



# FORMALE SYSTEME

## 24. Vorlesung: Horn-Logik und Komplexitätstheorie

Markus Krötzsch

Professur für Wissensbasierte Systeme

TU Dresden, 18. Januar 2018

# Rückblick

# Logisches Schließen

## Allgemeine Fragestellungen des logischen Schließens

- Logische Folgerung: Gilt  $\mathcal{F} \models G$ ?
- Allgemeingültigkeit: Gilt  $\models F$ ?
- Unerfüllbarkeit: Gilt  $\models \neg F$ , d.h.  $F \models \perp$ ?

## Einige einfache Methoden des logischen Schließens

- Wahrheitstabelle
- Erfüllbarkeitstest mit DNF
- Widerlegbarkeitstest mit KNF
- Erfüllbarkeitstest mit Resolution

↪ Alle schlimmstenfalls exponentiell und oft zu ineffizient

# Horn-Logik

# Rückblick: Horn-Logik

Eine **Horn-Klausel**\* ist eine Klausel, die höchstens ein nicht-negiertes Literal enthält.  
Eine **Horn-Formel** ist eine Formel in KNF, welche nur Horn-Klauseln enthält.

\*) nach Alfred Horn, 1918–2001, US-amerikanischer Mathematiker

Man kann Horn-Formeln als Implikationen auffassen  
(sogenannte **Horn-Regeln**):

Beispiele:

$$\begin{array}{lcl} \neg p \vee \neg q \vee r & \equiv & (p \wedge q) \rightarrow r \\ p & \equiv & \top \rightarrow p \\ \neg p \vee \neg q & \equiv & (p \wedge q) \rightarrow \perp \end{array}$$

- Wir verwenden  $\top$  für die leere Prämisse und  $\perp$  für die leere Konsequenz (weil so die gewünschte logische Äquivalenz gilt)
- Es ist üblich, die Klammern der Konjunktion in der Prämisse wegzulassen

# Beispiel

Viele einfache rekursive Algorithmen können in Horn-Aussagenlogik beschrieben werden.

Beispiel: Berechnung der Variablen  $V_\epsilon$  einer CFG, welche in  $\epsilon$  umgeschrieben werden können (vgl. Vorlesung 2, Folie 32). Wir betrachten die folgende Menge  $\mathcal{F}$  von Horn-Regeln:

$$\begin{array}{ll} \top \rightarrow p_A & \text{für jede Grammatikregel } A \rightarrow \epsilon \\ p_{A_1} \wedge \dots \wedge p_{A_n} \rightarrow p_B & \text{für jede Grammatikregel } B \rightarrow A_1 \dots A_n \end{array}$$

Dann gilt  $\mathcal{F} \models p_A$  genau dann wenn  $A \in V_\epsilon$ .

# Resolution bei Horn-Klauseln

Die Resolution von Horn-Klauseln = Verknüpfung von Regeln:

$$\frac{p_1 \wedge \dots \wedge p_n \rightarrow q \quad q \wedge q_1 \wedge \dots \wedge q_m \rightarrow r}{p_1 \wedge \dots \wedge p_n \wedge q_1 \wedge \dots \wedge q_m \rightarrow r}$$

# Resolution bei Horn-Klauseln

Die Resolution von Horn-Klauseln = Verknüpfung von Regeln:

$$\frac{p_1 \wedge \dots \wedge p_n \rightarrow q \quad q \wedge q_1 \wedge \dots \wedge q_m \rightarrow r}{p_1 \wedge \dots \wedge p_n \wedge q_1 \wedge \dots \wedge q_m \rightarrow r}$$

Man kann zeigen (ohne Beweis):

Satz: Das Resolutionskalkül für Horn-Formeln bleibt auch dann vollständig, wenn man sich auf Resolventen beschränkt, bei denen eine Klausel die Form  $\{p\}$  (also  $\top \rightarrow p$ ) hat.

Diese Einschränkung entspricht der Entfernung von bereits erfüllten Vorbedingungen aus der Prämisse einer Regel:

$$\frac{\top \rightarrow q \quad q \wedge q_1 \wedge \dots \wedge q_m \rightarrow r}{q_1 \wedge \dots \wedge q_m \rightarrow r}$$



# Hyperresolution = Anwendung von Regeln

**Es ist leicht zu sehen:** Im eingeschränkten Resolutionskalkül für Horn-Formeln kann eine Resolvente aus einer Regel  $q_1 \wedge \dots \wedge q_m \rightarrow r$  nur dann zur Ableitung von  $\perp$  nützlich sein, wenn letztlich alle Vorbedingungen  $q_1, \dots, q_m$  erfüllt sind.

