Vorlesung

Theoretische Informatik II

Bernhard Beckert

Institut für Informatik

UNIVERSITÄT KOBLENZ · LANDAU

Wintersemester 2007/2008

Dank

Diese Vorlesungsmaterialien basieren zum Teil auf den Folien zu den Vorlesungen von

Katrin Erk (gehalten an der Universität Koblenz-Landau)

Jürgen Dix (gehalten an der TU Clausthal)

Christoph Kreitz (gehalten an der Universität Potsdam)

Ihnen gilt mein herzlicher Dank.

– Bernhard Beckert, Oktober 2007

Teil I

Einführung

Teil I

Einführung

Kontakt

Bernhard Beckert

Email: beckert@uni-koblenz.de

Webseite: www.uni-koblenz.de/~beckert

Raum: B 218

Sprechstunde:

Dienstags, 16 Uhr, nach der Vorlesung
(sonst auch immer, wenn die Tür offen steht)

Ulrich Koch

Email: koch@uni-koblenz.de

Webseite: www.uni-koblenz.de/~koch

Raum: A 219

Kontakt

Bernhard Beckert

Email: beckert@uni-koblenz.de

Webseite: www.uni-koblenz.de/~beckert

Raum: B 218

Sprechstunde:

Dienstags, 16 Uhr, nach der Vorlesung
(sonst auch immer, wenn die Tür offen steht)

Ulrich Koch

Email: koch@uni-koblenz.de

Webseite: www.uni-koblenz.de/~koch

Raum: A 219

Webseite

http://www.uni-koblenz.de/~beckert/Lehre/TheoretischeInformatikII/

Alle relevante Information auf der Webseite

- Folien
- Weitere Materialien
- Termine usw.

Webseite

http://www.uni-koblenz.de/~beckert/Lehre/TheoretischeInformatikII/

Alle relevante Information auf der Webseite

- Folien
- Weitere Materialien
- Termine usw.

Übungen

Termine und Einteilung

Gruppe A: Mittwochs, 16 Uhr c.t., Raum C208

Gruppe B: Mittwochs, 18 Uhr c.t., Raum C208

MeToo-Registrierung bis Sonntag 28.10.!

Übungsblätter

- Wöchentlich
- Werden in den Übungen der darauffolgenden Woche besprochen
- Kein Einfluss auf Scheinvergabe

Übungen

Termine und Einteilung

Gruppe A: Mittwochs, 16 Uhr c.t., Raum C208

Gruppe B: Mittwochs, 18 Uhr c.t., Raum C208

MeToo-Registrierung bis Sonntag 28.10.!

Übungsblätter

- Wöchentlich
- Werden in den Übungen der darauffolgenden Woche besprochen
- Kein Einfluss auf Scheinvergabe

Prüfungen und Scheinvergabe

Zwei Teilklausuren während des Semesters

- Während des Semesters
- zwei Teilklausuren (je 45 Minuten)
- zur Mitte und zum Ende der Vorlesungszeit
- Schein bei 50% der insgesamt in den Teilklausuren zu erzielenden Punkte
- Freiversuch gilt für alle Teilklausuren zusammen
- Wiederholung einzelner Teilklausuren nicht möglich

Nachklausur zum Ende der Semesters

- Gegen Ende des Semesters (Ende März)
- Nachklausur hat gleichen „Wert“ wie die Teilklausuren zusammen
- 90 Minuten Dauer
- Schein bei 50% der Punkte in der Nachklausur

Prüfungen und Scheinvergabe

Zwei Teilklausuren während des Semesters

- Während des Semesters
- zwei Teilklausuren (je 45 Minuten)
- zur Mitte und zum Ende der Vorlesungszeit
- Schein bei 50% der insgesamt in den Teilklausuren zu erzielenden Punkte
- Freiversuch gilt für alle Teilklausuren zusammen
- Wiederholung einzelner Teilklausuren nicht möglich

Nachklausur zum Ende der Semesters

- Gegen Ende des Semesters (Ende März)
- Nachklausur hat gleichen „Wert“ wie die Teilklausuren zusammen
- 90 Minuten Dauer
- Schein bei 50% der Punkte in der Nachklausur

Prüfungen und Scheinvergabe

Anmeldung zu und Teilnahme an der ersten Prüfung (also den Teilklausuren während des Semesters) ist Voraussetzung für die Teilnahme an der Nachklausur.

Zudem darf an der Nachklausur (wie auch an den Teilklausuren) teilnehmen, wer an einer Prüfung zu „Einführung in die Theoretische Informatik II“ in frühren Semestern teilgenommen und diese nicht bestanden oder einen Freiversuch gesetzt hat.

Prüfungen und Scheinvergabe

Anmeldung zu und Teilnahme an der ersten Prüfung (also den Teilklausuren während des Semesters) ist Voraussetzung für die Teilnahme an der Nachklausur.

Zudem darf an der Nachklausur (wie auch an den Teilklausuren) teilnehmen, wer an einer Prüfung zu „Einführung in die Theoretische Informatik II“ in frühren Semestern teilgenommen und diese nicht bestanden oder einen Freiversuch gesetzt hat.

Literatur

Buch zur Vorlesung

Erk, Katrin and Priese, Lutz (2002).
Theoretische Informatik: Eine umfassende Einführung.
2. Auflage.
Springer-Verlag.

Literatur

Weitere Literatur


J. Hopcroft, R. Motwani, and J. Ullman (2002).
Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie.
Pearson.


G. Vossen and K.-U. Witt (2004).
Grundkurs Theoretische Informatik.
Vieweg.


U. Schöning (1994).
Theoretische Informatik: kurzgefaßt.
Spektrum-Verlag.

Teil I

Einführung

Teil I

Einführung

Theoretische Informatik befasst sich mit . . .

Grundlegende Konzepte der Informatik

- Probleme und ihre Beschreibung
- Systeme/Automaten/Maschinen, die Probleme lösen
- Lösbarkeit von Problemen
  (Entscheidbarkeit/Berechenbarkeit und deren Grenzen)
- Schwierigkeit (Komplexität) der Lösung von Problemen

Theoretische Informatik befasst sich mit . . .

Grundlegende Konzepte der Informatik

- Probleme und ihre Beschreibung
- Systeme/Automaten/Maschinen, die Probleme lösen
- Lösbarkeit von Problemen
  (Entscheidbarkeit/Berechenbarkeit und deren Grenzen)
- Schwierigkeit (Komplexität) der Lösung von Problemen

Theoretische Informatik befasst sich mit . . .

Grundlegende Konzepte der Informatik

- Probleme und ihre Beschreibung
- Systeme/Automaten/Maschinen, die Probleme lösen
- Lösbarkeit von Problemen (Entscheidbarkeit/Berechenbarkeit und deren Grenzen)
- Schwierigkeit (Komplexität) der Lösung von Problemen

Theoretische Informatik befasst sich mit . . .

Grundlegende Konzepte der Informatik

- Probleme und ihre Beschreibung
- Systeme/Automaten/Maschinen, die Probleme lösen
- Lösbarkeit von Problemen
  (Entscheidbarkeit/Berechenbarkeit und deren Grenzen)
- Schwierigkeit (Komplexität) der Lösung von Problemen

Teilgebiete der Theoretischen Informatik

Teilgebiete

- Formale Sprachen
- Automatentheorie
- Berechenbarkeitstheorie
- Komplexitätstheorie
- (Logik)

Schwerpunkte von Theoretische Informatik II

Teilgebiete der Theoretischen Informatik

Teilgebiete

- Formale Sprachen
- Automatentheorie
- Berechenbarkeitstheorie
- Komplexitätstheorie
- (Logik)

Schwerpunkte von Theoretische Informatik II

Teilgebiete der Theoretischen Informatik

Teilgebiete

- Formale Sprachen
- Automatentheorie
- Berechenbarkeitstheorie
- Komplexitätstheorie
- (Logik)

Schwerpunkte von Theoretische Informatik II

Teilgebiete der Theoretischen Informatik

Teilgebiete

- Formale Sprachen
- Automatentheorie
- Berechenbarkeitstheorie
- Komplexitätstheorie
- (Logik)

Schwerpunkte von Theoretische Informatik II

Teilgebiete der Theoretischen Informatik

Teilgebiete

- Formale Sprachen
- Automatentheorie
- Berechenbarkeitstheorie
- Komplexitätstheorie
- (Logik)

Schwerpunkte von Theoretische Informatik II

Teilgebiete der Theoretischen Informatik

Teilgebiete

- Formale Sprachen
- Automatentheorie
- Berechenbarkeitstheorie
- Komplexitätstheorie
- (Logik)

Schwerpunkte von Theoretische Informatik II

Teilgebiete der Theoretischen Informatik

Teilgebiete

- Formale Sprachen
- Automatentheorie
- **Berechenbarkeitstheorie**
- **Komplexitätstheorie**
- (Logik)

Schwerpunkte von Theoretische Informatik II

Warum Theoretische Informatik?

Theoretische Informatik . . .

- ist die Grundlage
- ist wichtig
  (bspw. für Algorithmentechnik, Software Engineering, Compilerbau)
- hilft, weitere Themen/Vorlesungen der Informatik zu verstehen
- veraltet nicht
- macht Spaß

Warum Theoretische Informatik?

Theoretische Informatik . . .

- ist die Grundlage
- ist wichtig
  (bspw. für Algorithmentechnik, Software Engineering, Compilerbau)
- hilft, weitere Themen/Vorlesungen der Informatik zu verstehen
- veraltet nicht
- macht Spaß

Warum Theoretische Informatik?

Theoretische Informatik . . .

- ist die Grundlage
- ist wichtig
  (bspw. für Algorithmentechnik, Software Engineering, Compilerbau)
- hilft, weitere Themen/Vorlesungen der Informatik zu verstehen
- veraltet nicht
- macht Spaß

Warum Theoretische Informatik?

Theoretische Informatik . . .

- ist die Grundlage
- ist wichtig
  (bspw. für Algorithmentechnik, Software Engineering, Compilerbau)
- hilft, weitere Themen/Vorlesungen der Informatik zu verstehen
- veraltet nicht
- macht Spaß

Warum Theoretische Informatik?

Theoretische Informatik . . .

- ist die Grundlage
- ist wichtig
  (bspw. für Algorithmentechnik, Software Engineering, Compilerbau)
- hilft, weitere Themen/Vorlesungen der Informatik zu verstehen
- veraltet nicht
- macht Spaß

Inhalt der Vorlesung

1. Wiederholung: Turing-Maschinen und Turing-Berechenbarkeit
2. Registermaschinen, LOOP, WHILE
3. Rekursive Funktionen
4. Der λ-Kalkül
5. Die Church-Turing-These
6. Berechenbarkeit, (Un-)Entscheidbarkeit
7. Komplexitätstheorie

Inhalt der Vorlesung

1. Wiederholung: Turing-Maschinen und Turing-Berechenbarkeit
2. Registermaschinen, LOOP, WHILE
3. Rekursive Funktionen
4. Der λ-Kalkül
5. Die Church-Turing-These
6. Berechenbarkeit, (Un-)Entscheidbarkeit
7. Komplexitätstheorie

Inhalt der Vorlesung

1. Wiederholung: Turing-Maschinen und Turing-Berechenbarkeit
2. Registermaschinen, LOOP, WHILE
3. Rekursive Funktionen
4. Der λ-Kalkül
5. Die Church-Turing-These
6. Berechenbarkeit, (Un-)Entscheidbarkeit
7. Komplexitätstheorie

Inhalt der Vorlesung

1. Wiederholung: Turing-Maschinen und Turing-Berechenbarkeit
2. Registermaschinen, LOOP, WHILE
3. Rekursive Funktionen
4. Der λ-Kalkül
5. Die Church-Turing-These
6. Berechenbarkeit, (Un-)Entscheidbarkeit
7. Komplexitätstheorie

Inhalt der Vorlesung

1. Wiederholung: Turing-Maschinen und Turing-Berechenbarkeit
2. Registermaschinen, LOOP, WHILE
3. Rekursive Funktionen
4. Der λ-Kalkül
5. Die Church-Turing-These
6. Berechenbarkeit, (Un-)Entscheidbarkeit
7. Komplexitätstheorie

Inhalt der Vorlesung

1. Wiederholung: Turing-Maschinen und Turing-Berechenbarkeit
2. Registermaschinen, LOOP, WHILE
3. Rekursive Funktionen
4. Der λ-Kalkül
5. Die Church-Turing-These
6. Berechenbarkeit, (Un-)Entscheidbarkeit
7. Komplexitätstheorie

Inhalt der Vorlesung

1. Wiederholung: Turing-Maschinen und Turing-Berechenbarkeit
2. Registermaschinen, LOOP, WHILE
3. Rekursive Funktionen
4. Der λ-Kalkül
5. Die Church-Turing-These
6. Berechenbarkeit, (Un-)Entscheidbarkeit
7. Komplexitätstheorie

Ausblick: Die Church-Turing-These (Churchsche These)

Informelle These

Die **intuitiv** berechenbaren Funktionen sind genau die Funktionen, die **Turing-berechenbar** sind.

