

# Grundlagen der Theoretischen Informatik

**Sommersemester 2016**

21.04.2016

Viorica Sofronie-Stokkermans

e-mail: [sofronie@uni-koblenz.de](mailto:sofronie@uni-koblenz.de)

# Bis jetzt

---

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

# Bis jetzt

---

- **Alphabete, Wörter**
  - Operationen auf Wörtern  
Konkatenation,  $i$ -te Potenz, Reverse
- **Sprache**
  - Operationen auf Sprachen  
Konkatenation,  $i$ -te Potenz, Reverse, Kleene-Hülle
- **Reguläre Ausdrücke**
- **Grammatiken**
  - Ableitung
  - Die von einer Grammatik erzeugte Sprache.

# Grammatik

---

Menge von Regeln, mit deren Hilfe man Wörter ableiten kann

**Definition.** Eine **Grammatik**  $G$  über einem Alphabet  $\Sigma$  ist ein Tupel

$$G = (V, T, R, S)$$

Dabei ist

- $V$  eine endliche Menge von **Variablen**
- $T \subseteq \Sigma$  eine endliche Menge von **Terminalen** mit  $V \cap T = \emptyset$
- $R$  eine endliche Menge von **Regeln**
- $S \in V$  das **Startsymbol**

**Konvention (meistens)**

- **Variablen** als **Großbuchstaben**
- **Terminale** als **Kleinbuchstaben**

# Rechnung einer Grammatik

---

Eine **Regel** ist ein Element  $(P, Q) \in ((V \cup T)^* V (V \cup T)^*) \times (V \cup T)^*$ .

- $P$  und  $Q$  sind Wörter über  $(V \cup T)$
- $P$  muss mindestens eine Variable enthalten
- $Q$  ist beliebig

Schreibweise:  $P \rightarrow_G Q, P \rightarrow Q$

( $P$ : Prämisse,  $Q$ : Conclusio )

## Ableitung, Rechnung.

- $w \Longrightarrow_G w'$  („ $w$  geht über in  $w'$ “) falls

$$\exists u, v \in (V \cup T)^* \exists P \rightarrow Q \in R \quad (w = uPv \text{ und } w' = uQv)$$

- $w \Longrightarrow_G^* w'$  falls es Wörter  $w_0, \dots, w_n \in (V \cup T)^*$  ( $n \geq 0$ ) gibt mit
  - $w = w_0$
  - $w_n = w'$
  - $w_i \Longrightarrow_G w_{i+1}$  für  $0 \leq i < n$

# Erzeugte Sprache, Äquivalenz

---

**Definition** (Erzeugte Sprache)      Gegeben: Eine Grammatik  $G$

Die von  $G$  erzeugte Sprache  $L(G)$  ist die Menge aller **terminalen** Wörter, die durch  $G$  vom Startsymbol  $S$  aus erzeugt werden können:

$$L(G) := \{w \in T^* \mid S \xRightarrow{*}_G w\}$$

**Definition** (Äquivalenz)

Zwei Grammatiken  $G_1, G_2$  heißen **äquivalent** gdw

$$L(G_1) = L(G_2)$$

# Warum Sprachen?

---

## Darstellung von Problemen

**Fakt:** So ziemlich alle Probleme können als Probleme über Sprachen formuliert werden.

# Warum Sprachen?

---

## Darstellung von Problemen

**Fakt:** So ziemlich alle Probleme können als Probleme über Sprachen formuliert werden.

## Beispiel: Primzahlen

**Alphabet**  $\Sigma_{\text{num}} := \{|\}$

**Sprache**  $L_{\text{primes}} := \{ \underbrace{|| \dots |}_{p \text{ mal}} \mid p \text{ prim} \}$



# Darstellung von Problemen

---

## Eingabealphabet

$$\Sigma = \{0, 1, \dots, n - 1\}$$

erlaubt Darstellung einer Ganzzahl zur Basis  $n$

# Darstellung von Problemen

---

## Eingabealphabet

$$\Sigma = \{0, 1, \dots, n - 1\}$$

erlaubt Darstellung einer Ganzzahl zur Basis  $n$

## Beispiel:

**5 binär:** 101

**5 unär:** ||||| (oder auch 11111)

# Darstellung von Problemen

---

## Speicheraufwand

$n$ -äre Darstellung ( $n > 1$ ) einer Zahl  $k$  führt zu einer Speicherersparnis:

$\log_n k$  ( $n$ -är) statt  $k$  (unär)

Nur der Schritt von unär auf binär ist wesentlich, denn

$$\log_n k = \frac{1}{\log_2 n} \cdot \log_2 k = c \cdot \log_2 k$$

(von binär auf  $n$ -är nur lineare Einsparung)

