  sind disjunkt, d.h.  
 $k = 2 \cdot 2^{m+1} = 2^{m+2}$ .
- Sei  $\mathbf{c} \in \mathcal{R}(m) \setminus \{\mathbf{0}, \mathbf{1}\}$ .
  - ▶ Für  $\mathbf{c}\mathbf{c}$  gilt  $w(\mathbf{c}\mathbf{c}) = 2w(\mathbf{c}) = 2 \cdot 2^{m-1} = 2^m$ .
  - ▶ Für  $\mathbf{c}\bar{\mathbf{c}}$  gilt  $w(\mathbf{c}\bar{\mathbf{c}}) = w(\mathbf{c}) + w(\bar{\mathbf{c}}) = 2^{m-1} + (2^m - 2^{m-1}) = 2^m$ .
- Für  $\mathbf{c} = \mathbf{0}$  gilt  $\mathbf{c}\bar{\mathbf{c}} = \mathbf{0}\mathbf{1}$  mit  $w(\mathbf{0}\mathbf{1}) = 2^m$ .
- Für  $\mathbf{c} = \mathbf{1}$  gilt  $\mathbf{c}\bar{\mathbf{c}} = \mathbf{1}\mathbf{0}$  mit  $w(\mathbf{1}\mathbf{0}) = 2^m$ .



# Reed-Muller Generatormatrizen

## Satz Generatormatrix für $\mathcal{R}(m)$

Sei  $R_m$  eine Generatormatrix für  $\mathcal{R}(m)$ . Dann ist

$$R_{m+1} = \left( \begin{array}{ccc|ccc} 0 & \dots & 0 & 1 & \dots & 1 \\ \hline & & R_m & & & R_m \end{array} \right)$$

eine Generatormatrix für  $\mathcal{R}(m+1)$ .

## Beweis:

- **Ann.:**  $\exists$  nicht-triviale Linearkombination, die  $\mathbf{0}$  liefert.
- Linearkombination kann nicht nur die erste Zeile enthalten.
- D.h. es gibt eine nicht-triviale Linearkombination der Zeilen  $2 \dots m+2$ , die den Nullvektor auf der ersten Hälfte liefert. (Widerspruch:  $R_m$  ist Generatormatrix für  $\mathcal{R}(m)$ .)
- Sei  $C$  der Code mit Generatormatrix  $R_{m+1}$ .
- Für  $\mathbf{c} \in \mathcal{R}(m)$  gilt:  $\mathbf{c}\mathbf{c} \in C$  und  $\mathbf{c}\bar{\mathbf{c}} \in C$ . D.h.  $\mathcal{R}(m+1) \subseteq C$ .
- $\dim(C) = m+1 = \dim(\mathcal{R}(m+1))$  und damit  $C = \mathcal{R}(m+1)$ .

# Charakterisierung der Generatormatrizen

**Bsp:**

$$R_3 = \left( \begin{array}{cccc|cccc} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array} \right)$$

Streiche Einserzeile aus  $R_m$ . Dann

- besitzen die Spaltenvektoren Länge  $m$  und
- bestehen aus Binärkodierungen von  $0, 1, \dots, 2^m - 1$ .
- D.h. Streichung der Einserzeile von  $R_m$  liefert die Hamming-Matrix  $H(m)$  mit einer zusätzlichen Nullspalte.

## Vergleich von Hamming, Simplex und Reed-Muller Codes

	$\mathcal{H}(m)$	$\mathcal{S}(m)$	$\mathcal{R}(m)$
Codewortlänge	$2^m - 1$	$2^m - 1$	$2^m$
Anzahl Codeworte	$2^{2^m - 1 - m}$	$2^m$	$2^{m+1}$
Distanz	3	$2^{m-1}$	$2^{m-1}$

# Dekodierung von Reed-Muller Codes

- $\mathcal{R}(m)$  kann  $\left\lfloor \frac{2^{m-1}-1}{2} \right\rfloor = 2^{m-2} - 1$  Fehler korrigieren.
- Syndrom-Tabelle besitzt  $\frac{2^n}{M} = \frac{2^{2^m}}{2^{m+1}} = 2^{2^m-m-1}$  Zeilen.