~> Wir können die Resolution aufschieben, bis dies gegeben ist

# Hyperresolution = Anwendung von Regeln

**Es ist leicht zu sehen:** Im eingeschränkten Resolutionskalkül für Horn-Formeln kann eine Resolvente aus einer Regel  $q_1 \wedge \dots \wedge q_m \rightarrow r$  nur dann zur Ableitung von  $\perp$  nützlich sein, wenn letztlich alle Vorbedingungen  $q_1, \dots, q_m$  erfüllt sind.

~> Wir können die Resolution aufschieben, bis dies gegeben ist

**Hyperresolution:** Mehrere Klauseln mit positiven Literalen werden parallel mit einer Klausel mit vielen negativen Literalen resolviert.

$$\frac{\begin{array}{c} \top \rightarrow q_1 \quad \dots \quad \top \rightarrow q_m \quad q_1 \wedge \dots \wedge q_m \rightarrow r \\ \hline \top \rightarrow r \end{array}}$$

Dies entspricht der intuitiven Idee einer **Regelanwendung**: „Wenn die Regel  $q_1 \wedge \dots \wedge q_m \rightarrow r$  und die Atome  $q_1, \dots, q_m$  gelten, dann kann  $r$  abgeleitet werden.“

# Komplexität des Schließens mit Horn-Formeln

## Vorteil der Hyperresolution bei Horn-Klauseln:

- Jede Resolvente hat die Form  $\top \rightarrow p$
- Es gibt nur linear viele solche Resolventen (für Atome  $p$ , die in der Formel vorkommen)

↪ Resolution muss nach linear vielen Schritten terminieren

# Komplexität des Schließens mit Horn-Formeln

## Vorteil der Hyperresolution bei Horn-Klauseln:

- Jede Resolvente hat die Form  $\top \rightarrow p$
- Es gibt nur linear viele solche Resolventen (für Atome  $p$ , die in der Formel vorkommen)

↪ Resolution muss nach linear vielen Schritten terminieren

Dies führt zu einer polynomiellen Laufzeit, nicht zu einer linearen (da jeder Resolutionsschritt einige Zeit benötigt).

Es geht aber auch noch besser:

Satz (Dowling & Gallier): Die Erfüllbarkeit einer Horn-Formel kann in linearer Zeit entschieden werden.

(Ohne Beweis)

# Logisches Schließen als Wortproblem

# Schließen als Entscheidungsproblem

Wir wissen bereits, dass Schließen ein Entscheidungsproblem ist, z.B.:

Erfüllbarkeit:

- **Eingabe:** Formel  $F$
- **Ausgabe:** „ja“ wenn  $F$  erfüllbar ist; sonst „nein.“

Wie wir wissen, können solche Probleme als Definition einer Sprache angesehen werden:

$$\mathbf{SAT} = \{\text{enc}(F) \mid F \text{ ist erfüllbar}\}$$

wobei  $\text{enc}(F)$  eine (vernünftige) Kodierung der Formel  $F$  als Wort über einem endlichen Alphabet (z.B.  $\{0, 1\}$ ) ist. Vor allem müssen wir beliebig viele Atome mit nur endlich vielen Zeichen kodieren.

# Schließen als Entscheidungsproblem

Wir wissen bereits, dass Schließen ein Entscheidungsproblem ist, z.B.:

Erfüllbarkeit:

- **Eingabe:** Formel  $F$
- **Ausgabe:** „ja“ wenn  $F$  erfüllbar ist; sonst „nein.“

Wie wir wissen, können solche Probleme als Definition einer Sprache angesehen werden:

$$\mathbf{SAT} = \{\text{enc}(F) \mid F \text{ ist erfüllbar}\}$$

wobei  $\text{enc}(F)$  eine (vernünftige) Kodierung der Formel  $F$  als Wort über einem endlichen Alphabet (z.B.  $\{0, 1\}$ ) ist. Vor allem müssen wir beliebig viele Atome mit nur endlich vielen Zeichen kodieren.

Was für eine Sprache ist **SAT**?  
Durch welches Automatenmodell wird sie erkannt?

# Schließen als Sprache (1)

Was für eine Sprache ist **SAT**?  
Durch welches Automatenmodell wird sie erkannt?

Wir wissen, dass **SAT** entscheidbar ist und daher von Typ 0 ist.



# Schließen als Sprache (1)

Was für eine Sprache ist **SAT**?  
Durch welches Automatenmodell wird sie erkannt?

Wir wissen, dass **SAT** entscheidbar ist und daher von Typ 0 ist.

Der folgende naive Algorithmus entscheidet **SAT** auf einer TM:

Naiver Erfüllbarkeitstest (Skizze):

- Wir prüfen systematisch jede Wertzuweisung auf den Atomen der gegebenen Formel
- Dazu iteriert die TM über alle Wertzuweisungen und berechnet für jede rekursiv den Wahrheitswert der Formel
- Falls eine erfüllende Zuweisung gefunden wird, dann hält die TM und akzeptiert
- Falls alle Wertzuweisungen ohne Erfolg durchlaufen worden sind, dann hält die TM und verwirft

## Schließen als Sprache (2)

Was für eine Sprache ist **SAT**?  
Durch welches Automatenmodell wird sie erkannt?

