Satz 1.1 (Äquivalenz von k- und 1-String TMs)
Zu jeder k-String-TM M mit Zeitschranke T gibt es eine 1-String-TM M' mit Zeitschranke $\mathcal{O}(T^2)$ und $L(M') = L(M)$

Klasse 1.2 (TIME und TIME_k)
Sei $f : \mathbb{N} \to \mathbb{R}$

$\text{TIME}_k(f)$:= *Menge aller $L \subseteq \Sigma^*$, für die es eine k-String TM M gibt mit*

1. $L = L(M)$
2. *M hat eine Zeitschranke T mit $T = \mathcal{O}(f)$*

$\text{TIME}(f) := \bigcup_{k \geq 1} \text{TIME}_k(f)$

Klasse 1.3 (P und EXPTIME)
$\text{P} := \bigcup_{c \geq 1} \text{TIME}(n^c)$

$\text{EXPTIME} := \bigcup_{c \geq 1} \text{TIME}(2^{n^c})$

Klasse 1.4 (NTIME und NTIME_k)
Sei $f : \mathbb{N} \to \mathbb{R}$

$\text{NTIME}_k(f)$:= *Menge aller $L \subseteq \Sigma^*$, für die es eine k-String TM M mit Zusatzeingabe gibt mit*

1. $L = L(M)$
2. *M hat eine Zeitschranke T mit $T = \mathcal{O}(f)$*

$\text{NTIME}(f) := \bigcup_{k \geq 1} \text{NTIME}_k(f)$

Klasse 1.5 (NP)
$\text{NP} := \bigcup_{c \geq 1} \text{NTIME}(n^c)$

Satz 3.7 (APTIME = PSPACE)
$\text{APTIME} = \text{PSPACE}$

Satz 3.9 (ATIME zu SPACE Beziehung)
Sei S eine platzkonstruierbare Funktion mit $S(n) \geq n$. Dann gilt:

$$\text{ATIME}(S) \subseteq \text{SPACE}(S) \subseteq \text{ATIME}(S^2)$$

Satz 4.4 (APSPACE = EXPTIME)
[Chandra 81]
$\text{APSPACE} = \text{EXPTIME}$

Satz 4.5 (APSPACE vs TIME)
[Chandra 81]

1. *Ist $S(n) = \Omega(\log n)$, so gilt*

$$\text{ASPACE}(S) \subseteq \text{TIME}(2^{\mathcal{O}(S)})$$

2. *Ist $T(n) = \Omega(n)$, so gilt*

$$\text{TIME}(S) \subseteq \text{ASPACE}(\log T)$$

Satz 4.6 (P = ALOGSPACE)
$\text{P} = \text{ALOGSPACE}$

Satz 4.7 (Komplementierung)

- $\text{co} - \text{ATIME}(f) = \text{ATIME}(f)$
- $\text{co} - \text{ASPACE}(f) = \text{ASPACE}(f)$
- $\text{co} - \text{SPACE}(f) = \text{SPACE}(f)$
- $\text{co} - \text{TIME}(f) = \text{TIME}(f)$

Satz 5.4 (Zeithierachie-Satz)
[Hartmanis,Starns 65]
Sei f, g zeitkonstruierbar, und

- $f(n) = \Omega(n)$
- $g(n) = \omega(f(n) \log f(n)$

so gilt $\text{TIME}(f) \subsetneq \text{TIME}(g)$

Satz 5.6 ($\text{NTIME}(n^r) \subsetneq \text{NTIME}(n^s)$)
[Cook 73]
Falls $1 \leq r < s$ (reelle Zahlen), so gilt: $\text{NTIME}(n^r) \subsetneq \text{NTIME}(n^s)$

Satz 5.7 (Platzhierarchiesätze)
[Stearns et al. 65, Szepietowski 94]
Seien f, g platzkonstruierbar,

- $f(n) = \Omega(\log n)$,
- $g(n) = \omega(f(n))$

Dann gelten:

- $\text{SPACE}(f) \subsetneq \text{SPACE}(g)$
- $\text{NSPACE}(f) \subsetneq \text{NSPACE}(g)$

Satz 5.8 (Lückensatz)
[Borodin 71, Trakhtenbrot 64]
Es gibt eine Funktion $T : \mathbb{N} \mapsto \mathbb{R}$ mit $\text{TIME}(T) = \text{TIME}(2^T)$