# Darstellung des Erfüllbarkeitsproblems SAT

---

## Problem SAT

**Gegeben:** Eine aussagenlogische Formel  $w$

**Frage:** Gibt es eine Belegung der booleschen Variablen in  $w$ ,  
so dass  $w$  zu *true* auswertet?

# Darstellung des Erfüllbarkeitsproblems SAT

---

## Problem SAT

**Gegeben:** Eine aussagenlogische Formel  $w$

**Frage:** Gibt es eine Belegung der booleschen Variablen in  $w$ ,  
so dass  $w$  zu *true* auswertet?

Signatur für aussagenlogische Formeln

**Signatur:**  $\Sigma_{\text{sat}} := \{\wedge, \vee, \neg, (, ), x, 0, 1\}$

Dabei Darstellung von boolescher Variablen  $x_i$  als  $x$  gefolgt von  $i$  binär kodiert.

Dadurch Formel der Länge  $n$  um (unerheblichen) Faktor  $\log n$  länger.

# Darstellung des Erfüllbarkeitsproblems SAT

---

## Satisfiability

Sprache

$L_{\text{sat}} := \{w \in \Sigma_{\text{sat}}^* : w \text{ ist eine aussagenlogische Formel,}$   
und es gibt eine Belegung für die  $x_i$ ,  
so dass die Formel  $w$  zu *true* auswertet }

# Darstellung des Erreichbarkeitsproblems in Graphen

---

## Erreichbarkeitsproblem

**Gegeben:** Ein Graph mit Ecken  $v_1$  bis  $v_n$

**Frage:** Gibt es einen Weg von Ecke  $v_1$  zu Ecke  $v_n$ ?

## Signatur für Graphen

**Signatur:**  $\Sigma_{\text{graph}} := \{v, e, 0, 1, (, ), \#\}$

Darstellung von

**Ecke**  $v_i$  als  $v$  gefolgt  $i$  binär kodiert

**Kante**  $e_{i,j}$  als  $e(\text{string}_1\#\text{string}_2)$ , wobei

- $\text{string}_1$  die binäre Darstellung von  $i$ ,
- $\text{string}_2$  die binäre Darstellung von  $j$

# Darstellung des Erreichbarkeitsproblems in Graphen

---

## Erreichbarkeitsproblem

Sprache

$L_{\text{reach}} := \{w \in \Sigma_{\text{graph}}^* : \text{es gibt einen Weg in } w$   
von der ersten Ecke  $v_1$   
zur letzten Ecke  $v_n\}$



# Die Chomsky-Hierarchie

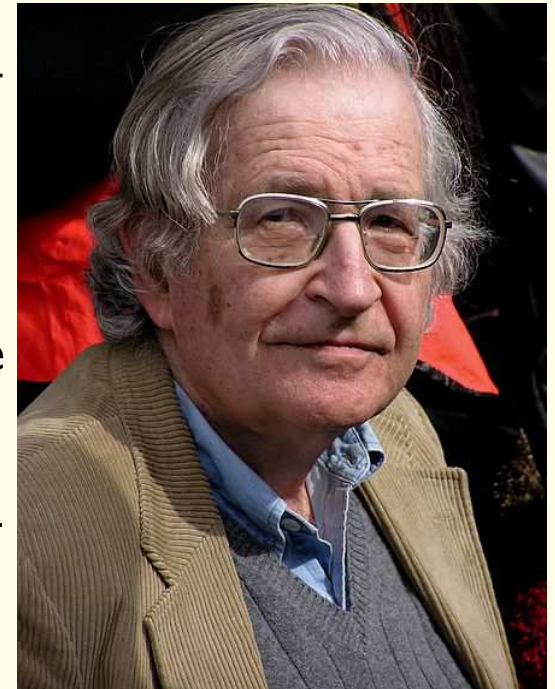
---

# Die Chomsky-Hierarchie

---

Noam Chomsky (born 1928)

- Professor für Linguistik und Philosophie am MIT
- Bedeutender Linguist
- Bedeutender Beitrag zur Informatik:  
Erste Beschreibung der Chomsky-Hierarchie  
(1956)
- Bedeutender linker Intellektueller und Globalisierungskritiker



# Die Chomsky-Hierarchie

---

## Was muss eine Grammatik erfüllen?

- Sie darf nur **endlich viele Regeln** haben
- Jede Regelprämisse muss **mindestens eine Variable** enthalten

# Die Chomsky-Hierarchie

---

## Was muss eine Grammatik erfüllen?

- Sie darf nur **endlich viele Regeln** haben
- Jede Regelprämisse muss **mindestens eine Variable** enthalten

Das Wort kann im Lauf der Ableitung beliebig wachsen und wieder schrumpfen.

