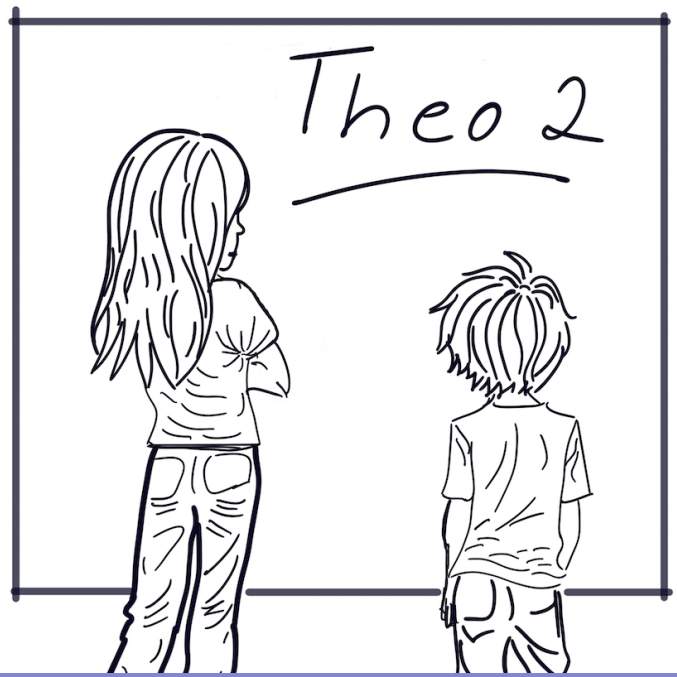
Theoretische Informatik 2

SoSe 2023

Berechenbarkeit und Komplexität

Prof. Dr. Sebastian Siebertz
AG Theoretische Informatik
MZH, Raum 3160
siebertz@uni-bremen.de

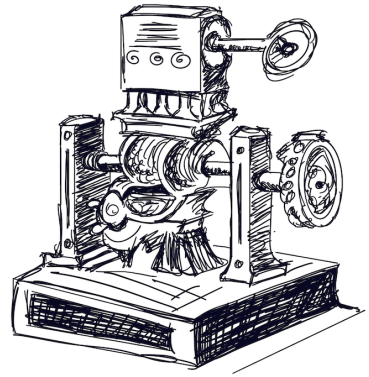
Universität Bremen



Berechenbarkeit und Komplexität – Motivation

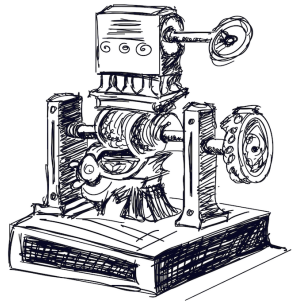
- Berechenbarkeit: Was kann ein Computer berechnen?
- Beispiele:
  - Eingabe: Zahl $n \in \mathbb{N}$
    Ausgabe: Der Wert 2^n
  - Berechnung:
    - falls $n = 1$: output 2
    - falls $n > 1$: berechne das Ergebnis für $n - 1$ und multipliziere es mit 2.
  - Eingabe: Ein mathematischer Satz S
    Ausgabe: 1 falls S ein gültiger Satz ist, 0 sonst.
  - Berechnung?
    - Probiere alle Beweise zu generieren ...
    - Terminiert diese Berechnung?

Berechenbarkeit und Komplexität – Motivation



- Berechenbarkeit: Was kann ein Computer berechnen?
- Die Antwort einer C++ Programmiererin: Jede Funktion, für die ein C++ Programm geschrieben werden kann!
- Sind für verschiedene Programmiersprachen oder für verschiedene Rechnerarchitekturen verschiedene Funktionen berechenbar?
- Wir brauchen formale Modelle der Berechenbarkeit!
- Dann können wir untersuchen
  - welche Funktionen in welchem Modell berechenbar sind,
  - ob ein Modell mächtiger ist als ein anderes, oder
  - ob es einen universellen Begriff der Berechenbarkeit gibt.

Berechenbarkeit und Komplexität – Motivation



- Wir möchten formale Modelle zur Verfügung stellen, die
  - von den spezifischen Gegebenheiten abstrahieren,
  - nur die wesentlichen Aspekte der Berechenbarkeit abbilden und
  - dabei so einfach und elegant sind wie möglich.

- Die Essenz einer Berechnung:
  - eine Eingabe liegt in einem Speicher;
  - es wird eine feste endliche Folge von Befehlen aus einem festen endlichen Befehlssatz ausgeführt;
  - jeder Befehl manipuliert den Speicher nach festen Regeln;
  - das Ergebnis der Berechnung wird aus dem Speicher abgelesen.

Berechenbarkeit und Komplexität – Motivation

- Seit den 1930er Jahren versuchen Mathematiker*innen und Informatiker*innen das Konzept der Berechenbarkeit zu formalisieren.
- Verschiedenste Berechnungsmodelle:
  - Turing Maschinen (Alan Turing),
  - Post Systeme (Emil Post),
  - μ-rekursive Funktionen (Kurt Gödel und Jaques Herbrand),
  - λ-Kalkül (Alonzo Church und Stephen C. Kleene),
  - ...
  - Jede hinreichend starke Programmiersprache.

Berechenbarkeit und Komplexität – Motivation

- Turing Maschine:
  - Unglaublich einfaches Modell für die endliche Kontrolle eines Speichers.
  - Trotzdem, oder vielleicht gerade deshalb, ist intuitiv sofort klar, dass jedes endliche, deterministische System zur Speichermanipulation simuliert werden kann!
  - Alle oben genannten Berechnungsmodelle sind gleich mächtig.

Church-Turing-These

Die Klasse der Turing-berechenbaren Funktionen stimmt mit der Klasse der intuitiv berechenbaren Funktionen überein.

- Umgekehrt heißt jedes System, das alle Turing-berechenbaren Funktionen berechnen kann turingvollständig.

Berechenbarkeit und Komplexität – Motivation

- In der Vorlesung betrachtete Berechnungsmodelle:
  - Turingmaschinen als intuitives und einfaches Berechnungsmodell,
  - WHILE-Programme als Abstraktion imperativer Programmiersprachen,
- Außerdem: der Zusammenhang zu Grammatiken

Berechenbarkeit und Komplexität – Motivation

- Mit diesem formalen Berechenbarkeitsbegriff können wir beweisen, dass es nicht berechenbare Funkionen gibt!
- Es handelt sich nicht um absurd konstruierte Probleme, sondern um äußerst wichtige Probleme in der Informatik und Mathematik.
  - Beispiel: Terminiert ein gegebenes Programm auf einer Eingabe?
  - Wäre extrem hilfreich z.B. im Compilerbau...