Klasse 6.4 (LOGSPACE, NL,$POLYLOGSPACE$)
$\mathbf{LOGSPACE} := \mathbf{SPACE}(\log n)$

$\mathbf{NL} := \mathbf{NSPACE}(\log n)$

$\mathbf{POLYLOGSPACE} := \bigcup_{c\geq 1} \mathbf{SPACE}((\log n)^c)$

Satz 6.5 (NSPACEvsSPACE)
[Savitch 70]
Ist S *platzkonstruierbar, und* $S(n) \geq \log n$*, so gilt:* $\mathbf{NSPACE}(S) \subseteq \mathbf{SPACE}(S^2)$

Satz 6.6 (NPSPACE = PSPACE)
$\mathbf{NPSPACE} = \mathbf{PSPACE}$

Satz 6.7 (NPSPACE = co − NSPACE)
Für jedes platzkonstruierbare S *mit* $S(n) \geq \log n$ *gilt:* $\mathbf{NPSPACE}(S) = \mathbf{co-NSPACE}(S)$

Satz 6.10 (TIME(T) $\subseteq$ SPACE($T/\log T$))
[Hopcroft et al. 75]
Falls T *zeitkonstruierbar ist, so gilt* $\mathbf{TIME}(T) \subseteq \mathbf{SPACE}(T/\log T)$

Satz 6.11 (TIME(T) $\subsetneq$ SPACE(T))
[Hopcroft et al. 75]
Falls T *zeitkonstruierbar ist, so gilt* $\mathbf{TIME}(T) \subsetneq \mathbf{SPACE}(T)$

Satz 6.12 (TIME(n) $\subsetneq$ NTIME(n))
[Paul et al. 83]
$\mathbf{TIME}(n) \subsetneq \mathbf{NTIME}(n)$

Definition 7.0 (Syntax von RAMs)
*Eine Random Access Machine (**RAM**)* $P = (p_1, \ldots, p_m)$ *ist eine endliche Folge von **Anweisungen** p_i, die jeweils von einer der folgenden Formen sind:*

LOAD i	*LOADN*	*HALF*
LOAD $R(i)$	*STORE* $R(i)$	*JUMP* j
LOAD $R(R(i))$	*STORE* $R(R(i))$	*JPOS* j
LOAD $I(i)$	*ADD* $R(i)$	*HALT*
LOAD $I(R(i))$	*SUB* $R(i)$	

Dabei ist $i \in \mathbb{N}$ *eine und* $j \in \mathbb{N}^{\leq m}$

Fakt 1
Der Zeitaufwand von Register- Programmen und TMs unterscheidet sich nur polynomiell.

Definition 7.1 (Konfigurationen von RAMs)
- *Die **Eingabe** einer RAM ist ein Tupel* $\vec{x} = (x_1, \ldots, x_n)$ *von natürlichen Zahlen.*
- *Eine **Konfiguration** einer RAM* $P(p_1, \ldots, p_m)$ *ist ein Paar* (z, f)*, wobei*
  - $z \leq m$*, (**Zeilennummer**) und*
  - $f : \mathbb{N}_0 \to \mathbb{N}_0$ *(**Registerbelegung**). Dabei:* $f(n) \neq 0$ *für endlich viele* n
- *Die **Startkonfiguration** ist* $(1, \hat{0})$*, wobei für alle* $n, \hat{0}(n) = 0$
- *Die Konfiguration* (z', f') *ist **Nachfolgekonfiguration** von* (z, f) *bei Eingabe* $\vec{x}$ *(Schreibweise* $(z, f) \overset{P,\vec{x}}{\to} (z', f')$*) falls einer der in der folgenden Tabelle aufgeführten Fälle gilt:*
- (z, f) *ist **Haltekonfiguration**, falls* $p_z =$*HALT*

Definition 7.2 (Nachfolgekonfiguration in RAMs)