Der naive Erfüllbarkeitstest kann in linearem Speicher ablaufen, d.h. auf einem LBA  $\rightsquigarrow$   
**SAT** ist von Typ 1



# Welches Berechnungsmodell passt zu **SAT**?

Erkenntnisse bisher:

- LBAs sind stark genug für **SAT** (d.h. **SAT** ist von Typ 1)
- **SAT** ist keine reguläre Sprache, da FAs korrekte Klammerung nicht erkennen können (d.h. **SAT** ist nicht von Typ 3)

# Welches Berechnungsmodell passt zu **SAT**?

Erkenntnisse bisher:

- LBAs sind stark genug für **SAT** (d.h. **SAT** ist von Typ 1)
- **SAT** ist keine reguläre Sprache, da FAs korrekte Klammerung nicht erkennen können (d.h. **SAT** ist nicht von Typ 3)

Man kann außerdem zeigen:

**SAT** ist nicht kontextfrei.

(zumindest wenn eine offensichtliche Kodierung verwendet wird; siehe <http://cstheory.stackexchange.com/questions/37322/is-sat-a-context-free-language>)

# Welches Berechnungsmodell passt zu **SAT**?

Erkenntnisse bisher:

- LBAs sind stark genug für **SAT** (d.h. **SAT** ist von Typ 1)
- **SAT** ist keine reguläre Sprache, da FAs korrekte Klammerung nicht erkennen können (d.h. **SAT** ist nicht von Typ 3)

Man kann außerdem zeigen:

**SAT** ist nicht kontextfrei.

(zumindest wenn eine offensichtliche Kodierung verwendet wird; siehe <http://cstheory.stackexchange.com/questions/37322/is-sat-a-context-free-language>)

Können wir **SAT** noch genauer charakterisieren als mit LBAs?

# Welches Berechnungsmodell passt zu **SAT**?

Erkenntnisse bisher:

- LBAs sind stark genug für **SAT** (d.h. **SAT** ist von Typ 1)
- **SAT** ist keine reguläre Sprache, da FAs korrekte Klammerung nicht erkennen können (d.h. **SAT** ist nicht von Typ 3)

Man kann außerdem zeigen:

**SAT** ist nicht kontextfrei.

(zumindest wenn eine offensichtliche Kodierung verwendet wird; siehe <http://cstheory.stackexchange.com/questions/37322/is-sat-a-context-free-language>)

Können wir **SAT** noch genauer charakterisieren als mit LBAs?

↪ Ja, mithilfe anderer Arten von eingeschränkten TMs.

# Komplexität



# Turingmaschinen beschränken

TMs verwenden zwei Ressourcen, die man beschränken kann:

- **Speicher:** die Zahl der verwendeten Speicherzellen
- **Zeit:** die Zahl der durchgeführten Berechnungsschritte

# Turingmaschinen beschränken

TMs verwenden zwei Ressourcen, die man beschränken kann:

- **Speicher:** die Zahl der verwendeten Speicherzellen
- **Zeit:** die Zahl der durchgeführten Berechnungsschritte

Feste Schranken ergeben wenig Sinn (endliche Automaten)

~> Schranken werden als Funktion in der Länge der Eingabe angegeben

Beispiel: LBAs beschränken den verfügbaren Speicher auf die Anzahl der Symbole in der Eingabe. Dies entspricht einer Funktion, welche die Länge  $n$  der Eingabe auf den Maximalwert von  $n$  Speicherzellen abbildet.

## Zur Erinnerung: $O$ -Notation

Die  $O$ -Notation (mit großem  $O$ !) charakterisiert Funktionen nach ihrem asymptotischen Verhalten und versteckt lineare Faktoren.

Für Funktionen  $f, g : \mathbb{N} \rightarrow \mathbb{R}$  schreiben wir  $f \in O(g)$  wenn gilt:

Es gibt eine Zahl  $c > 0$  und eine Zahl  $n_0 \in \mathbb{N}$ ,  
so dass für jedes  $n > n_0$  gilt:  $f(n) \leq c \cdot g(n)$

Das bedeutet,  $f$  wächst nicht wesentlich schneller als  $g$ .

# Zur Erinnerung: $O$ -Notation

Die  $O$ -Notation (mit großem  $O$ !) charakterisiert Funktionen nach ihrem asymptotischen Verhalten und versteckt lineare Faktoren.