(Weitere) Beschränkung der Form, die Regeln haben dürfen, führt zu

- **Grammatiktypen** und damit auch zu
- **Sprachtypen**

von verschiedenen Schwierigkeitsgraden.

# Die Chomsky-Hierarchie

---

**Definition** (Rechtlineare Grammatik)

Eine Grammatik  $G = (V, T, R, S)$  heißt **rechtslinear** gdw

$$\forall (P \rightarrow Q) \in R \quad (P \in V \text{ und } Q \in T^* \cup T^+V)$$

# Die Chomsky-Hierarchie

---

**Definition** (Rechtlineare Grammatik)

Eine Grammatik  $G = (V, T, R, S)$  heißt **rechtslinear** gdw

$$\forall (P \rightarrow Q) \in R \quad (P \in V \text{ und } Q \in T^* \cup T^+V)$$

Das heißt, bei jeder Regelanwendung:

- Links eine **einzelne Variable**
- Rechts **höchstens eine Variable**
- Wenn rechts eine Variable steht, steht sie **ganz rechts im Wort**.

# Die Chomsky-Hierarchie

---

**Definition** (Kontextfreie Grammatik)

Eine Grammatik  $G = (V, T, R, S)$  heißt **kontextfrei** gdw

$$\forall (P \rightarrow Q) \in R \quad (P \in V \text{ und } Q \in (V \cup T)^*)$$

# Die Chomsky-Hierarchie

---

**Definition** (Kontextfreie Grammatik)

Eine Grammatik  $G = (V, T, R, S)$  heißt **kontextfrei** gdw

$$\forall (P \rightarrow Q) \in R \quad (P \in V \text{ und } Q \in (V \cup T)^*)$$

Das heißt, bei jeder Regelanwendung:

- Links eine **einzelne Variable**
- Die Prämisse macht keine Aussage, was der Kontext dieser Variablen ist („kontextfrei“)
- Rechts steht etwas beliebiges



# Die Chomsky-Hierarchie

---

**Definition** (Kontextsensitive Grammatik)

Eine Grammatik  $G = (V, T, R, S)$  heißt **kontextsensitiv** gdw

$\forall (P \rightarrow Q) \in R:$

1.  $\exists u, v, \alpha \in (V \cup T)^* \exists A \in V (P = uAv \text{ und } Q = u\alpha v \text{ mit } |\alpha| \geq 1),$   
**oder** die Regel hat die Form  $S \rightarrow \varepsilon$
2.  $S$  nicht in  $Q$

# Die Chomsky-Hierarchie

---

**Definition** (Kontextsensitive Grammatik)

Eine Grammatik  $G = (V, T, R, S)$  heißt **kontextsensitiv** gdw

$\forall (P \rightarrow Q) \in R:$

1.  $\exists u, v, \alpha \in (V \cup T)^* \exists A \in V (P = uAv \text{ und } Q = u\alpha v \text{ mit } |\alpha| \geq 1)$ ,  
**oder** die Regel hat die Form  $S \rightarrow \varepsilon$
2.  $S$  nicht in  $Q$

Das heißt, bei jeder Regelanwendung:

- Eine Variable  $A$  wird in einen String  $\alpha$  mit  $|\alpha| \geq 1$  überführt
- Die Ersetzung von  $A$  durch  $\alpha$  findet nur statt, wenn der in der Regel geforderte **Kontext** ( $u$  und  $v$ ), **vorhanden ist**
- **Das Wort wird nicht kürzer, außer bei  $\varepsilon \in L$**

# Beschränkte Grammatik

---

**Definition** (Beschränkte Grammatik)

Eine Grammatik  $G = (V, T, R, S)$  heißt **beschränkt** gdw

$\forall (P \rightarrow Q) \in R:$

1.  $|P| \leq |Q|$ , **oder**  
die Regel hat die Form  $S \rightarrow \varepsilon$
2.  $S$  nicht in  $Q$

# Beschränkte Grammatik

---

**Definition** (Beschränkte Grammatik)

Eine Grammatik  $G = (V, T, R, S)$  heißt **beschränkt** gdw

$\forall (P \rightarrow Q) \in R:$

1.  $|P| \leq |Q|$ , **oder**  
die Regel hat die Form  $S \rightarrow \varepsilon$
2.  $S$  nicht in  $Q$

Das heißt, bei jeder Regelanwendung:

- Die Conclusio ist mindestens so lang wie die Prämisse, außer bei  $\varepsilon \in L$ .
- Das Wort wird nicht kürzer, außer bei  $\varepsilon \in L$

