Theoretische Informatik

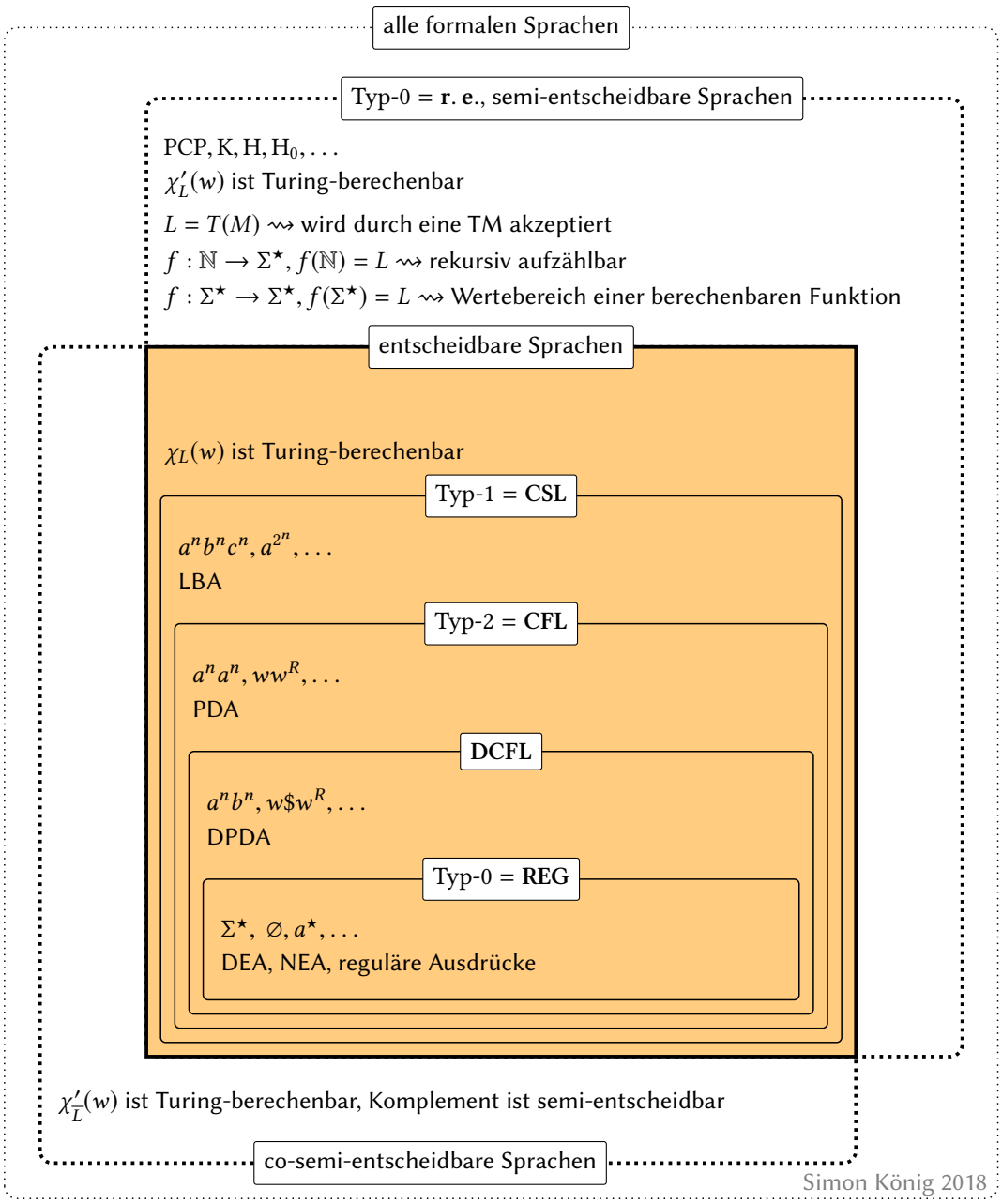
Zusammenfassung der Module TI 1 und 2

Simon König

Inhaltsverzeichnis

1: Überblick

Ein kurzer Gesamtüberblick über den Zusammenhang von formalen Sprachen zur Komplexität und Berechenbarkeitstheorie:



1

Kapitel 1: Grundlagen

1: Grundlagen der Aussagenlogik

1.1 Syntax der Aussagenlogik

- Atomare Formeln: A_i mit $i \in \mathbb{N}$
- F und G Formeln $\rightarrow (F \wedge G)$, $(F \vee G)$ und $\neg F$ auch Formeln

1.2 Semantik der Aussagenlogik

$D \subseteq \{A_1, A_2, \ldots\}$
Eine Abbildung $\mathcal{A} : D \rightarrow \{0, 1\}$ heißt Belegung. Weiter

- Eine Belegung $\mathcal{A}$ ist **passend** zu einer Formel F, falls alle in F vorkommenden atomaren Variablen zum Definitionsbereich von $\mathcal{A}$ gehören.
- Eine Belegung $\mathcal{A}$ ist **Modell** für eine Formel, falls $\mathcal{A}$ zu F passend ist und $\mathcal{A}(F) = 1$ gilt.
  Man schreibt dann $\mathcal{A} \models F$.
- F ist **erfüllbar**, falls ein Modell für F existiert.
- F ist **gültig**, falls alle passenden Belegungen Modelle sind. $\rightarrow F$ nennt man dann eine **Tautologie**.
  Das Komplement einer Tautologie ist unerfüllbar.
- Zwei Formeln F und G heißen **semantisch äquivalent**, wenn alle zu beiden passenden Belegungen $\mathcal{A}$ gilt: $\mathcal{A}(F) = \mathcal{A}(G)$. Man schreibt dann $F \equiv G$.

1.3 Prädikatenlogik

2

Kapitel 2: Formale Sprachen und Automatentheorie

3

Kapitel 3: Berechenbarkeitstheorie und Komplexität

1: Rekursiv aufzählbare Sprachen

Typ-0 = **r**. **e**.

Eine Turingmaschine M *akzeptiert* die Sprache L, wenn sie nach endlicher Zeit hält. Bei Eingaben, die nicht zu L gehören, rechnet sie unendlich lang weiter (dieses Verhalten führt später auf das Halteproblem). Solche Sprachen L gehören dann zu **r**. **e**., sie sind *semi-entscheidbar*.

1.1 Sätze zu r. e.

- Die Klasse der Typ-0 Sprachen ist abgeschlossen unter Sternopertaion, Vereinigung, Schnitt und Konkatenation.
- Die Klasse der durch Turingmaschinen erkennbaren/akzeptierten Sprachen ist gleich der Klasse der rekursiv aufzählbaren.

2: Entscheidbare Sprachen

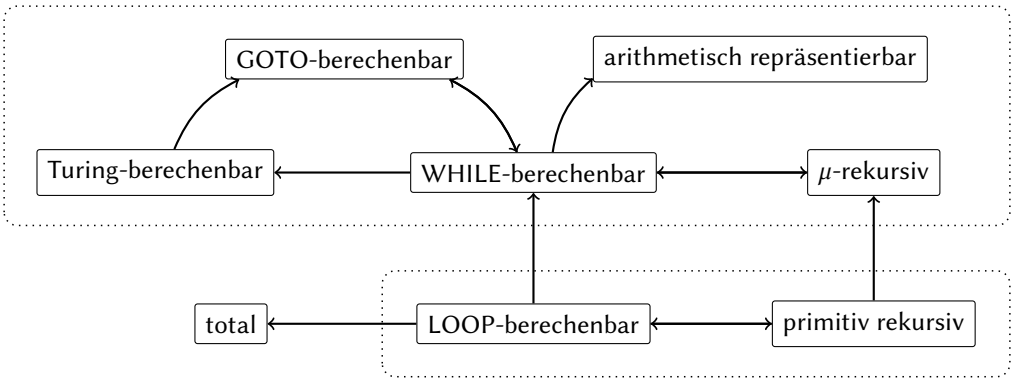
REC $\subset$ **r. e.**

Eine Turingmaschine *entscheidet* die Sprache L, wenn sie für jede Eingabe nach endlicher Zeit stoppt und JA oder NEIN ausgibt, je nachdem ob $w \in L$ gilt oder nicht. Diese Sprachen L gehören dann zur Klasse der entscheidbaren Sprachen **REC** (rekursive Sprachen).