Für Funktionen  $f, g : \mathbb{N} \rightarrow \mathbb{R}$  schreiben wir  $f \in O(g)$  wenn gilt:

Es gibt eine Zahl  $c > 0$  und eine Zahl  $n_0 \in \mathbb{N}$ ,  
so dass für jedes  $n > n_0$  gilt:  $f(n) \leq c \cdot g(n)$

Das bedeutet,  $f$  wächst nicht wesentlich schneller als  $g$ .

**Notation 1:** Manchmal schreibt man statt  $f \in O(g)$  auch  $f = O(g)$  (allerdings ist  $=$  dann eine asymmetrische Relation).

**Notation 2:** Manchmal schreibt man statt  $f \in O(g)$  (oder  $f = O(g)$ ) oft auch  $f(n) \in O(g(n))$  (oder  $f(n) = O(g(n))$ ).

- Beispiele:
- $(10n^3 + 42n^2 - n + 100) \in O(n^3)$
  - $(2^n + n^{2000}) \in O(2^n)$
  - $2^{729} \in O(1)$

# Schranken für Zeit und Raum

Die  $O$ -Notation wird verwendet, um allgemeine Ressourcenschranken für TMs anzugeben.

Sei  $f : \mathbb{N} \rightarrow \mathbb{R}$  eine Funktion und  $\mathcal{M}$  eine Turingmaschine.

- $\mathcal{M}$  heißt  **$O(f)$ -zeitbeschränkt** wenn es eine Funktion  $g \in O(f)$  gibt, so dass  $\mathcal{M}$  für eine beliebige Eingabe  $w \in \Sigma^*$  nach maximal  $g(|w|)$  Schritten anhält.
- $\mathcal{M}$  heißt  **$O(f)$ -speicherbeschränkt** wenn es eine Funktion  $g \in O(f)$  gibt, so dass  $\mathcal{M}$  für eine beliebige Eingabe  $w \in \Sigma^*$  hält und zuvor maximal  $g(|w|)$  Speicherzellen verwendet.

# Schranken für Zeit und Raum

Die  $O$ -Notation wird verwendet, um allgemeine Ressourcenschranken für TMs anzugeben.

Sei  $f : \mathbb{N} \rightarrow \mathbb{R}$  eine Funktion und  $\mathcal{M}$  eine Turingmaschine.

- $\mathcal{M}$  heißt  **$O(f)$ -zeitbeschränkt** wenn es eine Funktion  $g \in O(f)$  gibt, so dass  $\mathcal{M}$  für eine beliebige Eingabe  $w \in \Sigma^*$  nach maximal  $g(|w|)$  Schritten anhält.
- $\mathcal{M}$  heißt  **$O(f)$ -speicherbeschränkt** wenn es eine Funktion  $g \in O(f)$  gibt, so dass  $\mathcal{M}$  für eine beliebige Eingabe  $w \in \Sigma^*$  hält und zuvor maximal  $g(|w|)$  Speicherzellen verwendet.

Beispiel: Ein LBA entspricht einer  $O(n)$ -speicherbeschränkten TM.\*

Beispiel: Der naive Erfüllbarkeitstest für **SAT** ist  $O(2^n)$ -zeitbeschränkt.

\*) Wobei man die bei LBAs erzwungene Speicherbeschränkung in das Programm der TM einbauen muss. Unsere LBA-Definition erlaubt keine linearen Faktoren für den Speicherbedarf. Diese haben keinen Einfluss auf die Rechenstärke von LBAs, da man lineare Speichergewinne durch ein größeres Arbeitsalphabet simulieren kann.

# Zeit und Raum, deterministisch

Beschränkte TMs können verwendet werden, um viele weitere Sprachklassen zu definieren.

Sei  $f : \mathbb{N} \rightarrow \mathbb{R}$  eine Funktion.

- **DTIME**( $f(n)$ ) ist die Klasse aller Sprachen **L**, welche durch eine  $O(f)$ -zeitbeschränkte Turingmaschine entschieden werden können.
- **DSPACE**( $f(n)$ ) ist die Klasse aller Sprachen **L**, welche durch eine  $O(f)$ -speicherbeschränkte Turingmaschine entschieden werden können.

# Zeit und Raum, deterministisch

Beschränkte TMs können verwendet werden, um viele weitere Sprachklassen zu definieren.

Sei  $f : \mathbb{N} \rightarrow \mathbb{R}$  eine Funktion.

- **DTIME**( $f(n)$ ) ist die Klasse aller Sprachen **L**, welche durch eine  $O(f)$ -zeitbeschränkte Turingmaschine entschieden werden können.
- **DSPACE**( $f(n)$ ) ist die Klasse aller Sprachen **L**, welche durch eine  $O(f)$ -speicherbeschränkte Turingmaschine entschieden werden können.

