TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut für Künstliche Intelligenz, Professur für Computational Logic

FORMALE SYSTEME

24. Vorlesung: Horn-Logik und Komplexitätstheorie

Hannes Straß

TU Dresden, 17. Januar 2022

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. $F \models \bot$?

Einige einfache Methoden des logischen Schließens

- Wahrheitswertetabelle
- Erfüllbarkeitstest mit DNF
- Widerlegbarkeitstest mit KNF
- Erfüllbarkeitstest mit Resolution

$\rightsquigarrow$ Alle schlimmstenfalls exponentiell und oft zu ineffizient.

Aufzeichnung startet . . .

Horn-Logik

Horn-Logik

Bisher waren alle unsere Ansätze für aussagenlogisches Schließen exponentiell – geht es auch einfacher?

Horn-Logik

Bisher waren alle unsere Ansätze für aussagenlogisches Schließen exponentiell – geht es auch einfacher?

Idee: Beschränke die Form von Formeln.

Eine Horn-Klausel[a] ist eine Klausel, die höchstens ein nichtnegiertes Literal enthält.
Eine Horn-Formel ist eine Formel in KNF, welche nur Horn-Klauseln enthält.

[a] Nach Alfred Horn, 1918–2001, US-amerikanischer Mathematiker.

Horn-Logik

Bisher waren alle unsere Ansätze für aussagenlogisches Schließen exponentiell – geht es auch einfacher?

Idee: Beschränke die Form von Formeln.

Eine Horn-Klausel[a] ist eine Klausel, die höchstens ein nichtnegiertes Literal enthält.
Eine Horn-Formel ist eine Formel in KNF, welche nur Horn-Klauseln enthält.

[a] Nach Alfred Horn, 1918–2001, US-amerikanischer Mathematiker.

Beispiele:

- $\neg p \vee \neg q \vee r$ ist eine Horn-Klausel
- $q \vee p$ ist keine Horn-Klausel
- p ist eine Horn-Klausel
- $\neg p \vee \neg q$ ist eine Horn-Klausel
- $p \vee p$ ist keine Horn-Klausel (aber äquivalent zu einer)

Horn-Klauseln als Implikationen

Jede Horn-Klausel kann als eine Implikation ohne Negation und Disjunktion ausgedrückt werden.

Beispiele:

$$\begin{aligned} \neg p \vee \neg q \vee r &\equiv (p \wedge q) \rightarrow r \\ p &\equiv \top \rightarrow p \\ \neg p \vee \neg q &\equiv (p \wedge q) \rightarrow \bot \end{aligned}$$

- Wir verwenden $\top$ für die leere Prämisse und $\bot$ für die leere Konsequenz (weil so die gewünschte logische Äquivalenz gilt).
- Als Implikationen geschriebene Horn-Klauseln werden oft als (Horn-)Regeln bezeichnet.
- 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_ϵ einer CFG, welche in ϵ umgeschrieben werden können (siehe Vorlesung 2, Folie 32). Wir verwenden ein aussagenlogisches Vokabular mit einem Atom p_V für jedes Nichtterminal V der Grammatik (wobei p_A ausdrückt, dass $\mathsf{A} \in V_\epsilon$ gilt) und betrachten die folgende Menge $\mathcal{F}$ von Horn-Regeln:

$$\top \to p_\mathsf{A} \qquad \text{für jede Grammatikregel } \mathsf{A} \to \epsilon$$

$$p_{\mathsf{A}_1} \wedge \ldots \wedge p_{\mathsf{A}_n} \to p_\mathsf{B} \qquad \text{für jede Grammatikregel } \mathsf{B} \to \mathsf{A}_1 \cdots \mathsf{A}_n$$

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

Resolution bei Horn-Klauseln

Die Resolution von Horn-Klauseln entspricht der Verknüpfung von Regeln:

$$\frac{p_1 \wedge \ldots \wedge p_n \to q \qquad q \wedge q_1 \wedge \ldots \wedge q_m \to r}{p_1 \wedge \ldots \wedge p_n \wedge q_1 \wedge \ldots \wedge q_m \to r}$$

Resolution bei Horn-Klauseln

Die Resolution von Horn-Klauseln entspricht der Verknüpfung von Regeln:

$$\frac{p_1 \wedge \ldots \wedge p_n \rightarrow q \qquad q \wedge q_1 \wedge \ldots \wedge q_m \rightarrow r}{p_1 \wedge \ldots \wedge p_n \wedge q_1 \wedge \ldots \wedge q_m \rightarrow r}$$

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:

$$\frac{\top \rightarrow q \qquad q \wedge q_1 \wedge \ldots \wedge q_m \rightarrow r}{q_1 \wedge \ldots \wedge q_m \rightarrow r}$$

Hyperresolution = Anwendung von Regeln

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

$$\frac{\top \to q_1 \qquad \ldots \qquad \top \to q_m \qquad q_1 \wedge \ldots \wedge q_m \to r}{\top \to r}$$

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

Komplexität des Schließens mit Horn-Formeln

Vorteil der Hyperresolution bei Horn-Klauseln:

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

$\rightsquigarrow$ Resolution muss nach linear vielen Schritten terminieren.