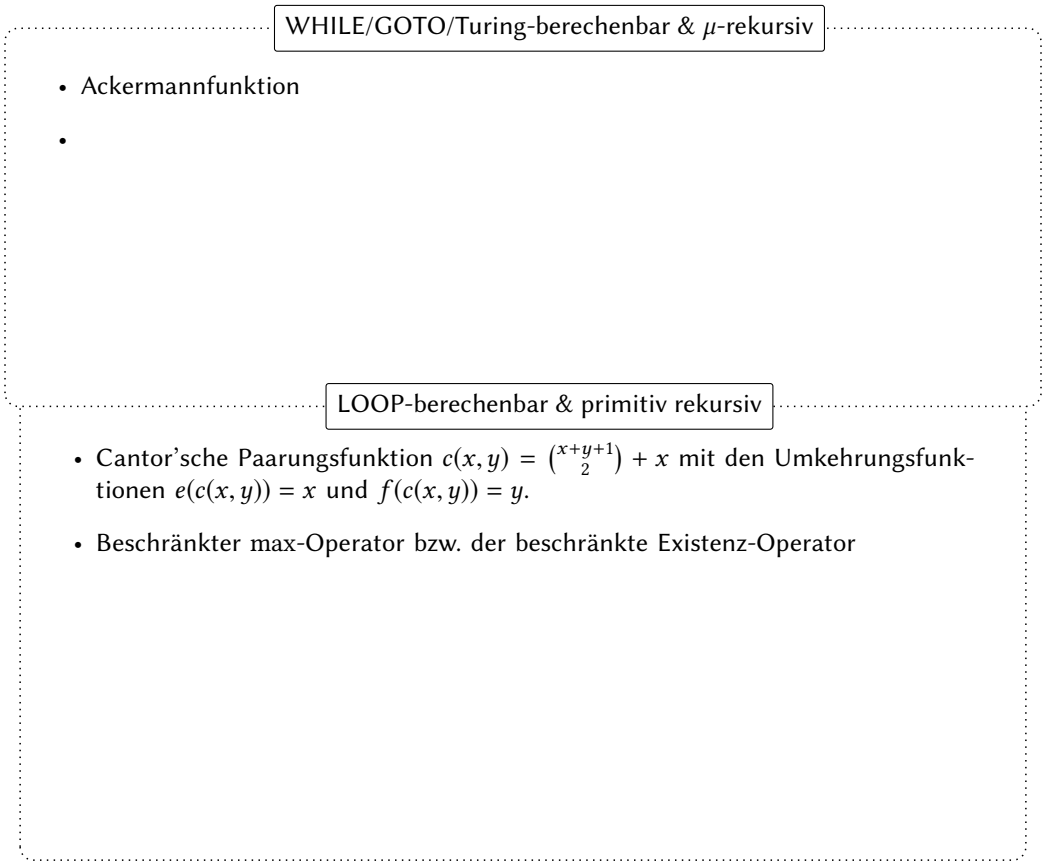
3: Berechenbarkeitstheorie

ACHTUNG: Dieser Abschnitt ist noch nicht abgeschlossen und enthält möglicherweise Fehler!

3.1 Berechenbarkeiten



Einige Funktionen, in den einzelnen Berechenbarkeitsklassen



3.1.1 LOOP-Berechenbarkeit

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt LOOP-berechenbar, falls es ein LOOP-Program P gibt, das gestartet auf der Eingabe $n_1, n_2, \ldots, n_k$ in den Variablen $x_1, x_2, \ldots, x_n$ nach endlich vielen Schritten hält und die Variable x_0 den Wert $f(n_1, \ldots, n_k)$ beinhaltet.

Erlaubte Basisanweisungen

- $x_i := x_j + c$ bzw. $x_i := x_j - c$ mit $c \in \mathbb{N}$
- `LOOP` x_i `DO` P `END`
- Hintereinanderausführung von LOOP-Programmen

Simulierbare Makros

- Wertzuweisungen $x_i := x_j$ und $x_i := c$
- `IF` $x_i > c$ `THEN` P `END`
- Alle üblichen arithmetischen Operationen, auch Modulrechnung

3.1.2 WHILE-Berechenbarkeit

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt WHILE-berechenbar, falls es ein WHILE-Program P gibt, das gestartet auf der Eingabe $n_1, n_2, \ldots, n_k$ in den Variablen $x_1, x_2, \ldots, x_n$ nach endlich vielen Schritten hält (falls das Ergebnis definiert ist) und die Variable x_0 den Wert $f(n_1, \ldots, n_k)$ beinhaltet. Ist $f(n_1, \ldots, n_k)$ undefiniert, so hält P nicht.