# Beispiele

Regel	rechtslinear	kontextfrei	kontextsensitiv	beschränkt
$S \rightarrow bA$	<input type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt	<input type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt	<input type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt	<input type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt
$S \rightarrow aSa$	<input type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt	<input type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt	<input type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt	<input type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt
$BC \rightarrow A$	<input type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt	<input type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt	<input type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt	<input type="checkbox"/> erlaubt <input checked="" type="checkbox"/> nicht erlaubt
$Ca \rightarrow CbC$	<input type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt	<input type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt	<input type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt	<input type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt
$B \rightarrow AB$	<input type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt	<input type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt	<input type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt	<input type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt

# Beispiele

Regel	rechtslinear	kontextfrei	kontextsensitiv	beschränkt
$S \rightarrow bA$	<input checked="" type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt	<input checked="" type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt	<input checked="" type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt	<input checked="" type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt
$S \rightarrow aSa$	<input type="checkbox"/> erlaubt <input checked="" type="checkbox"/> nicht erlaubt	<input checked="" type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt	<input type="checkbox"/> erlaubt <input checked="" type="checkbox"/> nicht erlaubt	<input type="checkbox"/> erlaubt <input checked="" type="checkbox"/> nicht erlaubt
$BC \rightarrow A$	<input type="checkbox"/> erlaubt <input checked="" type="checkbox"/> nicht erlaubt	<input type="checkbox"/> erlaubt <input checked="" type="checkbox"/> nicht erlaubt	<input type="checkbox"/> erlaubt <input checked="" type="checkbox"/> nicht erlaubt	<input type="checkbox"/> erlaubt <input checked="" type="checkbox"/> nicht erlaubt
$Ca \rightarrow CbC$	<input type="checkbox"/> erlaubt <input checked="" type="checkbox"/> nicht erlaubt	<input type="checkbox"/> erlaubt <input checked="" type="checkbox"/> nicht erlaubt	<input type="checkbox"/> erlaubt <input checked="" type="checkbox"/> nicht erlaubt	<input checked="" type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt
$B \rightarrow AB$	<input type="checkbox"/> erlaubt <input checked="" type="checkbox"/> nicht erlaubt	<input checked="" type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt	<input checked="" type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt	<input checked="" type="checkbox"/> erlaubt <input type="checkbox"/> nicht erlaubt

# Die Chomsky-Hierarchie

---

Aufbauend auf den Grammatikarten kann man Sprachklassen definieren.

**Definition** (Sprachklassen)

Klasse	definiert als	Sprache heißt
$L_3$ , REG	$\{L(G) \mid G \text{ ist rechtslinear}\}$	Typ 3, <b>regulär</b>
$L_2$ , CFL	$\{L(G) \mid G \text{ ist kontextfrei}\}$	Typ 2, <b>kontextfrei</b>
$L_1$ , CSL	$\{L(G) \mid G \text{ ist kontextsensitiv}\}$	Typ 1, <b>kontextsensitiv</b>
$L_1$ , CSL	$\{L(G) \mid G \text{ ist beschränkt}\}$	Typ 1, <b>beschränkt</b>
$L_0$ , r.e.	$\{L(G) \mid G \text{ beliebig}\}$	Typ 0, <b>aufzählbar</b>
$L$	$\{L \mid L \subseteq \Sigma^*\}$	<b>beliebige</b> Sprache

# Die Chomsky-Hierarchie

---

Grammatiken können kompliziert sein!

**Beispiel** (Grammatik für  $\{a^n b^n c^n \mid n \geq 1\}$ ):

Grammatik  $G_{abc} = (\{S, X_1, X_2\}, \{a, b, c\}, \{R_1, \dots, R_5\}, S)$  mit

$$R_1 = S \rightarrow abc \mid aX_1bc$$

$$R_2 = X_1b \rightarrow bX_1$$

$$R_3 = X_1c \rightarrow X_2bcc$$

$$R_4 = bX_2 \rightarrow X_2b$$

$$R_5 = aX_2 \rightarrow aa \mid aaX_1$$

- Ist diese Grammatik **kontextsensitiv**?
- Ist sie **beschränkt**?



# Probleme über Sprachen

---

# Probleme über Sprachen

---

## Interessante Probleme (informell)

- Ist ein gegebenes Wort in einer Sprache (definiert durch eine Grammatik) enthalten?
- Erzeugen zwei gegebene Grammatiken dieselbe Sprache?

Mit welchen Algorithmen können diese Probleme gelöst werden?

# Probleme und Algorithmen im allgemeine

---

## **Definition** (Problem, Algorithmus)

Ein **Problem**  $P$  ist die Frage, ob eine bestimmte Eigenschaft auf gegebene Objekte zutrifft.