Beweisbare Instanzen der These

Alle bekannten Berechnungsmodelle

- Turing-Maschinen
- Rekursive Funktionen
- λ-Funktionen
- alle bekannten Programmiersprachen (imperativ, logisch, funktional)
- ...

liefern den gleichen Berechenbarkeitsbegriff.

Ausblick: Die Church-Turing-These (Churchsche These)

Informelle These

Die **intuitiv** berechenbaren Funktionen sind genau die Funktionen, die **Turing-berechenbar** sind.

Beweisbare Instanzen der These

Alle bekannten Berechnungsmodelle

- Turing-Maschinen
- Rekursive Funktionen
- λ-Funktionen
- alle bekannten Programmiersprachen (imperativ, logisch, funktional)
- ...

liefern den gleichen Berechenbarkeitsbegriff.

Alonzo Church: Begründer der Theoretischen Informatik

Alonzo Church ⋆1903, †1995

- Studierte und promovierte in Princeton
- Postdoc in Göttingen
- Professor in Princeton und an der UCLA
- Legte Grundlagen der Theoretischen Informatik, u. a. Einführung des λ-Kalküls
- Doktorvater vieler der bedeutendsten Informatiker

Alonzo Church: Begründer der Theoretischen Informatik

Alonzo Church ⋆1903, †1995

- Studierte und promovierte in Princeton
- Postdoc in Göttingen
- Professor in Princeton und an der UCLA
- Legte Grundlagen der Theoretischen Informatik, u. a. Einführung des λ-Kalküls
- Doktorvater vieler der bedeutendsten Informatiker

Alonzo Church: Begründer der Theoretischen Informatik

Alonzo Church ⋆1903, †1995

- Studierte und promovierte in Princeton
- Postdoc in Göttingen
- Professor in Princeton und an der UCLA
- Legte Grundlagen der Theoretischen Informatik, u. a. Einführung des λ-Kalküls
- Doktorvater vieler der bedeutendsten Informatiker

Alonzo Church: Begründer der Theoretischen Informatik

Alonzo Church ⋆1903, †1995

- Studierte und promovierte in Princeton
- Postdoc in Göttingen
- Professor in Princeton und an der UCLA
- Legte Grundlagen der Theoretischen Informatik, u. a. Einführung des λ-Kalküls
- Doktorvater vieler der bedeutendsten Informatiker

Alonzo Church: Begründer der Theoretischen Informatik

Alonzo Church ⋆1903, †1995

- Studierte und promovierte in Princeton
- Postdoc in Göttingen
- Professor in Princeton und an der UCLA
- Legte Grundlagen der Theoretischen Informatik, u. a. Einführung des λ-Kalküls
- Doktorvater vieler der bedeutendsten Informatiker

Alonzo Church: Begründer der Theoretischen Informatik

Doktoranden Alonzo Churchs

Peter Andrews Grundlagen des automatischen Theorembeweisens

Martin Davis Davis-Putnam-Verfahren (automatisches Beweisen)

Leon Henkin (Standard-)Beweis der Vollständikeit der Prädikatenlogik

Stephen Kleene Reguläre Ausdrücke

Dana Scott Denotationale Semantik, Automatentheorie

Raymond Smullyan Tableaukalkül, Autor

Alan Turing Turing-Maschinen, Unentscheidbarkeit des Halteproblems

(und andere mehr)

Alonzo Church: Begründer der Theoretischen Informatik

Doktoranden Alonzo Churchs

Peter Andrews Grundlagen des automatischen Theorembeweisens

Martin Davis Davis-Putnam-Verfahren (automatisches Beweisen)

Leon Henkin (Standard-)Beweis der Vollständikeit der Prädikatenlogik

Stephen Kleene Reguläre Ausdrücke

Dana Scott Denotationale Semantik, Automatentheorie

Raymond Smullyan Tableaukalkül, Autor

Alan Turing Turing-Maschinen, Unentscheidbarkeit des Halteproblems

(und andere mehr)

Alonzo Church: Begründer der Theoretischen Informatik

Doktoranden Alonzo Churchs

Peter Andrews Grundlagen des automatischen Theorembeweisens

Martin Davis Davis-Putnam-Verfahren (automatisches Beweisen)

Leon Henkin (Standard-)Beweis der Vollständikeit der Prädikatenlogik

Stephen Kleene Reguläre Ausdrücke

Dana Scott Denotationale Semantik, Automatentheorie

Raymond Smullyan Tableaukalkül, Autor

Alan Turing Turing-Maschinen, Unentscheidbarkeit des Halteproblems

(und andere mehr)

Alonzo Church: Begründer der Theoretischen Informatik

Doktoranden Alonzo Churchs

Peter Andrews Grundlagen des automatischen Theorembeweisens

Martin Davis Davis-Putnam-Verfahren (automatisches Beweisen)

Leon Henkin (Standard-)Beweis der Vollständikeit der Prädikatenlogik

Stephen Kleene Reguläre Ausdrücke

Dana Scott Denotationale Semantik, Automatentheorie

Raymond Smullyan Tableaukalkül, Autor

Alan Turing Turing-Maschinen, Unentscheidbarkeit des Halteproblems

(und andere mehr)

Alonzo Church: Begründer der Theoretischen Informatik

Doktoranden Alonzo Churchs

Peter Andrews Grundlagen des automatischen Theorembeweisens

Martin Davis Davis-Putnam-Verfahren (automatisches Beweisen)

Leon Henkin (Standard-)Beweis der Vollständikeit der Prädikatenlogik

Stephen Kleene Reguläre Ausdrücke

Dana Scott Denotationale Semantik, Automatentheorie

Raymond Smullyan Tableaukalkül, Autor

Alan Turing Turing-Maschinen, Unentscheidbarkeit des Halteproblems

(und andere mehr)

Alonzo Church: Begründer der Theoretischen Informatik

Doktoranden Alonzo Churchs

Peter Andrews Grundlagen des automatischen Theorembeweisens

Martin Davis Davis-Putnam-Verfahren (automatisches Beweisen)

Leon Henkin (Standard-)Beweis der Vollständikeit der Prädikatenlogik

Stephen Kleene Reguläre Ausdrücke

Dana Scott Denotationale Semantik, Automatentheorie

Raymond Smullyan Tableaukalkül, Autor

Alan Turing Turing-Maschinen, Unentscheidbarkeit des Halteproblems

(und andere mehr)

Alonzo Church: Begründer der Theoretischen Informatik

Doktoranden Alonzo Churchs

Peter Andrews Grundlagen des automatischen Theorembeweisens

Martin Davis Davis-Putnam-Verfahren (automatisches Beweisen)

Leon Henkin (Standard-)Beweis der Vollständikeit der Prädikatenlogik

Stephen Kleene Reguläre Ausdrücke

Dana Scott Denotationale Semantik, Automatentheorie

Raymond Smullyan Tableaukalkül, Autor

Alan Turing Turing-Maschinen, Unentscheidbarkeit des Halteproblems

(und andere mehr)

Alonzo Church: Begründer der Theoretischen Informatik

Doktoranden Alonzo Churchs

Peter Andrews Grundlagen des automatischen Theorembeweisens

Martin Davis Davis-Putnam-Verfahren (automatisches Beweisen)

Leon Henkin (Standard-)Beweis der Vollständikeit der Prädikatenlogik

Stephen Kleene Reguläre Ausdrücke

Dana Scott Denotationale Semantik, Automatentheorie

Raymond Smullyan Tableaukalkül, Autor

Alan Turing Turing-Maschinen, Unentscheidbarkeit des Halteproblems

(und andere mehr)

Alonzo Church: Begründer der Theoretischen Informatik

Doktoranden Alonzo Churchs

Peter Andrews Grundlagen des automatischen Theorembeweisens

Martin Davis Davis-Putnam-Verfahren (automatisches Beweisen)

Leon Henkin (Standard-)Beweis der Vollständikeit der Prädikatenlogik

Stephen Kleene Reguläre Ausdrücke

Dana Scott Denotationale Semantik, Automatentheorie

Raymond Smullyan Tableaukalkül, Autor

Alan Turing Turing-Maschinen, Unentscheidbarkeit des Halteproblems

(und andere mehr)

Teil I

Einführung

Teil I

Einführung

Teil I

Einführung

Turing-Maschinen

Überblick: Turing-Maschinen

- akzeptieren Sprachen **vom Typ 0**.
- Erster Speicher: der Zustand (endlich)
- Zweiter Speicher: Band
  (unbeschränkte Größe, **Zugriff an beliebiger Stelle**
- Turing-Maschine hat einen Schreib-/Lesekopf, den sie über diesem Band in einem Rechenschritt um ein Feld nach rechts oder links bewegen kann.
- Das Eingabewort steht (am Anfang) auf dem Band.
  Die Maschine kann es **beliebig oft lesen**.

Turing-Maschinen

Überblick: Turing-Maschinen

- akzeptieren Sprachen **vom Typ 0**.
- Erster Speicher: der Zustand (endlich)
- Zweiter Speicher: Band
  (unbeschränkte Größe, **Zugriff an beliebiger Stelle**
- Turing-Maschine hat einen Schreib-/Lesekopf, den sie über diesem Band in einem Rechenschritt um ein Feld nach rechts oder links bewegen kann.
- Das Eingabewort steht (am Anfang) auf dem Band.
  Die Maschine kann es **beliebig oft lesen**.

Turing-Maschinen

Überblick: Turing-Maschinen

- akzeptieren Sprachen **vom Typ 0**.
- Erster Speicher: der Zustand (endlich)
- Zweiter Speicher: Band
  (unbeschränkte Größe, **Zugriff an beliebiger Stelle**
- Turing-Maschine hat einen Schreib-/Lesekopf, den sie über diesem Band in einem Rechenschritt um ein Feld nach rechts oder links bewegen kann.
- Das Eingabewort steht (am Anfang) auf dem Band.
  Die Maschine kann es **beliebig oft lesen**.

Turing-Maschinen

Überblick: Turing-Maschinen

- akzeptieren Sprachen **vom Typ 0**.
- Erster Speicher: der Zustand (endlich)
- Zweiter Speicher: Band
  (unbeschränkte Größe, **Zugriff an beliebiger Stelle**
- Turing-Maschine hat einen Schreib-/Lesekopf, den sie über diesem Band in einem Rechenschritt um ein Feld nach rechts oder links bewegen kann.
- Das Eingabewort steht (am Anfang) auf dem Band.
  Die Maschine kann es **beliebig oft lesen**.

Turing-Maschinen

Überblick: Turing-Maschinen

- akzeptieren Sprachen **vom Typ 0**.
- Erster Speicher: der Zustand (endlich)
- Zweiter Speicher: Band
  (unbeschränkte Größe, **Zugriff an beliebiger Stelle**
- Turing-Maschine hat einen Schreib-/Lesekopf, den sie über diesem Band in einem Rechenschritt um ein Feld nach rechts oder links bewegen kann.
- Das Eingabewort steht (am Anfang) auf dem Band.
  Die Maschine kann es **beliebig oft lesen**.

Turing-Maschine

Definition 3.1 (Turing-Maschine (DTM))

Eine **determinierte Turing-Maschine (DTM)** $\mathcal{M}$ ist ein Tupel

$$\mathcal{M} = (K, \Sigma, \delta, s)$$

Dabei ist

- K eine endliche Menge von Zuständen mit $h \notin K$,
  (h ist der **Haltezustand**)
- Σ ein Alphabet mit $L, R \notin \Sigma$, $\# \in \Sigma$,
- $\delta : K \times \Sigma \rightarrow (K \cup \{h\}) \times (\Sigma \cup \{L, R\})$ eine Übergangs**funktion**
- $s \in K$ ein Startzustand.

Anzahl der Zustände: $|K| - 1$
(Startzustand wird nicht mitgezählt).

Turing-Maschine

Definition 3.1 (Turing-Maschine (DTM))

Eine **determinierte Turing-Maschine (DTM)** $\mathcal{M}$ ist ein Tupel

$$\mathcal{M} = (K, \Sigma, \delta, s)$$

Dabei ist

- K eine endliche Menge von Zuständen mit $h \notin K$,
  (h ist der **Haltezustand**)
- Σ ein Alphabet mit $L, R \notin \Sigma$, $\# \in \Sigma$,
- $\delta : K \times \Sigma \to (K \cup \{h\}) \times (\Sigma \cup \{L, R\})$ eine Übergangs**funktion**
- $s \in K$ ein Startzustand.

Anzahl der Zustände: $|K| - 1$
(Startzustand wird nicht mitgezählt).

Turing-Maschine

Definition 3.1 (Turing-Maschine (DTM))

Eine **determinierte Turing-Maschine (DTM)** $\mathcal{M}$ ist ein Tupel

$$\mathcal{M} = (K, \Sigma, \delta, s)$$

Dabei ist

- K eine endliche Menge von Zuständen mit $h \notin K$,
  (h ist der **Haltezustand**)
- Σ ein Alphabet mit $L, R \notin \Sigma$, $\# \in \Sigma$,
- $\delta : K \times \Sigma \to (K \cup \{h\}) \times (\Sigma \cup \{L, R\})$ eine Übergangs**funktion**
- $s \in K$ ein Startzustand.

