RWTH AACHEN UNIVERSITY

Berechenbarkeit und Komplexität

Vorlesung im Wintersemester 22/23

Martin Grohe
Lehrstuhl Informatik 7

Folien bis auf kleine Änderungen von Pascal Schweitzer, basierend auf früheren Folien von Martin Grohe und Bertold Vöcking.

Vorlesung 1
Einführung

In BuK beschäftigen wir uns mit den
Grenzen der Berechenbarkeit.

Organisatorisches

Komponenten und Materialien

Die Vorlesung und alle dazugehörigen Veranstaltungen finden in Präsenz statt.

BuK hat folgende Komponenten

- Vorlesungen,
- Globalübungen,
- Kleingruppenübungen (Tutorien),
- Übungsblätter,
- E-Tests.

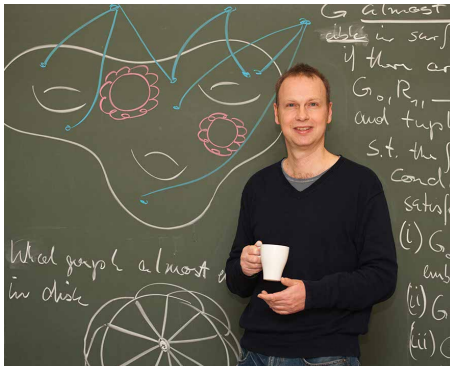
Alle Materialien und ein FAQ sind im Moodle-Raum zu finden.

Bitte sehen Sie regelmäßig in Moodle nach aktuellen Informationen.

Vorlesungen

- Die Vorlesung wird von Martin Grohe gehalten. Die Vorlesungen finden
  - Dienstags (ab 11.10.) um 16:30 Uhr im Großen Hörsaal (AM)
  - Mittwochs (ab 12.10.) um 08:30 Uhr im Großen Hörsaal (AM)

  statt.



Martin Grohe

- Die Vorlesungen werden von der Video-AG der Fachschaft Informatik aufgezeichnet.

Zulassungskriterien

Zulassung zur Klausur

- Es gibt 11 Übungsblätter zu je 15 Punkten.
  - Bonusblatt über Weihnachten
  - Zur Zulassung benötigt man je mindestens 50 % der Punkte aus den Blättern 1–6 (Berechenbarkeit) und aus den Blättern 7–11 (Komplexität).
- Es gibt 12 E-Tests zu je 10 Punkten.
  - Zur Zulassung benötigt man mindestens 50 % der Punkte aus den E-Tests.
  - Weiterhin müssen alle bis auf einen E-Test bearbeitet (mit mehr als 0 Punkten abgeschlossen) werden; bei bis zu 3 nicht bearbeiteten Tests muss ein Bonus-E-Test (nach den regulären E-Tests) bestanden werden.

Klausur

- Erster Termin: Dienstag, der 21.02.2023 um 08:30
- Zweiter Termin: Montag, der 27.03.2023 um 17:00
- Die Anmeldung ist ab dem 15.11. bis zum 15.01. möglich

Übungen

Übungsblätter

- Ausgabe wöchentlich montags bis 18:00 Uhr im Moodle, beginnend nächste Woche (17.10.)
- Abgabe in der folgenden Woche mittwochs um 14:30
- Bearbeitung in Gruppen zu 3 Studierenden
- Abgabe via Moodle als 1 PDF (max 15 MB) **mit Übungsgruppe, Namen und Matrikelnummern im Dokument**
  - Rückgabe der korrigierten Blätter über Moodle
  - Musterlösungen für die Hausaufgaben werden nach der Globalübung hochgeladen.
- Tutoriumsaufgaben werden in Tutorien gemeinsam erarbeitet. Es gibt *keine* Musterlösung für Tutoriumsaufgaben.

E-Tests

- Veröffentlichung und Abgabe montags um 18:00 Uhr, Bearbeitungszeit eine Woche
- Individuell zu bearbeiten

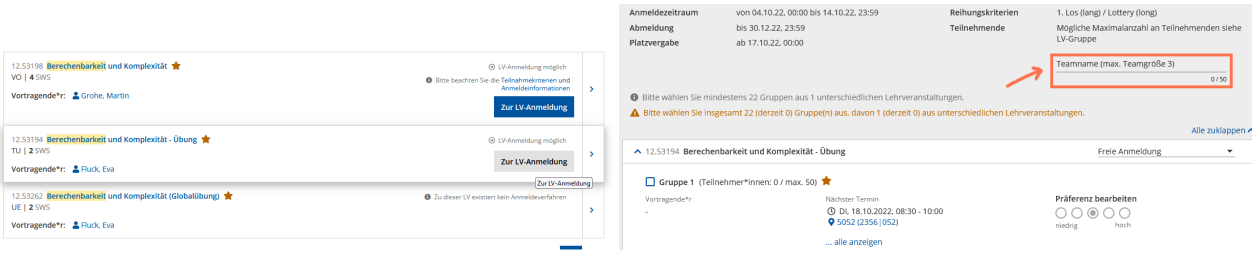
Globalübung

- Mittwochs (ab 26.10.) um 14:30 Uhr im Hörsaal H03 (CARL)
- Ausführliche Diskussion der abgegebenen Übung
  - Wie kommt man auf eine Lösung?
  - Wie sehen alternative Lösungswege aus?
  - Was sind häufige Fehler?

Diese Woche (12.10.): Besprechung Blatt 0, Unterstützung bei der Anmeldung, LaTeX-Einführung

Tutorien

- 22 Tutorien, verteilt auf Dienstag, Donnerstag und Freitag (siehe RWTHonline).
- Englischsprachiges Tutorium: Gruppe 19, Freitag um 14:30 im 5056
- Beginnen in der nächsten Woche (17.10.-21.10.).
- Anmeldung zu Tutorien in RWTHonline
  - Unabhängig von der Anmeldung zur Vorlesung
  - Prioritäten und Teams (siehe Bild rechts) in RWTHonline angeben
  - Abgabegruppen zu 3 Studierenden aus dem selben Tutorium

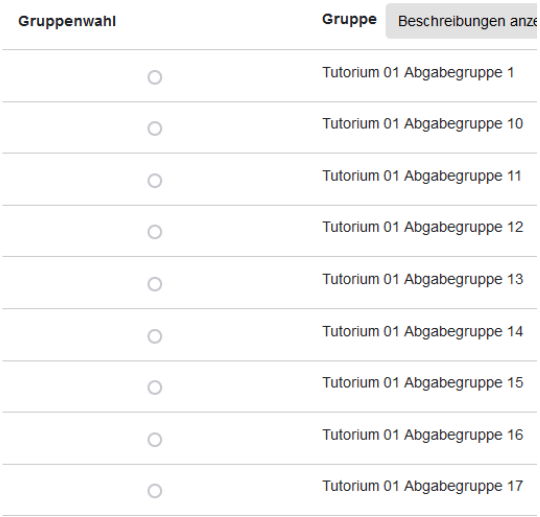


Anmeldefrist: Freitag, der 14. Oktober um 23:59.