Beispiel: **SAT**  $\in$  DTIME( $2^n$ ) und **SAT**  $\in$  DSPACE( $n$ ).



# Zeit und Raum, deterministisch

Beschränkte TMs können verwendet werden, um viele weitere Sprachklassen zu definieren.

Sei  $f : \mathbb{N} \rightarrow \mathbb{R}$  eine Funktion.

- **DTIME**( $f(n)$ ) ist die Klasse aller Sprachen **L**, welche durch eine  $O(f)$ -zeitbeschränkte Turingmaschine entschieden werden können.
- **DSPACE**( $f(n)$ ) ist die Klasse aller Sprachen **L**, welche durch eine  $O(f)$ -speicherbeschränkte Turingmaschine entschieden werden können.

Beispiel: **SAT**  $\in$  DTIME( $2^n$ ) und **SAT**  $\in$  DSPACE( $n$ ).

Beispiel: Das Halteproblem ist in keiner der Klassen DTIME( $f(n)$ ) oder DSPACE( $f(n)$ ). Unentscheidbare Probleme benötigen uneingeschränkten Zugang zu beliebig vielen Ressourcen einer TM.

# Maschinenmodelle

Es gibt viele unterschiedliche Versionen von deterministischen Turingmaschinen (z.B. Mehrband-Maschinen).

Sind  $\text{DTIME}(f)$  und  $\text{DSpace}(f)$  für jedes TM-Modell gleich?

# Maschinenmodelle

Es gibt viele unterschiedliche Versionen von deterministischen Turingmaschinen (z.B. Mehrband-Maschinen).

Sind  $DTIME(f)$  und  $DSPACE(f)$  für jedes TM-Modell gleich?

**Antwort:** „Bei vielen kleineren Variationen sind sie es, aber nicht in allen Fällen (oder zumindest ist dies nicht immer bekannt).“

Beispiel: Jede  $O(f(n))$ -zeitbeschränkte  $k$ -Band-TM kann durch eine  $O(k \cdot f^2(n))$ -zeitbeschränkte 1-Band TM simuliert werden (wie in Vorlesung 18 gezeigt). Einfacher gesagt: Der Verzicht auf mehrere Bänder verursacht maximal quadratische Zeitkosten ( $k$  ist hier ein linearer Faktor).

# Maschinenmodelle

Es gibt viele unterschiedliche Versionen von deterministischen Turingmaschinen (z.B. Mehrband-Maschinen).

Sind  $DTIME(f)$  und  $DSPACE(f)$  für jedes TM-Modell gleich?

**Antwort:** „Bei vielen kleineren Variationen sind sie es, aber nicht in allen Fällen (oder zumindest ist dies nicht immer bekannt).“

Beispiel: Jede  $O(f(n))$ -zeitbeschränkte  $k$ -Band-TM kann durch eine  $O(k \cdot f^2(n))$ -zeitbeschränkte 1-Band TM simuliert werden (wie in Vorlesung 18 gezeigt). Einfacher gesagt: Der Verzicht auf mehrere Bänder verursacht maximal quadratische Zeitkosten ( $k$  ist hier ein linearer Faktor).

↪ Es ist sinnvoll, **noch allgemeinere Sprachklassen** zu betrachten, die auch gegenüber polynomiellen Änderungen der Ressourcen robust sind

# Wichtige Komplexitätsklassen

Die wichtigen deterministischen Komplexitätsklassen fassen jeweils ganze Familien von zeit- oder speicherbeschränkten Klassen zusammen. Wir erwähnen hier nur die praktisch wichtigsten:

$$P = PTime = \bigcup_{d \geq 1} DTime(n^d)$$

polynomielle Zeit

$$Exp = ExpTime = \bigcup_{d \geq 1} DTime(2^{n^d})$$

exponentielle Zeit\*

$$L = LogSpace = DSpace(\log n)$$

logarithmischer Speicher

$$PSpace = \bigcup_{d \geq 1} DSpace(n^d)$$

polynomieller Speicher

\*) Anmerkung: Dies ist die praktisch wichtigste Definition von „exponentieller Zeit“. Es gibt daneben auch  $E = ETime = \bigcup_{d \geq 1} DTime(2^{dn})$  (exponentielle Zeit mit linearem Exponenten).

# LogSpace? Wie soll das gehen?

Für  $n > 1$  gilt  $\log(n) < n$ . Auch beliebige lineare Faktoren können das nur für kleine  $n$  kompensieren.

Eine  $O(\log(n))$ -speicherbeschränkte TM darf also weniger Speicher verwenden als ihre Eingabe benötigt.  $\leadsto$  Wie soll das gehen?

# LogSpace? Wie soll das gehen?

Für  $n > 1$  gilt  $\log(n) < n$ . Auch beliebige lineare Faktoren können das nur für kleine  $n$  kompensieren.