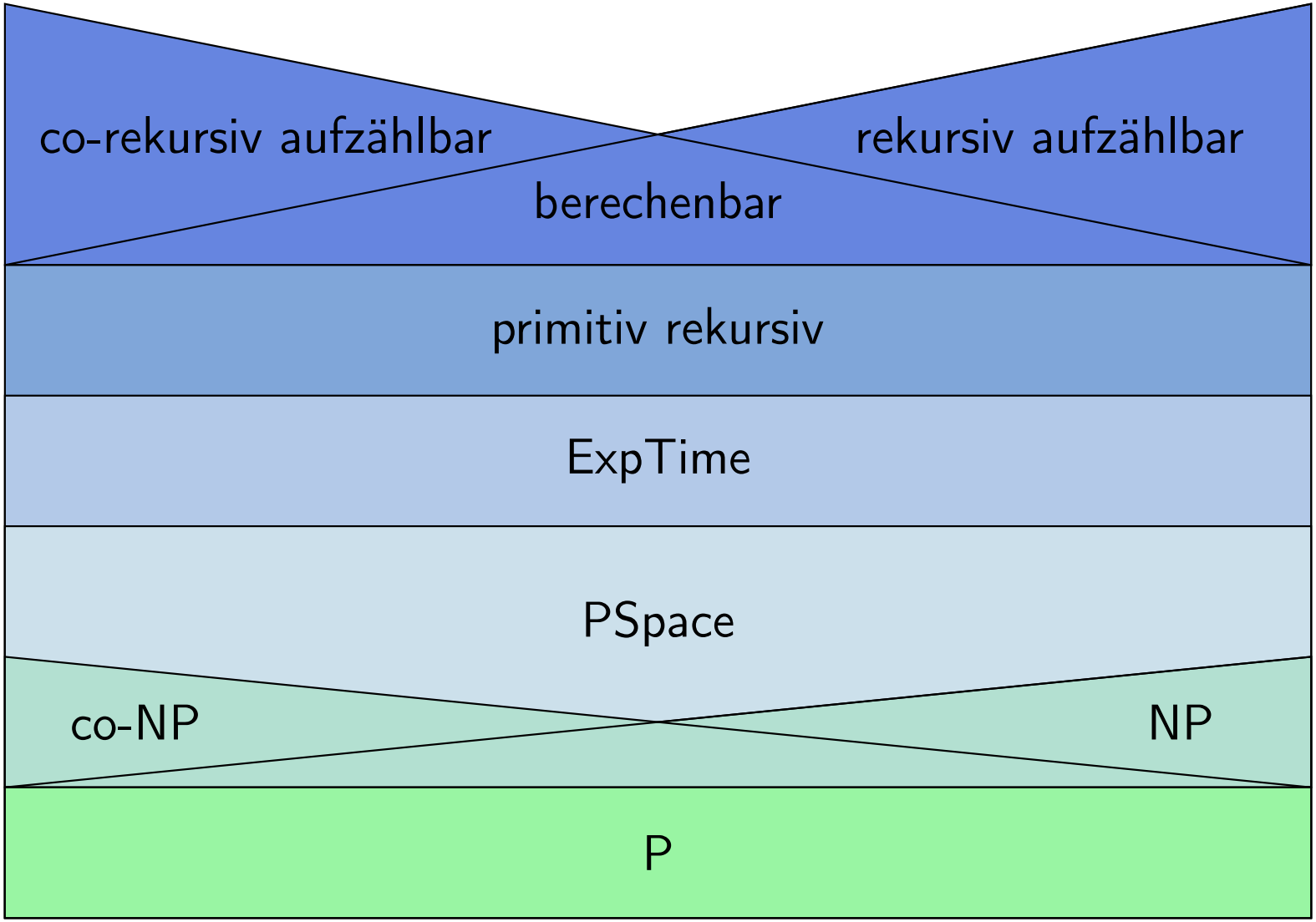
Berechenbarkeit und Komplexität – Motivation

- Komplexität: Welche Ressourcen (Zeit oder Speicher) brauchen wir, um eine Funktion zu berechnen?
- Theoretisch effizient berechenbare Funktionen:
  - die Klasse PTime, kurz P, der in Polynomialzeit (bzgl. Eingabegröße) berechenbaren Funktionen.
  - Das stimmt nicht mit den praktisch berechenbaren Funktionen überein! Probleme mit Laufzeit $\mathcal{O}(n^{100})$ sind für große Eingaben praktisch unlösbar...
  - ist aber eine robuste, vom Maschinenmodell unabhängige Klasse.
- Für viele wichtige Probleme kennen wir keinen Polynomialzeitalgorithmus, können aber auch nicht beweisen, dass es keinen gibt.

Berechenbarkeit und Komplexität – Motivation

- Wir definieren für diese Probleme weitere Komplexitätsklassen.
- Eine wichtige Klasse: NPTime, kurz NP, der Probleme, die von einer nichtdeterministischen Turingmaschine in Polynomialzeit gelöst werden können.
  - Wir gehen davon aus, dass die schwersten Probleme in NP nicht effzient gelöst werden können.
- Das Problem ob P $\neq$ NP gehört zu den wichtigsten offenen Problemen der Informatik.
- Wir identifizieren die schwersten Probleme in einer Klasse mittels Reduktionen.
  - Wenn wir ein NP-schweres Problem auf ein Problem P reduzieren können, nehmen wir an, dass auch P nicht effizient lösbar ist.

Berechenbarkeit und Komplexität – Motivation



Los geht's! Berechnungen

- Die Essenz einer Berechnung:
  - eine Eingabe liegt in einem Speicher;
  - es wird eine feste endliche Folge von Befehlen aus einem festen endlichen Befehlssatz ausgeführt;
  - jeder Befehl manipuliert den Speicher nach festen Regeln;
  - das Ergebnis der Berechnung wird aus dem Speicher abgelesen.
- Was sind unsere Eingaben und Ausgaben?
- Wie modellieren wir einen Speicher?
- Wie modellieren wir die Manipulation/Kontrolle des Speichers?

Eingabe und Ausgabe

- Alle (sinnvollen) Ein- und Ausgaben können als Wörter über Alphabeten kodiert werden.
  - Z.B. kann jede Zahl $n \in \mathbb{N}$ in Binärdarstellung als Wort kodiert werden.
  - Dagegen kann nicht jede Zahl $x \in \mathbb{R}$ kodiert werden, sondern muss möglicherweise angenähert werden.

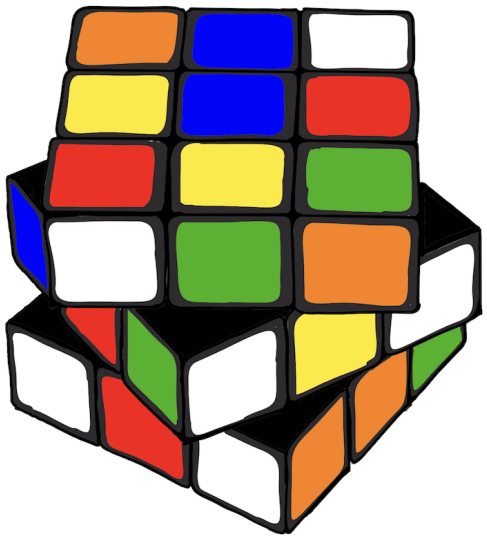
Alphabet

- Ein Alphabet ist eine endliche, nichtleere Menge von Symbolen.
- Wir benutzen griechische Großbuchstaben für Alphabete: $\Sigma, \Gamma, \ldots$
- und meistens römische Kleinbuchstaben für Symbole: $a, b, c, \ldots$
- Beispiele:
  - $\Sigma = \{a, b, c, \ldots, z\}$
  - $\Gamma = \{0, 1\}$

Wörter

- $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$ und $\mathbb{N} = \mathbb{N}_0 \setminus \{0\}$.
- $[n] = \{1, \ldots, n\}$ für $n \in \mathbb{N}$
- $[0] = \varnothing$.
- Ein Wort über einem Alphabet Σ ist eine endliche Folge $w = w_1 \cdots w_n$ von Symbolen aus Σ.
  - Formal: eine endliche Folge ist eine Abbildung $w : [n] \to \Sigma : i \mapsto w_i$ für ein $n \in \mathbb{N}_0$. Die Zahl n bezeichnet die Länge der Folge.
  - Die Folge mit Definitionsbereich $[0] = \varnothing$ heißt leere Folge oder leeres Wort. Es wird mit ε bezeichnet.