Tutorien (forts.)

Abgabegruppen können in Moodle gebildet werden, sobald die Tutorien übertragen wurden.

1. Nutzt ggf. die Abgabegruppen-Börse eures Tutoriums, um eine Gruppe zu finden
2. Tragt euch unter "Abgabegruppen" dann gemeinsam in eine freie Gruppe ein (siehe Bild)



- Ohne Gruppe ist keine Abgabe möglich!
- Frist für Moodle-Gruppen: Montag, der 24.10. um 15:00 Uhr, erst danach Abgabe der Übungen möglich

Anmeldungen

Anmeldungen

Für die Vorlesung brauchen Sie drei unabhängige Anmeldungen

1. Anmeldeverfahren der Vorlesung: Zugang zum Moodle-Raum
2. Anmeldeverfahren Übungen: Zuordnung zu Tutorien
3. Anmeldung zur Klausur

Scheine und Auflagen

- Für einen Schein nimmt man wie alle anderen an der Klausur teil und muss auch die Zulassung erlangen.
- Studierende, die die Vorlesung als Master-Auflage absolvieren müssen, nehmen ebenfalls ganz normal an Übungen und Klausur teil. Das ZPA schaltet in diesem Fall das Anmeldeverfahren in RWTHonline gesondert frei.

Beteiligte

Dozent

Martin Grohe

Assistent*innen

Eva Fluck

Athena Riazsadri

Julia Feith

Tutor*innen

Tabea Hegewald, André Kuslido, Simon Wilmes, Lennard Schober, János Arpasi, Naomi Barth, David Philipps, Kevin Lu, Matthäus Micun, Yaman Faour, Imogen Hergeth, Jonas Groven, Georg Albrecht, Florian Cramer, Jannik Hiller, Larissa Meffert, Magnus Giesbert, Tomáš Novotný, Geonho Yun, Klara Pakhomenko, Arian Kulmer

Was tun bei Fragen?

1. Ist die Frage auch interessant für andere Studierende? Dann poste die Frage bitte ins Diskussionsforum im Moodle.
2. Für Fragen bezüglich der Korrektur wendet euch bitte an die Tutor*in eurer Kleingruppe.
3. Euere Tutor*innen sind auch ansonsten euer erster Anlaufpunkt.
4. Dann erst, oder in dringenden(!) Fällen E-Mail an:

   `buk@lists.rwth-aachen.de`

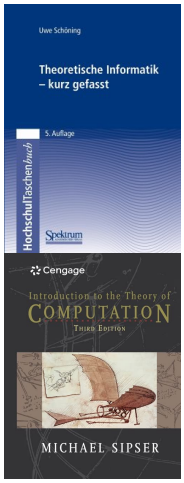
Bitte keine E-Mail direkt an die Assistent*innen oder mich. Diese werden ungelesen gelöscht!

Wie können Sie mich erreichen?

- Immer nach der Vorlesung
- Sprechstunde Dienstags, 14–15 Uhr

Bitte keine E-Mail an mich und keine Telefonanrufe.

Buchempfehlungen



Uwe Schöning. Theoretische Informatik - kurzgefasst. Spektrum Akademischer Verlag, 2001.

Michael Sipser. Introduction to the Theory of Computation, 3rd Edition. Cengage Learning, 2012.

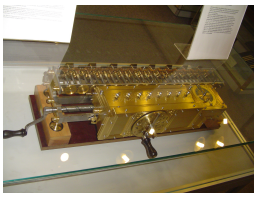
Diese und noch weitere Bücher zum Thema sind zu finden in der Informatikbibliothek, im Handapparat unter BuK.

Zwei weiterführende Bücher sind

- Nigel J. Cutland. Computability: An Introduction to Recursive Function Theory. Cambridge University Press, 1980.
- Sanjeev Arora, Boaz Barak. Computational Complexity. Cambridge University Press, 2009.

Motivation und Übersicht

Rechenmaschinen



1672/1700

User:Kolossos/Wikimedia Commons/CC-BY-SA-3.0

1923

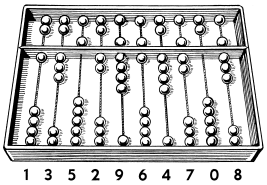
Greg Goebel/Wikimedia Commons/Public Domain

?

~2014

1980

2013



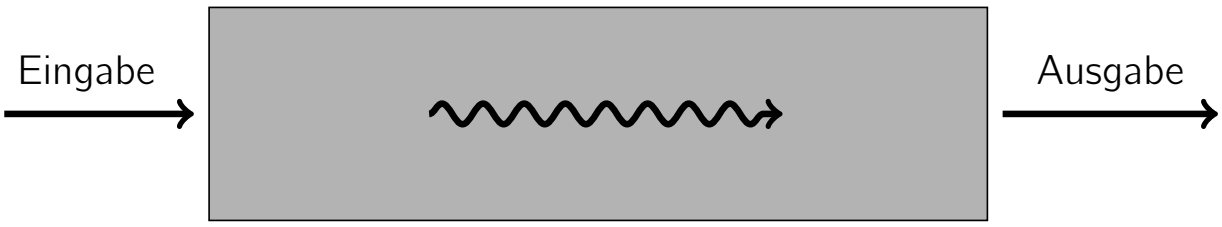
~3000 AC

Pearson Scott Foresman/Wikimedia Commons/Public Domain

1983

Berechenbarkeit – die absoluten Grenzen des Computers

In der Vorlesung interessieren wir uns allgemein für Berechnungsprobleme.



Gibt es überhaupt Probleme, die wir nicht mit dem Computer lösen können?