Komplexität des Schließens mit Horn-Formeln

Vorteil der Hyperresolution bei Horn-Klauseln:

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

$\leadsto$ 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:

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

wobei $\text{enc}(F)$ eine (vernünftige) Kodierung der Formel F als Wort über einem endlichen Alphabet (z.B. $\{0, 1, \#, \vee, \neg\}$) ist. Vor allem müssen wir beliebig viele Atome mit nur endlich vielen Zeichen kodieren.
Z.B. könnte die KNF $p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$ kodiert werden als $1\#\neg 10 \vee 11\#10\#$.

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:

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

wobei enc(F) eine (vernünftige) Kodierung der Formel F als Wort über einem endlichen Alphabet (z.B. $\{0, 1, \#, \vee, \neg\}$) ist. Vor allem müssen wir beliebig viele Atome mit nur endlich vielen Zeichen kodieren.
Z.B. könnte die KNF $p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$ kodiert werden als $1\#\neg 10 \vee 11\#10\#$.

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 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 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 verwirft die TM (aber hält).

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.

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.

Naive Erfüllbarkeit auf LBA (Skizze):

- Wir verwenden ein Arbeitsalphabet mit mehreren „Spuren“:
- (1) Eingabe,
  (2) aktuell ermittelte Wahrheitswerte aller Teilformeln (Literale und Klauseln),
  (3) aktuell getestete Wertzuweisung.
- Skizze der möglichen Kodierung einer KNF $p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$:

- Man kann nun systematisch Spur 3 iterieren, Spur 2 rekursiv berechnen und den Wert der Gesamtformel prüfen (machbar, wir verzichten auf die Details).

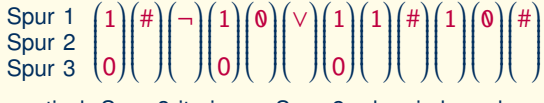
Schließen als Sprache (2)

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.

Naive Erfüllbarkeit auf LBA (Skizze):

- Wir verwenden ein Arbeitsalphabet mit mehreren „Spuren“:
- (1) Eingabe,
  (2) aktuell ermittelte Wahrheitswerte aller Teilformeln (Literale und Klauseln),
  (3) aktuell getestete Wertzuweisung.
- Skizze der möglichen Kodierung einer KNF $p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$:



- Man kann nun systematisch Spur 3 iterieren, Spur 2 rekursiv berechnen und den Wert der Gesamtformel prüfen (machbar, wir verzichten auf die Details).

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.

Naive Erfüllbarkeit auf LBA (Skizze):

- Wir verwenden ein Arbeitsalphabet mit mehreren „Spuren“:
- (1) Eingabe,
  (2) aktuell ermittelte Wahrheitswerte aller Teilformeln (Literale und Klauseln),
  (3) aktuell getestete Wertzuweisung.
- Skizze der möglichen Kodierung einer KNF $p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$:

- Man kann nun systematisch Spur 3 iterieren, Spur 2 rekursiv berechnen und den Wert der Gesamtformel prüfen (machbar, wir verzichten auf die Details).

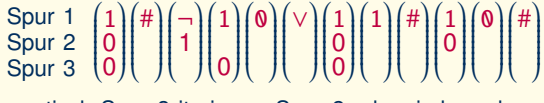
Schließen als Sprache (2)

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.

Naive Erfüllbarkeit auf LBA (Skizze):

- Wir verwenden ein Arbeitsalphabet mit mehreren „Spuren“:
- (1) Eingabe,
  (2) aktuell ermittelte Wahrheitswerte aller Teilformeln (Literale und Klauseln),
  (3) aktuell getestete Wertzuweisung.
- Skizze der möglichen Kodierung einer KNF $p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$:



- Man kann nun systematisch Spur 3 iterieren, Spur 2 rekursiv berechnen und den Wert der Gesamtformel prüfen (machbar, wir verzichten auf die Details).

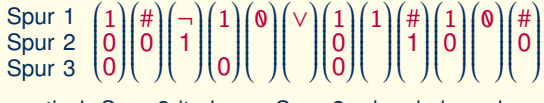
Schließen als Sprache (2)

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.