Probleme und Entscheidungsprobleme



- Σ^* ist die Menge aller Wörter über Σ.
- $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$.
- Seien Σ und Γ Alphabete.
- Ein Problem mit Eingaben aus Σ^* und Ausgaben aus Γ^* ist eine Funktion

$$f : \Sigma^* \to \Gamma^*.$$

- Ein Entscheidungsproblem über Σ^* ist eine Teilmenge $L \subseteq \Sigma^*$.
  - Spezialfall einer Funktion f, die nur zwei verschiedene Werte in der Ausgabe hat, z.B. 0 und 1.
  - Dann können wir $f : \Sigma^* \to \{0,1\}$ mit $L = \{w \in \Sigma^* \ : \ f(w) = 1\} \subseteq \Sigma^*$ identifizieren.

Probleme und Entscheidungsprobleme

- Beispiele:
  - Sei $\Sigma = \{0, 1\}$ und $\mathrm{bin}(n)$ für $n \in \mathbb{N}_0$ die Binärdarstellung von n über Σ. Dann ist
    $$f: \Sigma^* \to \Sigma^* : \mathrm{bin}(n) \mapsto \mathrm{bin}(2^n)$$
    ein Problem mit Eingaben und Ausgaben aus Σ^*.
  - Sei $\Sigma = \{0, 1\}$ und $\mathrm{code}(\cdot)$ eine Kodierungsfunktion, die endliche Graphen eindeutig auf Bitstrings abbildet. Dann ist
    $$\{\mathrm{code}(G) \ : \ G \text{ zusammenhängender Graph}\}$$
    ein Entscheidungsproblem über Σ^*.
- Übung: Sei Σ ein Alphabet. Gib eine Bijektion $f: \Sigma^* \to \mathbb{N}$ an.
  - Wir können also natürliche Zahlen über beliebigen Alphabeten kodieren.

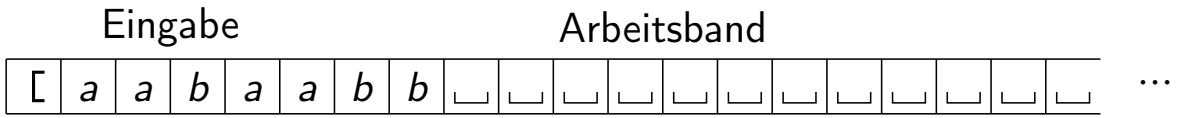
Turingmaschinen

- Was sind unsere Eingaben und Ausgaben? ✓
- Wie modellieren wir einen Speicher?
- Wie modellieren wir die Manipulation/Kontrolle des Speichers?

Turingmaschinen

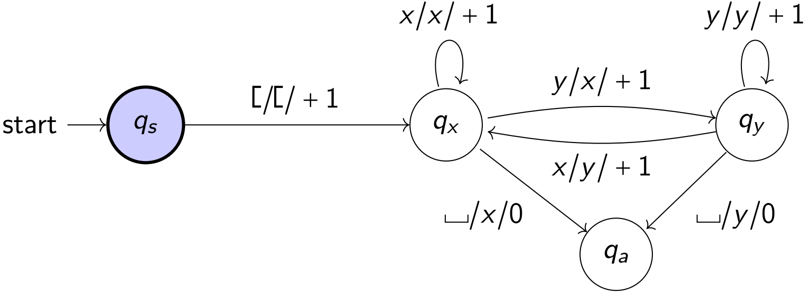
Turingmaschinen

- Der Speicher: unendliches Band mit Speicherzellen $B_1, B_2, B_3, \ldots$
  - Zelle B_1 ist mit [markiert.
  - Die Eingabe w befindet sich in den Zellen $B_2, \ldots, B_{|w|+1}$
  - Die Zellen $B_{|w|+2}, \ldots$ sind mit Blanks $\sqcup$ gefüllt (unbeschrieben).



- Die Kontrolle: endlich viele Zustände mit einer Übergangsfunktion.
- Die Manipulation des Speichers: Lese- und Schreibkopf, der sich auf dem Band bewegen kann.

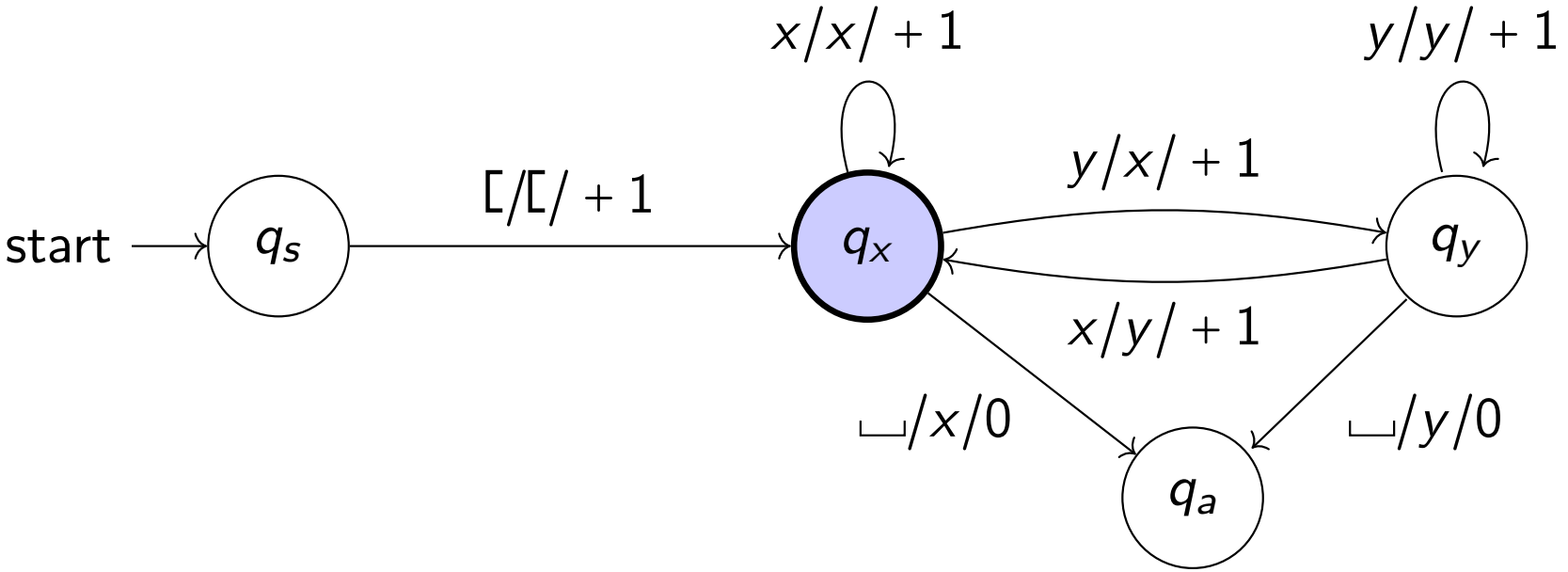
Beispiel Turingmaschine

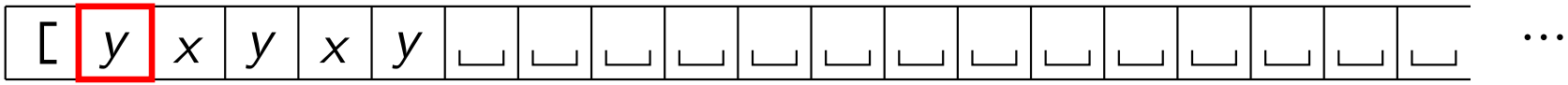


[	y	x	y	x	y	␣	␣	␣	␣	␣	␣	␣	␣	␣	␣	␣	␣	␣	␣	…

- Die Maschine soll ein x an die erste Position schreiben und das Eingabewort dafür eine Position nach rechts schieben.