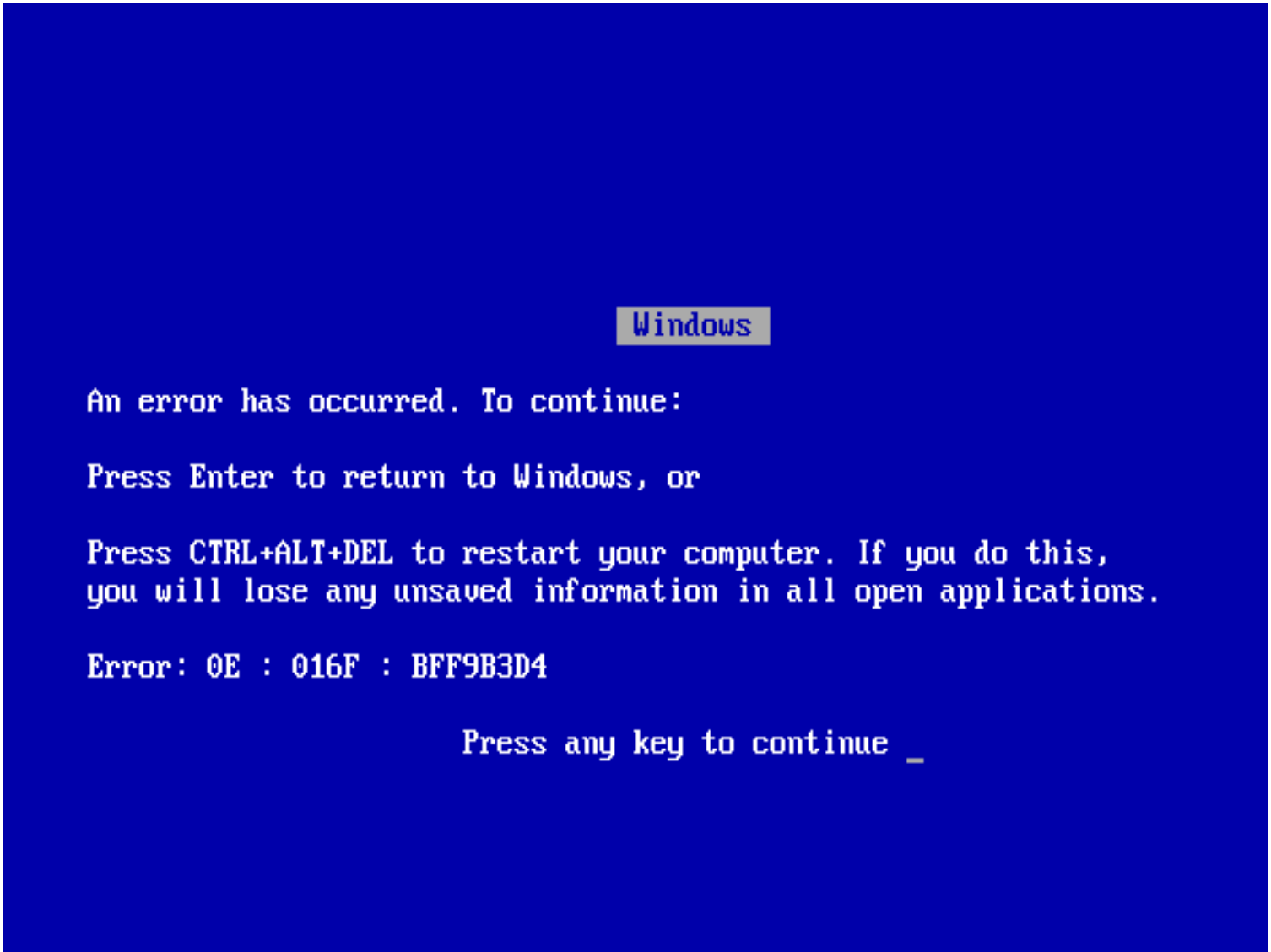
Bessere Frage

Gibt es "algorithmische Probleme" (oder "Berechnungsprobleme"), die ein Computer nicht lösen kann?

Die absoluten Grenzen des Computers

Wir werden sehen:

- Es gibt keinen Algorithmus, der entscheidet, ob ein gegebenes Programm in einen bestimmten Zustand läuft.
  (`Error: 0E : 016F : BFF9B3D4`.)
- Allgemein lässt sich die Funktionsweise von Programmen nur schwer algorithmisch überprüfen.
- Zum Beispiel gibt es keinen Algorithmus, der entscheidet, ob ein gegebenes Programm immer die Summe zweier eingegebenen Zahlen berechnet.



Das Halteproblem

Allgemeines Halteproblem

- Eingabe: Ein Programm in einer wohldefinierten, universellen Programmiersprache (z.B. Java, C++, Python, Scala).
- Frage: Terminiert dieses Programm?

Wir werden beweisen, dass es keinen Algorithmus gibt, der dieses Problem entscheiden kann.

Komplexitätstheorie — die Grenzen der effizienten Berechenbarkeit

Frage:

Lässt sich ein gegebenes algorithmisches Problem effizient lösen, das heißt, in vernünftiger Laufzeit, mit vernünftigem Speicherbedarf und vernünftiger Verwendung anderer Ressourcen?

Beispiel 1: Zauberdodekaeder

Aufgabe

Löse den Dodekaeder.

Beispiel 2: Rush Hour

Aufgabe

Befreie das rote Auto aus dem Verkehrsstau.

Dieses algorithmische Problem (und ähnliche) lassen sich offensichtlich mit dem Computer lösen. Man kann zum Beispiel alle Möglichkeiten durchprobieren.

Leider lassen sie sich (bisher) oft nur durch extrem ineffiziente Algorithmen lösen.

Beispiel 3: Traveling Salesperson

Aufgabe

Finde eine kurze Rundreise durch die größten deutschen Städte und zurück zum Ausgangsort.

Die angegebene Route ist die kürzeste von 43.589.145.600 möglichen.

Wieder kann man das Problem lösen, indem man alle Möglichkeiten durchprobiert, wieder ist dies extrem ineffizient.

Die Suche nach einem effizienten Algorithmus für das Problem begann schon vor fünfzig Jahren . . .

Traveling Salesperson (Forts.)

Traveling Salesperson Problem (TSP)

- Eingabe: vollständiger Graph G mit Kantenlängen
- Ausgabe: eine Rundreise, die alle Knoten in G besucht und dabei so kurz wie möglich ist

Wir werden zeigen, dass es unter einer gewissen Hypothese ($P \neq NP$) keinen effizienten Algorithmus für dieses Problem gibt.