1. Jedes GOTO-Programm kann durch ein WHILE-Programm mit nur einer einzigen WHILE-Schleife simuliert werden.
2. Jede WHILE-Berechenbare Funktion kann durch ein WHILE-Programm mit nur einer einzigen WHILE-Schleife berechnet werden (Kleene'sches Normalform-Theorem)

Erlaubte Anweisungen

- Alle Anweisungen von LOOP-Programmen
- `WHILE` $x_i \neq 0$ `DO` P `END`

3.1.3 GOTO-Berechenbarkeit

Erlaubte Anweisungen

- Berechnungen und Zuweisungen: $x_i := x_j \pm c$
- Marken: M_i
- `GOTO` M_i
- `HALT`
- `IF` $x_i = c$ `THEN GOTO` M_j

Simulierbare Makros

- Die `WHILE`-Schleife

3.1.4 Turing-Berechenbarkeit

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt Turing-berechenbar, falls eine deterministische Turingmaschine existiert, die $f(n_1, \ldots, n_k) = m$ berechnet indem, sie gestartet auf dem k-Tupel $(n_1, \ldots, n_k)$ nach endlich vielen Berechnungsschritten einen Endzustand erreicht und dann m auf dem Band steht. Falls das Ergebnis für die Eingabe undefiniert ist, terminiert die Maschine nie.

3.1.5 Primitive Rekursion

Eine Funktion ist genau dann primitiv rekursiv, wenn sie LOOP-berechenbar ist.

- Konstante Funktionen sind primitiv rekursiv
- Projektionen sind primitiv rekursiv
- Die Nachfolgerfunktion $s : \mathbb{N} \to \mathbb{N}, n \mapsto n + 1$ ist primitiv rekursiv
- Verkettungen von primitiv rekursiven Funktionen sind primitiv rekursiv
- Funktionen, die durch primitive Rekursion aus primitiv rekursiven Funktionen entstehen, sind primitiv rekursiv:

$$f(0, x_1, \ldots, x_k) = g(x_1, \ldots, x_k) \qquad \text{(Startwert)}$$
$$f(n+1, x_1, \ldots, x_k) = h(f(n, x_1, \ldots, x_k), n, x_1, \ldots, x_k) \qquad \text{(Rekursionsschritt)}$$

Das heißt, g, h primitiv rekursiv $\Rightarrow f$ primitiv rekursiv.

3.1.6 μ-Rekursion

$$\mu f(x_1, \ldots, x_k) = \min \{n \in \mathbb{N} \mid f(n, x_1, \ldots, x_k) = 0 \wedge \forall m < n : f(m, x_1, \ldots, x_k) > 0\}$$

Der μ-Operator liefert den kleinsten Eingabewert n der ersten Variable, bei der die Funktion 0 ausgibt. Dabei muss insbesondere f für alle Werte kleiner als n definiert sein, da sonst eine Berechnung nicht möglich wäre.

Satz von Kleene: Eine μ-rekursive Funktion $f : \mathbb{N}^n \to \mathbb{N}$ lässt sich immer durch zwei $n+1$-stellige primitiv rekursive Funktionen darstellen:

$$f(x_1, \ldots, x_n) = p(x_1, \ldots, x_n, \mu q(x_1, \ldots, x_n))$$

3.1.7 Arithmetische Repräsentierbarkeit

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt arithmetisch repräsentierbar, falls es eine $(k+1)$ stellige arithmetische Formel F gibt, so dass gilt

$$F(n_1, \ldots, n_k, m) \Leftrightarrow f(n_1, \ldots, n_k) = m$$

Hierfür besteht eine arithmetische Formel zunächst aus Termen:

- Alle $n \in \mathbb{N}$ sind Terme
- Für alle $i \in \mathbb{N}$ ist x_i ein Term
- Verknüpfung zweier Terme mit $+$ oder $*$ sind ebenfalls Terme

Aus den Termen werden Formeln gebildet

- Gleichheit zweier Terme $(t_1 = t_2)$ ist eine Formel
- Für F, G Formeln sind auch $\neg F, (F \wedge G)$ und $(F \vee G)$ Formeln
- Für F eine Formel und $i \in \mathbb{N}$ sind auch $\forall x_i F$ und $\exists x_i F$ Formeln

Gödel'scher Unvollständigkeitssatz Jedes Beweissystem, das nur wahre Formeln der Arithmetik beweist muss unvollständig sein.