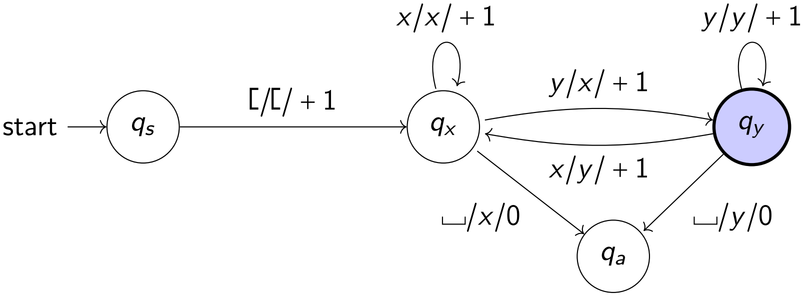
Beispiel Turingmaschine

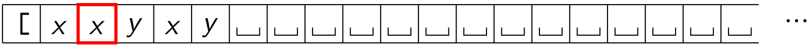




- Die Maschine soll ein x an die erste Position schreiben und das Eingabewort dafür eine Position nach rechts schieben.

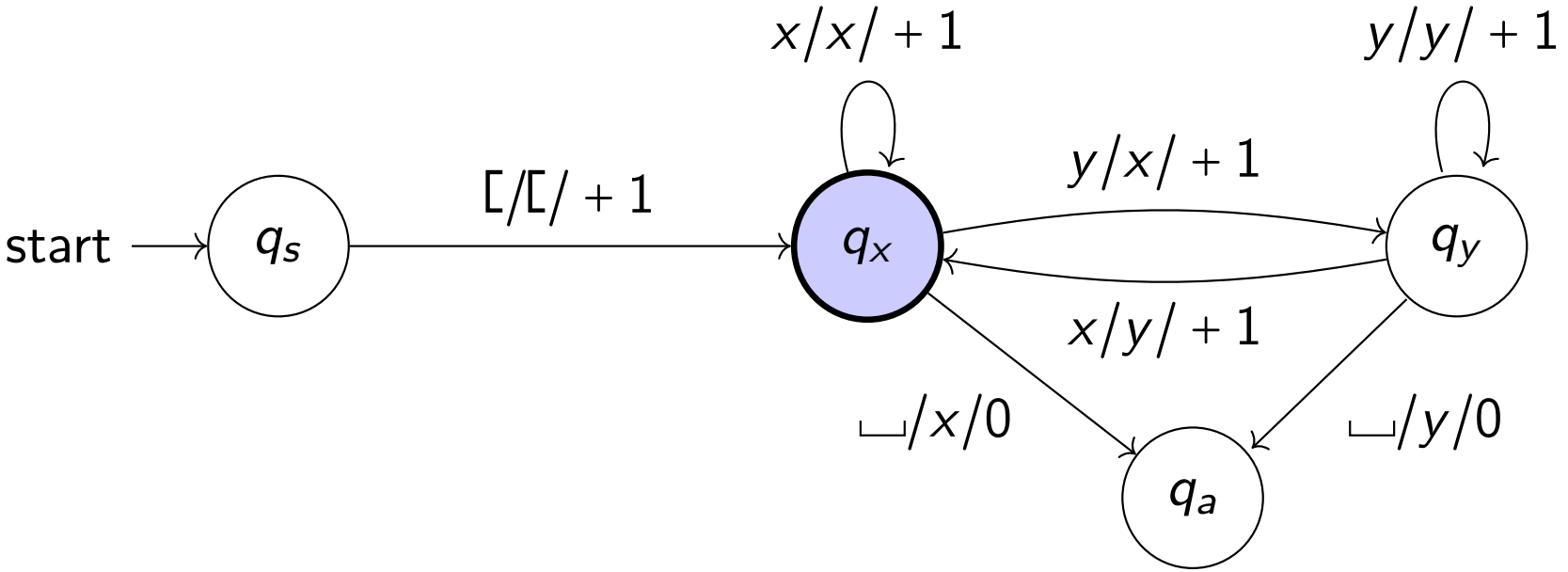
Beispiel Turingmaschine

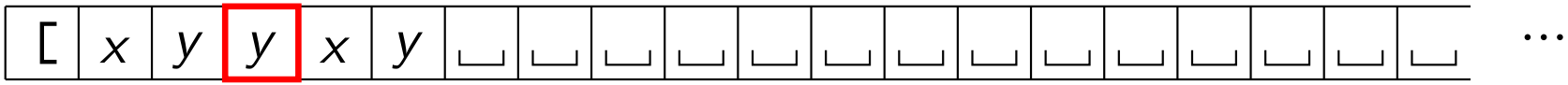




- Die Maschine soll ein x an die erste Position schreiben und das Eingabewort dafür eine Position nach rechts schieben.