Naive Erfüllbarkeit auf LBA (Skizze):

- Wir verwenden ein Arbeitsalphabet mit mehreren „Spuren“:
- (1) Eingabe,
  (2) aktuell ermittelte Wahrheitswerte aller Teilformeln (Literale und Klauseln),
  (3) aktuell getestete Wertzuweisung.
- Skizze der möglichen Kodierung einer KNF $p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$:



- Man kann nun systematisch Spur 3 iterieren, Spur 2 rekursiv berechnen und den Wert der Gesamtformel prüfen (machbar, wir verzichten auf die Details).

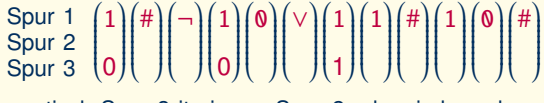
Schließen als Sprache (2)

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.

Naive Erfüllbarkeit auf LBA (Skizze):

- Wir verwenden ein Arbeitsalphabet mit mehreren „Spuren“:
- (1) Eingabe,
  (2) aktuell ermittelte Wahrheitswerte aller Teilformeln (Literale und Klauseln),
  (3) aktuell getestete Wertzuweisung.
- Skizze der möglichen Kodierung einer KNF $p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$:



- Man kann nun systematisch Spur 3 iterieren, Spur 2 rekursiv berechnen und den Wert der Gesamtformel prüfen (machbar, wir verzichten auf die Details).

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; dies betrifft bereits die Syntax der Aussagenlogik, ist also unabhängig von Semantik/Erfüllbarkeit).

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; dies betrifft bereits die Syntax der Aussagenlogik, ist also unabhängig von Semantik/Erfüllbarkeit).

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; dies betrifft bereits die Syntax der Aussagenlogik, ist also unabhängig von Semantik/Erfüllbarkeit).

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; dies betrifft bereits die Syntax der Aussagenlogik, ist also unabhängig von Semantik/Erfüllbarkeit).

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?

$\rightsquigarrow$ Ja, mit Hilfe 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 (sie führen zu endlichen Automaten).

$\rightsquigarrow$ 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, also $f : \mathbb{N} \to \mathbb{R}, \quad n \mapsto n$.

Zur Erinnerung: O-Notation

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

Für Funktionen $f, g : \mathbb{N} \to \mathbb{R}$ schreiben wir $f \in O(g)$ genau dann, 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 höchstens so schnell wie g.

Zur Erinnerung: O-Notation

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

Für Funktionen $f, g : \mathbb{N} \to \mathbb{R}$ schreiben wir $f \in O(g)$ genau dann, 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 höchstens so schnell wie g.

Notation 1: Manche Autor:innen schreiben statt $f \in O(g)$ auch $f = O(g)$ (allerdings ist = dann eine asymmetrische Relation).

Notation 2: Oft schreiben wir statt $f \in O(g)$ auch $f(n) \in O(g(n))$.

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

- $\mathcal{M}$ heißt genau dann $O(f)$-zeitbeschränkt, wenn es eine Funktion $g \in O(f)$ gibt, sodass für alle $w \in \Sigma^*$ gilt: $\mathcal{M}$ hält auf Eingabe w nach maximal $g(|w|)$ Schritten.
- $\mathcal{M}$ heißt genau dann $O(f)$-speicherbeschränkt, wenn es eine Funktion $g \in O(f)$ gibt, sodass für alle $w \in \Sigma^*$ gilt:
  $\mathcal{M}$ hält auf Eingabe w und verwendet dabei maximal $g(|w|)$ Speicherzellen.

Schranken für Zeit und Raum

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

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

- $\mathcal{M}$ heißt genau dann $O(f)$-zeitbeschränkt, wenn es eine Funktion $g \in O(f)$ gibt, sodass für alle $w \in \Sigma^*$ gilt: $\mathcal{M}$ hält auf Eingabe w nach maximal $g(|w|)$ Schritten.
- $\mathcal{M}$ heißt genau dann $O(f)$-speicherbeschränkt, wenn es eine Funktion $g \in O(f)$ gibt, sodass für alle $w \in \Sigma^*$ gilt:
  $\mathcal{M}$ hält auf Eingabe w und verwendet dabei maximal $g(|w|)$ Speicherzellen.

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 aber 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 $f : \mathbb{N} \to \mathbb{R}$ 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 $f : \mathbb{N} \to \mathbb{R}$ 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 $f : \mathbb{N} \to \mathbb{R}$ 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($f(n)$). Per Definition enthalten diese Klassen nur entscheidbare Probleme. Intuitiv heißt das: 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 DTIME(f) und 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).