Anzahl der Zustände: $|K| - 1$
(Startzustand wird nicht mitgezählt).

Turing-Maschine

Definition 3.1 (Turing-Maschine (DTM))

Eine **determinierte Turing-Maschine (DTM)** $\mathcal{M}$ ist ein Tupel

$$\mathcal{M} = (K, \Sigma, \delta, s)$$

Dabei ist

- K eine endliche Menge von Zuständen mit $h \notin K$,
  (h ist der **Haltezustand**)
- Σ ein Alphabet mit $L, R \notin \Sigma$, $\# \in \Sigma$,
- $\delta : K \times \Sigma \to (K \cup \{h\}) \times (\Sigma \cup \{L, R\})$ eine Übergangs**funktion**
- $s \in K$ ein Startzustand.

Anzahl der Zustände: $|K| - 1$
(Startzustand wird nicht mitgezählt).

Turing-Maschine

Definition 3.1 (Turing-Maschine (DTM))

Eine **determinierte Turing-Maschine (DTM)** $\mathcal{M}$ ist ein Tupel

$$\mathcal{M} = (K, \Sigma, \delta, s)$$

Dabei ist

- K eine endliche Menge von Zuständen mit $h \notin K$,
  (h ist der **Haltezustand**)
- Σ ein Alphabet mit $L, R \notin \Sigma$, $\# \in \Sigma$,
- $\delta : K \times \Sigma \rightarrow (K \cup \{h\}) \times (\Sigma \cup \{L, R\})$ eine Übergangs**funktion**
- $s \in K$ ein Startzustand.

Anzahl der Zustände: $|K| - 1$
(Startzustand wird nicht mitgezählt).

Turing-Maschine

Definition 3.1 (Turing-Maschine (DTM))

Eine **determinierte Turing-Maschine (DTM)** $\mathcal{M}$ ist ein Tupel

$$\mathcal{M} = (K, \Sigma, \delta, s)$$

Dabei ist

- K eine endliche Menge von Zuständen mit $h \notin K$,
  (h ist der **Haltezustand**)
- Σ ein Alphabet mit $L, R \notin \Sigma$, $\# \in \Sigma$,
- $\delta : K \times \Sigma \rightarrow (K \cup \{h\}) \times (\Sigma \cup \{L, R\})$ eine Übergangs**funktion**
- $s \in K$ ein Startzustand.

Anzahl der Zustände: $|K| - 1$
(Startzustand wird nicht mitgezählt).

Turing-Maschine

Definition 3.1 (Turing-Maschine (DTM))

Eine **determinierte Turing-Maschine (DTM)** $\mathcal{M}$ ist ein Tupel

$$\mathcal{M} = (K, \Sigma, \delta, s)$$

Dabei ist

- K eine endliche Menge von Zuständen mit $h \notin K$,
  (h ist der **Haltezustand**)
- Σ ein Alphabet mit $L, R \notin \Sigma$, $\# \in \Sigma$,
- $\delta : K \times \Sigma \to (K \cup \{h\}) \times (\Sigma \cup \{L, R\})$ eine Übergangs**funktion**
- $s \in K$ ein Startzustand.

Anzahl der Zustände: $|K| - 1$
(Startzustand wird nicht mitgezählt).

Turing-Maschine

Arbeitsschritt einer Turing-Maschine

Übergang

$$\delta(q, a) = (q', x)$$

bedeutet:

In Abhängigkeit

- vom aktuellen Zustand $q \in K$
- von dem Zeichen $a \in \Sigma$, das unter dem Schreib-/Lesekopf steht

geschieht folgendes:

- entweder ein **Schritt nach links**, falls $x = L$ ist
- oder ein **Schritt nach rechts**, falls $x = R$ ist
- oder **das Zeichen** a, das momentan unter dem Schreib-/Lesekopf steht, wird **durch** $b \in \Sigma$ **überschreiben**, falls $x = b \in \Sigma$
- der **Zustand** wird zu $q' \in K \cup \{h\}$ **geändert**,

Turing-Maschine

Arbeitsschritt einer Turing-Maschine

Übergang

$$\delta(q, a) = (q', x)$$

bedeutet:

In Abhängigkeit

- vom aktuellen Zustand $q \in K$
- von dem Zeichen $a \in \Sigma$, das unter dem Schreib-/Lesekopf steht

geschieht folgendes:

- entweder ein **Schritt nach links**, falls $x = L$ ist
- oder ein **Schritt nach rechts**, falls $x = R$ ist
- oder **das Zeichen** a, das momentan unter dem Schreib-/Lesekopf steht, wird **durch** $b \in \Sigma$ **überschreiben**, falls $x = b \in \Sigma$
- der **Zustand** wird zu $q' \in K \cup \{h\}$ **geändert**,

Turing-Maschine

Arbeitsschritt einer Turing-Maschine

Übergang

$$\delta(q, a) = (q', x)$$

bedeutet:

In Abhängigkeit

- vom aktuellen Zustand $q \in K$
- von dem Zeichen $a \in \Sigma$, das unter dem Schreib-/Lesekopf steht

geschieht folgendes:

- entweder ein **Schritt nach links**, falls $x = L$ ist
- oder ein **Schritt nach rechts**, falls $x = R$ ist
- oder **das Zeichen** a, das momentan unter dem Schreib-/Lesekopf steht, wird **durch** $b \in \Sigma$ **überschreiben**, falls $x = b \in \Sigma$
- der **Zustand** wird zu $q' \in K \cup \{h\}$ **geändert**,

Turing-Maschine

Arbeitsschritt einer Turing-Maschine

Übergang

$$\delta(q, a) = (q', x)$$

bedeutet:

In Abhängigkeit

- vom aktuellen Zustand $q \in K$
- von dem Zeichen $a \in \Sigma$, das unter dem Schreib-/Lesekopf steht

geschieht folgendes:

- entweder ein **Schritt nach links**, falls $x = L$ ist
- oder ein **Schritt nach rechts**, falls $x = R$ ist
- oder **das Zeichen** a, das momentan unter dem Schreib-/Lesekopf steht, wird **durch** $b \in \Sigma$ **überschreiben**, falls $x = b \in \Sigma$
- der **Zustand** wird zu $q' \in K \cup \{h\}$ **geändert**,

Turing-Maschine

Arbeitsschritt einer Turing-Maschine

Übergang

$$\delta(q, a) = (q', x)$$

bedeutet:

In Abhängigkeit

- vom aktuellen Zustand $q \in K$
- von dem Zeichen $a \in \Sigma$, das unter dem Schreib-/Lesekopf steht

geschieht folgendes:

- entweder ein **Schritt nach links**, falls $x = L$ ist
- oder ein **Schritt nach rechts**, falls $x = R$ ist
- oder **das Zeichen** a, das momentan unter dem Schreib-/Lesekopf steht, wird **durch** $b \in \Sigma$ **überschreiben**, falls $x = b \in \Sigma$
- der **Zustand** wird zu $q' \in K \cup \{h\}$ **geändert**,

Turing-Maschine

Arbeitsschritt einer Turing-Maschine

Übergang

$$\delta(q, a) = (q', x)$$

bedeutet:

In Abhängigkeit

- vom aktuellen Zustand $q \in K$
- von dem Zeichen $a \in \Sigma$, das unter dem Schreib-/Lesekopf steht

geschieht folgendes:

- entweder ein **Schritt nach links**, falls $x = L$ ist
- oder ein **Schritt nach rechts**, falls $x = R$ ist
- oder **das Zeichen** a, das momentan unter dem Schreib-/Lesekopf steht, wird **durch** $b \in \Sigma$ **überschreiben**, falls $x = b \in \Sigma$
- der **Zustand** wird zu $q' \in K \cup \{h\}$ **geändert**,

Turing-Maschine

Arbeitsschritt einer Turing-Maschine

Übergang

$$\delta(q, a) = (q', x)$$

bedeutet:

In Abhängigkeit

- vom aktuellen Zustand $q \in K$
- von dem Zeichen $a \in \Sigma$, das unter dem Schreib-/Lesekopf steht

geschieht folgendes:

- entweder ein **Schritt nach links**, falls $x = L$ ist
- oder ein **Schritt nach rechts**, falls $x = R$ ist
- oder **das Zeichen** a, das momentan unter dem Schreib-/Lesekopf steht, wird **durch** $b \in \Sigma$ **überschreiben**, falls $x = b \in \Sigma$
- der **Zustand** wird zu $q' \in K \cup \{h\}$ **geändert**,

Turing-Maschine

Arbeitsschritt einer Turing-Maschine

Übergang

$$\delta(q, a) = (q', x)$$

bedeutet:

In Abhängigkeit

- vom aktuellen Zustand $q \in K$
- von dem Zeichen $a \in \Sigma$, das unter dem Schreib-/Lesekopf steht

geschieht folgendes:

- entweder ein **Schritt nach links**, falls $x = L$ ist
- oder ein **Schritt nach rechts**, falls $x = R$ ist
- oder **das Zeichen** a, das momentan unter dem Schreib-/Lesekopf steht, wird **durch** $b \in \Sigma$ **überschreiben**, falls $x = b \in \Sigma$
- der **Zustand** wird zu $q' \in K \cup \{h\}$ **geändert**,

Turing-Maschine

Leerzeichen

Das spezielle Zeichen # (*blank*) ist das Leerzeichen.

Es ist nie Teil des Eingabeworts; man kann es u.a. dazu benutzen, Wörter voneinander abzugrenzen.

Turing-Maschine

Begrenzung des Bandes

Das Band einer DTM ist **einseitig unbeschränkt**:

- Nach rechts ist es unendlich lang.
- Nach links hat es ein Ende.
- Wenn eine DTM versucht, das Ende zu überschreiten, bleibt sie „hängen“.

  In diesem Fall hält sie nicht.

Turing-Maschine

Begrenzung des Bandes

Das Band einer DTM ist **einseitig unbeschränkt**:

- Nach rechts ist es unendlich lang.
- Nach links hat es ein Ende.
- Wenn eine DTM versucht, das Ende zu überschreiten, bleibt sie „hängen“.

  In diesem Fall hält sie nicht.

Turing-Maschine

Begrenzung des Bandes

Das Band einer DTM ist **einseitig unbeschränkt**:

- Nach rechts ist es unendlich lang.
- Nach links hat es ein Ende.
- Wenn eine DTM versucht, das Ende zu überschreiten, bleibt sie „hängen“.

  In diesem Fall hält sie nicht.

Turing-Maschine

Anfangskonfiguration

- Ganz links auf dem Band steht ein Blank
- Direkt rechts davon steht das Eingabewort
- Wenn eine DTM mehrere Eingabewörter hintereinander bekommt, sind sie durch Blanks getrennt.
- Rechts vom letzten Eingabewort stehen nur noch Blanks.
- Der Schreib-/Lesekopf der DTM steht auf dem Blank direkt rechts neben dem (letzten) Eingabewort.

Merke

Das Band enthält immer nur endlich viele Symbole, die keine Blanks sind.

Turing-Maschine

Anfangskonfiguration

- Ganz links auf dem Band steht ein Blank
- Direkt rechts davon steht das Eingabewort
- Wenn eine DTM mehrere Eingabewörter hintereinander bekommt, sind sie durch Blanks getrennt.
- Rechts vom letzten Eingabewort stehen nur noch Blanks.
- Der Schreib-/Lesekopf der DTM steht auf dem Blank direkt rechts neben dem (letzten) Eingabewort.

Merke

Das Band enthält immer nur endlich viele Symbole, die keine Blanks sind.

Turing-Maschine

Anfangskonfiguration

- Ganz links auf dem Band steht ein Blank
- Direkt rechts davon steht das Eingabewort
- Wenn eine DTM mehrere Eingabewörter hintereinander bekommt, sind sie durch Blanks getrennt.
- Rechts vom letzten Eingabewort stehen nur noch Blanks.
- Der Schreib-/Lesekopf der DTM steht auf dem Blank direkt rechts neben dem (letzten) Eingabewort.

Merke

Das Band enthält immer nur endlich viele Symbole, die keine Blanks sind.

Turing-Maschine

Anfangskonfiguration

- Ganz links auf dem Band steht ein Blank
- Direkt rechts davon steht das Eingabewort
- Wenn eine DTM mehrere Eingabewörter hintereinander bekommt, sind sie durch Blanks getrennt.
- Rechts vom letzten Eingabewort stehen nur noch Blanks.
- Der Schreib-/Lesekopf der DTM steht auf dem Blank direkt rechts neben dem (letzten) Eingabewort.

Merke

Das Band enthält immer nur endlich viele Symbole, die keine Blanks sind.

Turing-Maschine

Anfangskonfiguration

- Ganz links auf dem Band steht ein Blank
- Direkt rechts davon steht das Eingabewort
- Wenn eine DTM mehrere Eingabewörter hintereinander bekommt, sind sie durch Blanks getrennt.
- Rechts vom letzten Eingabewort stehen nur noch Blanks.
- Der Schreib-/Lesekopf der DTM steht auf dem Blank direkt rechts neben dem (letzten) Eingabewort.

Merke

Das Band enthält immer nur endlich viele Symbole, die keine Blanks sind.