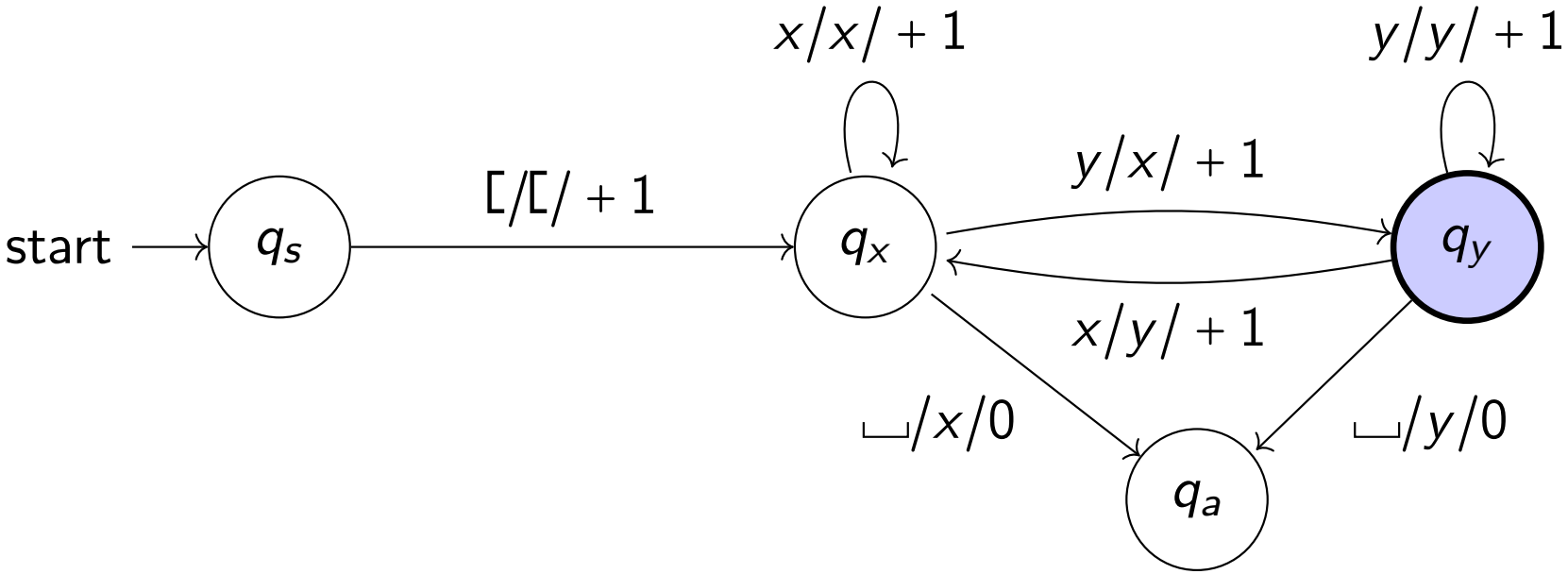
Beispiel Turingmaschine

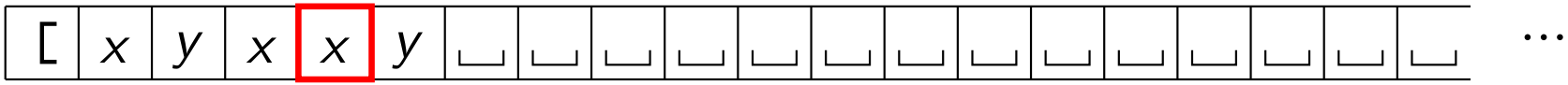




- Die Maschine soll ein x an die erste Position schreiben und das Eingabewort dafür eine Position nach rechts schieben.

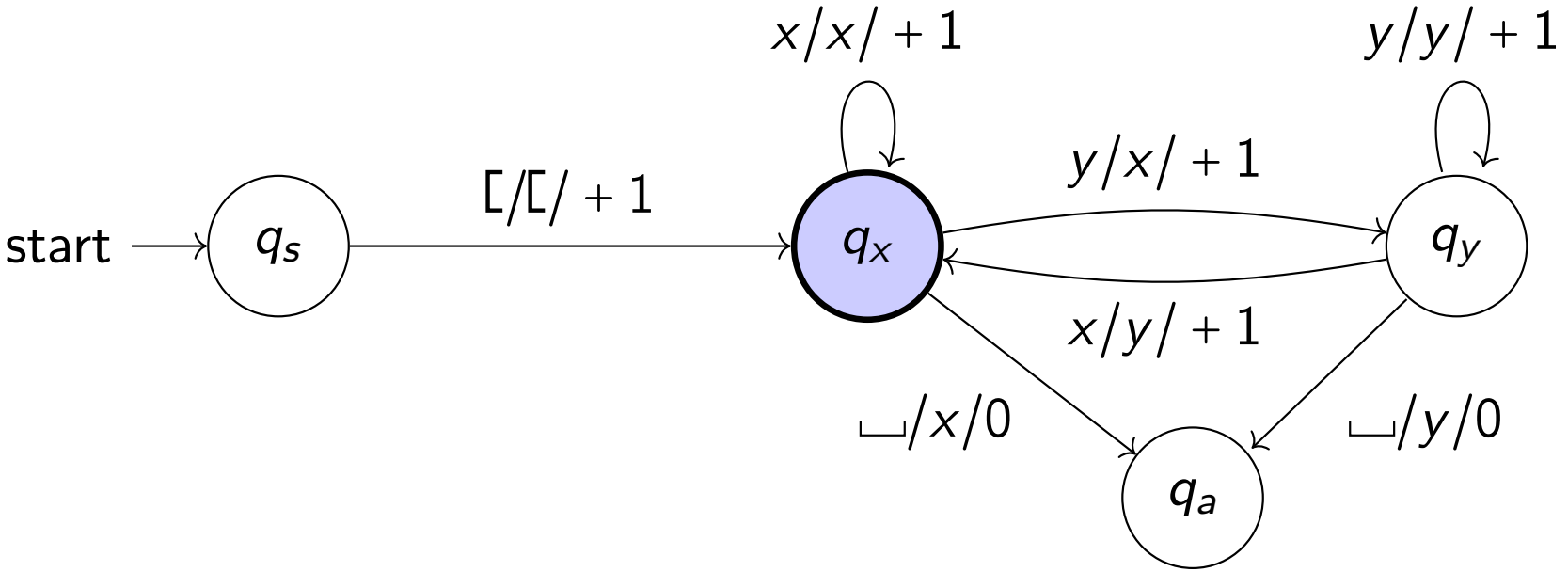
Beispiel Turingmaschine

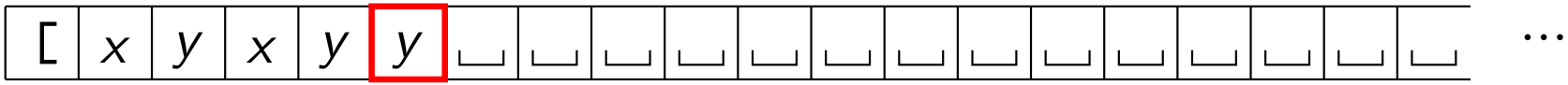




- Die Maschine soll ein x an die erste Position schreiben und das Eingabewort dafür eine Position nach rechts schieben.

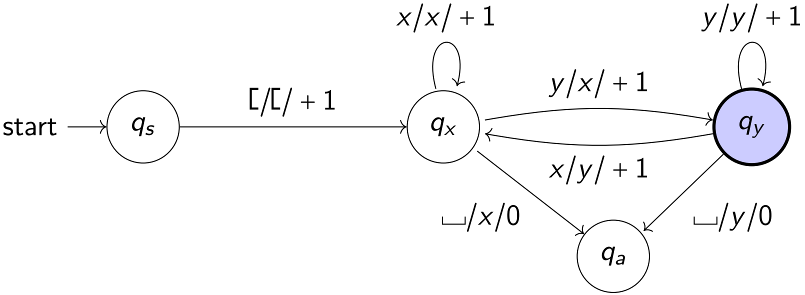
Beispiel Turingmaschine

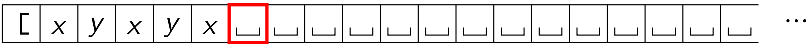




- Die Maschine soll ein x an die erste Position schreiben und das Eingabewort dafür eine Position nach rechts schieben.