Dabei ist eine bekannte, abzählbaren Grundmenge solcher Objekte gegeben.

Für jedes Objekt  $o$  gilt: die Eigenschaft trifft auf  $o$  zu oder nicht.

Ein **Algorithmus** für ein Problem  $P$  ist eine Vorschrift (ein Programm), die zu beliebigem Objekt  $o$  berechnet, ob die Eigenschaft für  $o$  zutrifft oder nicht.

# Probleme und Algorithmen im allgemeinen

---

## Beispiel (Einige Probleme)

- Für  $n \in \mathbb{N}$ :  
Ist  $n$  eine Primzahl?
- Für ein Wort  $w \in \Sigma^*$  und  
ein Element  $G$  aus der Menge aller Grammatiken über  $\Sigma$ :  
Gilt  $w \in L(G)$ ?
- Für ein Element  $G$  aus der Menge aller Grammatiken:  
Ist  $L(G)$  leer (endlich, unendlich)?
- Für  $(a, b, c) \in \mathbb{N}^3$ :  
Hat  $a^n + b^n = c^n$  eine Lösung in den natürlichen Zahlen?
- Für ein Programm  $p$  aus der Menge aller Java-Programme:  
Terminiert  $p$ ?

# Endlich, unendlich und dann?

---

# Abzählbarkeit

---

## **Definition** (Abzählbarkeit)

Eine Menge  $M$  heißt **abzählbar**, wenn

- es eine **surjektive** Funktion  $f : \mathbb{N} \rightarrow M$  gibt,
- oder  $M$  leer ist.

## **Intuition**

Eine Menge ist abzählbar, wenn sie höchstens so mächtig wie  $\mathbb{N}$  ist.

# Abzählbarkeit

---

**Lemma.** Eine Menge  $M$  ist abzählbar, wenn es eine **injektive** Funktion

$$f : M \rightarrow \mathbb{N}$$

gibt.

# Abzählbarkeit

---

## Beispiel:

Abzählbar sind:

- $\mathbb{N}$
- $\mathbb{Q}$
- alle endlichen Mengen
- die Vereinigung zweier abzählbarer Mengen
- die Vereinigung abzählbar vieler abzählbarer Mengen



# David Hilbert

---

## David Hilbert (1862-1943)

- Einer der bedeutendsten und einflußreichsten Mathematiker aller Zeiten
- Professor in Königsberg und Göttingen
- Wichtige Beiträge zu
  - Logik
  - Funktionalanalysis
  - Zahlentheorie
  - Mathematische Grundlagen der Physik
  - uvm.



# Hilbert's Hotel

---

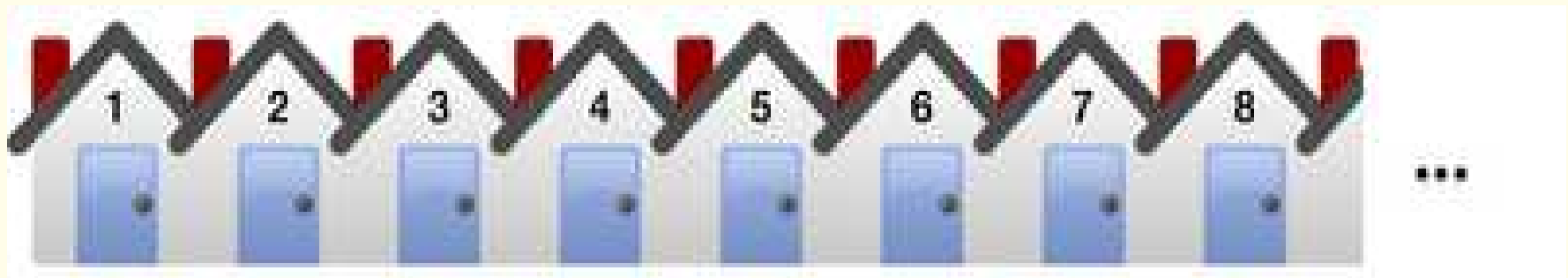
In einem Hotel mit endlich vielen Zimmern können keine Gäste mehr aufgenommen werden, sobald alle Zimmer belegt sind (Schubfachprinzip).

# Hilbert's Hotel

---

In einem Hotel mit endlich vielen Zimmern können keine Gäste mehr aufgenommen werden, sobald alle Zimmer belegt sind (Schubfachprinzip).