Turing-Maschine

Anfangskonfiguration

- Ganz links auf dem Band steht ein Blank
- Direkt rechts davon steht das Eingabewort
- Wenn eine DTM mehrere Eingabewörter hintereinander bekommt, sind sie durch Blanks getrennt.
- Rechts vom letzten Eingabewort stehen nur noch Blanks.
- Der Schreib-/Lesekopf der DTM steht auf dem Blank direkt rechts neben dem (letzten) Eingabewort.

Merke

Das Band enthält immer nur endlich viele Symbole, die keine Blanks sind.

Turing-Maschine

Anfangskonfiguration

- Ganz links auf dem Band steht ein Blank
- Direkt rechts davon steht das Eingabewort
- Wenn eine DTM mehrere Eingabewörter hintereinander bekommt, sind sie durch Blanks getrennt.
- Rechts vom letzten Eingabewort stehen nur noch Blanks.
- Der Schreib-/Lesekopf der DTM steht auf dem Blank direkt rechts neben dem (letzten) Eingabewort.

Merke

Das Band enthält immer nur endlich viele Symbole, die keine Blanks sind.

Turing-Maschine

Beispiel 3.2 (Copy)

Die folgende DTM $\mathcal{C}$ erhält als Eingabe einen String Einsen.

Dieser String wird kopiert:

Falls n Einsen auf dem Band stehen, stehen nach Ausführung von $\mathcal{C}$ $2n$ Einsen auf dem Band stehen, (getrennt durch ein Blank #).

state	#	1	c
q_0	$\langle q_1, c\rangle$	—	—
q_1	$\langle q_2, R\rangle$	$\langle q_1, L\rangle$	$\langle q_1, L\rangle$
q_2	—	$\langle q_3, \#\rangle$	$\langle q_7, \#\rangle$
q_3	$\langle q_4, R\rangle$	—	—
q_4	$\langle q_5, 1\rangle$	$\langle q_4, R\rangle$	$\langle q_4, R\rangle$
q_5	$\langle q_6, 1\rangle$	$\langle q_5, L\rangle$	$\langle q_5, L\rangle$
q_6	—	$\langle q_2, R\rangle$	—
q_7	$\langle q_8, R\rangle$	—	—
q_8	$\langle h, \#\rangle$	$\langle q_8, R\rangle$	—

Turing-Maschine

Beispiel 3.2 (Copy)

Die folgende DTM $\mathcal{C}$ erhält als Eingabe einen String Einsen.
Dieser String wird kopiert:
Falls n Einsen auf dem Band stehen, stehen nach Ausführung von $\mathcal{C}$ $2n$ Einsen auf dem Band stehen, (getrennt durch ein Blank #).

state	#	1	c
q_0	$\langle q_1, c\rangle$	—	—
q_1	$\langle q_2, R\rangle$	$\langle q_1, L\rangle$	$\langle q_1, L\rangle$
q_2	—	$\langle q_3, \#\rangle$	$\langle q_7, \#\rangle$
q_3	$\langle q_4, R\rangle$	—	—
q_4	$\langle q_5, 1\rangle$	$\langle q_4, R\rangle$	$\langle q_4, R\rangle$
q_5	$\langle q_6, 1\rangle$	$\langle q_5, L\rangle$	$\langle q_5, L\rangle$
q_6	—	$\langle q_2, R\rangle$	—
q_7	$\langle q_8, R\rangle$	—	—
q_8	$\langle h, \#\rangle$	$\langle q_8, R\rangle$	—

Turing-Maschine

Beispiel 3.2 (Copy)

Die folgende DTM $\mathcal{C}$ erhält als Eingabe einen String Einsen.
Dieser String wird kopiert:
Falls n Einsen auf dem Band stehen, stehen nach Ausführung von $\mathcal{C}$ $2n$ Einsen auf dem Band stehen, (getrennt durch ein Blank #).

state	#	1	c
q_0	$\langle q_1, c\rangle$	–	–
q_1	$\langle q_2, R\rangle$	$\langle q_1, L\rangle$	$\langle q_1, L\rangle$
q_2	–	$\langle q_3, \#\rangle$	$\langle q_7, \#\rangle$
q_3	$\langle q_4, R\rangle$	–	–
q_4	$\langle q_5, 1\rangle$	$\langle q_4, R\rangle$	$\langle q_4, R\rangle$
q_5	$\langle q_6, 1\rangle$	$\langle q_5, L\rangle$	$\langle q_5, L\rangle$
q_6	–	$\langle q_2, R\rangle$	–
q_7	$\langle q_8, R\rangle$	–	–
q_8	$\langle h, \#\rangle$	$\langle q_8, R\rangle$	–

Turing-Maschine

Begriff der Konfigurationen

- **Konfiguration** beschreibt die *komplette* aktuelle Situation der Maschine in einer Rechnung.
- Eine **Rechnung** ist eine Folge von Konfigurationen, wobei immer von einer Konfiguration zu einer Nachfolgekonfiguration übergegangen wird.

Konfiguration einer DTM

Besteht aus 4 Elementen:

- das aktuellen Zustand q,
- das Wort w links vom Schreib-/Lesekopf,
- das Zeichen a, auf dem der Kopf gerade steht,
- das Wort u rechts von der aktuellen Kopfposition.

Turing-Maschine

Begriff der Konfigurationen

- **Konfiguration** beschreibt die *komplette* aktuelle Situation der Maschine in einer Rechnung.
- Eine **Rechnung** ist eine Folge von Konfigurationen, wobei immer von einer Konfiguration zu einer Nachfolgekonfiguration übergegangen wird.

Konfiguration einer DTM

Besteht aus 4 Elementen:

- das aktuellen Zustand q,
- das Wort w links vom Schreib-/Lesekopf,
- das Zeichen a, auf dem der Kopf gerade steht,
- das Wort u rechts von der aktuellen Kopfposition.

Turing-Maschine

Begriff der Konfigurationen

- **Konfiguration** beschreibt die *komplette* aktuelle Situation der Maschine in einer Rechnung.
- Eine **Rechnung** ist eine Folge von Konfigurationen, wobei immer von einer Konfiguration zu einer Nachfolgekonfiguration übergegangen wird.

Konfiguration einer DTM

Besteht aus 4 Elementen:

- das aktuellen Zustand q,
- das Wort w links vom Schreib-/Lesekopf,
- das Zeichen a, auf dem der Kopf gerade steht,
- das Wort u rechts von der aktuellen Kopfposition.

Turing-Maschine

Begriff der Konfigurationen

- **Konfiguration** beschreibt die *komplette* aktuelle Situation der Maschine in einer Rechnung.
- Eine **Rechnung** ist eine Folge von Konfigurationen, wobei immer von einer Konfiguration zu einer Nachfolgekonfiguration übergegangen wird.

Konfiguration einer DTM

Besteht aus 4 Elementen:

- das aktuellen Zustand q,
- das Wort w links vom Schreib-/Lesekopf,
- das Zeichen a, auf dem der Kopf gerade steht,
- das Wort u rechts von der aktuellen Kopfposition.

Turing-Maschine

Begriff der Konfigurationen

- **Konfiguration** beschreibt die *komplette* aktuelle Situation der Maschine in einer Rechnung.
- Eine **Rechnung** ist eine Folge von Konfigurationen, wobei immer von einer Konfiguration zu einer Nachfolgekonfiguration übergegangen wird.

Konfiguration einer DTM

Besteht aus 4 Elementen:

- das aktuellen Zustand q,
- das Wort w links vom Schreib-/Lesekopf,
- das Zeichen a, auf dem der Kopf gerade steht,
- das Wort u rechts von der aktuellen Kopfposition.

Turing-Maschine

Begriff der Konfigurationen

- **Konfiguration** beschreibt die *komplette* aktuelle Situation der Maschine in einer Rechnung.
- Eine **Rechnung** ist eine Folge von Konfigurationen, wobei immer von einer Konfiguration zu einer Nachfolgekonfiguration übergegangen wird.

Konfiguration einer DTM

Besteht aus 4 Elementen:

- das aktuellen Zustand q,
- das Wort w links vom Schreib-/Lesekopf,
- das Zeichen a, auf dem der Kopf gerade steht,
- das Wort u rechts von der aktuellen Kopfposition.

Turing-Maschine

Begriff der Konfigurationen

- **Konfiguration** beschreibt die *komplette* aktuelle Situation der Maschine in einer Rechnung.
- Eine **Rechnung** ist eine Folge von Konfigurationen, wobei immer von einer Konfiguration zu einer Nachfolgekonfiguration übergegangen wird.

Konfiguration einer DTM

Besteht aus 4 Elementen:

- das aktuellen Zustand q,
- das Wort w links vom Schreib-/Lesekopf,
- das Zeichen a, auf dem der Kopf gerade steht,
- das Wort u rechts von der aktuellen Kopfposition.

Turing-Maschine

Definition 3.3 (Eingabe)

w heißt **Eingabe** (*input*) für $\mathcal{M}$, falls $\mathcal{M}$ mit der **Startkonfiguration**

$$C_0 = s, \#w\underline{\#}$$

startet.

$(w_1, \ldots, w_n)$ heißt **Eingabe** für $\mathcal{M}$, falls $\mathcal{M}$ mit der **Startkonfiguration**

$$C_0 = s, \#w_1\# \ldots \#w_n\underline{\#}$$

startet.

Turing-Maschine

Definition 3.3 (Eingabe)

w heißt **Eingabe** (*input*) für $\mathcal{M}$, falls $\mathcal{M}$ mit der **Startkonfiguration**

$$C_0 = s, \#w\underline{\#}$$

startet.

$(w_1, \ldots, w_n)$ heißt **Eingabe** für $\mathcal{M}$, falls $\mathcal{M}$ mit der **Startkonfiguration**

$$C_0 = s, \#w_1\# \ldots \#w_n\underline{\#}$$

startet.

Turing-Maschine

Definition 3.3 (Eingabe)

w heißt **Eingabe** (*input*) für $\mathcal{M}$, falls $\mathcal{M}$ mit der **Startkonfiguration**

$$C_0 = s, \#w\underline{\#}$$

startet.

$(w_1, \ldots, w_n)$ heißt **Eingabe** für $\mathcal{M}$, falls $\mathcal{M}$ mit der **Startkonfiguration**

$$C_0 = s, \#w_1\# \ldots \#w_n\underline{\#}$$

startet.

Turing-Maschine

Definition 3.4 (Halten, Hängen)

Sei $\mathcal{M}$ eine Turing-Maschine.

- $\mathcal{M}$ **hält** in $C = q, w\underline{a}u$ **gdw.** $q = h$.
- $\mathcal{M}$ **hängt** in $C = q, w\underline{a}u$ **gdw.** es keine Nachfolgekonfiguration gibt
  Insbesondere: wenn $w = \varepsilon \wedge \exists q' \; \delta(q, a) = (q', L)$.

Turing-Maschine

Definition 3.4 (Halten, Hängen)

Sei $\mathcal{M}$ eine Turing-Maschine.

- $\mathcal{M}$ **hält** in $C = q, w\underline{a}u$ **gdw.** $q = h$.
- $\mathcal{M}$ **hängt** in $C = q, w\underline{a}u$ **gdw.** es keine Nachfolgekonfiguration gibt
  Insbesondere: wenn $w = \varepsilon \wedge \exists q'\ \delta(q, a) = (q', L)$.

Turing-Maschine

Definition 3.5 (Rechnung)

Sei $\mathcal{M}$ eine Turing-Maschine. Man schreibt

$$C \vdash_{\mathcal{M}}^{*} C'$$

gdw.:
es gibt eine Reihe von Konfigurationen

$$C_0, C_1, \ldots, C_n \qquad (n \geq 0)$$

so daß

- $C = C_0$ und $C' = C_n$
- für alle $i < n$ gilt: $C_i \vdash_{\mathcal{M}} C_{i+1}$

Dann heißt $C_0, C_1, \ldots, C_n$ eine **Rechnung** der Länge n von C_0 nach C_n.

Turing-Maschine

Definition 3.5 (Rechnung)

Sei $\mathcal{M}$ eine Turing-Maschine. Man schreibt

$$C \vdash^*_{\mathcal{M}} C'$$

gdw.:
es gibt eine Reihe von Konfigurationen

$$C_0, C_1, \ldots, C_n \qquad (n \geq 0)$$

so daß

- $C = C_0$ und $C' = C_n$
- für alle $i < n$ gilt: $C_i \vdash_{\mathcal{M}} C_{i+1}$

Dann heißt $C_0, C_1, \ldots, C_n$ eine **Rechnung** der Länge n von C_0 nach C_n.