Eine  $O(\log(n))$ -speicherbeschränkte TM darf also weniger Speicher verwenden als ihre Eingabe benötigt.  $\leadsto$  Wie soll das gehen?

Man definiert  $O(\log(n))$ -speicherbeschränkte Turingmaschinen als besondere Mehrband-TMs:

- Das erste Band ist das **Eingabeband**. Es enthält die Eingabe und darf nur gelesen aber nicht beschrieben werden.
- Das zweite Band ist das **Arbeitsband**. Es darf beliebig gelesen und beschrieben werden, aber es ist auf  $O(\log(n))$ -Speicherzellen beschränkt.

Das genügt zur Erkennung von Sprachen. Wenn die TM eine Ausgabe berechnen soll, dann wird dafür ein drittes **Ausgabeband** verwendet, auf dem man beliebig viele Zeichen einmalig schreiben aber nicht lesen kann.

# Beziehungen der Komplexitätsklassen

Eine wichtige Frage der Komplexitätstheorie ist, was man über die Beziehungen der Komplexitätsklassen aussagen kann.



# Beziehungen der Komplexitätsklassen

Eine wichtige Frage der Komplexitätstheorie ist, was man über die Beziehungen der Komplexitätsklassen aussagen kann.

Offensichtlich führen (asymptotisch) höhere Ressourcenschranken zu größeren Sprachklassen. Oft ist aber nicht klar, ob man mit mehr Ressourcen auch wirklich mehr (oder einfach nur gleich viele) Probleme lösen kann. Bei unseren Klassen ist das aber bekannt:

Fakt: Es gilt  $P \subsetneq \text{Exp}$  und  $\text{LogSpace} \subsetneq \text{PSpace}$ .

# Beziehungen der Komplexitätsklassen

Eine wichtige Frage der Komplexitätstheorie ist, was man über die Beziehungen der Komplexitätsklassen aussagen kann.

Offensichtlich führen (asymptotisch) höhere Ressourcenschranken zu größeren Sprachklassen. Oft ist aber nicht klar, ob man mit mehr Ressourcen auch wirklich mehr (oder einfach nur gleich viele) Probleme lösen kann. Bei unseren Klassen ist das aber bekannt:

Fakt: Es gilt  $P \subsetneq \text{Exp}$  und  $\text{LogSpace} \subsetneq \text{PSpace}$ .

Weiterhin kann man Speicher mit Zeit in Beziehung bringen:

- In Zeit  $n$  kann man nur  $n$  Speicherzellen nutzen
- Alle möglichen Konfigurationen auf  $n$  Speicherzellen kann man in exponentieller Zeit (bezüglich  $n$ ) berechnen

(für LBAs haben wir das in Vorlesung 20 besprochen; siehe „Konfigurationsgraph“)

Fakt: Es gilt  $\text{LogSpace} \subseteq P \subseteq \text{PSpace} \subseteq \text{Exp}$ .

# Beispiele

Unsere Klassen sind recht robust: Details der Implementierung haben oft keinen Einfluss auf die Einordnung eines Problems.

Oft genügt eine Implementierungsskizze um zu zeigen, dass eine Sprache in einer dieser Klassen liegt.

# Beispiele

Unsere Klassen sind recht robust: Details der Implementierung haben oft keinen Einfluss auf die Einordnung eines Problems.

Oft genügt eine Implementierungsskizze um zu zeigen, dass eine Sprache in einer dieser Klassen liegt.

Beispiel: Erfüllbarkeit von propositionaler Horn-Logik ist in P. Unser Resolutionsalgorithmus liefert allerdings keinen Hinweis auf Machbarkeit in LogSpace.

# Beispiele

Unsere Klassen sind recht robust: Details der Implementierung haben oft keinen Einfluss auf die Einordnung eines Problems.

Oft genügt eine Implementierungsskizze um zu zeigen, dass eine Sprache in einer dieser Klassen liegt.

Beispiel: Erfüllbarkeit von propositionaler Horn-Logik ist in  $P$ . Unser Resolutionsalgorithmus liefert allerdings keinen Hinweis auf Machbarkeit in  $\text{LogSpace}$ .

Beispiel: **SAT** ist in  $\text{ExpTime}$  aber auch in  $\text{PSPACE}$ .

# Beispiele

Unsere Klassen sind recht robust: Details der Implementierung haben oft keinen Einfluss auf die Einordnung eines Problems.

Oft genügt eine Implementierungsskizze um zu zeigen, dass eine Sprache in einer dieser Klassen liegt.

Beispiel: Erfüllbarkeit von propositionaler Horn-Logik ist in P. Unser Resolutionsalgorithmus liefert allerdings keinen Hinweis auf Machbarkeit in LogSpace.

Beispiel: **SAT** ist in ExpTime aber auch in PSpace.