LOAD i	$f'(0) = i$	$z+1$
LOAD $R(i)$	$f'(0) = f(i)$	$z+1$
LOAD $R(R(i))$	$f'(0) = f(f(i))$	$z+1$
LOAD $I(i)$	$f'(0) = x_i$	$z+1$
LOAD $I(R(i))$	$f'(0) = x_{f(i)}$	$z+1$
LOADN	$f'(0) = n$	$z+1$
STORE $R(i)$	$f'(i) = f(0)$	$z+1$
STORE $R(R(i))$	$f'(f(i)) = f(0)$	$z+1$
ADD $R(i)$	$f'(0) = f(0) + f(i)$	$z+1$
SUB $R(i)$	$f'(0) = \max\{0, f(0) - f(i)\}$	$z+1$
HALF	$f'(0) = \lfloor \frac{f(0)}{2} \rfloor$	$z+1$
JUMP j	$f'(i) = f(i)$, $\forall i$	j
JPOS j	$f'(i) = f(i)$, $\forall i$	j, $f(0) > 0$ $z+1$, *sonst*

Definition 7.3 (Semantik und Aufwand von RAMs)
- P ***akzeptiert*** $\vec{x}$*, falls*

$$(1, \hat{0}) \overset{P,\vec{x}}{\to} (z_1, f_1) \overset{P,\vec{x}}{\to} \ldots \overset{P,\vec{x}}{\to} (z_t, f_t)$$

  sowie
  - (z_t, f_t) *Haltekonfiguration ist und*
  - $f_t(0) > 0$
- *Falls* $f_t(0) = 0$*,* ***lehnt*** P ***ab***
- ***Zeitaufwand:*** $t_P(\vec{x}) \overset{def}{=} t$
- ***Platzaufwand:***

$$s_P(\vec{x}) \overset{def}{=} \sum_{i=1}^{\infty} \max_{j \in \{1,\ldots t\}} \lceil \log(f_j(i) + 1) \rceil$$

Definition 7.4 (PRAMs)
Eine $\boldsymbol{PRAM}$ *ist ein Algorithmus (eine TM)* $\boldsymbol{P}$ *, der bei Eingabe* $\mathbf{1^m 01^n}$ *mit logarithmischem Speicherplatz eine Folge* $\boldsymbol{\Pi(m,n) = \pi_1, \ldots, \pi_q}$ *von Register-Programmen berechnet.*

- *Dabei*
  - $\boldsymbol{m}$*: Anzahl der Eingabezahlen*
  - $\boldsymbol{n}$*: Gesamtlänge der Eingabezahlen (*$\boldsymbol{m \leq n}$*)*
  - *Anm: Meistens hängt* $\boldsymbol{\Pi(m,n}$ *nur von* $\boldsymbol{m}$ *ab*
- *Jedes Programm* $\boldsymbol{\pi_j}$ *kann außer den globalen Registern* $\boldsymbol{R_0, R_1, \ldots}$ *lokale Register* $\boldsymbol{L_0^j, L_1^j, \ldots}$ *verwenden.*
- *Die erlaubten Befehle sind wie bei RAMs.* $\boldsymbol{L_0^j}$ *dient als Akkumulator für* $\boldsymbol{\pi_j}$

Definition 7.5 (Semantik von PRAMs)
- $\boldsymbol{Konfiguration}$*: Tripel* $\boldsymbol{(\vec{z}, f, g)}$*, wobei gilt:*
  - $\boldsymbol{\vec{z} = (z_1, \ldots, z_q)}$ *gibt für jedes Programm die aktuelle Zeilennummer an.*
  - $\boldsymbol{f : \mathbb{N}_0 \to \mathbb{N}_0}$*, mit* $\boldsymbol{f(i) =}$ *Inhalt von* $\boldsymbol{R_i}$
  - $\boldsymbol{g : \{1, \ldots, q\} \times \mathbb{N}_0 \to \mathbb{N}_0}$*, mit* $\boldsymbol{g(i,j) =}$ *Inhalt von* $\boldsymbol{L_i^j}$
  - *jeweils nur endlich viele* $\boldsymbol{f(i)}$ *und* $\boldsymbol{g(i,j)}$ *sind von* $\mathbf{0}$ *verschieden.*
- $\boldsymbol{Eingabe}$*: Tupel* $\boldsymbol{\vec{x} = (x_1, \ldots, x_m) \in \mathbb{N}_0^*}$
- $\boldsymbol{Startkonfiguration}$*:* $\boldsymbol{K_0(\vec{x}) = (\vec{z}, \hat{0}, \hat{0}')}$ *mit*
  - *für alle* $\boldsymbol{i}$*:* $\boldsymbol{z_i = 1}$
  - *für alle* $\boldsymbol{i,j}$*:* $\boldsymbol{\hat{0}'(i,j) = 0}$
- $\boldsymbol{Nachfolgekonfiguration}$*: analog zur Definition von Nachfolgekonfigurationen für RAMs*
  - *mehrere Programme können vom selben (Eingabe- oder Arbeits-) Register lesen*
  - *wenn mehrere Programme ins selbe Register zu schreiben versuchen, so wird der Wert des Programms mit der kleinsten Nummer genommen*
- $\boldsymbol{Haltekonfiguration}$*: Konfiguration, in der mindestens ein Programm anhält*
- $\boldsymbol{\Pi}$ $\boldsymbol{akzeptiert}$*, falls das minimale Programm, das angehalten hat, akzeptiert*
- $\boldsymbol{Rechenzeit}$ $\boldsymbol{t_P(\vec{x})}$*: Anzahl der ausgeführten Schritte. (*$\boldsymbol{t_p(\vec{x})}$ *hängt nicht von der Größe der Eingabezahlen ab*
- *Anmerkung: PRAMs lassen sich auf realistischen Parallelrechnern mit relativ geringem Zeitverlust simulieren.*