Turing-Maschine können Funktionen berechnen

Definition 3.6 (TM-berechenbare Funktion)

Sei Σ_0 ein Alphabet mit $\# \notin \Sigma_0$.
Eine (partielle) Funktion

$$f : (\Sigma_0{}^*)^m \rightarrow (\Sigma_0{}^*)^n$$

heißt **DTM-berechenbar**, falls:

Es existiert eine determinierte Turing-Maschine $\mathcal{M} = (K, \Sigma, \delta, s)$

- mit $\Sigma_0 \subseteq \Sigma$,
- so daß für alle $w_1, \ldots, w_m, u_1, \ldots, u_n \in \Sigma_0{}^*$ gilt:

Turing-Maschine können Funktionen berechnen

Definition 3.6 (TM-berechenbare Funktion)

Sei Σ_0 ein Alphabet mit $\# \notin \Sigma_0$.
Eine (partielle) Funktion

$$f : (\Sigma_0{}^*)^m \rightarrow (\Sigma_0{}^*)^n$$

heißt **DTM-berechenbar**, falls:

Es existiert eine determinierte Turing-Maschine $\mathcal{M} = (K, \Sigma, \delta, s)$

- mit $\Sigma_0 \subseteq \Sigma$,
- so daß für alle $w_1, \ldots, w_m,\ u_1, \ldots, u_n \in \Sigma_0{}^*$ gilt:
  - $f(w_1, \ldots, w_m) = (u_1, \ldots, u_n)$ gdw
    $s, \#w_1\# \ldots \#w_m\# \vdash_{\mathcal{M}}^* h, \#u_1\# \ldots \#u_n\#$
  - $f(w_1, \ldots, w_m)$ ist undefiniert gdw
    $\mathcal{M}$ gestartet mit $s, \#w_1\# \ldots \#w_m\#$ hält nicht (läuft unendlich oder hängt)

Turing-Maschine können Funktionen berechnen

Definition 3.6 (TM-berechenbare Funktion)

Sei Σ_0 ein Alphabet mit $\# \notin \Sigma_0$.
Eine (partielle) Funktion

$$f : (\Sigma_0{}^*)^m \rightarrow (\Sigma_0{}^*)^n$$

heißt **DTM-berechenbar**, falls:

Es existiert eine determinierte Turing-Maschine $\mathcal{M} = (\ K, \Sigma, \delta, s\ )$

- mit $\Sigma_0 \subseteq \Sigma$,
- so daß für alle $w_1, \ldots, w_m,\ u_1, \ldots, u_n \in \Sigma_0{}^*$ gilt:
  - $f(w_1, \ldots, w_m) = (u_1, \ldots, u_n)$ gdw
    $s, \#w_1\# \ldots \#w_m\# \vdash_{\mathcal{M}}^* h, \#u_1\# \ldots \#u_n\#$
  - $f(w_1, \ldots, w_m)$ ist undefiniert gdw
    $\mathcal{M}$ gestartet mit $s, \#w_1\# \ldots \#w_m\#$ hält nicht (läuft unendlich oder hängt)

Turing-Maschine können Funktionen berechnen

Definition 3.6 (TM-berechenbare Funktion)

Sei Σ_0 ein Alphabet mit $\# \notin \Sigma_0$.
Eine (partielle) Funktion

$$f : ({\Sigma_0}^*)^m \rightarrow ({\Sigma_0}^*)^n$$

heißt **DTM-berechenbar**, falls:

Es existiert eine determinierte Turing-Maschine $\mathcal{M} = (\ K, \Sigma, \delta, s\ )$

- mit $\Sigma_0 \subseteq \Sigma$,
- so daß für alle $w_1, \ldots, w_m,\ u_1, \ldots, u_n \in {\Sigma_0}^*$ gilt:
  - $f(w_1, \ldots, w_m) = (u_1, \ldots, u_n)$ gdw
    $s, \#w_1\# \ldots \#w_m\# \vdash^*_{\mathcal{M}} h, \#u_1\# \ldots \#u_n\#$
  - $f(w_1, \ldots, w_m)$ ist undefiniert gdw
    $\mathcal{M}$ gestartet mit $s, \#w_1\# \ldots \#w_m\#$ hält nicht (läuft unendlich oder hängt)

Turing-Maschine können Funktionen berechnen

Definition 3.6 (TM-berechenbare Funktion)

Sei Σ_0 ein Alphabet mit $\# \notin \Sigma_0$.
Eine (partielle) Funktion

$$f : ({\Sigma_0}^*)^m \rightarrow ({\Sigma_0}^*)^n$$

heißt **DTM-berechenbar**, falls:

Es existiert eine determinierte Turing-Maschine $\mathcal{M} = (K, \Sigma, \delta, s)$

- mit $\Sigma_0 \subseteq \Sigma$,
- so daß für alle $w_1, \ldots, w_m,\ u_1, \ldots, u_n \in {\Sigma_0}^*$ gilt:
  - $f(w_1, \ldots, w_m) = (u_1, \ldots, u_n)$ gdw
    $s, \#w_1\# \ldots \#w_m\# \vdash_{\mathcal{M}}^* h, \#u_1\# \ldots \#u_n\#$
  - $f(w_1, \ldots, w_m)$ ist undefiniert gdw
    $\mathcal{M}$ gestartet mit $s, \#w_1\# \ldots \#w_m\#$ hält nicht (läuft unendlich oder hängt)

Turing-Maschine können Funktionen berechnen

Vorsicht

Wir betrachten Turing-Maschinen hier unter einem anderen Aspekt als alle bisherigen Automaten:

- Bei endlichen Automaten und Pushdown-Automaten haben wir untersucht, **welche Sprachen sie akzeptieren**.
- Bei Turing-Maschinen untersuchen wir,
  - welche Sprachen sie akzeptieren und
  - welche Funktionen sie berechnen.

Akzeptieren ist Spezialfall von Berechnen

Turing-Maschine können Funktionen berechnen

Vorsicht

Wir betrachten Turing-Maschinen hier unter einem anderen Aspekt als alle bisherigen Automaten:

- Bei endlichen Automaten und Pushdown-Automaten haben wir untersucht, **welche Sprachen sie akzeptieren**.
- Bei Turing-Maschinen untersuchen wir,
  - welche Sprachen sie akzeptieren und
  - **welche Funktionen sie berechnen.**

Akzeptieren ist Spezialfall von Berechnen

Turing-Maschine können Funktionen berechnen

Vorsicht

Wir betrachten Turing-Maschinen hier unter einem anderen Aspekt als alle bisherigen Automaten:

- Bei endlichen Automaten und Pushdown-Automaten haben wir untersucht, **welche Sprachen sie akzeptieren**.
- Bei Turing-Maschinen untersuchen wir,
  - welche Sprachen sie akzeptieren und
  - **welche Funktionen sie berechnen.**

Akzeptieren ist Spezialfall von Berechnen

Turing-Maschine können Funktionen berechnen

Vorsicht

Wir betrachten Turing-Maschinen hier unter einem anderen Aspekt als alle bisherigen Automaten:

- Bei endlichen Automaten und Pushdown-Automaten haben wir untersucht, **welche Sprachen sie akzeptieren**.
- Bei Turing-Maschinen untersuchen wir,
  - welche Sprachen sie akzeptieren und
  - welche Funktionen sie berechnen.

Akzeptieren ist Spezialfall von Berechnen

Turing-Maschine können Funktionen berechnen

Vorsicht

Wir betrachten Turing-Maschinen hier unter einem anderen Aspekt als alle bisherigen Automaten:

- Bei endlichen Automaten und Pushdown-Automaten haben wir untersucht, **welche Sprachen sie akzeptieren**.
- Bei Turing-Maschinen untersuchen wir,
  - welche Sprachen sie akzeptieren und
  - **welche Funktionen sie berechnen.**

Akzeptieren ist Spezialfall von Berechnen

Turing-Maschine können Funktionen berechnen

Vorsicht

Wir betrachten Turing-Maschinen hier unter einem anderen Aspekt als alle bisherigen Automaten:

- Bei endlichen Automaten und Pushdown-Automaten haben wir untersucht, **welche Sprachen sie akzeptieren**.
- Bei Turing-Maschinen untersuchen wir,
  - welche Sprachen sie akzeptieren und
  - **welche Funktionen sie berechnen**.

Akzeptieren ist Spezialfall von Berechnen

Turing-Maschine können Funktionen berechnen

Vorsicht

Wir betrachten Turing-Maschinen hier unter einem anderen Aspekt als alle bisherigen Automaten:

- Bei endlichen Automaten und Pushdown-Automaten haben wir untersucht, **welche Sprachen sie akzeptieren**.
- Bei Turing-Maschinen untersuchen wir,
  - welche Sprachen sie akzeptieren und
  - **welche Funktionen sie berechnen**.

Akzeptieren ist Spezialfall von Berechnen

Turing-Maschine: Akzeptierte Sprache

Definition 3.7 (Von einer DTM akzeptierte Sprache)

Ein Wort w wird **akzeptiert von einer DTM $\mathcal{M}$**,
falls $\mathcal{M}$ auf Eingabe von w hält
(wobei am Ende der Kopf auf dem ersten Blank rechts von w steht).

Eine Sprache $L \subseteq \Sigma^*$ **wird akzeptiert von einer DTM $\mathcal{M}$**, wenn genau die Wörter aus L aus $\mathcal{M}$ und keine anderen akzeptiert werden.

Achtung

Bei nicht akzeptierten Wörtern muss die DTM nicht halten
Sie darf es sogar nicht!

Turing-Maschine: Akzeptierte Sprache

Definition 3.7 (Von einer DTM akzeptierte Sprache)

Ein Wort w wird **akzeptiert von einer DTM** $\mathcal{M}$,
falls $\mathcal{M}$ auf Eingabe von w hält
(wobei am Ende der Kopf auf dem ersten Blank rechts von w steht).

Eine Sprache $L \subseteq \Sigma^*$ **wird akzeptiert von einer DTM** $\mathcal{M}$, wenn genau die Wörter aus L aus $\mathcal{M}$ und keine anderen akzeptiert werden.

Achtung

Bei nicht akzeptierten Wörtern muss die DTM nicht halten
Sie darf es sogar nicht!

Turing-Maschine: Akzeptierte Sprache

Definition 3.7 (Von einer DTM akzeptierte Sprache)

Ein Wort w wird **akzeptiert von einer DTM $\mathcal{M}$**,
falls $\mathcal{M}$ auf Eingabe von w hält
(wobei am Ende der Kopf auf dem ersten Blank rechts von w steht).

Eine Sprache $L \subseteq \Sigma^*$ **wird akzeptiert von einer DTM $\mathcal{M}$**, wenn genau die Wörter aus L aus $\mathcal{M}$ und keine anderen akzeptiert werden.

Achtung

Bei nicht akzeptierten Wörtern muss die DTM nicht halten
Sie darf es sogar nicht!

Turing-Maschine: Funktionen auf natürlichen Zahlen

Funktionen auf natürlichen Zahlen

- Wir verwenden die **Unärdarstellung**
  Eine Zahl n wird auf dem Band der Maschine durch n senkrechte Striche dargestellt.
- Eine Turing-Maschine $\mathcal{M}$ berechnet eine Funktion

$$f : \mathbb{N}^k \to \mathbb{N}^n$$

in Unärdarstellung wie folgt:

- Wenn $f(i_1, \ldots, i_k) = (j_1, \ldots, j_n)$ ist, dann rechnet $\mathcal{M}$

$$s, \#|^{i_1}\# \ldots \#|^{i_k}\underline{\#} \vdash^*_{\mathcal{M}} h, \#|^{j_1}\# \ldots \#|^{j_n}\underline{\#}$$

- Ist $f(i_1, \ldots, i_k)$ undefiniert, dann hält $\mathcal{M}$ bei Input $\#|^{i_1}\# \ldots \#|^{i_k}\#$ nicht.

Turing-Maschine: Funktionen auf natürlichen Zahlen

Funktionen auf natürlichen Zahlen

- Wir verwenden die **Unärdarstellung**
  Eine Zahl n wird auf dem Band der Maschine durch n senkrechte Striche dargestellt.
- Eine Turing-Maschine $\mathcal{M}$ berechnet eine Funktion

$$f : \mathbb{N}^k \to \mathbb{N}^n$$

in Unärdarstellung wie folgt:

- Wenn $f(i_1, \ldots, i_k) = (j_1, \ldots, j_n)$ ist, dann rechnet $\mathcal{M}$

$$s, \#|^{i_1}\#\ldots\#|^{i_k}\underline{\#} \vdash^*_{\mathcal{M}} h, \#|^{j_1}\#\ldots\#|^{j_n}\underline{\#}$$

- Ist $f(i_1, \ldots, i_k)$ undefiniert, dann hält $\mathcal{M}$ bei Input $\#|^{i_1}\#\ldots\#|^{i_k}\#$ nicht.

Turing-Maschine: Funktionen auf natürlichen Zahlen

Funktionen auf natürlichen Zahlen

- Wir verwenden die **Unärdarstellung**
  Eine Zahl n wird auf dem Band der Maschine durch n senkrechte Striche dargestellt.
- Eine Turing-Maschine $\mathcal{M}$ berechnet eine Funktion

$$f: \mathbb{N}^k \to \mathbb{N}^n$$

in Unärdarstellung wie folgt:

- Wenn $f(i_1, \ldots, i_k) = (j_1, \ldots, j_n)$ ist, dann rechnet $\mathcal{M}$

$$s, \#|^{i_1}\#\ldots\#|^{i_k}\underline{\#} \vdash^*_{\mathcal{M}} h, \#|^{j_1}\#\ldots\#|^{j_n}\underline{\#}$$

- Ist $f(i_1, \ldots, i_k)$ undefiniert, dann hält $\mathcal{M}$ bei Input $\#|^{i_1}\#\ldots\#|^{i_k}\#$ nicht.