Übersicht

Teil 1: Grundlagen

- Modellierung von Problemen
- Einführung der Turingmaschine (TM)
- Einführung der Registermaschine (RAM)
- Vergleich TM – RAM
- Church-Turing-These

Übersicht

Teil 2: Berechenbarkeit

- Existenz unentscheidbarer Probleme
- Unentscheidbarkeit des Halteproblems
- Diagonalisierung / Unterprogrammtechnik / Reduktion
- Hilberts zehntes Problem
- Das Postsche Korrespondenzproblem
- WHILE- und LOOP-Programme

Übersicht

Teil 3: Komplexität

- Die Komplexitätsklassen P und NP
- NP-Vollständigkeit und der Satz von Cook und Levin
- Kochrezept für NP-Vollständigkeitsbeweise (Polynomielle Reduktion)
- NP-Vollständigkeit zahlreicher Probleme
- Auswege aus der NP-Vollständigkeit

Teil I
Grundlagen

Vorlesung 2
Berechnungsprobleme und Turingmaschinen

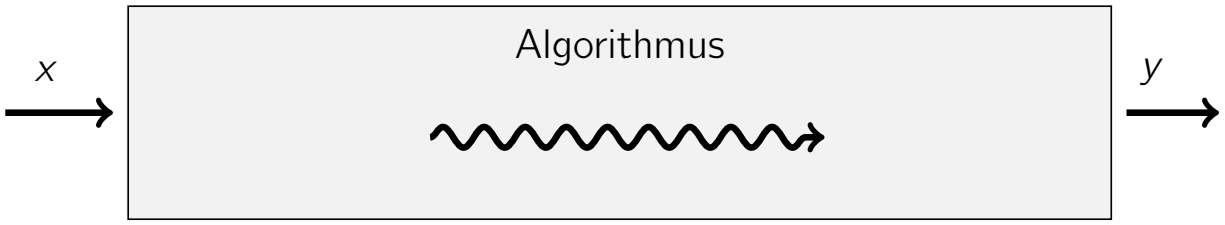
Berechnungsprobleme

Was ist ein Berechnungsproblem?

Informelle Umschreibung des Begriffes *Berechnungsproblem*:

Für gegebene Eingaben sollen bestimmte Ausgaben produziert werden.

Die Berechnung geschieht mit einem Algorithmus (Handlungsvorschrift).



- Mit „Problem“ meinen wir ab jetzt immer ein Berechnungsproblem.

Wir benötigen eine präzisere Definition ...

Alphabete und Wörter

- Ein- und Ausgaben sind Wörter über einem Alphabet Σ.
- Beispiele für Alphabete: $\Sigma = \{0, 1, \#\}$,
  $\Sigma = \{a, b, c, \ldots, z\}$,
  $\Sigma = \{$あ,い,う,え,お,か,き,く,...,わ$\}$,
  $\Sigma = \{$  $\}$.
- Σ^k ist die Menge aller Wörter der Länge k, z.B.

$$\{0,1\}^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$$

- Das leere Wort, also das Wort der Länge 0, bezeichnen wir mit ϵ.

  Dann gilt: $\Sigma^0 = \{\epsilon\}$.

- $\Sigma^* = \bigcup_{k \in \mathbb{N}_0} \Sigma^k$ ist der Kleenesche Abschluss von Σ und enthält alle Wörter über Σ. Diese kann man z.B. der Länge nach aufzählen können

$$\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, \ldots$$

Problem als Relation

- Im Allgemeinen entspricht ein Problem einer Relation $R \subseteq \Sigma^* \times \Sigma'^*$.
- Ein Paar (x, y) liegt in R, wenn y eine zulässige Ausgabe zur Eingabe x ist.

Beispielproblem: Primfaktorbestimung

Beispiel: Primfaktorbestimmung

Zu einer natürlichen Zahl $q \geq 2$ suchen wir einen Primfaktor.

Wir einigen uns darauf, Zahlen binär zu kodieren. Die Binärkodierung einer natürlichen Zahl i bezeichnen wir mit $\mathsf{bin}(i)$.

Also zum Beispiel: $\mathsf{bin}(0) = 0, \mathsf{bin}(6) = 110$

Die entsprechende Relation ist

$$R \quad = \quad \{(x, y) \in \{0,1\}^* \times \{0,1\}^* \mid x = \mathsf{bin}(q), y = \mathsf{bin}(p), \\ q, p \in \mathbb{N}, q \geq 2, p \text{ prim}, p \text{ teilt } q\} \quad .$$

Also zum Beispiel $(110, 11) \in R$, aber $(101, 11) \notin R$.

Problem als Funktion

- Bei vielen Problemen gibt es zu jeder Eingabe eine eindeutige Ausgabe.
- Dann können wir das Problem durch eine Funktion $f : \Sigma^* \to \Sigma'^*$ beschreiben.
- Die zur Eingabe $x \in \Sigma^*$ gesuchte Ausgabe ist $f(x) \in \Sigma'^*$.

Beispiel: Multiplikation

Zu zwei natürlichen Zahlen $i_1, i_2 \in \mathbb{N}$ suchen wir das Produkt.

Um die Zahlen i_1 und i_2 in der Eingabe voneinander trennen zu können, erweitern wir das Alphabet um ein Trennsymbol $\#$, d.h. $\Sigma = \{0, 1, \#\}$.

Die entsprechende Funktion $f : \Sigma^* \to \Sigma^*$ ist gegeben durch

$$f(\mathsf{bin}(i_1) \# \mathsf{bin}(i_2)) \quad = \quad \mathsf{bin}(i_1 \cdot i_2) \quad .$$

Entscheidungsprobleme als Sprachen

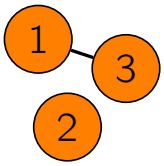
- Viele Probleme lassen sich als Ja-Nein-Fragen formulieren.
- Derartige Entscheidungsprobleme sind von der Form $f : \Sigma^* \to \{0, 1\}$, wobei wir 0 als „Nein“ und 1 als „Ja“ interpretieren.
- Sei $L = f^{-1}(1) \subseteq \Sigma^*$ die Menge derjenigen Eingaben, die mit „Ja“ beantwortet werden.
- L ist eine Teilmenge der Wörter über dem Alphabet Σ.
- Eine Teilmenge von Σ^* wird allgemein als Sprache bezeichnet.
- Die Sprache L ist die zu dem durch f definierten Entscheidungsproblem gehörende Sprache.