Definition 7.6 (Schaltkreis)
- *Sei* $\boldsymbol{X}$ *eine Menge von Variablen*
- *Ein* $\boldsymbol{Schaltkreis}$ $\boldsymbol{C = (G, m, v_0)}$ *über* $\boldsymbol{X}$ *besteht aus*
  - *einem gerichteten, azyklischen Graphen* $\boldsymbol{G = (V, E)}$
  - *einer Kontenmarkierung* $\boldsymbol{m : V \to \{0, 1, \vee, \wedge, \neg\} \cup X}$*,*
  - *und einem ausgezeichneten Knoten* $\boldsymbol{v_0 \in V}$
- *Wir nennen die Knoten eines Schaltkreises auch* $\boldsymbol{Gatter}$ *(*$\boldsymbol{v_0}$ *heißt* $\boldsymbol{Ausgabegatter}$*)*
- *Ist* $\boldsymbol{m(v) \in \{0,1\} \cup X}$*, so hat* $\boldsymbol{v}$ *Eingrad 0*
- *Ist* $\boldsymbol{m(v) = \neg}$*, so hat* $\boldsymbol{v}$ *Eingrad 1*
- *Gatter* $\boldsymbol{v}$ *mit* $\boldsymbol{m(v) \in X}$ *heißen* $\boldsymbol{Eingabegatter}$
- *Eine* $\boldsymbol{Belegung}$ *für einen Schaltkreis* $\boldsymbol{C}$ *über* $\boldsymbol{X}$ *ist eine Funktion* $\boldsymbol{\alpha : X \to \{0,1\}}$
- *Der Wert* $\boldsymbol{w_\alpha(g)}$ *eines Gatters* $\boldsymbol{g}$ *unter der Belegung* $\boldsymbol{\alpha}$ *ist* $\mathbf{1}$*, falls*
  - $\boldsymbol{m(g) = 1}$*,*
  - $\boldsymbol{m(g) = x}$ *und* $\boldsymbol{\alpha(x) = 1}$*,*
  - $\boldsymbol{m(g) = \neg}$*,* $\boldsymbol{(g', g) \in E}$ *und* $\boldsymbol{w_\alpha(g') = 0}$*,*
  - $\boldsymbol{m(g) = \wedge}$ *und für alle* $\boldsymbol{g' \in V}$ *mit* $\boldsymbol{(g', g) \in E}$ *gilt* $\boldsymbol{w_\alpha(g') = 1}$*, oder*
  - $\boldsymbol{m(g) = \vee}$ *und es gibt* $\boldsymbol{g' \in V}$ *mit* $\boldsymbol{(g', g) \in E}$ *gilt* $\boldsymbol{w_\alpha(g') = 1}$*, oder*
- *Andernfalls ist* $\boldsymbol{w_\alpha(g) = 0}$
- $\boldsymbol{w_\alpha(C) \stackrel{def}{=} w_\alpha(v_0)}$
- *Die* $\boldsymbol{Größe}$ $\boldsymbol{s(C)}$ *von* $\boldsymbol{C}$ *ist die Anzahl der Knoten von* $\boldsymbol{G}$
- *Die* $\boldsymbol{Tiefe}$ $\boldsymbol{d(C)}$ *ist die maximale Länge eines (gerichteten) Pfades in* $\boldsymbol{G}$