4: Entscheidbarkeitstheorie

ACHTUNG: Dieser Abschnitt ist noch nicht abgeschlossen und enthält möglicherweise Fehler!

4.1 Definitionen

Die Berechenbarkeitstheorie spielt sich unterhalb der Entscheidbarkeitstheorie ab. Wenn eine Sprache entscheidbar ist, macht es Sinn ihre Komplexität zu bestimmen. Andersherum heißt das, dass jede Sprache, die in einer der Komplexitätsklassen enthalten ist entscheidbar ist (also insbesondere semi- und co-semi-entscheidbar).

4.1.1 Charakteristische Funktionen

Für jede Menge A existiert die sogenannte *charakteristische Funktion* $\chi_A(w)$. Diese Funktion entscheidet für jedes Wort w aus einer festgelegten Grundmenge, ob $w \in A$ gilt.

$$\chi_A(w) = \begin{cases} 1, \text{ falls } w \in A \\ 0, \text{ sonst} \end{cases}$$

Ebenso lässt sich die semi-charakteristische Funktion $\chi'_A(w)$ definieren

$$\chi'_A(w) = \begin{cases} 1, \text{ falls } w \in A \\ \text{undefiniert, sonst} \end{cases}$$

Eine Menge heißt entscheidbar, wenn ihre zugehörige charakteristische Funktion berechenbar ist. Eine Menge heißt semi-entscheidbar, falls die semi-charakteristische Funktion berechenbar ist.

4.1.2 Rekursive Aufzählbarkeit

Eine Sprache A ist rekursiv aufzählbar, wenn eine totale, berechenbare Funktion $c : \mathbb{N} \to \Sigma^\star$ existiert sodass $A = \{c(n) \mid n \in \mathbb{N}\}$ gilt. Rekursive Aufzählbarkeit ist äquivalent zu semi-Entscheidbarkeit.

4.1.3 Entscheidbarkeitsprobleme

Hierbei seien die Gödelisierungen der Maschinen in einer geeigneten Art und Weise kodiert. Beispielsweise $w \in \{0,1\}^\star$ für eine Maschine M_w.

Spezielles Halteproblem	$K = \{w \mid M_w \text{ hält auf Eingabe } w\}$	semi-entscheidbar
Allgemeines Halteproblem	$H = \{w\#x \mid M_w \text{ hält auf Eingabe } x\}$	semi-entscheidbar
Halteproblem auf leerem Band	$H_0 = \{w \mid M_w \text{ hält auf Eingabe } \epsilon\}$	semi-entscheidbar
PCP	?	semi-entscheidbar
MPCP	Für alle Lösungen gilt $i_1 = 1$	semi-entscheidbar

Satz von Rice Sei $\mathcal{R}$ die Klasse der Turing-berechenbaren Funktionen und $\mathcal{S}$ eine nichttriviale Teilmenge von $\mathcal{R}$. Dann ist die Menge $C(\mathcal{S}) = \{w \mid M_w \text{ berechnet eine Funktion aus } \mathcal{S}\}$ unentscheidbar.

Probleme in der Theorie der formalen Sprachen

- Für deterministisch kontextfreie Sprachen $L, K \in \mathbf{DCFL}$ sind die Fragen $L \cap K = \emptyset$, $|L \cap K| < \infty$, $L \cap K \in \mathbf{CFL}$ sowie $L \subseteq K$ unentscheidbar.
- Für eine kontextfreie Grammatik sind die Frage nach Mehrdeutigkeit, $\overline{L(G)} \in \mathbf{CFL}$, $L(G) \in \mathbf{REG}$ und $L(G) \in \mathbf{DCFL}$ alle unentscheidbar.
- Für eine kontextsensitive-Grammatik ist die Leerheit sowie die Endlichkeit der Sprache unentscheidbar.