Entscheidungsprobleme – Beispiel

Beispiel: Graphzusammenhang

Problemstellung: Für einen gegebenen Graphen G soll bestimmt werden, ob G zusammenhängend ist.

Der Graph G liege dabei in einer geeigneten Kodierung $\mathsf{code}(G) \in \Sigma^*$ vor, z. B. als binär kodierte Adjazenzmatrix.

Beispiel: 

$\rightsquigarrow \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \rightsquigarrow 001000100$

Die zu diesem Entscheidungsproblem gehörende Sprache ist

$$L = \{ w \in \Sigma^* \mid \exists \text{ Graph } G\text{: } w = \mathsf{code}(G) \text{ und } G \text{ ist zusammenhängend} \} .$$

Zentrale Fragestellung

Welche Berechnungsprobleme sind durch einen Algorithmus lösbar?

bzw.

Welche Sprachen können von einem Algorithmus entschieden werden?

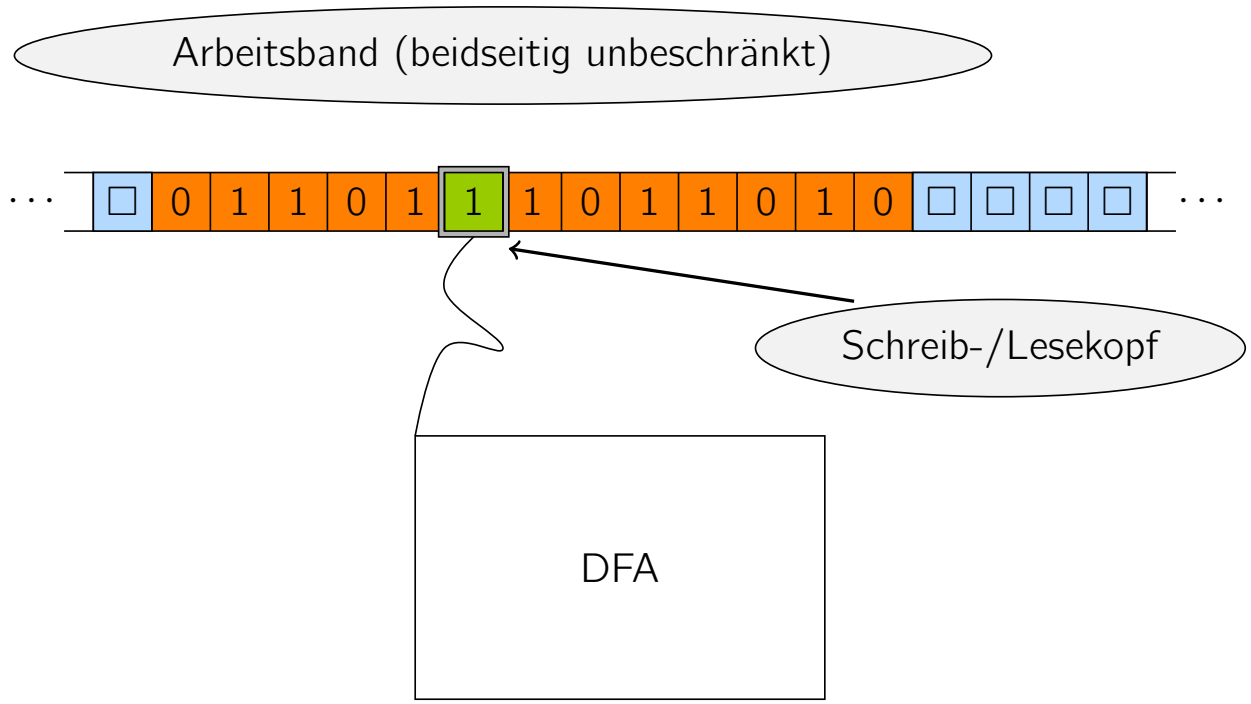
- Um diese Fragen in einem mathematisch exakten Sinne klären zu können, müssen wir festlegen, was eigentlich ein Algorithmus ist.

- Zu diesem Zweck definieren wir ein einfaches „Computer“-Modell:

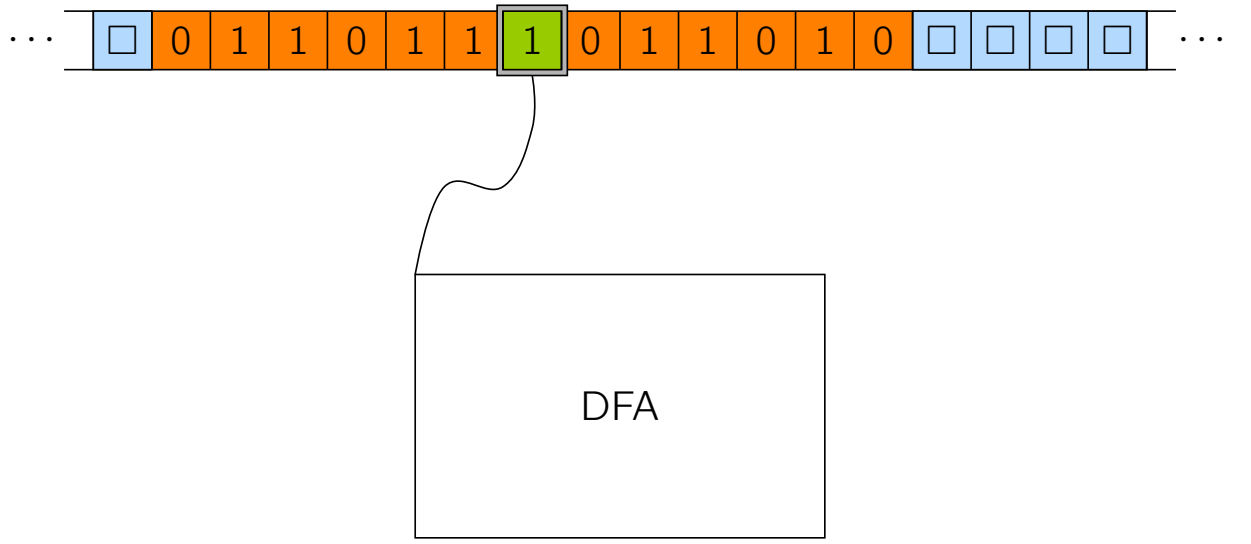
  Die Turingmaschine (TM).