Definition 7.7 (Schaltkreisfamilie)
Wir betrachten $\boldsymbol{Familien}$ $\boldsymbol{\mathcal{C} = (C_n)_{n \geq 1}}$ *von Schaltkreisen mit einem Schaltkreis* $\boldsymbol{C_n}$ *für jede Eingabelänge* $\boldsymbol{n}$

- *Konvention:* $\boldsymbol{C_n}$ *hat Variablen* $\boldsymbol{x_1, \ldots, x_n}$
- *Jeder Schaltkreis Cn definiert dann in natürlicher Weise eine Funktion* $\boldsymbol{f_n : \{0,1\}^n \to \{0,1\}}$
- *Außerdem definiert jeder Schaltkreis Cn in natürlicher Weise eine Menge* $\boldsymbol{L(C_n)}$ *von 0-1-Strings der Länge* $\boldsymbol{n}$
- $\boldsymbol{L(\mathcal{C}) : \bigcup_{n \geq 0} L(C_n)}$

Definition 7.8 (uniforme Schaltkreisfamilie)
$(C_n)_{n\geq 1}$ *heißt* ***uniform****, wenn die Funktion* $1^n \to C_n$ *mit logarithmischer Speicherplatz-Schranke berechenbar ist.*

Satz 7.9 (Schaltkreis $\subseteq$ PRAM)
Wird eine Funktion von einer uniformen Schaltkreis-Familie $\mathcal{C} = (C_n)_{n\geq 1}$ *mit Größe* $f(n)$ *und Tiefe* $g(n)$ *berechnet, so auch von einer PRAM mit* $\mathcal{O}(f(n))$ *Prozessoren und Zeitschranke* $\mathcal{O}(g(n))$

Satz 7.10 (Schaltkreis $\supseteq$ PRAM)
Seien h_1 *und* h_2 *log-platz-berechenbar und polynomiell beschränkt und es gelte* $h_1(n) \geq \log n$*. Wird eine Funktion* F *von einer PRAM* P *mit Zeitschranke* $h1(n)$ *und* $h2(m)$ *Prozessoren berechnet (*m*: Anzahl Eingabe-Zahlen,* n*: Eingabelänge), so auch von einer Schaltkreis-familie* $\mathcal{C}$ *mit*

- *Eingrad 2,*
- *Tiefe* $\mathcal{O}(h_1(n)\log n)$*, und*
- *polynomieller Größe in* $h_1(n) + h_2(m)$

Klasse 7.11 ($\mathbf{NC^k}$ und NC)
Für $k > 0$ *sei* $\mathbf{NC^k}$ *die Klasse aller Sprachen, die von einer uniformen Familie von Schaltkreisen polynomieller Größe mit Gattern vom Eingrad* ≤ 2 *und Tiefe* $\mathcal{O}((\log n)^k)$ *entschieden werden.*

$$\mathbf{NC} := \bigcup_{k>0} \mathbf{NC^k}$$

Satz 7.12 ($\mathbf{NC} \subseteq \mathbf{P}$)
$\mathbf{NC} \subseteq \mathbf{P}$

Satz 7.13 (CVP ist vollständig für P)
Sei CVP *das Problem, den Wahrheitswert eines gegebenen Schaltkreises ohne Variablen zu berechnen. Dann ist* CVP *vollständig für* $\mathbf{P}$

Satz 8.1 (NC vs LOGSPACE vs PLOYLOGSPACE)
(a) $\mathbf{NC^1} \subseteq \mathbf{LOGSPACE}$

(b) $\mathbf{NL} \subseteq \mathbf{NC^2}$

(c) $\mathbf{NC} \subseteq \mathbf{POLYLOGSPACE}$

Definition 8.2 ($\mathbf{AC^k}$ und AC)
Für $k \geq 0$ *sei* $\mathbf{AC^k}$ *die Klasse aller Sprachen, die von einer uniformen Familie von Schaltkreisen polynomieller Größe mit unbeschränkten Gattern und Tiefe* $\mathcal{O}((\log n)^k)$ *entschieden werden.*

$$\mathbf{AC} := \bigcup_{k} \mathbf{AC^k}$$

Satz 8.3
(a) $\mathbf{AC^0} \subseteq \mathbf{NC^1} \subseteq \mathbf{AC^1} \subseteq \mathbf{NC^2} \ldots$