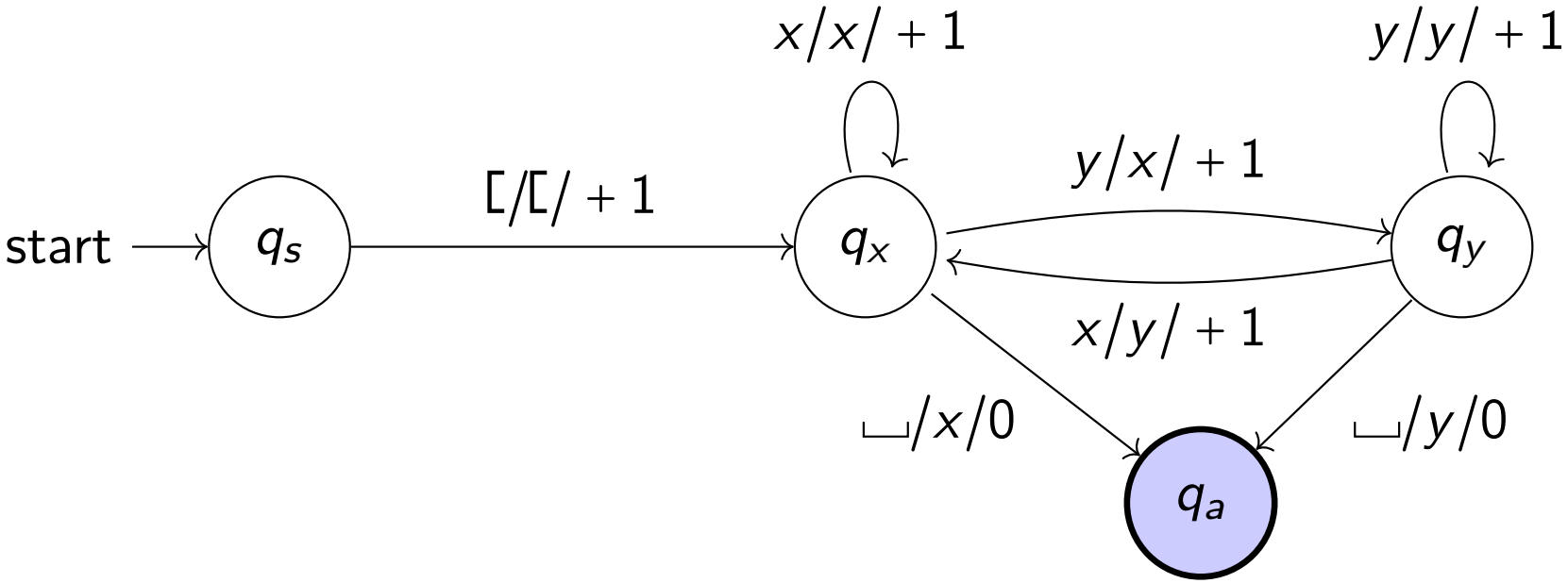
Beispiel Turingmaschine

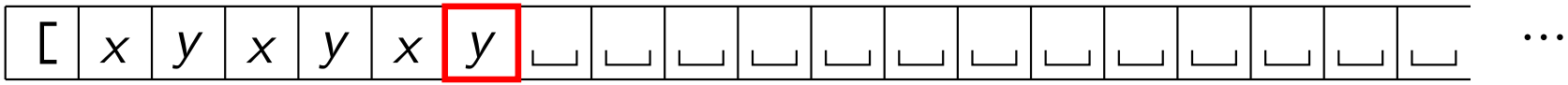




- Die Maschine soll ein x an die erste Position schreiben und das Eingabewort dafür eine Position nach rechts schieben.

Beispiel Turingmaschine





- Die Maschine soll ein x an die erste Position schreiben und das Eingabewort dafür eine Position nach rechts schieben.

Turingmaschinen formal definiert

- Eine deterministische Einbandturingmaschine ist ein Tupel $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, q_s, q_a, q_r)$, wobei
  - Q endliche Zustandsmenge ist,
  - Σ das Eingabealphabet ist,
  - Γ das Arbeitsalphabet ist mit $\Sigma \subseteq \Gamma$,
  - $\vdash \in \Gamma \setminus \Sigma$ der linke Endmarker ist,
  - $\sqcup \in \Gamma \setminus \Sigma$ das Blanksymbol ist,
  - $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{-1, 0, +1\}$ die Übergangsfunktion ist, wobei gilt, dass
    (1) für alle $p \in Q$ ein $q \in Q$ und $d \in \{0, +1\}$ existieren, so dass $\delta(p, \vdash) = (q, \vdash, d)$ und
    (2) für alle $a \in \Gamma$ gilt $\delta(q_a, a) = (q_a, a, 0)$ und $\delta(q_r, a) = (q_r, a, 0)$,
  - $q_s \in Q$ der Anfangszustand ist,
  - $q_a \in Q$ der akzeptierende Zustand ist und
  - $q_r \in Q$ der verwerfende Zustand ist, $q_a \neq q_r$.

Turingmaschinen

- $\delta(p, a) = (q, b, d)$ für $p, q \in Q, a, b \in \Gamma$ und $d \in \{-1, 0, +1\}$ bedeutet, dass die Maschine,
  - wenn sie im Zustand p
  - an der aktuellen Kopfposition k
  - das Symbol a liest,
  - so wechselt sie in den Zustand q,
  - schreibt das Symbol b an der Kopfposition auf das Band und
  - bewegt den Kopf auf Position $k + d$.
    - $d = -1 \rightarrow$ nach links
    - $d = 0 \rightarrow$ stehenbleiben
    - $d = +1 \rightarrow$ nach rechts

Turingmaschinen

- Für die Übergangsfunktion $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{-1, 0, +1\}$ gilt
  (1) für alle $p \in Q$ ein $q \in Q$ und $d \in \{0, +1\}$ existieren, so dass $\delta(p, \vdash) = (q, \vdash, d)$ und
  (2) für alle $a \in \Gamma$ gilt $\delta(q_a, a) = (q_a, a, 0)$ und $\delta(q_r, a) = (q_r, a, 0)$,
- Die Bedingungen stellen sicher, dass
  (1) die Maschine niemals über den linken Rand des Arbeitsbandes läuft und das Symbol $\vdash$ niemals entfernt, und dass sie
  (2) falls sie den akzeptierenden Zustand q_a oder verwerfenden Zustand q_r erreicht, ihre Berechnung beendet indem sie im gleichen Zustand bleibt, das Bandsymbol nicht mehr ändert und den Kopf nicht mehr bewegt.

Einschub: unendliche Wörter

- Ein unendliches Wort über einem Alphabet Σ ist eine Abbildung

$$w\colon \mathbb{N} \to \Sigma.$$

- Vergleiche: Ein endliches Wort ist eine Abbildung

$$w\colon [n] \to \Sigma.$$

- Wir schreiben kurz $w = w_1 w_2 w_3 \cdots$ für das unendliche Wort $w\colon \mathbb{N} \to \Sigma\colon i \mapsto w_i$.
- Mit $\Sigma^{\mathbb{N}}$ (oder auch Σ^{ω}) bezeichnen wir die Menge aller unendlichen Wörter über einem Alphabet Σ.

Konfigurationen und Berechnungen

- Eine Konfiguration beschreibt vollständig den Zustand einer Turingmaschine.
- Eine Konfiguration von M ist ein Tupel

$$\alpha = (q, w, k) \in Q \times \Gamma^{\mathbb{N}} \times \mathbb{N},$$

mit $w(1) = \text{[}$ und so dass eine Position $n \in \mathbb{N}$ existiert mit $w(i) = \sqcup$ für alle $i \geq n$.

- Die Konfiguration (q, w, k), wobei $w = v \sqcup \sqcup \sqcup \ldots$ für ein $v \in \Gamma^*$ bedeutet, dass die Maschine sich in Zustand q befindet, das Band mit

$$v \sqcup \sqcup \sqcup \ldots$$

beschriftet ist, und dass der Lese- und Schreibkopf sich an Position k auf dem Arbeitsband befindet.

Ersetzung eines Symbols in einem unendlichen Wort

- Sei Γ ein Alphabet.
- Für ein (unendliches) Wort w über Γ, $k \in \mathbb{N}$ und $a \in \Gamma$ definiere

$$w[k \mapsto a] : i \mapsto \begin{cases} a & \text{falls } i = k \\ w(i) & \text{sonst.} \end{cases}$$

- Das Wort $w[k \mapsto a]$ ist also das Wort, das wir aus w erhalten, indem wir das Symbol an der Stelle k durch das Symbol a ersetzen.

Folgekonfiguration

- Sei $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, q_s, q_a, q_r)$ eine Turingmaschine und sei

$$\alpha = (q, w, k) \in Q \times \Gamma^{\mathbb{N}} \times \mathbb{N}$$

  eine Konfiguration.