Beispiel: Das Wortproblem jeder regulären Sprache ist in LogSpace. Tatsächlich benötigt ein DFA gar keinen Speicher.

# Beispiele

Unsere Klassen sind recht robust: Details der Implementierung haben oft keinen Einfluss auf die Einordnung eines Problems.

Oft genügt eine Implementierungsskizze um zu zeigen, dass eine Sprache in einer dieser Klassen liegt.

Beispiel: Erfüllbarkeit von propositionaler Horn-Logik ist in P. Unser Resolutionsalgorithmus liefert allerdings keinen Hinweis auf Machbarkeit in LogSpace.

Beispiel: **SAT** ist in ExpTime aber auch in PSpace.

Beispiel: Das Wortproblem jeder regulären Sprache ist in LogSpace. Tatsächlich benötigt ein DFA gar keinen Speicher.

Beispiel: Die Transformation einer Formel in Negationsnormalform ist in LogSpace möglich. Dies ist kein Spracherkennungsproblem sondern eine Berechnung mit Ein- und Ausgabe.

# Ressourcen nichtdeterministischer TMs

Bei NTMs gibt es viele mögliche Berechnungspfade.

↪ Welche Pfade meinen wir, wenn wir Ressourcen beschränken?



# Ressourcen nichtdeterministischer TMs

Bei NTMs gibt es viele mögliche Berechnungspfade.

→ Welche Pfade meinen wir, wenn wir Ressourcen beschränken?

– Alle!

Sei  $f : \mathbb{N} \rightarrow \mathbb{R}$  eine Funktion und  $\mathcal{M}$  eine nichtdeterministische TM.

- $\mathcal{M}$  heißt  **$O(f)$ -zeitbeschränkt** wenn es eine Funktion  $g \in O(f)$  gibt, so dass  $\mathcal{M}$  für eine beliebige Eingabe  $w \in \Sigma^*$  auf jedem Berechnungspfad nach maximal  $g(|w|)$  Schritten anhält.
- $\mathcal{M}$  heißt  **$O(f)$ -speicherbeschränkt** wenn es eine Funktion  $g \in O(f)$  gibt, so dass  $\mathcal{M}$  für eine beliebige Eingabe  $w \in \Sigma^*$  hält und zuvor auf jedem Berechnungspfad maximal  $g(|w|)$  Speicherzellen verwendet.

Eine zeit- oder speicherbeschränkte NTM muss also auch auf erfolglosen Pfaden („falsch geratene Übergänge“) garantiert innerhalb der Ressourcengrenzen halten.

# Zeit und Raum, nichtdeterministisch

Die entsprechenden Sprachklassen werden genau wie bei deterministischen TMs definiert:

Sei  $f : \mathbb{N} \rightarrow \mathbb{R}$  eine Funktion.

- **NTIME**( $f(n)$ ) ist die Klasse aller Sprachen  $\mathbf{L}$ , welche durch eine  $O(f)$ -zeitbeschränkte NTM entschieden werden können.
- **NSPACE**( $f(n)$ ) ist die Klasse aller Sprachen  $\mathbf{L}$ , welche durch eine  $O(f)$ -speicherbeschränkte NTM entschieden werden können.

# Nichtdeterministische Komplexitätsklassen

Auch hier beschränken wir uns auf einige wichtige Fälle:

$$\text{NP} = \text{NPTime} = \bigcup_{d \geq 1} \text{NTime}(n^d)$$

nichtdet. polynomielle Zeit

$$\text{NExp} = \text{NExpTime} = \bigcup_{d \geq 1} \text{NTime}(2^{n^d})$$

nichtdet. exponentielle Zeit

$$\text{NL} = \text{NLogSpace} = \text{NSpace}(\log n)$$

nichtdet. logarithmischer Speicher

$$\text{NPSpace} = \bigcup_{d \geq 1} \text{NSpace}(n^d)$$

nichtdet. polynomieller Speicher

# Nichtdeterministische Komplexitätsklassen

Auch hier beschränken wir uns auf einige wichtige Fälle:

$$\text{NP} = \text{NPTime} = \bigcup_{d \geq 1} \text{NTime}(n^d)$$

nichtdet. polynomielle Zeit

$$\text{NExp} = \text{NExpTime} = \bigcup_{d \geq 1} \text{NTime}(2^{n^d})$$

nichtdet. exponentielle Zeit

$$\text{NL} = \text{NLogSpace} = \text{NSpace}(\log n)$$

nichtdet. logarithmischer Speicher

$$\text{NPSpace} = \bigcup_{d \geq 1} \text{NSpace}(n^d)$$

nichtdet. polynomieller Speicher

Beispiel: **SAT**  $\in$  NP. Man kann unseren naiven Erfüllbarkeitstest auf einer NTM so implementieren, dass nur eine (erfüllende) Wertzuweisung geraten wird. Der Wahrheitswert der Formel für diese Zuweisung kann in polynomieller Zeit überprüft werden.