$\rightsquigarrow$ Es ist sinnvoll, noch allgemeinere Sprachklassen zu betrachten, die auch gegenüber polynomiellen Änderungen der Ressourcen robust sind.

Wichtige Komplextitätsklassen

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

$$\mathsf{P} = \mathsf{PTime} = \bigcup_{d \geq 1} \mathsf{DTime}\left(n^d\right) \qquad \text{polynomielle Zeit}$$

$$\mathsf{Exp} = \mathsf{ExpTime} = \bigcup_{d \geq 1} \mathsf{DTime}\left(2^{n^d}\right) \qquad \text{exponentielle Zeit}^*$$

$$\mathsf{L} = \mathsf{LogSpace} = \mathsf{DSpace}(\log(n)) \qquad \text{logarithmischer Speicher}$$

$$\mathsf{PSpace} = \bigcup_{d \geq 1} \mathsf{DSpace}\left(n^d\right) \qquad \text{polynomieller Speicher}$$

*) Anmerkung: Dies ist die praktisch wichtigste Definition von „exponentieller Zeit“. Es gibt daneben auch $\mathsf{E} = \mathsf{ETime} = \bigcup_{d \geq 1} \mathsf{DTime}\left(2^{dn}\right)$ (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. $\rightsquigarrow$ 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. $\rightsquigarrow$ 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 $O(\log(n))$ viele 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.
Für die hier genannten Klassen ist aber Folgendes bekannt:

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

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.
Für die hier genannten Klassen ist aber Folgendes bekannt:

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

Weiterhin kann man Speicher mit Zeit in Beziehung bringen:

- In n Berechnungsschritten kann eine TM 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 $\mathrm{LogSpace} \subseteq \mathrm{P} \subseteq \mathrm{PSpace} \subseteq \mathrm{Exp}$.

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 aussagenlogischer 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 aussagenlogischer Horn-Logik ist in P. Unser Resolutionsalgorithmus liefert allerdings keinen Hinweis auf Machbarkeit in LogSpace.

Beispiel: **SAT** ist in ExpTime, aber auch in 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 aussagenlogischer 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 aussagenlogischer 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.

$\rightsquigarrow$ Welche Pfade meinen wir, wenn wir Ressourcen beschränken?

Ressourcen nichtdeterministischer TMs

Bei NTMs gibt es viele mögliche Berechnungspfade.
$\rightsquigarrow$ Welche Pfade meinen wir, wenn wir Ressourcen beschränken?

Alle!

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

- $\mathcal{M}$ heißt genau dann $O(f)$-zeitbeschränkt, wenn es eine Funktion $g \in O(f)$ gibt, sodass für alle $w \in \Sigma^*$ gilt:
  $\mathcal{M}$ hält bei Eingabe w auf jedem Berechnungspfad nach maximal $g(|w|)$ Schritten.
- $\mathcal{M}$ heißt genau dann $O(f)$-speicherbeschränkt, wenn es eine Funktion $g \in O(f)$ gibt, sodass für alle $w \in \Sigma^*$ gilt: $\mathcal{M}$ hält bei Eingabe w auf jedem Berechnungspfad und verwendet dabei maximal $g(|w|)$ Speicherzellen.

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

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

Nichtdeterministische Komplextitätsklassen

Auch hier beschränken wir uns auf einige wichtige Fälle:

$$\mathsf{NP} = \mathsf{NPTime} = \bigcup_{d \geq 1} \mathsf{NTime}\left(n^d\right) \qquad \text{nichtdet. polynomielle Zeit}$$

$$\mathsf{NExp} = \mathsf{NExpTime} = \bigcup_{d \geq 1} \mathsf{NTime}\left(2^{n^d}\right) \qquad \text{nichtdet. exponentielle Zeit}$$

$$\mathsf{NL} = \mathsf{NLogSpace} = \mathsf{NSpace}(\log n) \qquad \text{nichtdet. logarithmischer Speicher}$$

$$\mathsf{NPSpace} = \bigcup_{d \geq 1} \mathsf{NSpace}\left(n^d\right) \qquad \text{nichtdet. polynomieller Speicher}$$

Beispiel: **SAT** $\in$ NP

Satz: SAT $\in$ NP.