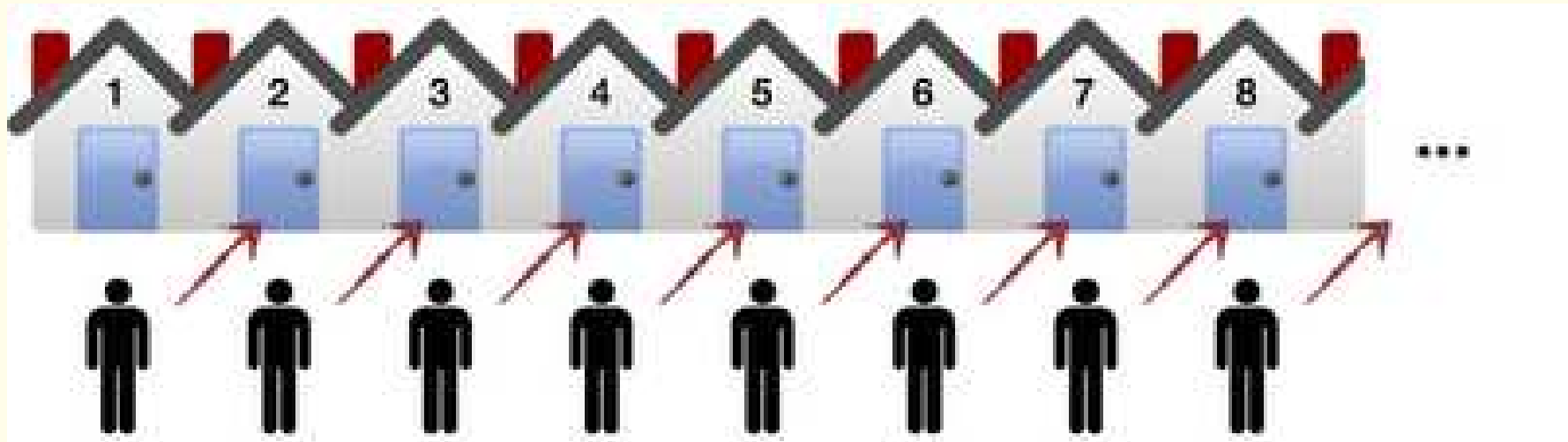
Hilberts Hotel hat nun unendlich viele Zimmer (durchnummeriert mit natürlichen Zahlen bei 1 beginnend).



Man könnte annehmen, dass dasselbe Problem auch hier auftreten würde, wenn alle Zimmer durch (unendlich viele) Gäste belegt sind.

# Hilbert's Hotel

---



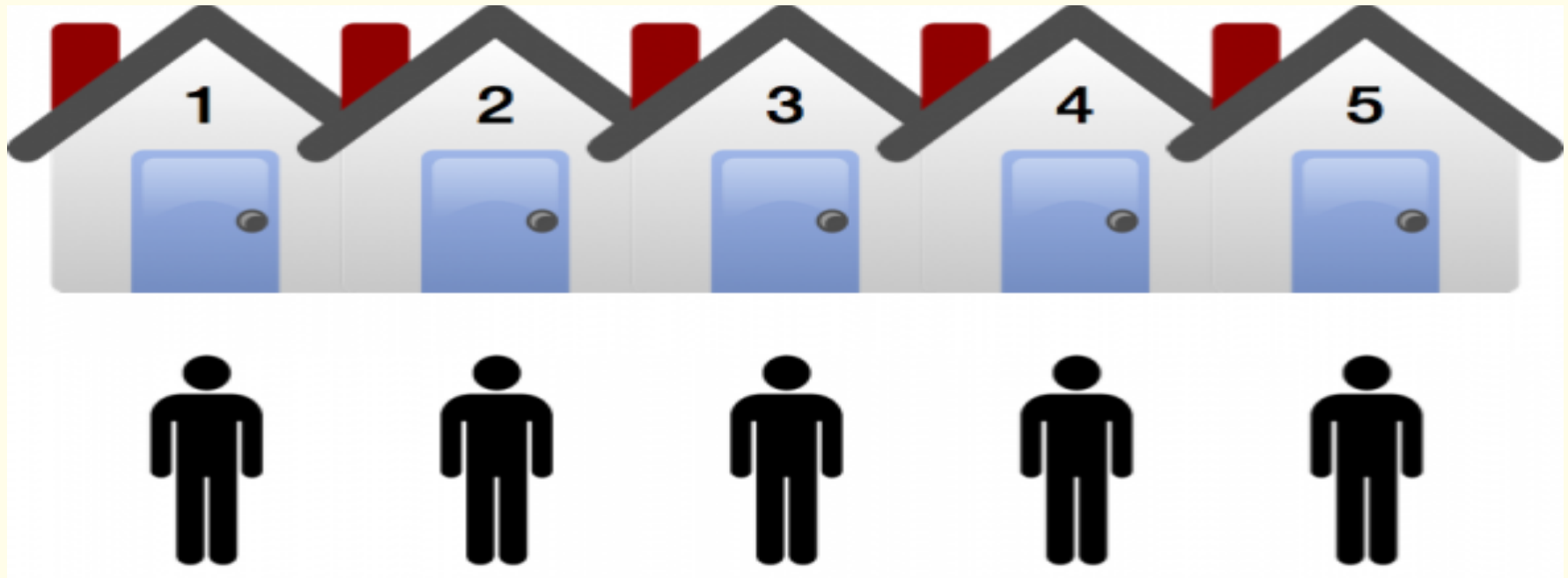
Es gibt einen Weg, Platz für einen weiteren Gast zu machen, obwohl alle Zimmer belegt sind:

- Der Gast von Zimmer 1 geht in Zimmer 2,
- der Gast von Zimmer 2 geht in Zimmer 3,
- der von Zimmer 3 nach Zimmer 4 usw.

Damit wird Zimmer 1 frei für den neuen Gast.

# Hilbert's Hotel

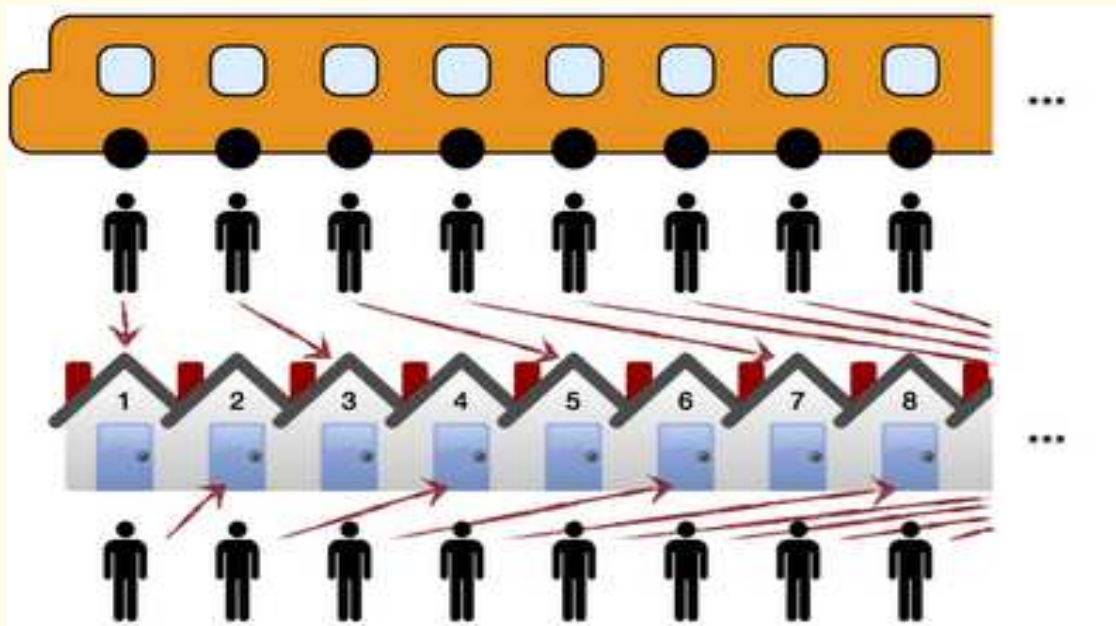
---



Da die Anzahl der Zimmer unendlich ist, gibt es keinen "letzten" Gast, der nicht in ein weiteres Zimmer umziehen könnte. Wiederholt man das, erhält man Platz für eine beliebige, aber endliche Zahl neuer Gäste.