(b) $\mathbf{AC} = \mathbf{NC}$

Satz 8.4 ($\mathbf{AC^0} \subsetneq \mathbf{NC^1}$)
[Ajtai 83; Furst, Saxe, Sipser 83]

(a) $\textsc{Parity} \notin \mathbf{AC^0}$

(b) $\textsc{Parity} \in \mathbf{NC^1}$*, daher* $\mathbf{AC^0} \subsetneq \mathbf{NC^1}$

Definition 8.5 (LOGCFL)
LOGCFL *sei die Klasse aller Sprachen* L*, für die es eine kontextfreie* L' *gibt, so dass* $L \leq_l L'$

Satz 8.6 (LOGCFL und TMmk(log,poly))
Für eine Sprache L *sind äquivalent:*

1. $L \in \mathbf{LOGCFL}$
2. $L = L(M)$ *für eine TM* M *mit logarithmischer Platzschranke, einem zusätzlichen Hilfskeller und nichtdeterministischem Akzeptiermechanismus.*

Satz 8.7 ($\mathbf{NL} \subseteq \mathbf{LOGCFL}$)
$\mathbf{NL} \subseteq \mathbf{LOGCFL}$

Definition 8.8 (ASTR($\mathcal{F}, \mathcal{G}$))
Für Funktionsklassen $\mathcal{F}$ *und* $\mathcal{G}$ *sei* $ASTR(\mathcal{F}, \mathcal{G})$ *die Klasse aller Sprachen, die von einer ZRO-TM* M *mit Platzschranke aus* $\mathcal{F}$ *entschieden werden, und für die es eine Funktion* $g \in \mathcal{G}$ *gibt, so dass für jedes* $x \in L(M)$ *für Spieler* $\exists$ *eine Gewinnstrategie hat, deren Größe (Anzahl der Knoten im Strategiebaum) durch* $g(|x|)$ *beschränkt ist*

Satz 8.9 (LOGCFL $\subseteq$ ASTR(log ,poly))
$\mathbf{LOGCFL} \subseteq \mathbf{ASTR}(log\ ,poly)$

Definition 8.10 (SAC)
Für $k > 0$ *sei* $\mathbf{SAC^k}$ *die Klasse aller Sprachen, die von einer uniformen Familie von Schaltkreisen polynomieller Größe mit Und-Gattern von Eingrad 2 und beliebigen Oder-Gattern und Tiefe* $\mathcal{O}((\log n)^k)$ *entschieden werden.*

Satz 8.11 (ASTR(log ,poly)$\subseteq$ SAC^1)
ASTR$(log\ ,poly) \subseteq$ **SAC**^{1}

Satz 8.12 (NL bis NC^2)
NL $\subseteq$ **LOGCFL** $\subseteq$ **SAC**^{1} $\subseteq$ **AC**^{1} $\subseteq$ **NC**^{2}

Satz 8.13 (LOGCFL = SAC^1)
LOGCFL = **SAC**^{1}

Definition 8.14 (TC^k und TC)
TC^k := *Menge aller Sprachen, die von uniformen Schaltkreisfamilien der Tiefe* $\mathcal{O}((\log n)^k)$ *mit* $\vee-, \wedge-$, *und Majority-Gattern erkannt werden.*

$$\mathbf{TC} := \bigcup_k \mathbf{TC}^k$$

Satz 8.15 (Mult, ItAdd, ItMult)
MULT, ITADD, ITMULT *sind vollständig für* **TC**^{0} *unter* **AC**^{0}*-berechenbaren Reduktionen.*

Satz 8.16 (AC^0 bis NC^1)
AC^0 $\subsetneq$ **TC**^{0} $\subseteq$ **NC**^{1}

Satz 8.17 (Beschreibungskomplexität für AC^0)
Eine Sprache L *ist genau dann in* $\mathbf{FO}[+,\times]$*, wenn sie in uniformem* **AC**^{0} *ist.*