- Sei

$$\delta(q, w_k) = (p, b, d) \in Q \times \Gamma \times \{-1, 0, +1\}.$$

- Dann ist die Folgekonfiguration von α die Konfiguration

$$\beta = (p, w[k \mapsto b], k + d)$$

- Wir schreiben $\alpha \vdash_M \beta$.

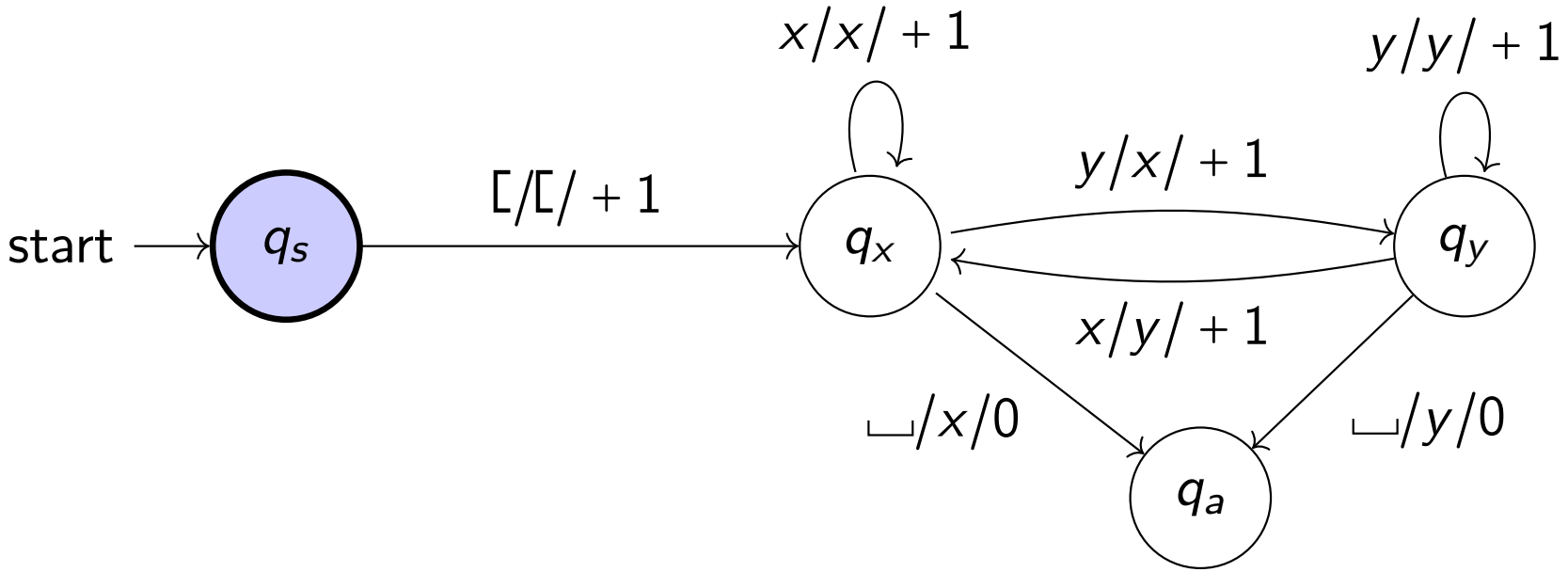
Startkonfiguration und Berechnungen

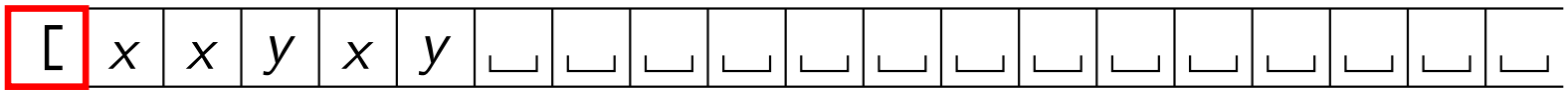
- Sei $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, q_s, q_a, q_r)$ eine TM und sei $w \in \Sigma^*$.
- $\alpha_1 = (q_s, \vdash w \sqcup^\omega, 1)$ heißt Startkonfiguration von M auf w.
- Eine Konfiguration $(q_a, v, k) \in \{q_a\} \times \Gamma^{\mathbb{N}} \times \mathbb{N}$ heißt akzeptierend.
- Eine Konfiguration $(q_r, v, k) \in \{q_r\} \times \Gamma^{\mathbb{N}} \times \mathbb{N}$ heißt verwerfend.
- Die Berechnung von M auf w ist die (eindeutig bestimmte) unendliche Konfigurationsfolge

$$\alpha_1 \vdash_M \alpha_2 \vdash_M \dots,$$

wobei α_1 die Startkonfiguration von M auf w ist.