Turing-Maschine: Funktionen auf natürlichen Zahlen

Funktionen auf natürlichen Zahlen

- Wir verwenden die **Unärdarstellung**
  Eine Zahl n wird auf dem Band der Maschine durch n senkrechte Striche dargestellt.
- Eine Turing-Maschine $\mathcal{M}$ berechnet eine Funktion

$$f: \mathbb{N}^k \to \mathbb{N}^n$$

  in Unärdarstellung wie folgt:
  - Wenn $f(i_1, \ldots, i_k) = (j_1, \ldots, j_n)$ ist, dann rechnet $\mathcal{M}$

$$s, \#|^{i_1}\#\ldots\#|^{i_k}\underline{\#} \vdash^*_{\mathcal{M}} h, \#|^{j_1}\#\ldots\#|^{j_n}\underline{\#}$$

  - Ist $f(i_1, \ldots, i_k)$ undefiniert, dann hält $\mathcal{M}$ bei Input $\#|^{i_1}\#\ldots\#|^{i_k}\#$ nicht.

Turing-Maschine: Funktionen auf natürlichen Zahlen

Definition 3.8

- $\mathbf{TM}^{part}$ ist die Menge der partiellen TM-berechenbaren Funktionen $f : \mathbb{N}^k \rightarrow \mathbb{N}$
- $\mathbf{TM}$ ist die Menge der totalen TM-berechenbaren Funktionen $f : \mathbb{N}^k \rightarrow \mathbb{N}$

Turing-Maschine: Funktionen auf natürlichen Zahlen

Achtung: Einschränkung

In der Definition von TM und TM^{part} haben wir uns eingeschränkt:

- nur Funktionen über natürliche Zahlen
- nur Funktionen mit einstelligem Wertebereich

Das ist keine echte Einschränkung

Elemente (Wörter) aus anderen Definitions- und Wertebereiche können als natürliche Zahlen kodiert werden.

Turing-Maschine: Funktionen auf natürlichen Zahlen

Achtung: Einschränkung

In der Definition von TM und TM^{part} haben wir uns eingeschränkt:

- nur Funktionen über natürliche Zahlen
- nur Funktionen mit einstelligem Wertebereich

Das ist keine echte Einschränkung

Elemente (Wörter) aus anderen Definitions- und Wertebereiche können als natürliche Zahlen kodiert werden.

Turing-Maschine: Funktionen auf natürlichen Zahlen

Achtung: Einschränkung

In der Definition von TM und TM^{part} haben wir uns eingeschränkt:

- nur Funktionen über natürliche Zahlen
- nur Funktionen mit einstelligem Wertebereich

Das ist keine echte Einschränkung

Elemente (Wörter) aus anderen Definitions- und Wertebereiche können als natürliche Zahlen kodiert werden.

Turing-Maschine: Funktionen auf natürlichen Zahlen

Achtung: Einschränkung

In der Definition von TM und TM^{part} haben wir uns eingeschränkt:

- nur Funktionen über natürliche Zahlen
- nur Funktionen mit einstelligem Wertebereich

Das ist keine echte Einschränkung

Elemente (Wörter) aus anderen Definitions- und Wertebereiche können als natürliche Zahlen kodiert werden.

Turing-Maschine: Funktionen auf natürlichen Zahlen

Achtung: Einschränkung

In der Definition von TM und TM^{part} haben wir uns eingeschränkt:

- nur Funktionen über natürliche Zahlen
- nur Funktionen mit einstelligem Wertebereich

Das ist keine echte Einschränkung

Elemente (Wörter) aus anderen Definitions- und Wertebereiche können als natürliche Zahlen kodiert werden.

Variationen von Turing-Maschinen

Standard-DTM

Die Turing-Maschine, die wir bisher kennen, . . .

- ist determiniert
- hat ein einseitig unbeschränktes Band (**Halbband**).

Ab jetzt nennen wir sie auch:
Standard-Turing-Maschine (Standard-DTM oder kurz DTM)

Variationen

- **zweiseitig** unbeschränktes Band (kein Hängen)
- **mehrere** Bänder
- **indeterminierte** Turing-Maschinen

Variationen von Turing-Maschinen

Standard-DTM

Die Turing-Maschine, die wir bisher kennen, . . .

- ist determiniert
- hat ein einseitig unbeschränktes Band (**Halbband**).

Ab jetzt nennen wir sie auch:
Standard-Turing-Maschine (Standard-DTM oder kurz DTM)

Variationen

- **zweiseitig** unbeschränktes Band (kein Hängen)
- **mehrere** Bänder
- **indeterminierte** Turing-Maschinen

Variationen von Turing-Maschinen

Standard-DTM

Die Turing-Maschine, die wir bisher kennen, . . .

- ist determiniert
- hat ein einseitig unbeschränktes Band (**Halbband**).

Ab jetzt nennen wir sie auch:
Standard-Turing-Maschine (Standard-DTM oder kurz DTM)

Variationen

- **zweiseitig** unbeschränktes Band (kein Hängen)
- **mehrere** Bänder
- **indeterminierte** Turing-Maschinen

Variationen von Turing-Maschinen

Standard-DTM

Die Turing-Maschine, die wir bisher kennen, . . .

- ist determiniert
- hat ein einseitig unbeschränktes Band (**Halbband**).

Ab jetzt nennen wir sie auch:
Standard-Turing-Maschine (Standard-DTM oder kurz DTM)

Variationen

- **zweiseitig** unbeschränktes Band (kein Hängen)
- **mehrere** Bänder
- **indeterminierte** Turing-Maschinen

Variationen von Turing-Maschinen

Standard-DTM

Die Turing-Maschine, die wir bisher kennen, . . .

- ist determiniert
- hat ein einseitig unbeschränktes Band (**Halbband**).

Ab jetzt nennen wir sie auch:
Standard-Turing-Maschine (Standard-DTM oder kurz DTM)

Variationen

- **zweiseitig** unbeschränktes Band (kein Hängen)
- **mehrere** Bänder
- **indeterminierte** Turing-Maschinen

Variationen von Turing-Maschinen

Standard-DTM

Die Turing-Maschine, die wir bisher kennen, . . .

- ist determiniert
- hat ein einseitig unbeschränktes Band (**Halbband**).

Ab jetzt nennen wir sie auch:
Standard-Turing-Maschine (Standard-DTM oder kurz DTM)

Variationen

- **zweiseitig** unbeschränktes Band (kein Hängen)
- **mehrere** Bänder
- **indeterminierte** Turing-Maschinen

Variationen von Turing-Maschinen

Standard-DTM

Die Turing-Maschine, die wir bisher kennen, . . .

- ist determiniert
- hat ein einseitig unbeschränktes Band (**Halbband**).

Ab jetzt nennen wir sie auch:
Standard-Turing-Maschine (Standard-DTM oder kurz DTM)

Variationen

- **zweiseitig** unbeschränktes Band (kein Hängen)
- **mehrere** Bänder
- **indeterminierte** Turing-Maschinen

Indeterminierte Turing-Maschine

Definition 3.9 (Indeterminierte Turing-Maschine, NTM)

Eine **indeterminierte Turing-Maschine** $\mathcal{M}$ ist ein Tupel

$$M = (K, \Sigma, \Delta, s)$$

Dabei sind K, Σ, s definiert wie bei determinierten Turing-Maschinen.

Übergangs**relation**:

$$\Delta \subseteq (K \times \Sigma) \times \Big((K \cup \{h\}) \times (\Sigma \cup \{L, R\})\Big)$$

Indeterminierte Turing-Maschine

Definition 3.9 (Indeterminierte Turing-Maschine, NTM)

Eine **indeterminierte Turing-Maschine** $\mathcal{M}$ ist ein Tupel

$$M = (K, \Sigma, \Delta, s)$$

Dabei sind K, Σ, s definiert wie bei determinierten Turing-Maschinen.

Übergangs**relation**:

$$\Delta \subseteq (K \times \Sigma) \times \big((K \cup \{h\}) \times (\Sigma \cup \{L, R\})\big)$$

Indeterminierte Turing-Maschine

Definition 3.9 (Indeterminierte Turing-Maschine, NTM)

Eine **indeterminierte Turing-Maschine** $\mathcal{M}$ ist ein Tupel

$$M = (K, \Sigma, \Delta, s)$$

Dabei sind K, Σ, s definiert wie bei determinierten Turing-Maschinen.

Übergangs**relation**:

$$\Delta \subseteq (K \times \Sigma) \times \Big((K \cup \{h\}) \times (\Sigma \cup \{L, R\}) \Big)$$

Indeterminierte Turing-Maschine

Mehrere Nachfolgekonfigurationen

Konfigurationen sind definiert wie bei DTMs.

Nun kann eine Konfiguration aber mehrere mögliche Nachfolgekonfigurationen haben.

Indeterminierte Turing-Maschine

Mehrere Nachfolgekonfigurationen

Konfigurationen sind definiert wie bei DTMs.

Nun kann eine Konfiguration aber mehrere mögliche Nachfolgekonfigurationen haben.

Indeterminierte Turing-Maschine

Definition 3.10 (NTM: Halten, Hängen, Akzeptieren)

Sei $\mathcal{M} = (K, \Sigma, \Delta, s_0)$ eine indeterminierte Turing-Maschine.

- $\mathcal{M}$ **hält** bei Input w, falls es **unter den möglichen Rechnungen**, die $\mathcal{M}$ wählen kann, **eine gibt**, so daß $\mathcal{M}$ eine Haltekonfiguration erreicht.
- $\mathcal{M}$ **hängt** in einer Konfiguration, wenn es keine (durch Δ definierte) Nachfolgekonfiguration gibt.
- $\mathcal{M}$ **akzeptiert** ein Wort w, falls sie **von $s, \#w\underline{\#}$ aus einen Haltezustand erreichen kann**, und $\mathcal{M}$ akzeptiert eine Sprache L, wenn sie genau alle Wörter $w \in L$ akzeptiert.

Indeterminierte Turing-Maschine

Definition 3.10 (NTM: Halten, Hängen, Akzeptieren)

Sei $\mathcal{M} = (K, \Sigma, \Delta, s_0)$ eine indeterminierte Turing-Maschine.

- $\mathcal{M}$ **hält** bei Input w, falls es **unter den möglichen Rechnungen**, die $\mathcal{M}$ wählen kann, **eine gibt**, so daß $\mathcal{M}$ eine Haltekonfiguration erreicht.
- $\mathcal{M}$ **hängt** in einer Konfiguration, wenn es keine (durch Δ definierte) Nachfolgekonfiguration gibt.
- $\mathcal{M}$ **akzeptiert** ein Wort w, falls sie **von** $s, \#w\underline{\#}$ **aus einen Haltezustand erreichen kann**, und $\mathcal{M}$ akzeptiert eine Sprache L, wenn sie genau alle Wörter $w \in L$ akzeptiert.

Indeterminierte Turing-Maschine

Definition 3.10 (NTM: Halten, Hängen, Akzeptieren)

Sei $\mathcal{M} = (K, \Sigma, \Delta, s_0)$ eine indeterminierte Turing-Maschine.

- $\mathcal{M}$ **hält** bei Input w, falls es **unter den möglichen Rechnungen**, die $\mathcal{M}$ wählen kann, **eine gibt**, so daß $\mathcal{M}$ eine Haltekonfiguration erreicht.
- $\mathcal{M}$ **hängt** in einer Konfiguration, wenn es keine (durch Δ definierte) Nachfolgekonfiguration gibt.
- $\mathcal{M}$ **akzeptiert** ein Wort w, falls sie **von** $s, \#w\underline{\#}$ **aus einen Haltezustand erreichen kann**, und $\mathcal{M}$ akzeptiert eine Sprache L, wenn sie genau alle Wörter $w \in L$ akzeptiert.

Indeterminierte Turing-Maschine

Bemerkung

Wenn es nicht nur darauf ankommt, ob die Maschine hält, sondern auch mit welchem Bandinhalt:

Welche der vielen Haltekonfigurationen sollte dann gelten?

Um dies Problem zu umgehen, übertragen wir die Begriffe des *Entscheidens* und *Aufzählens* **nicht** auf NTM. Im Allgemeinen verwendet man NTM auch nicht dazu, Funktionen zu berechnen.

Indeterminierte Turing-Maschine

Bemerkung

Wenn es nicht nur darauf ankommt, ob die Maschine hält, sondern auch mit welchem Bandinhalt:

Welche der vielen Haltekonfigurationen sollte dann gelten?

Um dies Problem zu umgehen, übertragen wir die Begriffe des *Entscheidens* und *Aufzählens* **nicht** auf NTM. Im Allgemeinen verwendet man NTM auch nicht dazu, Funktionen zu berechnen.

Indeterminierte Turing-Maschine

Wie rechnet eine indeterminierte Turing-Maschine?

- Die Regeln einer determinierten DTM kann man sich als Programm (aus sehr einfachen Schritten) vorstellen.
- **Bei NTM ist das anders!**
- **Eine NTM ist nicht einfach eine Maschine, die immer richtig rät!**

Dieselbe Diskussion hatten wir bei indeterminierten endlichen Automaten.