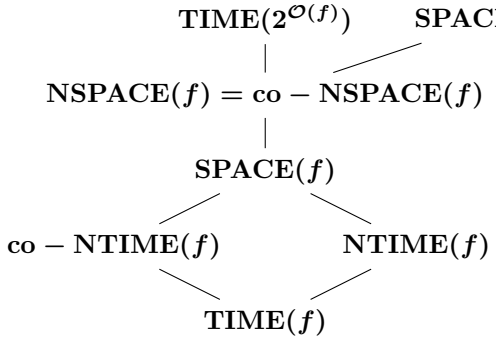
Definition 8.18 (stark-uniform)
Ob ein gegebenes Gatter mit einem gegebenen anderen Gatter verbunden ist, lässt sich in poly-logarithmischer Zeit entscheiden (in einem erweitertem TM-Modell)

Satz 8.19 ()
Es gibt keine Folge $C_1, C_2, \ldots$ *von Schaltkreisen, so dass gilt:*

(a) Die Tiefe von C_n *ist* $\mathcal{O}(1)$

(b) Die Größe von C_n *ist polynomiell, und*

(c) Jeder Schaltkreis C_n *akzeptiert die Strings der Länge* n *mit ungerade vielen 1en*



Problem 1 (Min-VertexCover)
Gegeben: *Graph* $G = (V, E)$

Gesucht: *Menge* $U \subseteq V$*, so dass gelten:*

(1) $(u, v) \in E \Rightarrow u \in U$ *oder* $v \in U$
(2) $|U|$ *minimal ?*

Problem 2 (VertexCover)
Gegeben: *Graph* $G = (V, E)$

Frage: *Gibt es* $U \subseteq V$*, so dass gelten:*

(1) $(u, v) \in E \Rightarrow u \in U$ *oder* $v \in U$
(2) $|U| \leq k$*?*

Satz 8.1 (VertexCover ist NP-schwierig)
VertexCover *ist* **NP***-schwierig. Für festes* k *ist* VertexCover *in* $\mathcal{O}(n^k)$ *lösbar.*

Definition 8.2 (Parametrisiertes Problem)
Ein ***parametrisiertes Problem*** *ist eine Menge* L *von Paaren* (x, k)*, wobei* $x \in \Sigma^*$ *und* $k \in \mathbb{N}$*.*
$L_k := \{(x, i) \in L \mid i = k\}$ *"*k*-te Scheibe" von* L

Problem 3 (P-VertexCover)
Gegeben: *Graph* $G = (V, E)$, k

Parameter: k

Frage: *Gibt es* $U \subseteq V$*, so dass gelten:*

(1) $(u, v) \in E \Rightarrow u \in U$ *oder* $v \in U$
(2) $|U| \leq k$*?*

Definition 8.3 (FPT)
Ein parametrisiertes Problem L *heißt* ***fp-effizient*** *("fixed parameter tractable"), falls es einen Algorithmus* $\mathcal{A}$*, ein Polynom* p *und ein berechenbare Funktion* $f : \mathbb{N} \to \mathbb{N}$ *gibt, do dass gilt:*

(a) $\mathcal{A}$ *entscheidet* L

(b) $\mathcal{A}$ *hat bei Eingabe* (x, k) *Laufzeit* $f(k)p(|x|)$

FPT *ist die Klasse aller fp-effizient parametrisierten Problem*

Lemma 8.4 (FPT mit additiver Laufzeitschranke)
Ist $L \in$ **FPT***, so gibt es ein berechenbares* f' *und ein Polynom* p'*, so dass* $(x, k) \in L$ *in Zeit* $f'(k) + p'(|x|)$ *getestet werden kann.*

Problem 4 (P-HittingSet)
Geben: *Hypergraph* $H = (V, E), k$

Parameter: k

Frage: *Gibt es* $U \subseteq V$*, so dass gelten:*

(1) $e \in E \Rightarrow U \cap e \neq \emptyset$
(2) $|U| \leq k$

Problem 5 (P-DominatingSet)
Geben: *Graph* $G = (V, E), k$

Parameter: k

Frage: *Gibt es* $U \subseteq V$*, so dass gelten:*

(1) für jedes $v \in V - U$ *gibt es ein* $u \in U$ *mit* $(u, v) \in E$
(2) $|U| \leq k$*?*