Turingmaschinen

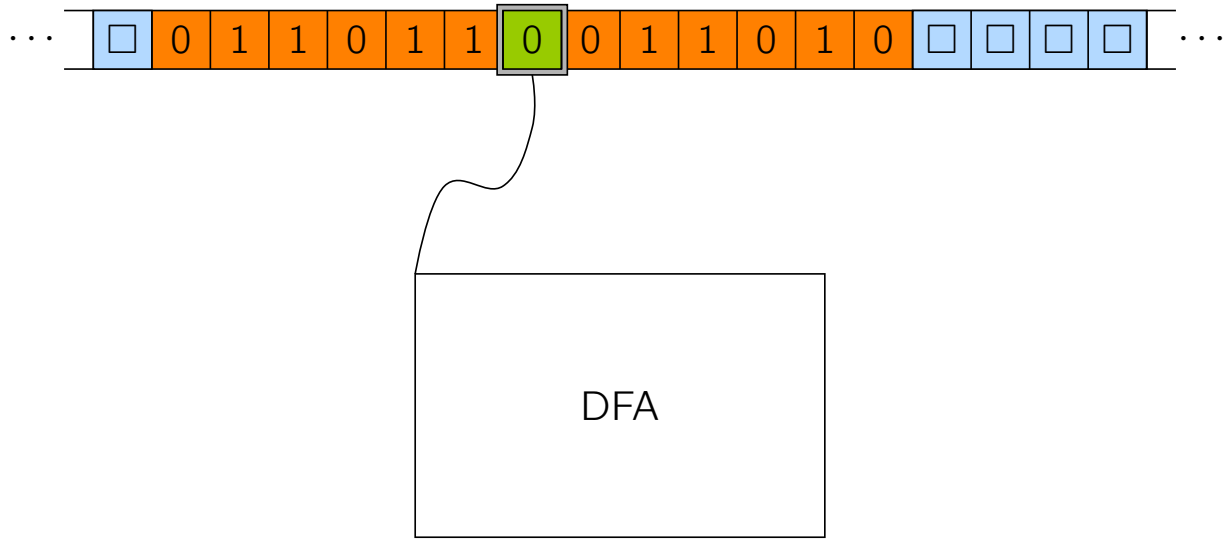
Deterministische Turingmaschine (TM bzw. DTM)



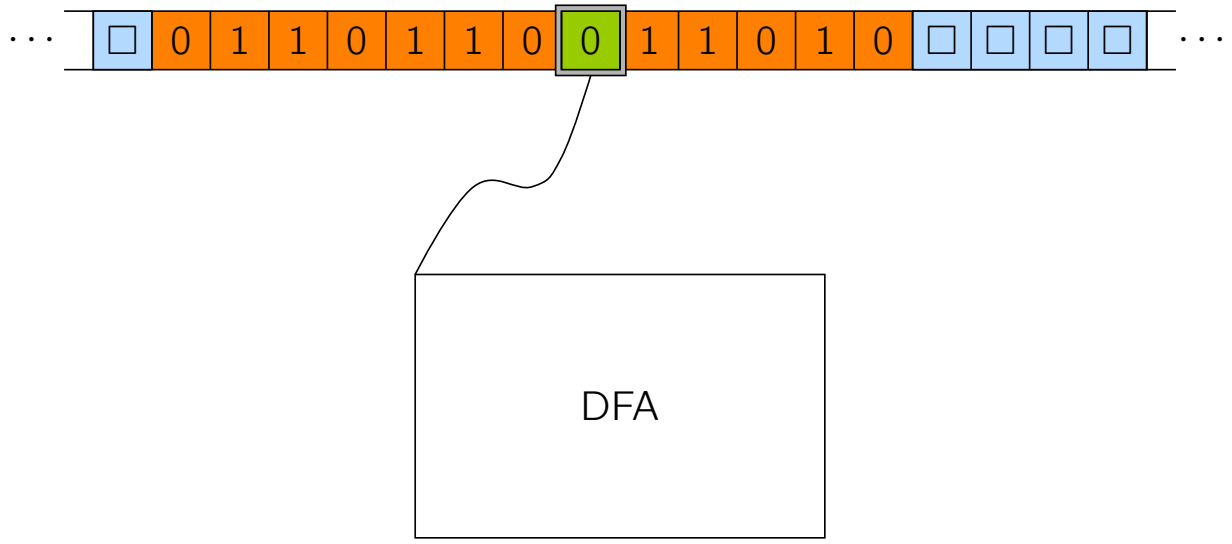
Deterministische Turingmaschine (TM bzw. DTM)



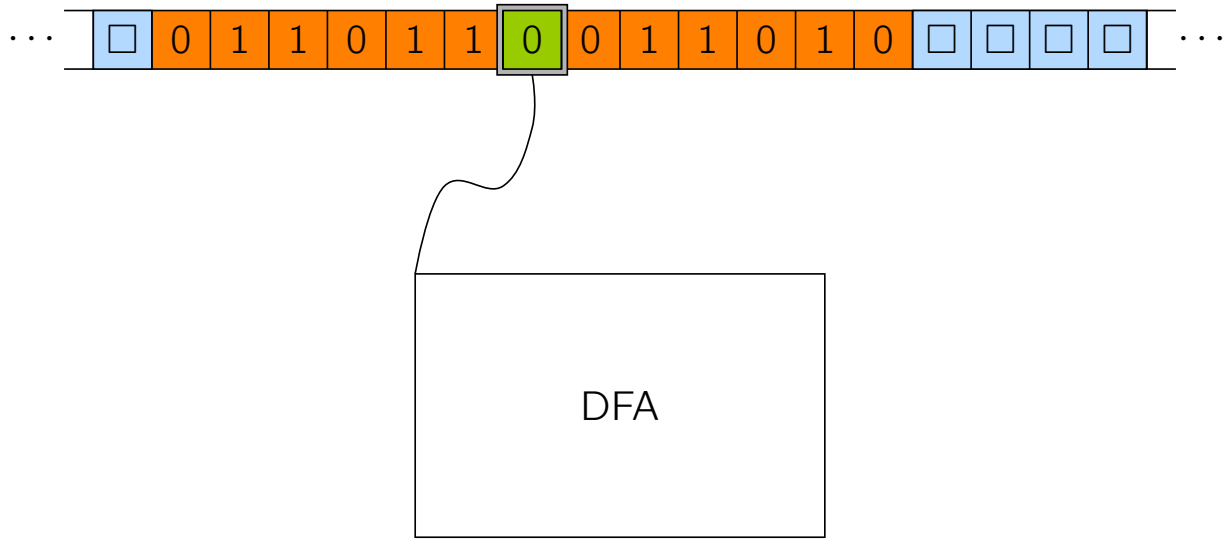
Deterministische Turingmaschine (TM bzw. DTM)