4.2 Reduktion von Problemen

Um die prinzipielle Lösbarkeit zweier Probleme zu betrachten, gibt es die sog. many-one-Reduktion: Seien $A \subseteq \Sigma^\star$ und $B \subseteq \Gamma^\star$ zwei Probleme. Kann man eine totale und berechenbare Funktion $f : \Sigma^\star \to \Gamma^\star$ finden, so dass gilt

$$x \in A \Leftrightarrow f(x) \in B,$$

so sagt man, A ist auf B reduzierbar. Man schreibt dann auch $A \leq B$.

1. $A \leq B$ und B entscheidbar $\Rightarrow$ A entscheidbar.
2. $A \leq B$ und B semi-entscheidbar $\Rightarrow$ A semi-entscheidbar.

Bei der Reduktion übertragen sich immer fehlende Eigenschaften von links nach rechts. D.h. reduziert man ein semi-entscheidbares, aber nicht co-semi-entscheidbares Problem A auf ein Problem B ($A \leq B$), so ist B ebenfalls nicht co-semi-entscheidbar. Über die semi-Entscheidbarkeit von B wird keine Aussage gemacht.

5: Komplexität

ACHTUNG: Dieser Abschnitt ist noch nicht abgeschlossen und enthält möglicherweise Fehler!
Die Frage in der Komplexitätstheorie ist, wie viel Aufwand benötigt wird, ein Problem zu lösen. Die theoretische Lösbarkeit wird in der Entscheidbarkeits- bzw. Berechenbarkeitstheorie behandelt.
Die Komplexitätstheorie befasst sich mit oberen und unteren Schranken an den Ressourcenaufwand zur Problemlösung. Wichtig sind hierbei auch inhärente Komplexitäten, also eine Beschränkung nach oben und unten.

5.1 Komplexitätsklassen

Diese Komplexitätsklassen liegen alle in der Menge der entscheidbaren Sprachen. Alle folgenden Klassen sind also Teilmengen von **REC**.
Für alle Komplexitätsklassen C ist zu unterscheiden:

$$\operatorname{co} C = \left\{ L \in \Sigma^\star \,\middle|\, \overline{L} \in C \right\} \qquad \overline{C} = \left\{ L \in \Sigma^\star \,\middle|\, L \notin C \right\}$$

5.1.1 Zeitklassen

Die Funktion $\text{time}_M : \Sigma^\star \to \mathbb{N}$ ordnet einer Eingabe die Anzahl Schritte zu, die eine deterministische Maschine M auf ihr ausführt.
Die Zeitklasse $\text{DTIME}(f(n))$ enthält alle Probleme, die sich durch eine deterministische Turingmaschine in einem Zeitaufwand kleiner als f lösen lassen. Das heißt, f ist eine obere Schranke für die Komplexität der Probleme in $\text{TIME}(f)$.

Die Funktion $\text{ntime}_M : \Sigma^\star \to \mathbb{N}$ ordnet einer Eingabe die Länge des kürzesten akzeptierenden Pfades der nichtdeterministischen Maschine M zu. Falls die Maschine nicht akzeptiert, ist $\text{ntime}_M(w) = 0$.
Die Zeitklasse $\text{NTIME}(f(n))$ enthält alle Probleme, die sich durch eine nichtdeterministische Turingmaschine in einem Zeitaufwand kleiner als f lösen lassen.

$$\mathbf{P} = \bigcup_{k \geq 1} \text{DTIME}(n^k)$$
$$\mathbf{NP} = \bigcup_{k \geq 1} \text{NTIME}(n^k)$$

5.1.2 Platzklassen

Analog zu den Zeitklassen sind Platzklassen definiert, die den maximal belegten Platz für eine Berechnung beschränken. Die Funktion $\text{space}_M : \Sigma^\star \to \mathbb{N}$ ordnet dabei jeder Eingabe auf einer deterministischen Turingmaschine die Anzahl der maximal verwendeten Bandelemente (auf einem Arbeitsband) zu.
So ist die Platzklasse $\text{SPACE}(f(n))$ definiert als die Klasse aller Probleme für die f eine Platzbeschränkung für alle Eingaben ist.
Analog wie bei Zeitklassen die Unterscheidung deterministische-nichtdeterministische Platzklassen.

$$\begin{aligned}\mathbf{L} &= \text{DSPACE}(\log)\\ \mathbf{NL} &= \text{NSPACE}(\log)\\ \mathbf{PSPACE} &= \bigcup_{k\geq 1}\text{DSPACE}(n^k) = \bigcup_{k\geq 1}\text{NSPACE}(n^k)\end{aligned}$$