Beweis (Skizze): Wir definieren eine NTM $\mathcal{M}_{\mathbf{SAT}}$ wie folgt:

- $\mathcal{M}_{\mathbf{SAT}}$ verwendet ein Arbeitsalphabet mit 3 „Spuren“ (siehe Folie 22).
- Die Eingabe befindet sich auf Spur 1.
- $\mathcal{M}_{\mathbf{SAT}}$ schreibt zuerst nichtdeterministisch eine Wertzuweisung w auf Spur 3;
- danach berechnet sie auf Spur 2 deterministisch die Wahrheitswerte aller Literale.
- Schließlich überprüft $\mathcal{M}_{\mathbf{SAT}}$ deterministisch, ob alle Klauseln erfüllt sind. □

Beispiel: Wir betrachten $F = p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$ und skizzieren mögliche Läufe:

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	#		
Spur 2														
Spur 3														

Beispiel: **SAT** $\in$ NP

Satz: SAT $\in$ NP.

Beweis (Skizze): Wir definieren eine NTM $\mathcal{M}_{\mathbf{SAT}}$ wie folgt:

- $\mathcal{M}_{\mathbf{SAT}}$ verwendet ein Arbeitsalphabet mit 3 „Spuren“ (siehe Folie 22).
- Die Eingabe befindet sich auf Spur 1.
- $\mathcal{M}_{\mathbf{SAT}}$ schreibt zuerst nichtdeterministisch eine Wertzuweisung w auf Spur 3;
- danach berechnet sie auf Spur 2 deterministisch die Wahrheitswerte aller Literale.
- Schließlich überprüft $\mathcal{M}_{\mathbf{SAT}}$ deterministisch, ob alle Klauseln erfüllt sind. □

Beispiel: Wir betrachten $F = p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$ und skizzieren mögliche Läufe:

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	#	
Spur 2													
Spur 3	1												

Beispiel: **SAT** $\in$ NP

Satz: SAT $\in$ NP.

Beweis (Skizze): Wir definieren eine NTM $\mathcal{M}_{\mathbf{SAT}}$ wie folgt:

- $\mathcal{M}_{\mathbf{SAT}}$ verwendet ein Arbeitsalphabet mit 3 „Spuren“ (siehe Folie 22).
- Die Eingabe befindet sich auf Spur 1.
- $\mathcal{M}_{\mathbf{SAT}}$ schreibt zuerst nichtdeterministisch eine Wertzuweisung w auf Spur 3;
- danach berechnet sie auf Spur 2 deterministisch die Wahrheitswerte aller Literale.
- Schließlich überprüft $\mathcal{M}_{\mathbf{SAT}}$ deterministisch, ob alle Klauseln erfüllt sind. □

Beispiel: Wir betrachten $F = p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$ und skizzieren mögliche Läufe:

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	#	
Spur 2													
Spur 3	1			0									

Beispiel: **SAT** $\in$ NP

Satz: SAT $\in$ NP.

Beweis (Skizze): Wir definieren eine NTM $\mathcal{M}_{\mathbf{SAT}}$ wie folgt:

- $\mathcal{M}_{\mathbf{SAT}}$ verwendet ein Arbeitsalphabet mit 3 „Spuren“ (siehe Folie 22).
- Die Eingabe befindet sich auf Spur 1.
- $\mathcal{M}_{\mathbf{SAT}}$ schreibt zuerst nichtdeterministisch eine Wertzuweisung w auf Spur 3;
- danach berechnet sie auf Spur 2 deterministisch die Wahrheitswerte aller Literale.
- Schließlich überprüft $\mathcal{M}_{\mathbf{SAT}}$ deterministisch, ob alle Klauseln erfüllt sind. □

Beispiel: Wir betrachten $F = p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$ und skizzieren mögliche Läufe:

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	#		
Spur 2														
Spur 3	1			0			1							

Beispiel: **SAT** $\in$ NP

Satz: SAT $\in$ NP.

Beweis (Skizze): Wir definieren eine NTM $\mathcal{M}_{\mathbf{SAT}}$ wie folgt:

- $\mathcal{M}_{\mathbf{SAT}}$ verwendet ein Arbeitsalphabet mit 3 „Spuren“ (siehe Folie 22).
- Die Eingabe befindet sich auf Spur 1.
- $\mathcal{M}_{\mathbf{SAT}}$ schreibt zuerst nichtdeterministisch eine Wertzuweisung w auf Spur 3;
- danach berechnet sie auf Spur 2 deterministisch die Wahrheitswerte aller Literale.
- Schließlich überprüft $\mathcal{M}_{\mathbf{SAT}}$ deterministisch, ob alle Klauseln erfüllt sind. □

Beispiel: Wir betrachten $F = p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$ und skizzieren mögliche Läufe:

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	0	#	
Spur 2	1		1				1			0				
Spur 3	1			0			1							

Beispiel: **SAT** $\in$ NP

Satz: SAT $\in$ NP.