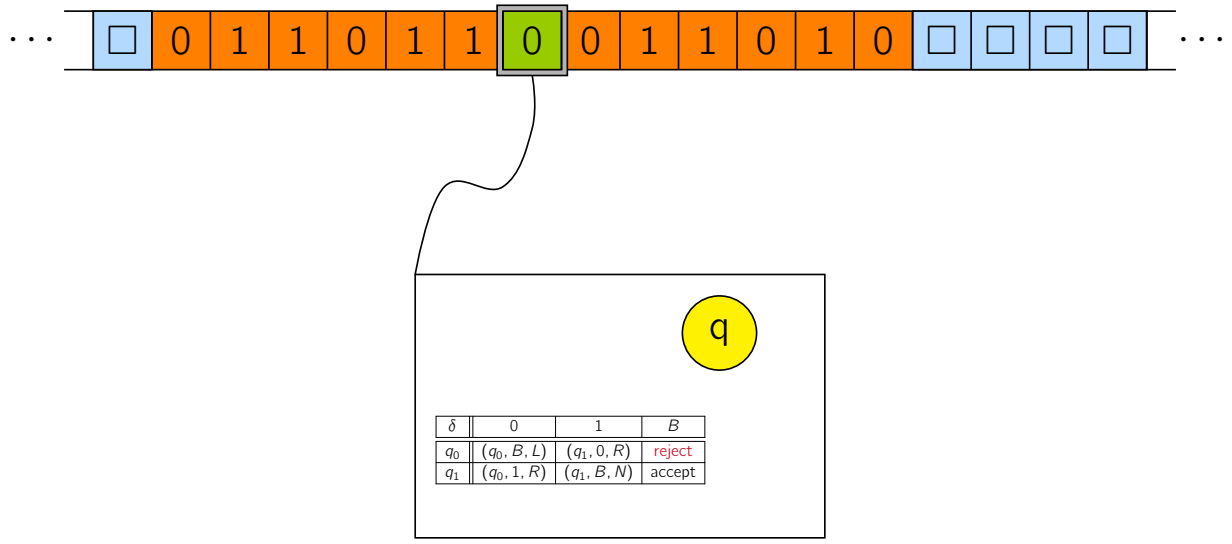
Deterministische Turingmaschine (TM bzw. DTM)



Deterministische Turingmaschine (TM bzw. DTM)



Deterministische Turingmaschine (TM bzw. DTM)



Komponenten der TM

- Q, die endliche Zustandsmenge
- Σ, das endliche Eingabealphabet
- $\Gamma \supset \Sigma$, das endliche Bandalphabet
- $B \in \Gamma \setminus \Sigma$, das Leerzeichen (Blank, in Bildern $\square$)
- $q_0 \in Q$, der Anfangszustand
- $\bar{q} \in Q$, der Endzustand
- $\delta \colon (Q \setminus \{\bar{q}\}) \times \Gamma \to Q \times \Gamma \times \{R, L, N\}$, die Zustandsüberführungsfunktion

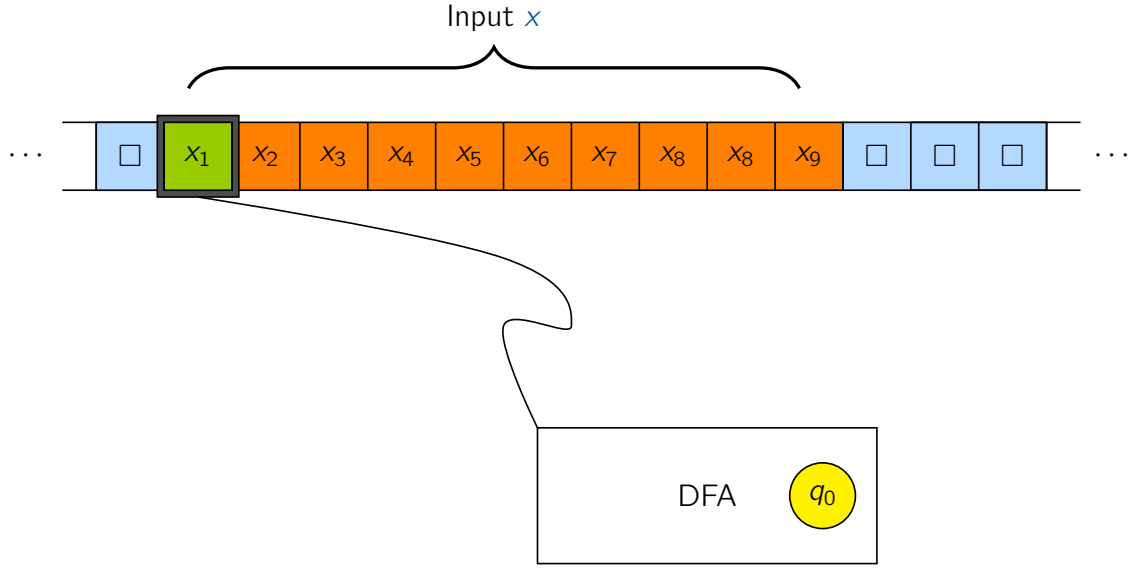
Definition

Eine Turingmaschine ist definiert durch das 7-Tupel $(Q, \Sigma, \Gamma, B, q_0, \bar{q}, \delta)$.