5.1.3 Wichtige Beziehungen zwischen den Klassen

- In den Platzklassen spielt die O-Notation keine Rolle, Konstanten können wegen Bandreduktion und Bandkompression vernachlässigt werden

  $\text{DSPACE}(O(f)) = \text{DSPACE}(f)$
  $\text{NSPACE}(O(f)) = \text{NSPACE}(f)$

- In nichtdeterministischen Zeitklassen spielt die O-Notation ebenfalls keine Rolle

  $\text{NTIME}(O(f)) = \text{NTIME}(f)$

- Bei deterministischen Zeitklassen gilt i.A. $\text{DTIME}(O(f)) \neq \text{DTIME}(f)$, nur für größer als lineare Funktionen gilt Gleichheit d.h.

  $\text{DTIME}(O(f)) = \text{DTIME}(f) \quad f(n) \geq (1+\epsilon)n$ für ein $\epsilon > 0$
  $\text{DTIME}(f) \subseteq \text{DTIME}(f \log f)$

- Für alle $f(n) \geq n$ gilt für die Zeitklassen

  $\text{DTIME}(f) \subseteq \text{NTIME}(f) \subseteq \text{DSPACE}(f)$

- Und für alle $f(n) \geq \log n$ gilt

  $\text{DSPACE}(f) \subseteq \text{NSPACE}(f) \subseteq \text{DTIME}(2^{O(f)})$

- **Satz von Savitch**: Sei $s \in \Omega(\log(n))$, dann gilt

  $\text{NSPACE}(s) \subseteq \text{DSPACE}(s^2)$

- Sei $s_1 \notin \Omega(s_2)$ und $s_2 \in \Omega(\log(n))$, dann gilt der **Platzhierarchiesatz**

  $\text{DSPACE}(s_2) \setminus \text{DSPACE}(s_1) \neq \emptyset$

- Sei $t_1 \log(t_1) \notin \Omega(t_2)$ und $t_2 \in \Omega(n \log(n))$, dann gilt der **Zeithierarchiesatz**

  $\text{DTIME}(t_2) \setminus \text{DTIME}(t_1) \neq \emptyset$

- **Lückensatz von Borodin**: Für jede totale berechenbare Funktion $r(n) \geq n$ existiert effektiv eine totale berechenbare Funktion $s(n) \geq n+1$ mit

  $\text{DTIME}(s(n)) = \text{DTIME}(r(s(n)))$

- Translationtechnik:
  Die Translationssätze werden verwendet, Separationen von größeren zu kleineren Klassen bzw. Gleichheiten oder Inklusionen von kleineren zu größeren Klassen zu übertragen. Die durch Padding aufgeblähte Sprache ist $Pad_f(L) := \left\{ w\$^{f(|w|)-|w|} \,\middle|\, w \in L \right\}$.

  1. Für zwei Funktionen $f(n), g(n) \geq n$ gilt der **Translationssatz für Zeitklassen**:

     $Pad_f(L) \in \text{DTIME}(O(g)) \Leftrightarrow L \in \text{DTIME}(O(g \circ f))$
     $Pad_f(L) \in \text{NTIME}(O(g)) \Leftrightarrow L \in \text{NTIME}(O(g \circ f))$

  2. Und analog für $g \in \Omega(\log)$ und $f(n) \geq n$ der **Translationssatz für Platzklassen**:

     $Pad_f(L) \in \text{DSPACE}(O(g)) \Leftrightarrow L \in \text{DSPACE}(O(g \circ f))$
     $Pad_f(L) \in \text{NSPACE}(O(g)) \Leftrightarrow L \in \text{NSPACE}(O(g \circ f))$