Problem 6 (P-Clique)
Geben: *Graph* $G = (V, E), k$

Parameter: k

Frage: *Gibt es* $U \subseteq V$*, so dass gelten:*

(1) $u, v \in U \Rightarrow (u, v) \in E$ *oder* $u = v$
(2) $|U| \geq k$*?*

Problem 7 (P-IndependentSet)
Geben: *Graph* $G = (V, E), k$

Parameter: k

Frage: *Gibt es* $U \subseteq V$*, so dass gelten:*

(1) $u, v \in U \Rightarrow (u, v) \notin E$
(2) $|U| \geq k$*?*

Problem 8 (P-Max-Sat)
Geben: k *und KNF-Formel* $\varphi = C_1 \wedge \ldots \wedge C_m$

Parameter: k

Frage: *Gibt es eine Wahrheitsbelgeung, mindestens* k *Klauseln wahr macht?*

Definition 8.5 (kern-reduzierbar)
Ein parametrisirtes Problem L *ist* ***kern-reduzierbar****, wenn es berechenbare Funktionen* f_S, f_K *und eine in polynomieller Zeit berechenbare Funktion* $(x,k) \mapsto (x',k')$ *gibt mit:*

- $|x'| \leq f_S(k)$,
- $k' \leq f_K(k)$, *und*
- $(x,k) \in L \Leftrightarrow (x',k') \in L$

Satz 8.6 (FPT und kern-redzierbarkeit)
Ein entscheidbares parametrisiertes Problem L *ist in* **FPT** *genau dann, wenn es kernreduzierbar ist.*

Satz 8.7 (P-Max-Sat ist in FPT)
P-Max-Sat *ist kernreduzierbar mit* $f_S(k) = 2k^2$ *und* $f_K(k) = k$

Definition 18.1 (Algorithmus für VertexCover)
(1) Falls $|V| \leq 3$ *oder* $k \leq 1$*, löse das Prolem direkt.*

(2) Falls möglich, wähle v *von Grad* 1 *und mache weiter mit* $G - N(v)$.

(3) Falls mögliche, wähle v *vom Grad* 2 *mit verbundenen Nachbarn und mache weiter mit* $G - N(v)$

(4) Falls möglich, wähle v *vom Grad* 2 *mit unverbundenen Nachbarn und verzweige zu* $G - N(N(v))$ *und* $G - N(v)$.

(5) Andernfalls wähle v *vom Grad* ≥ 3 *und verzweige zu* $G - \{v\}$ *und* $G - N(v)$

Satz 18.2
Algorithmus 18.1 ist korrekt. Die Anzahl der Blätter im Suchbaum von Alorithmus 18.1 ist bei Eingaben mit Parameter k *beschränkt durch* $1,47^k$

Satz 18.3
Es gibt einen Algorithmus führt zu einem Suchbaum der Größe $\leq 1,33^k$

Definition 18.4 (Baumzerlegung)
Sei $G = (V,E)$ *ein Graph.*

Eine ***Baumzerlegung*** (T,X) *besteht aus einem Baum* T *mit Knotenmenge* I *und einer Menge* $X(i) \subseteq V$ *für jedes* $i \in I$*, so das gelten:*

(i) $V = \bigcup_{i \in I} X(i)$

(ii) Ist $(u,v) \in E$*, so gibt es ein* $i \in I$ *mit* $\{u,v\} \subseteq X(i)$

(iii) Für jedes $v \in V$ *ist die Menge* $\{i \mid v \in X(i)\}$ *zusammenhängend in* T

Die ***Weite*** $w(T,X) := \max_{i \in I} |X(i)| - 1$
Die ***Baumweite*** $tw(G)$ *von* G *ist die minimale Weite einer Baumzerlegung für* G
Mit $\mathbf{BTW}_k$ *bezeichnen wir die Klasse aller Strukturen mit Baumweite* $\leq k$

Satz 18.5 ()
(a) Die Menge $\{(G,k) \mid tw(G) \leq k\}$ *ist* **NP**-*vollständig*

(b) Für jedes feste k*, ist die Menge aller* G *mit* $tw(G) \leq k$ *in linearer Zeit erkennbar*

(c) Für jedes k *gibt es einem Algorithmus, der zu jedem Graphen* $G \in \mathbf{BTW}_k$ *eine Baumzerlgeung der Weiter* k *in linearer Zeit konstruiert*

Satz 18.6 ()
Für Graphen G *mit gegebener Baumzerlegung* (T,X) *der Weite* k *lässt sich in Zeit* $\mathcal{O}(2^k k|I|)$ *ein minimales Vertex-Cover berechnen.*