Beweis (Skizze): Wir definieren eine NTM $\mathcal{M}_{\mathbf{SAT}}$ wie folgt:

- $\mathcal{M}_{\mathbf{SAT}}$ verwendet ein Arbeitsalphabet mit 3 „Spuren“ (siehe Folie 22).
- Die Eingabe befindet sich auf Spur 1.
- $\mathcal{M}_{\mathbf{SAT}}$ schreibt zuerst nichtdeterministisch eine Wertzuweisung w auf Spur 3;
- danach berechnet sie auf Spur 2 deterministisch die Wahrheitswerte aller Literale.
- Schließlich überprüft $\mathcal{M}_{\mathbf{SAT}}$ deterministisch, ob alle Klauseln erfüllt sind. □

Beispiel: Wir betrachten $F = p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$ und skizzieren mögliche Läufe:

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	#		
Spur 2	1	1	1				1							
Spur 3	1			0			1			0				

Beispiel: **SAT** $\in$ NP

Satz: SAT $\in$ NP.

Beweis (Skizze): Wir definieren eine NTM $\mathcal{M}_{\mathbf{SAT}}$ wie folgt:

- $\mathcal{M}_{\mathbf{SAT}}$ verwendet ein Arbeitsalphabet mit 3 „Spuren“ (siehe Folie 22).
- Die Eingabe befindet sich auf Spur 1.
- $\mathcal{M}_{\mathbf{SAT}}$ schreibt zuerst nichtdeterministisch eine Wertzuweisung w auf Spur 3;
- danach berechnet sie auf Spur 2 deterministisch die Wahrheitswerte aller Literale.
- Schließlich überprüft $\mathcal{M}_{\mathbf{SAT}}$ deterministisch, ob alle Klauseln erfüllt sind. □

Beispiel: Wir betrachten $F = p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$ und skizzieren mögliche Läufe:

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	0	#	
Spur 2	1	1	1				1		1					
Spur 3	1			0			1							

Beispiel: **SAT** $\in$ NP

Satz: SAT $\in$ NP.

Beweis (Skizze): Wir definieren eine NTM $\mathcal{M}_{\mathbf{SAT}}$ wie folgt:

- $\mathcal{M}_{\mathbf{SAT}}$ verwendet ein Arbeitsalphabet mit 3 „Spuren“ (siehe Folie 22).
- Die Eingabe befindet sich auf Spur 1.
- $\mathcal{M}_{\mathbf{SAT}}$ schreibt zuerst nichtdeterministisch eine Wertzuweisung w auf Spur 3;
- danach berechnet sie auf Spur 2 deterministisch die Wahrheitswerte aller Literale.
- Schließlich überprüft $\mathcal{M}_{\mathbf{SAT}}$ deterministisch, ob alle Klauseln erfüllt sind. □

Beispiel: Wir betrachten $F = p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$ und skizzieren mögliche Läufe:

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	0	#
Spur 2	1	1	1				1						
Spur 3	1			0			1						

(verwerfender Lauf)

Beispiel: **SAT** $\in$ NP

Satz: SAT $\in$ NP.

Beweis (Skizze): Wir definieren eine NTM $\mathcal{M}_{\mathbf{SAT}}$ wie folgt:

- $\mathcal{M}_{\mathbf{SAT}}$ verwendet ein Arbeitsalphabet mit 3 „Spuren“ (siehe Folie 22).
- Die Eingabe befindet sich auf Spur 1.
- $\mathcal{M}_{\mathbf{SAT}}$ schreibt zuerst nichtdeterministisch eine Wertzuweisung w auf Spur 3;
- danach berechnet sie auf Spur 2 deterministisch die Wahrheitswerte aller Literale.
- Schließlich überprüft $\mathcal{M}_{\mathbf{SAT}}$ deterministisch, ob alle Klauseln erfüllt sind. □

Beispiel: Wir betrachten $F = p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$ und skizzieren mögliche Läufe:

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	0	#	(verwerfender Lauf)
Spur 2	1	1					1		1					
Spur 3	1		1	0			1							

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	#		
Spur 2														
Spur 3														

Beispiel: **SAT** $\in$ NP

Satz: SAT $\in$ NP.

Beweis (Skizze): Wir definieren eine NTM $\mathcal{M}_{\mathbf{SAT}}$ wie folgt:

- $\mathcal{M}_{\mathbf{SAT}}$ verwendet ein Arbeitsalphabet mit 3 „Spuren“ (siehe Folie 22).
- Die Eingabe befindet sich auf Spur 1.
- $\mathcal{M}_{\mathbf{SAT}}$ schreibt zuerst nichtdeterministisch eine Wertzuweisung w auf Spur 3;
- danach berechnet sie auf Spur 2 deterministisch die Wahrheitswerte aller Literale.
- Schließlich überprüft $\mathcal{M}_{\mathbf{SAT}}$ deterministisch, ob alle Klauseln erfüllt sind. □

Beispiel: Wir betrachten $F = p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$ und skizzieren mögliche Läufe:

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	0	#	(verwerfender Lauf)
Spur 2	1	1					1		1					
Spur 3	1			0			1							

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	#	
Spur 2													
Spur 3	1												

Beispiel: **SAT** $\in$ NP

Satz: SAT $\in$ NP.