Indeterminierte Turing-Maschine

Wie rechnet eine indeterminierte Turing-Maschine?

- Die Regeln einer determinierten DTM kann man sich als Programm (aus sehr einfachen Schritten) vorstellen.
- **Bei NTM ist das anders!**
- **Eine NTM ist nicht einfach eine Maschine, die immer richtig rät!**

Dieselbe Diskussion hatten wir bei indeterminierten endlichen Automaten.

Indeterminierte Turing-Maschine

Wie rechnet eine indeterminierte Turing-Maschine?

- Die Regeln einer determinierten DTM kann man sich als Programm (aus sehr einfachen Schritten) vorstellen.
- **Bei NTM ist das anders!**
- **Eine NTM ist nicht einfach eine Maschine, die immer richtig rät!**

Dieselbe Diskussion hatten wir bei indeterminierten endlichen Automaten.

Indeterminierte Turing-Maschine

Wie rechnet eine indeterminierte Turing-Maschine?

- Die Regeln einer determinierten DTM kann man sich als Programm (aus sehr einfachen Schritten) vorstellen.
- **Bei NTM ist das anders!**
- **Eine NTM ist nicht einfach eine Maschine, die immer richtig rät!**

Dieselbe Diskussion hatten wir bei indeterminierten endlichen Automaten.

Indeterminierte Turing-Maschine

Vorstellung von einer NTM

- Korrekte Vorstellung:
  - Übergänge von Konfiguration zu Nachfolgekonfiguration entspr. Regelmenge
  - **plus Suchverfahren!**

  oder: Eine NTM beschreitet alle möglichen Rechenwege parallel.
- **Eine NTM akzeptiert ein Wort, wenn es mindestens einen Berechnungsweg gibt, der in einer Haltekonfiguration endet.**
- Sprechweise **“Die NTM rät”**: Wir verwenden diese Sprechweise, sie ist aber mit Vorsicht zu genießen!

Indeterminierte Turing-Maschine

Vorstellung von einer NTM

- Korrekte Vorstellung:
  - Übergänge von Konfiguration zu Nachfolgekonfiguration entspr. Regelmenge
  - **plus Suchverfahren!**

  oder: Eine NTM beschreitet alle möglichen Rechenwege parallel.

- **Eine NTM akzeptiert ein Wort, wenn es mindestens einen Berechnungsweg gibt, der in einer Haltekonfiguration endet.**

- Sprechweise **“Die NTM rät”**: Wir verwenden diese Sprechweise, sie ist aber mit Vorsicht zu genießen!

Indeterminierte Turing-Maschine

Vorstellung von einer NTM

- Korrekte Vorstellung:
  - Übergänge von Konfiguration zu Nachfolgekonfiguration entspr. Regelmenge
  - **plus Suchverfahren!**

  oder: Eine NTM beschreitet alle möglichen Rechenwege parallel.

- **Eine NTM akzeptiert ein Wort, wenn es mindestens einen Berechnungsweg gibt, der in einer Haltekonfiguration endet.**

- Sprechweise **“Die NTM rät”**: Wir verwenden diese Sprechweise, sie ist aber mit Vorsicht zu genießen!

Indeterminierte Turing-Maschine

Vorstellung von einer NTM

- Korrekte Vorstellung:
  - Übergänge von Konfiguration zu Nachfolgekonfiguration entspr. Regelmenge
  - **plus Suchverfahren!**

  oder: Eine NTM beschreitet alle möglichen Rechenwege parallel.

- **Eine NTM akzeptiert ein Wort, wenn es mindestens einen Berechnungsweg gibt, der in einer Haltekonfiguration endet.**

- Sprechweise **“Die NTM rät”**: Wir verwenden diese Sprechweise, sie ist aber mit Vorsicht zu genießen!

Indeterminierte Turing-Maschine

Vorstellung von einer NTM

- Korrekte Vorstellung:
  - Übergänge von Konfiguration zu Nachfolgekonfiguration entspr. Regelmenge
  - **plus Suchverfahren!**

  oder: Eine NTM beschreitet alle möglichen Rechenwege parallel.

- **Eine NTM akzeptiert ein Wort, wenn es mindestens einen Berechnungsweg gibt, der in einer Haltekonfiguration endet.**

- Sprechweise **“Die NTM rät”**: Wir verwenden diese Sprechweise, sie ist aber mit Vorsicht zu genießen!

Indeterminierte Turing-Maschine

Vorstellung von einer NTM

- Korrekte Vorstellung:
  - Übergänge von Konfiguration zu Nachfolgekonfiguration entspr. Regelmenge
  - **plus Suchverfahren!**

  oder: Eine NTM beschreitet alle möglichen Rechenwege parallel.

- **Eine NTM akzeptiert ein Wort, wenn es mindestens einen Berechnungsweg gibt, der in einer Haltekonfiguration endet.**

- Sprechweise **"Die NTM rät"**: Wir verwenden diese Sprechweise, sie ist aber mit Vorsicht zu genießen!

Indeterminierte Turing-Maschine

Beispiel 3.11

Sei

$$L = \{|^n \mid n \text{ ist nicht prim und } n \geq 2\}$$

Eine NTM kann diese Sprache wie folgt azeptieren:

1. Eine Zahl „raten“ und (nach rechts) aufs Band schreiben.
2. Noch eine Zahl „raten“ und daneben schreiben.
3. Die beiden Zahlen miteinander multiplizieren.
4. Das Ergebnis mit der Eingabe vergleichen.
5. Genau dann, wenn beide gleich sind, anhalten

Indeterminierte Turing-Maschine

Beispiel 3.11

Sei

$$L = \{|^n \mid n \text{ ist nicht prim und } n \geq 2\}$$

Eine NTM kann diese Sprache wie folgt azeptieren:

1. Eine Zahl „raten“ und (nach rechts) aufs Band schreiben.
2. Noch eine Zahl „raten“ und daneben schreiben.
3. Die beiden Zahlen miteinander multiplizieren.
4. Das Ergebnis mit der Eingabe vergleichen.
5. Genau dann, wenn beide gleich sind, anhalten

Indeterminierte Turing-Maschine

Beispiel 3.11

Sei

$$L = \{|^n \mid n \text{ ist nicht prim und } n \geq 2\}$$

Eine NTM kann diese Sprache wie folgt azeptieren:

1. Eine Zahl „raten“ und (nach rechts) aufs Band schreiben.
2. Noch eine Zahl „raten“ und daneben schreiben.
3. Die beiden Zahlen miteinander multiplizieren.
4. Das Ergebnis mit der Eingabe vergleichen.
5. Genau dann, wenn beide gleich sind, anhalten

Indeterminierte Turing-Maschine

Beispiel 3.11

Sei

$$L = \{|^n \mid n \text{ ist nicht prim und } n \geq 2\}$$

Eine NTM kann diese Sprache wie folgt azeptieren:

1. Eine Zahl „raten“ und (nach rechts) aufs Band schreiben.
2. Noch eine Zahl „raten“ und daneben schreiben.
3. Die beiden Zahlen miteinander multiplizieren.
4. Das Ergebnis mit der Eingabe vergleichen.
5. Genau dann, wenn beide gleich sind, anhalten

Indeterminierte Turing-Maschine

Beispiel 3.11

Sei

$$L = \{|^n \mid n \text{ ist nicht prim und } n \geq 2\}$$

Eine NTM kann diese Sprache wie folgt azeptieren:

1. Eine Zahl „raten“ und (nach rechts) aufs Band schreiben.
2. Noch eine Zahl „raten“ und daneben schreiben.
3. Die beiden Zahlen miteinander multiplizieren.
4. Das Ergebnis mit der Eingabe vergleichen.
5. Genau dann, wenn beide gleich sind, anhalten

Indeterminierte Turing-Maschine

Beispiel 3.11

Sei

$$L = \{|^n \mid n \text{ ist nicht prim und } n \geq 2\}$$

Eine NTM kann diese Sprache wie folgt azeptieren:

1. Eine Zahl „raten“ und (nach rechts) aufs Band schreiben.
2. Noch eine Zahl „raten“ und daneben schreiben.
3. Die beiden Zahlen miteinander multiplizieren.
4. Das Ergebnis mit der Eingabe vergleichen.
5. Genau dann, wenn beide gleich sind, anhalten

Indeterminierte Turing-Maschine

Theorem 3.12 (Simulation von NTM durch DTM)

Jede Sprache, die von einer indeterminierten Turing-Maschine akzeptiert wird, wird auch von einer Standard-DTM akzeptiert.

Universelle Turing-Maschine

Vergleich Turing-Maschine / „normaler“ Computer

Turing-Maschinen sind sehr mächtig.

Wie mächtig sind sie wirklich?

- Eine Turing-Maschine hat eine vorgegebenes „Programm“ (Regelmenge)
- „Normale“ Computer können beliebige Programme ausführen.

Tatsächlich geht das mit Turing-Maschinen auch!

Universelle Turing-Maschine

Vergleich Turing-Maschine / „normaler“ Computer

Turing-Maschinen sind sehr mächtig.

Wie mächtig sind sie wirklich?

- Eine Turing-Maschine hat eine vorgegebenes „Programm“ (Regelmenge)
- „Normale“ Computer können beliebige Programme ausführen.

Tatsächlich geht das mit Turing-Maschinen auch!

Universelle Turing-Maschine

Vergleich Turing-Maschine / „normaler“ Computer

Turing-Maschinen sind sehr mächtig.

Wie mächtig sind sie wirklich?

- Eine Turing-Maschine hat eine vorgegebenes „Programm“ (Regelmenge)
- „Normale“ Computer können beliebige Programme ausführen.

Tatsächlich geht das mit Turing-Maschinen auch!

Universelle Turing-Maschine

Turing-Maschine, die andere TMen simuliert

- Universelle TM $\mathcal{U}$ bekommt als Eingabe:
  - die Regelmenge einer beliebigen Turing-Maschine $\mathcal{M}$ und
  - ein Wort w, auf dem $\mathcal{M}$ rechnen soll.
- $\mathcal{U}$ simuliert $\mathcal{M}$, indem sie jeweils nachschlägt, welchen δ-Übergang $\mathcal{M}$ machen würde.

Universelle Turing-Maschine

Turing-Maschine, die andere TMen simuliert

- Universelle TM $\mathcal{U}$ bekommt als Eingabe:
  - die Regelmenge einer beliebigen Turing-Maschine $\mathcal{M}$ und
  - ein Wort w, auf dem $\mathcal{M}$ rechnen soll.
- $\mathcal{U}$ simuliert $\mathcal{M}$, indem sie jeweils nachschlägt, welchen δ-Übergang $\mathcal{M}$ machen würde.

Universelle Turing-Maschine

Turing-Maschine, die andere TMen simuliert

- Universelle TM $\mathcal{U}$ bekommt als Eingabe:
  - die Regelmenge einer beliebigen Turing-Maschine $\mathcal{M}$ und
  - ein Wort w, auf dem $\mathcal{M}$ rechnen soll.
- $\mathcal{U}$ simuliert $\mathcal{M}$, indem sie jeweils nachschlägt, welchen δ-Übergang $\mathcal{M}$ machen würde.

Universelle Turing-Maschine

Turing-Maschine, die andere TMen simuliert

- Universelle TM $\mathcal{U}$ bekommt als Eingabe:
  - die Regelmenge einer beliebigen Turing-Maschine $\mathcal{M}$ und
  - ein Wort w, auf dem $\mathcal{M}$ rechnen soll.
- $\mathcal{U}$ simuliert $\mathcal{M}$, indem sie jeweils nachschlägt, welchen δ-Übergang $\mathcal{M}$ machen würde.

Universelle Turing-Maschine

Gödelisierung

Ein Verfahren, jeder Turing-Maschine eine Zahl oder ein Wort (**Gödelzahl** bzw. **Gödelwort**) so zuzuordnen, daß man aus der Zahl bzw. dem Wort die Turing-Maschine effektiv rekonstruieren kann.

Akzeptierbarkeit und Entscheidbarkeit

Akzeptieren

Eine DTM **akzeptiert** eine Sprache L, wenn sie

- für jedes Eingabe-Wort $w \in L$ irgendwann hält
- für jedes Wort $v \notin L$ unendlich lang rechnet oder hängt

Entscheiden

Eine DTM **entscheidet** eine Sprache L, wenn sie

- für jedes Eingabe-Wort $w \in L$ hält mit dem Bandinhalt Y (“Yes”)
- für jedes Wort $v \notin L$ hält mit dem Bandinhalt N (“No”)

Akzeptierbarkeit und Entscheidbarkeit

Akzeptieren

Eine DTM **akzeptiert** eine Sprache L, wenn sie

- für jedes Eingabe-Wort $w \in L$ irgendwann hält
- für jedes Wort $v \notin L$ unendlich lang rechnet oder hängt

Entscheiden

Eine DTM **entscheidet** eine Sprache L, wenn sie

- für jedes Eingabe-Wort $w \in L$ hält mit dem Bandinhalt Y (“Yes”)
- für jedes Wort $v \notin L$ hält mit dem Bandinhalt N (“No”)

Akzeptierbarkeit und Entscheidbarkeit

Definition 3.13 (Entscheidbar)

L sei eine Sprache über Σ_0 mit $\#, N, Y \notin \Sigma_0$.