5.2 Reduktionen

Reduktionen können n icht nur verwendet werden um Entscheidbarkeiten festzustellen, sondern können genauso bei der Einstufung von Problemen in Komplexitätsklassen helfen. So kann man durch beschränken der Reduktionsfunktion beispielsweise eine Lösbarkeit in einer bestimmen Zeit übertragen.
Ähnlich wie in der Berechenbarkeitstheorie kann man zur Analyse der Komplexität von Problemen Reduktionen zwischen diesen entwickeln. Hier ist das Ergebnis allerdings sogar eine Beschränkung des Aufwands nach oben oder unten.
Allerdings ist die many-one-Reduktion zu grob um sinnvolle Reduktionen zwischen Problemen mit Zeitbeschränkung durchzuführen. (Ein Problem, das in Polynomialzeit lösbar ist, könnte durch eine many-one-Reduktionsfunktion gelöst werden. So kann man eigentlich alle berechenbaren Probleme aufeinander many-one-reduzieren, diese Aussage ist jedoch für eine Komplexitätsbetrachtung uninteressant.)
Deshalb gibt es die sog. Polynomialzeitreduktion. Hier ist eine Beschränkung an die Reduktionsfunktion gesetzt, nämlich muss diese zusätzlich in Polynomialzeit berechenbar sein. Man schreibt $A \leq_p B$.

1. $A \leq_p B \wedge B \in \mathbf{P} \Rightarrow A \in \mathbf{P}$
2. $A \leq_p B \wedge B \in \mathbf{NP} \Rightarrow A \in \mathbf{NP}$

5.2.1 Härte und Vollständigkeit

NP-Härte Eine Sprache ist **NP**-hart falls sich alle Probleme aus **NP** in Polynomialzeit darauf reduzieren lassen. Das heißt dieses Problem ist mindestens so schwierig zu lösen wie ein Problem aus **NP**. Es gilt: A **NP**-hart und $A \leq_p B \Rightarrow B$ **NP**-hart.

NP-Vollständigkeit Ist eine Sprache **NP**-hart und selbst in **NP** enthalten, so nennt man sie **NP**-vollständig. Diese Sprachen gehören zu den schwierigsten Sprachen aus **NP**. Kann man von einer **NP**-vollständigen Sprache zeigen, dass sie sogar in **P** liegt, so hat man **P**=**NP** gezeigt.
Einige **NP**-vollständige Probleme:

SAT	$\{w \mid w$ kodiert eine erfüllbare Formel$\}$
3KNF-SAT	Ist eine Formel in KNF mit max. 3 Literalen pro Klausel erfüllbar?
CLIQUE	Enthält ein Graph eine Clique der Größe k?
FÄRBBARKEIT	Gibt es eine Knotenfärbung mit k Farben?

PSPACE-Vollständigkeit **QBF** $= \{F \mid F$ ist eine geschlossene **QBF**-Formel, die sich zum Wert TRUE auswerten lässt.$\}$ ist PSPACE-vollständig.

5.2.2 Logspace-Reduktion

Logspace-Transducer Ein Logspace-Transducer ist eine deterministische Turingmaschine mit einem read-only Eingabeband und einem logarithmisch beschränkten Arbeitsband, sowie einem write-only Ausgabeband auf dem der Schreib-/ Lesekopf nicht nach links bewegt werden kann.
Eine Funktion heißt logspace-berechenbar, wenn ein Logspace-Transducer existiert, der sie berechnet.
Auf Basis dieser Definitionen gibt es nun als verfeinerung der Polynomialzeitreduktion die Logspace-Reduktion:
Seien $A, B \subseteq \Sigma^\star$ Sprachen. A heißt auf B logspace-reduzierbar, falls eine logspace-berechenbare Funktion $f : \Sigma^\star \longrightarrow \Sigma^\star$ existiert, so dass

$$x \in A \Leftrightarrow f(x) \in B$$

Man schreibt dann $A \leq_{\log} B$.
So kann man nun Begriffe wie **NL**- bzw. **P**-Härte/-Vollständigkeit sinnvoll (und analog zur **NP**-Vollständigkeit) definieren.
Einige Probleme, die **NL**-vollständig bezüglich logspace-Reduktion sind:

GAP	Grapherreichbarkeitsproblem
2-SAT	erfüllbare boole'sche Formeln in 2KNF-Darstellung
2-NSAT	nicht erfüllbare Formeln in 2KNF-Darstellung

Einige Probleme, die **P**-vollständig bezüglich logspace-Reduktion sind:

CFE	$L_{cfe} = \{w \mid w$ ist Kodierung einer Typ-2 Grammatik G mit $L(G) = \varnothing\}$
CVP	Circuit Value Problem (Ausgabe des Schaltnetzes)
MCVP	Wie **CVP**, allerdings ohne Negationen