# Hilbert's Hotel

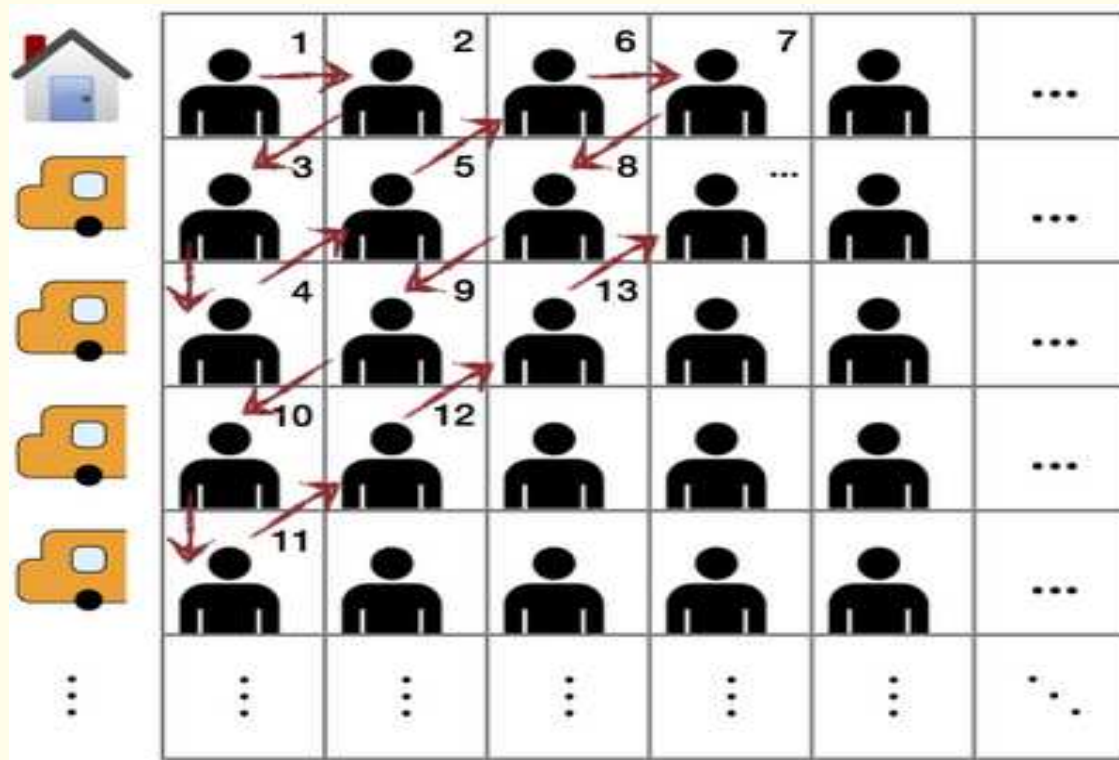
Es ist sogar möglich, Platz für abzählbar unendlich viele neue Gäste zu machen.



Der Gast von Zimmer 1 geht in Zimmer 2, der Gast von Zimmer 2 aber in Zimmer 4, der von Zimmer 3 in Zimmer 6 usw. Damit werden alle Zimmer mit ungerader Nummer frei für die abzählbar unendlich vielen Neuankömmlinge.

# Hilbert's Hotel

Wenn abzählbar unendlich viele Busse mit je abzählbar unendlich vielen Gästen vorfahren, können auch diese Gäste alle im bereits vollen Hotel untergebracht werden.



# Hilbert's Hotel

---

Andere Möglichkeit:

- die Zimmer mit ungeraden Nummern wie eben beschrieben wird frei gemacht
- die Gäste aus Bus 1 in die Zimmer 3, 9, 27, ... schickt (also in jene Zimmer, die mit Potenzen von 3 nummeriert sind;  $3 = 3^1$ ,  $9 = 3^2$ ,  $27 = 3^3$ , ...),
- die Gäste aus Bus 2 in die Zimmer 5, 25, 125, 625, etc., usw.,

also die Gäste aus Bus  $i$  in die Zimmer  $p_i$ ,  $p_i^2$ ,  $p_i^3$  etc., wobei  $p_i$  die  $i + 1$ -te Primzahl ist.

Dadurch sind alle angekommenen Gäste im Hotel untergebracht und sogar noch unendlich viele Zimmer (wie zum Beispiel das Zimmer 15, dessen Nummer keine Potenz einer Primzahl ist) frei.



# Hilbert's Hotel

---

## Weitere Möglichkeit:

Eine andere, effizientere Möglichkeit wäre die, die Hotelgäste jeweils aus den Zimmern  $n$  in die Zimmer  $2n - 1$  umziehen zu lassen, sodass alle geraden Zimmer frei werden.

Dann können die neuen Gäste aus dem Bus mit der Nummer  $n$  die Zimmer belegen, deren Zimmernummern durch  $2^n$ , nicht aber durch  $2^{n+1}$  teilbar sind, sodass kein Zimmer frei bliebe.

# Korollar

---

$\mathbb{N}$  ist abzählbar

Alle endlichen Mengen sind abzählbar

## Korollar

- die Vereinigung zweier abzählbarer Mengen ist abzählbar

Falls  $\Sigma_1, \Sigma_2$  abzählbar, so ist  $\Sigma_1 \cup \Sigma_2$  abzählbar

- die Vereinigung abzählbar vieler abzählbarer Mengen ist abzählbar

Falls  $\Sigma_1, \Sigma_2, \dots$  abzählbar, so ist  $\bigcup_{i \in \mathbb{N}} \Sigma_i$  abzählbar

Falls  $\Sigma$  abzählbar, so ist  $\Sigma^i$  abzählbar für alle  $i \in \mathbb{N}$ .