$M = (\, K, \Sigma, \delta, s \,)$ sei eine DTM mit $\Sigma_0 \subseteq \Sigma$.

M **entscheidet** L, falls für alle $w \in \Sigma_0^*$ gilt:

$$s, \#w\underline{\#} \vdash_M^* \begin{cases} h, \#Y\underline{\#} & \text{falls } w \in L \\ h, \#N\underline{\#} & \text{sonst} \end{cases}$$

L heißt **entscheidbar**, falls es eine DTM gibt, die L entscheidet.

Akzeptierbarkeit und Entscheidbarkeit

Definition 3.13 (Entscheidbar)

L sei eine Sprache über Σ_0 mit $\#, N, Y \notin \Sigma_0$.

$M = (K, \Sigma, \delta, s)$ sei eine DTM mit $\Sigma_0 \subseteq \Sigma$.

M **entscheidet** L, falls für alle $w \in \Sigma_0^*$ gilt:

$$s, \#w\underline{\#} \vdash_M^* \begin{cases} h, \#Y\underline{\#} & \text{falls } w \in L \\ h, \#N\underline{\#} & \text{sonst} \end{cases}$$

L heißt **entscheidbar**, falls es eine DTM gibt, die L entscheidet.

Akzeptierbarkeit und Entscheidbarkeit

Definition 3.13 (Entscheidbar)

L sei eine Sprache über Σ_0 mit $\#, N, Y \notin \Sigma_0$.

$M = (K, \Sigma, \delta, s)$ sei eine DTM mit $\Sigma_0 \subseteq \Sigma$.

M **entscheidet** L, falls für alle $w \in \Sigma_0^*$ gilt:

$$s, \#w\underline{\#} \vdash_M^* \begin{cases} h, \#Y\underline{\#} & \text{falls } w \in L \\ h, \#N\underline{\#} & \text{sonst} \end{cases}$$

L heißt **entscheidbar**, falls es eine DTM gibt, die L entscheidet.

Akzeptierbarkeit und Entscheidbarkeit

Definition 3.14 (Akzeptierbar)

L sei eine Sprache über Σ_0 mit $\#, N, Y \notin \Sigma_0$.

$M = (\, K, \Sigma, \delta, s \,)$ sei eine DTM mit $\Sigma_0 \subseteq \Sigma$.

$\mathcal{M}$ **akzeptiert** ein Wort $w \in \Sigma_0^*$,
falls $\mathcal{M}$ bei Input w hält.

$\mathcal{M}$ **akzeptiert die Sprache** L, falls für alle $w \in \Sigma_0^*$ gilt:

$$\mathcal{M} \text{ akzeptiert } w \quad \textit{genau dann wenn} \quad w \in L$$

L heißt **akzeptierbar** (oder auch **semi-entscheidbar**),
falls es eine DTM gibt, die L akzeptiert.

Akzeptierbarkeit und Entscheidbarkeit

Definition 3.14 (Akzeptierbar)

L sei eine Sprache über Σ_0 mit $\#, N, Y \notin \Sigma_0$.

$M = (\, K, \Sigma, \delta, s \,)$ sei eine DTM mit $\Sigma_0 \subseteq \Sigma$.

$\mathcal{M}$ **akzeptiert** ein Wort $w \in \Sigma_0^*$,
falls $\mathcal{M}$ bei Input w hält.

$\mathcal{M}$ **akzeptiert die Sprache** L, falls für alle $w \in \Sigma_0^*$ gilt:

$$\mathcal{M} \text{ akzeptiert } w \quad \textit{genau dann wenn} \quad w \in L$$

L heißt **akzeptierbar** (oder auch **semi-entscheidbar**),
falls es eine DTM gibt, die L akzeptiert.

Akzeptierbarkeit und Entscheidbarkeit

Definition 3.14 (Akzeptierbar)

L sei eine Sprache über Σ_0 mit $\#, N, Y \notin \Sigma_0$.

$M = (\, K, \Sigma, \delta, s \,)$ sei eine DTM mit $\Sigma_0 \subseteq \Sigma$.

$\mathcal{M}$ **akzeptiert** ein Wort $w \in \Sigma_0^*$,
falls $\mathcal{M}$ bei Input w hält.

$\mathcal{M}$ **akzeptiert die Sprache** L, falls für alle $w \in \Sigma_0^*$ gilt:

$$\mathcal{M} \text{ akzeptiert } w \quad \textit{genau dann wenn} \quad w \in L$$

L heißt **akzeptierbar** (oder auch **semi-entscheidbar**),
falls es eine DTM gibt, die L akzeptiert.

Akzeptierbarkeit und Entscheidbarkeit

Definition 3.14 (Akzeptierbar)

L sei eine Sprache über Σ_0 mit $\#, N, Y \notin \Sigma_0$.

$M = (K, \Sigma, \delta, s)$ sei eine DTM mit $\Sigma_0 \subseteq \Sigma$.

$\mathcal{M}$ **akzeptiert** ein Wort $w \in \Sigma_0^*$,
falls $\mathcal{M}$ bei Input w hält.

$\mathcal{M}$ **akzeptiert die Sprache** L, falls für alle $w \in \Sigma_0^*$ gilt:

$$\mathcal{M} \text{ akzeptiert } w \quad \textit{genau dann wenn} \quad w \in L$$

L heißt **akzeptierbar** (oder auch **semi-entscheidbar**),
falls es eine DTM gibt, die L akzeptiert.

Rekursiv aufzählbar

Definition 3.15 (Rekursiv Aufzählbar (recursively enumerable))

L sei eine Sprache über Σ_0 mit $\#, N, Y \notin \Sigma_0$.

$M = (\, K, \Sigma, \delta, s \,)$ sei eine DTM mit $\Sigma_0 \subseteq \Sigma$.

$\mathcal{M}$ **zählt** L **auf**, falls es einen Zustand $q_B \in K$ gibt (den **Blinkzustand**), so daß:

$$L = \{ w \in \Sigma_0^* \mid \exists u \in \Sigma^* : s, \underline{\#} \vdash_{\mathcal{M}}^* q_B, \#w\#u \}$$

L heißt **rekursiv aufzählbar**, falls es eine DTM gibt, die L aufzählt.

Rekursiv aufzählbar

Definition 3.15 (Rekursiv Aufzählbar (recursively enumerable))

L sei eine Sprache über Σ_0 mit $\#, N, Y \notin \Sigma_0$.

$M = (\, K, \Sigma, \delta, s \,)$ sei eine DTM mit $\Sigma_0 \subseteq \Sigma$.

$\mathcal{M}$ **zählt** L **auf**, falls es einen Zustand $q_B \in K$ gibt (den **Blinkzustand**), so daß:

$$L = \{ w \in \Sigma_0^* \mid \exists u \in \Sigma^* : s, \underline{\#} \vdash_{\mathcal{M}}^* q_B, \#w\underline{\#}u \}$$

L heißt **rekursiv aufzählbar**, falls es eine DTM gibt, die L aufzählt.

Rekursiv aufzählbar

Definition 3.15 (Rekursiv Aufzählbar (recursively enumerable))

L sei eine Sprache über Σ_0 mit $\#, N, Y \notin \Sigma_0$.

$M = (\, K, \Sigma, \delta, s \,)$ sei eine DTM mit $\Sigma_0 \subseteq \Sigma$.

$\mathcal{M}$ **zählt** L **auf**, falls es einen Zustand $q_B \in K$ gibt (den **Blinkzustand**), so daß:

$$L = \{ w \in \Sigma_0^* \mid \exists u \in \Sigma^* : s, \underline{\#} \vdash_{\mathcal{M}}^* q_B, \#w\underline{\#}u \}$$

L heißt **rekursiv aufzählbar**, falls es eine DTM gibt, die L aufzählt.

Rekursiv aufzählbar

Achtung: aufzählbar $\neq$ abzählbar.

Unterschied

M abzählbar: Es gibt eine surjektive Abbildung der natürlichen Zahlen auf M

M aufzählbar: Diese Abbildung kann von einer Turing-Maschine berechnet werden.

Wegen Endlichkeit der Wörter und des Alphabets sind alle Sprachen abzählbar.
Aber nicht alle Sprachen sind aufzählbar.

Rekursiv aufzählbar

Achtung: aufzählbar $\neq$ abzählbar.

Unterschied

M **abzählbar:** Es gibt eine surjektive Abbildung der natürlichen Zahlen auf *M*

M **aufzählbar:** Diese Abbildung kann von einer Turing-Maschine berechnet werden.

Wegen Endlichkeit der Wörter und des Alphabets sind alle Sprachen abzählbar.
Aber nicht alle Sprachen sind aufzählbar.

Rekursiv aufzählbar

Achtung: aufzählbar $\neq$ abzählbar.

Unterschied

M **abzählbar:** Es gibt eine surjektive Abbildung der natürlichen Zahlen auf *M*

M **aufzählbar:** Diese Abbildung kann von einer Turing-Maschine berechnet werden.

Wegen Endlichkeit der Wörter und des Alphabets sind alle Sprachen abzählbar.
Aber nicht alle Sprachen sind aufzählbar.

Rekursiv aufzählbar

Achtung: aufzählbar $\neq$ abzählbar.

Unterschied

M **abzählbar:** Es gibt eine surjektive Abbildung der natürlichen Zahlen auf *M*

M **aufzählbar:** Diese Abbildung kann von einer Turing-Maschine berechnet werden.

Wegen Endlichkeit der Wörter und des Alphabets sind alle Sprachen abzählbar.
Aber nicht alle Sprachen sind aufzählbar.

Rekursiv aufzählbar

Achtung: aufzählbar $\neq$ abzählbar.

Unterschied

M **abzählbar:** Es gibt eine surjektive Abbildung der natürlichen Zahlen auf *M*

M **aufzählbar:** Diese Abbildung kann von einer Turing-Maschine berechnet werden.

Wegen Endlichkeit der Wörter und des Alphabets sind alle Sprachen abzählbar.
Aber nicht alle Sprachen sind aufzählbar.

Rekursiv aufzählbar: Beispiele

Beispiel 3.16 (Rekursiv aufzählbar aber nicht entscheidbar)

Folgende Mengen sind rekursiv aufzählbar aber *nicht* entscheidbar:

- Die Menge der Gödelisierungen aller haltenden Turing-Maschinen
- Die Menge aller terminierenden Programme
- Die Menge aller allgemeingültigen prädikatenlogischen Formeln

Rekursiv aufzählbar: Beispiele

Beispiel 3.16 (Rekursiv aufzählbar aber nicht entscheidbar)

Folgende Mengen sind rekursiv aufzählbar aber *nicht* entscheidbar:

- Die Menge der Gödelisierungen aller haltenden Turing-Maschinen
- Die Menge aller terminierenden Programme
- Die Menge aller allgemeingültigen prädikatenlogischen Formeln

Rekursiv aufzählbar: Beispiele

Beispiel 3.16 (Rekursiv aufzählbar aber nicht entscheidbar)

Folgende Mengen sind rekursiv aufzählbar aber *nicht* entscheidbar:

- Die Menge der Gödelisierungen aller haltenden Turing-Maschinen
- Die Menge aller terminierenden Programme
- Die Menge aller allgemeingültigen prädikatenlogischen Formeln

Akzeptierbar gleich rekursiv aufzählbar

Satz 3.17 (Akzeptierbar = Rekursiv Aufzählbar)

Eine Sprache ist genau dann rekursiv aufzählbar, wenn sie akzeptierbar ist.

Entscheidbarkeit und Akzeptierbarkeit

Satz 3.18 (Entscheidbar und akzeptierbar)

Jede entscheidbare Sprache ist akzeptierbar.

Entscheidbarkeit und Akzeptierbarkeit

Satz 3.19 (Komplement einer entscheidbaren Sprache ist entscheidbar)

Das Komplement einer entscheidbaren Sprache ist entscheidbar.

Entscheidbarkeit und Akzeptierbarkeit

Satz 3.20 (Charakterisierung von Entscheidbarkeit)

Eine Sprache L ist genau dann entscheidbar, wenn sie und ihr Komplement akzeptierbar sind.

Rekursiv Aufzählbar = Typ 0

Zur Erinnerung

Formale Sprachen sind vom **Typ 0**, wenn sie durch beliebige Grammatiken (keinerlei Einschränkungen) erzeugt werden können.

Rekursiv Aufzählbar = Typ 0

Satz 3.21 (Rekursiv aufzählbar = Typ 0)

Die rekursiv aufzählbaren Sprachen
(also die durch DTMn akzeptierbaren Sprachen)
sind genau die durch beliebige Grammatiken erzeugten Sprachen
(also die vom Typ 0).

Rekursiv Aufzählbar = Typ 0

Satz 3.21 (Rekursiv aufzählbar = Typ 0)

Die rekursiv aufzählbaren Sprachen
(also die durch DTMn akzeptierbaren Sprachen)
sind genau die durch beliebige Grammatiken erzeugten Sprachen
(also die vom Typ 0).