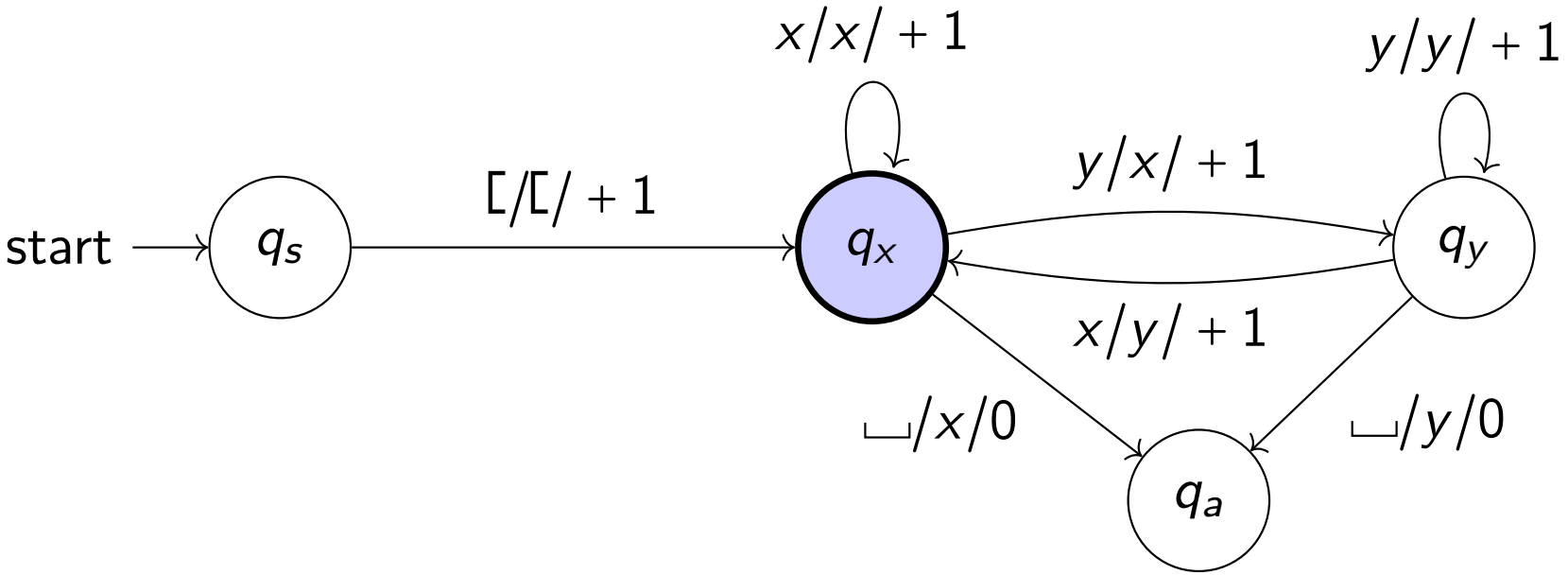
Beispiel Berechnung





Startkonfiguration: $(q_s, [xxyxy\sqcup\sqcup\sqcup\ldots, 1)$

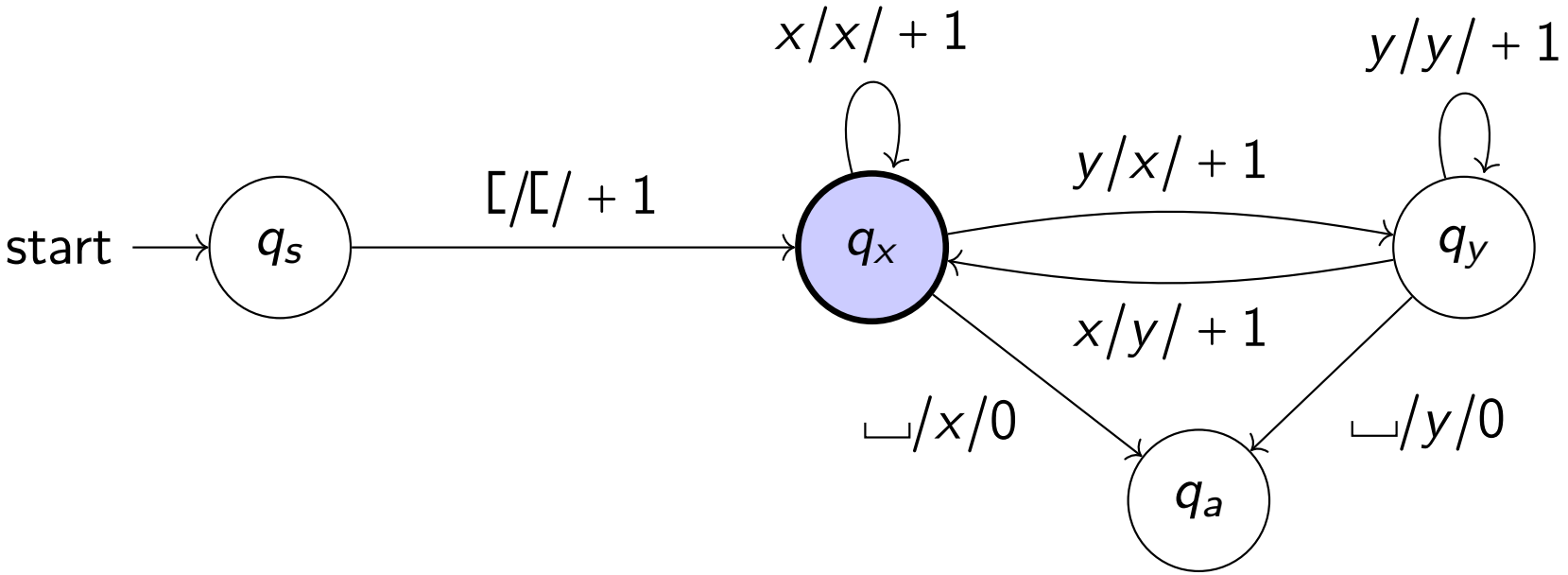
Beispiel Berechnung

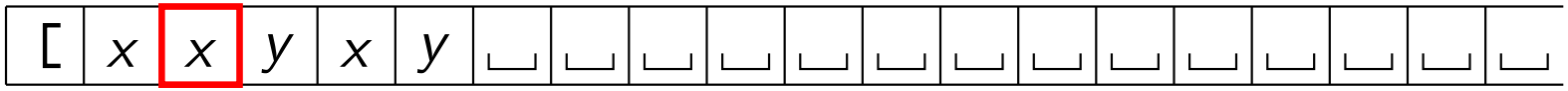


Startkonfiguration: $(q_s, \vdash xxyxy\sqcup\sqcup\sqcup\ldots, 1)$

$\vdash_M (q_x, \vdash xxyxy\sqcup\sqcup\sqcup\ldots, 2)$

Beispiel Berechnung





Startkonfiguration: $(q_s, [xxyxy\sqcup\sqcup\sqcup\ldots, 1)$

$\vdash_M (q_x, [xxyxy\sqcup\sqcup\sqcup\ldots, 2)$

$\vdash_M (q_x, [xxyxy\sqcup\sqcup\sqcup\ldots, 3)$

Berechnungen

- Eine Berechnung $\alpha_1 \vdash_M \alpha_2 \vdash_M \ldots$ ist akzeptierend, falls α_i akzeptierend ist für ein $i \geq 1$ und verwerfend, falls α_i verwerfend ist für ein $i \geq 1$.
  - Eine Berechnung kann nicht akzeptierend und verwerfend gleichzeitig sein, da $\alpha_i \vdash_M \alpha_i$ für eine akzeptierende oder verwerfende Konfiguration α_i und $q_a \neq q_r$.
  - Dies folgt aus Bedingung (2) für die Funktion δ.
  - Also erreicht eine Berechnung einen Fixpunkt (sie *terminiert*), sobald sie eine akzeptierende oder verwerfende Konfiguration erreicht.
- Die Maschine M hält oder terminiert auf w falls es eine akzeptierende oder verwerfende Berechnung von M auf w gibt. Sie akzeptiert w im ersten Fall und verwirft w im zweiten Fall.

Erkennbare Sprachen

- Die von einer Turingmaschine $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, q_s, q_a, q_r)$ erkannte Sprache ist die Menge $L(M) = \{w \in \Sigma^* \ : \ M \text{ akzeptiert } w\}$.
- Eine Sprache L heißt erkennbar, rekursiv aufzählbar (r.e. für recursively enumerable) oder semi-entscheidbar, wenn es eine Turingmaschine M gibt, die L erkennt, d.h. es gilt $L = L(M)$.
  - Beachte den Unterschied, ob eine Maschine M auf einem Wort w terminiert und es verwirft, oder ob sie auf w nicht terminiert.
  - In beiden Fällen akzeptiert sie das Wort nicht.
  - Im zweiten Fall können wir aber möglicherweise zu keinem Zeitpunkt der Berechnung sagen, ob M auf w terminieren wird und ob wir noch auf die Antwort warten sollen.
- Eine Sprache $L \subseteq \Sigma^*$ heißt co-rekursiv aufzählbar (co-r.e.) wenn das Komplement $\overline{L} = \{w \in \Sigma^* \ : \ w \notin L\}$ rekursiv aufzählbar ist.

Entscheidbare Sprachen, berechenbare Funktionen

- Eine Sprache $L \subseteq \Sigma^*$ heißt berechenbar, rekursiv oder entscheidbar, wenn es eine Turingmaschine M gibt, die auf jedem $w \in \Sigma^*$ terminiert und w akzeptiert genau dann, wenn $w \in L$.
- Eine partielle Funktion $f : \Sigma^* \to \Gamma^*$ heißt berechenbar oder rekursiv, wenn es eine Turingmaschine $M = (Q, \Sigma, \Gamma, [, \sqcup, \delta, q_s, q_a, q_r)$ gibt, die
  - auf allen Wörtern $w \in \Sigma^*$ auf denen f definiert ist in einer akzeptierenden Konfiguration (q_a, v, k) terminiert und $v_2 \dots v_k = f(w)$ gilt (das Symbol $v_1 = [$ wird ignoriert)
  - und auf allen Wörtern $w \in \Sigma^*$ auf denen f nicht definiert ist, nicht akzeptiert.