Beweis (Skizze): Wir definieren eine NTM $\mathcal{M}_{\mathbf{SAT}}$ wie folgt:

- $\mathcal{M}_{\mathbf{SAT}}$ verwendet ein Arbeitsalphabet mit 3 „Spuren“ (siehe Folie 22).
- Die Eingabe befindet sich auf Spur 1.
- $\mathcal{M}_{\mathbf{SAT}}$ schreibt zuerst nichtdeterministisch eine Wertzuweisung w auf Spur 3;
- danach berechnet sie auf Spur 2 deterministisch die Wahrheitswerte aller Literale.
- Schließlich überprüft $\mathcal{M}_{\mathbf{SAT}}$ deterministisch, ob alle Klauseln erfüllt sind. □

Beispiel: Wir betrachten $F = p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$ und skizzieren mögliche Läufe:

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	0	#	(verwerfender Lauf)
Spur 2	1	1					1		1					
Spur 3	1			0			1							

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	#	
Spur 2													
Spur 3	1			1									

Beispiel: **SAT** $\in$ NP

Satz: SAT $\in$ NP.

Beweis (Skizze): Wir definieren eine NTM $\mathcal{M}_{\mathbf{SAT}}$ wie folgt:

- $\mathcal{M}_{\mathbf{SAT}}$ verwendet ein Arbeitsalphabet mit 3 „Spuren“ (siehe Folie 22).
- Die Eingabe befindet sich auf Spur 1.
- $\mathcal{M}_{\mathbf{SAT}}$ schreibt zuerst nichtdeterministisch eine Wertzuweisung w auf Spur 3;
- danach berechnet sie auf Spur 2 deterministisch die Wahrheitswerte aller Literale.
- Schließlich überprüft $\mathcal{M}_{\mathbf{SAT}}$ deterministisch, ob alle Klauseln erfüllt sind. □

Beispiel: Wir betrachten $F = p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$ und skizzieren mögliche Läufe:

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	0	#	(verwerfender Lauf)
Spur 2	1	1					1							
Spur 3	1			0			1							

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	#	
Spur 2													
Spur 3	1			1			1						

Beispiel: **SAT** $\in$ NP

Satz: SAT $\in$ NP.

Beweis (Skizze): Wir definieren eine NTM $\mathcal{M}_{\mathbf{SAT}}$ wie folgt:

- $\mathcal{M}_{\mathbf{SAT}}$ verwendet ein Arbeitsalphabet mit 3 „Spuren“ (siehe Folie 22).
- Die Eingabe befindet sich auf Spur 1.
- $\mathcal{M}_{\mathbf{SAT}}$ schreibt zuerst nichtdeterministisch eine Wertzuweisung w auf Spur 3;
- danach berechnet sie auf Spur 2 deterministisch die Wahrheitswerte aller Literale.
- Schließlich überprüft $\mathcal{M}_{\mathbf{SAT}}$ deterministisch, ob alle Klauseln erfüllt sind. □

Beispiel: Wir betrachten $F = p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$ und skizzieren mögliche Läufe:

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	0	#	(verwerfender Lauf)
Spur 2	1	1	1				1		1				0	
Spur 3	1			0			1							

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	0	#	
Spur 2	1		0				1			1				
Spur 3	1			1			1							

Beispiel: **SAT** $\in$ NP

Satz: SAT $\in$ NP.

Beweis (Skizze): Wir definieren eine NTM $\mathcal{M}_{\mathbf{SAT}}$ wie folgt:

- $\mathcal{M}_{\mathbf{SAT}}$ verwendet ein Arbeitsalphabet mit 3 „Spuren“ (siehe Folie 22).
- Die Eingabe befindet sich auf Spur 1.
- $\mathcal{M}_{\mathbf{SAT}}$ schreibt zuerst nichtdeterministisch eine Wertzuweisung w auf Spur 3;
- danach berechnet sie auf Spur 2 deterministisch die Wahrheitswerte aller Literale.
- Schließlich überprüft $\mathcal{M}_{\mathbf{SAT}}$ deterministisch, ob alle Klauseln erfüllt sind. □

Beispiel: Wir betrachten $F = p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$ und skizzieren mögliche Läufe:

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	0	#		
Spur 2	1	1	1				1		1				0		
Spur 3	1			0			1								

(verwerfender Lauf)

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	0	#		
Spur 2	1	1	0				1		1				0		
Spur 3	1			1			1								

Beispiel: **SAT** $\in$ NP

Satz: SAT $\in$ NP.