**Bsp:**  $\mathcal{R}(3)$  ist 1-fehlerkorrigierend.

$$R_3 = \begin{pmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \mathbf{r}_3 \\ \mathbf{r}_4 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Sei  $\mathbf{c} = \alpha_1 \mathbf{r}_1 + \alpha_2 \mathbf{r}_2 + \alpha_3 \mathbf{r}_3 + \alpha_4 \mathbf{r}_4$ . Es gilt

- $c_1 + c_5 = \alpha_1(r_{11} + r_{15}) + \alpha_2(r_{21} + r_{25}) + \alpha_3(r_{31} + r_{35}) + \alpha_4(r_{41} + r_{45}) = \alpha_1$
- $c_2 + c_6 = \alpha_1(r_{12} + r_{16}) + \alpha_2(r_{22} + r_{26}) + \alpha_3(r_{32} + r_{36}) + \alpha_4(r_{42} + r_{46}) = \alpha_1$
- Ebenso  $\alpha_1 = c_3 + c_7 = c_4 + c_8$ .

# Mehrheitsdekodierung

- Suche für jede Zeile  $i$  Spaltenpaar  $(u, v)$ , so dass sich die Spalten  $u, v$  nur in der  $i$ -ten Zeile unterscheiden. Liefert Gleichung für  $\alpha_j$ .
  - Für Zeile 1:  $(1, 5), (2, 6), (3, 7), (4, 8)$ , d.h. im Abstand 4.
  - Für Zeile 2:  $(1, 3), (2, 4), (5, 7), (6, 8)$ , d.h. im Abstand 2.
  - Für Zeile 3:  $(1, 2), (3, 4), (5, 6), (7, 8)$ , d.h. im Abstand 1.
  - Für Zeile 4: nicht möglich.
- 
- Erhalten für  $\alpha_1, \alpha_2, \alpha_3$  jeweils 4 Gleichungen in verschiedenen  $c_j$ .
  - Falls  $\mathbf{x} = \mathbf{c} + \mathbf{e}_i$ , ist genau 1 von 4 Gleichungen inkorrekt.

## Algorithmus Mehrheitsdekodierung Reed-Muller Code $\mathcal{R}(m)$

- 1 Bestimme  $\alpha_1, \dots, \alpha_m$  per Mehrheitsentscheid.
- 2 Berechne  $\mathbf{e} = \mathbf{x} - \sum_{j=1}^m \alpha_j \mathbf{r}_j$ .
- 3 Falls  $w(\mathbf{e}) \leq 2^{m-2} - 1$ , dekodiere  $\mathbf{c} = \mathbf{x} + \mathbf{e}$ . (d.h.  $\alpha_{m+1} = 0$ )
- 4 Falls  $w(\bar{\mathbf{e}}) \leq 2^{m-2} - 1$ , dekodiere  $\mathbf{c} = \mathbf{x} + \bar{\mathbf{e}}$ . (d.h.  $\alpha_{m+1} = 1$ )