# Deterministisch vs. nichtdeterministisch

Welche Beziehungen haben diese Klassen zu anderen?

# Deterministisch vs. nichtdeterministisch

Welche Beziehungen haben diese Klassen zu anderen?

- Die Beziehungen zwischen nichtdeterministischen Klassen sind analog zu denen im deterministischen Fall:

$$NL \subseteq NP \subseteq NPSPACE \subseteq NEXP$$

# Deterministisch vs. nichtdeterministisch

Welche Beziehungen haben diese Klassen zu anderen?

- Die Beziehungen zwischen nichtdeterministischen Klassen sind analog zu denen im deterministischen Fall:

$$NL \subseteq NP \subseteq NPSpace \subseteq NExp$$

- Eine DTM kann als NTM aufgefasst werden, d.h. die nichtdeterministischen Klassen sind immer stärker:

$$L \subseteq NL \quad P \subseteq NP \quad PSpace \subseteq NPSpace \quad Exp \subseteq NExp$$

# Deterministisch vs. nichtdeterministisch

Welche Beziehungen haben diese Klassen zu anderen?

- Die Beziehungen zwischen nichtdeterministischen Klassen sind analog zu denen im deterministischen Fall:

$$NL \subseteq NP \subseteq NPSpace \subseteq NExp$$

- Eine DTM kann als NTM aufgefasst werden, d.h. die nichtdeterministischen Klassen sind immer stärker:

$$L \subseteq NL \quad P \subseteq NP \quad PSpace \subseteq NPSpace \quad Exp \subseteq NExp$$

- Man kann NTMs mit DTMs simulieren, aber das ist oft mit exponentiellen Mehrkosten verbunden (Vorlesung 19).



# Deterministisch vs. nichtdeterministisch

Welche Beziehungen haben diese Klassen zu anderen?

- Die Beziehungen zwischen nichtdeterministischen Klassen sind analog zu denen im deterministischen Fall:

$$NL \subseteq NP \subseteq NPSpace \subseteq NExp$$

- Eine DTM kann als NTM aufgefasst werden, d.h. die nichtdeterministischen Klassen sind immer stärker:

$$L \subseteq NL \quad P \subseteq NP \quad PSpace \subseteq NPSpace \quad Exp \subseteq NExp$$

- Man kann NTMs mit DTMs simulieren, aber das ist oft mit exponentiellen Mehrkosten verbunden (Vorlesung 19).
- Der berühmte [Satz von Savitch](#) besagt, dass speicherbeschränkte NTMs durch DTMs mit nur quadratischen Mehrkosten simuliert werden können. Insbesondere gilt damit  $PSpace = NPSpace$ .

Zusammenfassung der wichtigsten bekannten Beziehungen:

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSpace = NPSpace \subseteq Exp \subseteq NExp$$

# Die Grenzen unseres Wissens

Wir wissen:

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSpace = NPSPACE \subseteq Exp \subseteq NExp$$

- Wir wissen nicht, ob irgendeines dieser  $\subseteq$  sogar  $\subsetneq$  ist.
- Insbesondere wissen wir nicht, ob  $P \subsetneq NP$  oder  $P = NP$ .
- Wir wissen nicht einmal, ob  $L \subsetneq NP$  oder  $L = NP$ .

# Die Grenzen unseres Wissens

Wir wissen:

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSpace = NPSPACE \subseteq Exp \subseteq NExp$$

- Wir wissen nicht, ob irgendeines dieser  $\subseteq$  sogar  $\subsetneq$  ist.
- Insbesondere wissen wir nicht, ob  $P \subsetneq NP$  oder  $P = NP$ .
- Wir wissen nicht einmal, ob  $L \subsetneq NP$  oder  $L = NP$ .

Es wird aber vermutet, dass alle  $\subseteq$  eigentlich  $\subsetneq$  sind. Bekannt ist das aber nur bei exponentiell großen Ressourcenunterschieden:

Es gilt:

- $NL \subsetneq PSpace$
- $P \subsetneq Exp$
- $NP \subsetneq NExp$

# Zusammenfassung und Ausblick

**Komplexitätsklassen** sind Mengen von Sprachen, die man (grob) einteilt entsprechend der Ressourcen, die eine TM zur Entscheidung ihres Wortproblems benötigt.

**Horn-Logik** erlaubt logisches Schließen in P.

Das propositionale Erfüllbarkeitsproblem **SAT** ist in NP.

Offene Fragen:

- Gibt es auch sub-exponentielle Algorithmen für aussagenlogisches Schließen?
- Gilt  $P \neq NP$ ?