Beweis (Skizze): Wir definieren eine NTM $\mathcal{M}_{\mathbf{SAT}}$ wie folgt:

- $\mathcal{M}_{\mathbf{SAT}}$ verwendet ein Arbeitsalphabet mit 3 „Spuren“ (siehe Folie 22).
- Die Eingabe befindet sich auf Spur 1.
- $\mathcal{M}_{\mathbf{SAT}}$ schreibt zuerst nichtdeterministisch eine Wertzuweisung w auf Spur 3;
- danach berechnet sie auf Spur 2 deterministisch die Wahrheitswerte aller Literale.
- Schließlich überprüft $\mathcal{M}_{\mathbf{SAT}}$ deterministisch, ob alle Klauseln erfüllt sind. □

Beispiel: Wir betrachten $F = p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$ und skizzieren mögliche Läufe:

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	0	#	(verwerfender Lauf)
Spur 2	1	1	1				1		1					
Spur 3	1			0			1							

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	0	#	
Spur 2	1	1	0				1		1					
Spur 3	1			1			1							

Beispiel: **SAT** $\in$ NP

Satz: SAT $\in$ NP.

Beweis (Skizze): Wir definieren eine NTM $\mathcal{M}_{\mathbf{SAT}}$ wie folgt:

- $\mathcal{M}_{\mathbf{SAT}}$ verwendet ein Arbeitsalphabet mit 3 „Spuren“ (siehe Folie 22).
- Die Eingabe befindet sich auf Spur 1.
- $\mathcal{M}_{\mathbf{SAT}}$ schreibt zuerst nichtdeterministisch eine Wertzuweisung w auf Spur 3;
- danach berechnet sie auf Spur 2 deterministisch die Wahrheitswerte aller Literale.
- Schließlich überprüft $\mathcal{M}_{\mathbf{SAT}}$ deterministisch, ob alle Klauseln erfüllt sind. □

Beispiel: Wir betrachten $F = p_1 \wedge (\neg p_2 \vee p_3) \wedge p_2$ und skizzieren mögliche Läufe:

Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	0	#	(verwerfender Lauf)
Spur 2	1	1	1				1		1	1			0		
Spur 3	1			0			1								
Spur 1	1	#	¬	1	0	∨	1	1	#	1	0	0	#	(akzeptierender Lauf)
Spur 2	1	1	0				1		1	1			1		
Spur 3	1			1			1								

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:

$$\mathsf{NL} \subseteq \mathsf{NP} \subseteq \mathsf{NPSpace} \subseteq \mathsf{NExp}$$

Deterministisch vs. nichtdeterministisch

Welche Beziehungen haben diese Klassen zu anderen?

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

$$\mathsf{NL} \subseteq \mathsf{NP} \subseteq \mathsf{NPSpace} \subseteq \mathsf{NExp}$$

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

$$\mathsf{L} \subseteq \mathsf{NL} \qquad \mathsf{P} \subseteq \mathsf{NP} \qquad \mathsf{PSpace} \subseteq \mathsf{NPSpace} \qquad \mathsf{Exp} \subseteq \mathsf{NExp}$$

Deterministisch vs. nichtdeterministisch

Welche Beziehungen haben diese Klassen zu anderen?

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

$$\mathsf{NL} \subseteq \mathsf{NP} \subseteq \mathsf{NPSpace} \subseteq \mathsf{NExp}$$

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

$$\mathsf{L} \subseteq \mathsf{NL} \qquad \mathsf{P} \subseteq \mathsf{NP} \qquad \mathsf{PSpace} \subseteq \mathsf{NPSpace} \qquad \mathsf{Exp} \subseteq \mathsf{NExp}$$

- 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:

$$\mathsf{NL} \subseteq \mathsf{NP} \subseteq \mathsf{NPSpace} \subseteq \mathsf{NExp}$$

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

$$\mathsf{L} \subseteq \mathsf{NL} \qquad \mathsf{P} \subseteq \mathsf{NP} \qquad \mathsf{PSpace} \subseteq \mathsf{NPSpace} \qquad \mathsf{Exp} \subseteq \mathsf{NExp}$$

- 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 $\mathsf{PSpace} = \mathsf{NPSpace}$.

Zusammenfassung der wichtigsten bekannten Beziehungen:

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

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 aussagenlogische Erfüllbarkeitsproblem **SAT** ist in NP.

Offene Fragen:

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