# Beispiel Mehrheitsdekodierung

## Bsp:

- Verwenden  $\mathcal{R}(3)$  und erhalten  $\mathbf{x} = 11011100$ .
  - ▶  $\alpha_1 = x_1 + x_5 = 0$
  - ▶  $\alpha_1 = x_2 + x_6 = 0$
  - ▶  $\alpha_1 = x_3 + x_7 = 0$
  - ▶  $\alpha_1 = x_4 + x_8 = 1$
- Mehrheitsentscheid liefert  $\alpha_1 = 0$ .
  - ▶  $\alpha_2 = x_1 + x_3 = 1$
  - ▶  $\alpha_2 = x_2 + x_4 = 0$
  - ▶  $\alpha_2 = x_5 + x_7 = 1$
  - ▶  $\alpha_2 = x_6 + x_8 = 1$
- Mehrheitsentscheid liefert  $\alpha_2 = 1$  und analog  $\alpha_3 = 0$ .
- $\mathbf{e} = \mathbf{x} - 0 \cdot \mathbf{r}_1 - 1 \cdot \mathbf{r}_2 - 0 \cdot \mathbf{r}_3 = 11011100 - 00110011 = 11101111$ .
- $w(\bar{\mathbf{e}}) \leq 1$ , d.h.  $\mathbf{c} = \mathbf{x} + \bar{\mathbf{e}} = 11001100$ .

# McEliece Verfahren (1978)

- **Dekodieren eines zufälligen linearen Codes ist NP-hart.**
- Verwende linearen Code  $C$  mit effizientem Dekodierverfahren (z.B. sogenannten Goppa-Code).
- Generatormatrix von  $C$  bildet den geheimen Schlüssel.
- $C$  wird in äquivalenten linearen Code  $C'$  transformiert.

## Algorithmus Schlüsselgenerierung McEliece

- 1 Wähle linearen  $[n, k, d]$ -Code  $C$  mit Generatormatrix  $G$ .
- 2 Wähle zufällige binäre  $(k \times k)$ -Matrix  $S$  mit  $\det(S) = 1$ .
- 3 Wähle zufällige binäre  $(n \times n)$ -Permutationsmatrix  $P$ .
- 4  $G' \leftarrow SGP$

öffentlicher Schlüssel:  $G'$ , geheimer Schlüssel  $S, G, P$ .

# McEliece Verschlüsselung

## Algorithmus McEliece Verschlüsselung

EINGABE: Plaintext  $\mathbf{m} \in \mathbb{F}_2^k$

- 1 Wähle zufälligen Fehlervektor  $\mathbf{e} \in \mathbb{F}_2^n$  mit  $w(\mathbf{e}) = \lfloor \frac{d-1}{2} \rfloor$ .
- 2  $\mathbf{c} \leftarrow \mathbf{m}G' + \mathbf{e}$ .

AUSGABE: Ciphertext  $\mathbf{c} \in \mathbb{F}_2^n$

Vorgeschlagene Parameter:

- [1024, 512, 101]-Goppacode  $C$ .
- Plaintextlänge: 512 Bit, Chiffretextlänge: 1024 Bit.
- Größe des öffentlichen Schlüssels:  $512 \times 1024$  Bit.

# McEliece Entschlüsselung

## Algorithmus McEliece Entschlüsselung

EINGABE: Ciphertext  $\mathbf{c} \in \mathbb{F}_2^n$

- 1  $\mathbf{x} \leftarrow \mathbf{c}P^{-1}$ .
- 2 Dekodiere  $\mathbf{x}$  mittels Dekodieralgorithmus für  $C$  zu  $\mathbf{m}'$ .
- 3  $\mathbf{m} \leftarrow \mathbf{m}'S^{-1}$

AUSGABE: Plaintext  $\mathbf{m} \in \mathbb{F}_2^k$

- **Korrektheit:**

$$\mathbf{x} = \mathbf{c}P^{-1} = (\mathbf{m}G' + \mathbf{e}) \cdot P^{-1} = (\mathbf{m}SGP + \mathbf{e}) \cdot P^{-1} = (\mathbf{m}S)G + \mathbf{e} \cdot P^{-1}.$$

- $\mathbf{e} \cdot P^{-1}$  besitzt Gewicht  $w(\mathbf{e}P^{-1}) = w(\mathbf{e}) = \lfloor \frac{d-1}{2} \rfloor$ .
- Dekodierung liefert  $\mathbf{m}' = \mathbf{m}S$ , d.h.  $\mathbf{m} = \mathbf{m}'S^{-1}$ .