Funktionsweise der TM

Ausgangssituation

- auf dem Band steht die Eingabe $x \in \Sigma^*$ eingerahmt von Blanks
- der initiale Zustand ist q_0
- der Kopf steht über dem ersten Symbol von x



Funktionsweise der TM

Ausgangssituation

- auf dem Band steht die Eingabe $x \in \Sigma^*$ eingerahmt von Blanks
- der initiale Zustand ist q_0
- der Kopf steht über dem ersten Symbol von x

Nummerierung der Zellen des Bandes

- die initiale Kopfposition wird als Position 0 bezeichnet
- bewegt sich der Kopf einen Schritt „nach rechts“, erhöht sich die Position um 1
- bewegt sich der Kopf um einen Schritt „nach links“, erniedrigt sich die Position um 1

Funktionsweise der TM

Durchführung eines Rechenschrittes

- $a \in \Gamma$ bezeichne das gelesene Symbol
- $q \in Q \setminus \{\bar{q}\}$ bezeichne den aktuellen Zustand
- angenommen $\delta(q, a) = (q', a', d)$, für $q' \in Q, a' \in \Gamma, d \in \{R, L, N\}$
- dann wird der Zustand auf q' gesetzt
- an der Kopfposition wird das Symbol a' geschrieben
- der Kopf

$$\text{bewegt sich} \begin{cases} \text{um eine Position nach rechts} & \text{falls } d = R \\ \text{um eine Position nach links} & \text{falls } d = L \\ \text{nicht} & \text{falls } d = N \end{cases}$$

Funktionsweise der TM

Ende der Berechnung

- die TM stoppt, wenn sie den Endzustand $\bar{q}$ erreicht
- das Ausgabewort $y \in \Sigma^*$ kann dann vom Band abgelesen werden: y beginnt an der Kopfposition und endet unmittelbar vor dem ersten Symbol aus $\Gamma \setminus \Sigma$
- *Spezialfall:* wenn wir es mit Entscheidungsproblemen zu tun haben, wird die Antwort wie folgt als JA oder NEIN interpretiert:
  - die TM akzeptiert das Eingabewort, wenn sie terminiert und das Ausgabewort mit einer 1 beginnt
  - die TM verwirft das Eingabewort, wenn sie terminiert und das Ausgabewort nicht mit einer 1 beginnt

Funktionsweise der TM

Bemerkungen

- Beachte, es gibt die Möglichkeit, dass die TM den Endzustand niemals erreicht. Wir sagen dann, die Berechnung terminiert nicht.
- Laufzeit = Anzahl Zustandsübergänge bis zur Terminierung
- Speicherbedarf = Anzahl Bandzellen, die während der Berechnung besucht worden sind

Funktionsweise der TM am Beispiel

Sei $L = \{w1 \mid w \in \{0,1\}^*\}$ die Sprache der 0/1-Wörter, die auf 1 enden.

L wird entschieden durch die TM $M = (Q, \Sigma, \Gamma, B, q_0, \bar{q}, \delta)$ mit

- $Q = \{q_0, q_1, \bar{q}\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, 1, B\}$
- δ gemäß Tabelle

δ	0	1	B
q_0	(q_0, B, R)	(q_1, B, R)	reject
q_1	(q_0, B, R)	(q_1, B, R)	accept

„accept“ steht als Abkürzung für $(\bar{q}, 1, N)$.
„reject“ steht als Abkürzung für $(\bar{q}, 0, N)$.

Allgemein: M entscheidet L, wenn M alle Wörter in L akzeptiert und alle Wörter, die nicht in L sind, verwirft.

- Die TM ist ein formales Modell zur Beschreibung von Algorithmen.

Funktionsweise der TM am Beispiel

Die Übergangsfunktion ist zentraler Bestandteil der Turingmaschine.

Beschreibung der Übergangsfunktion als Tabelle:

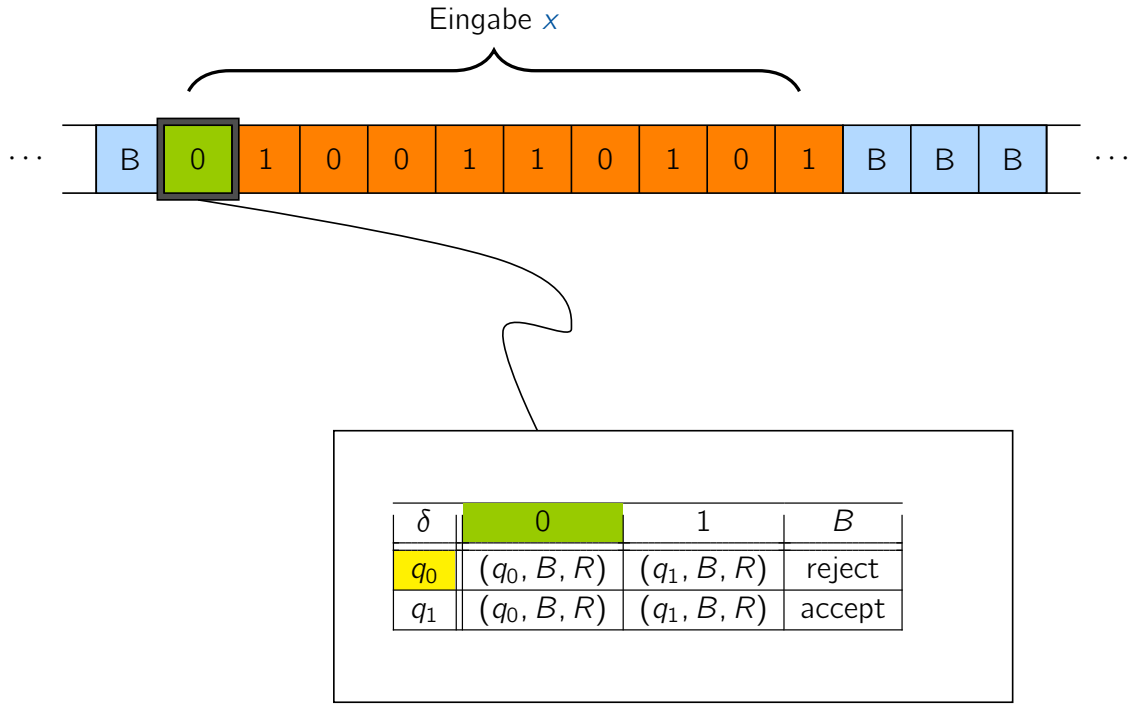
δ	0	1	B
q_0	(q_0, B, R)	(q_1, B, R)	reject
q_1	(q_0, B, R)	(q_1, B, R)	accept

Verbale Beschreibung des Algorithmus der TM:

- Solange ein Symbol aus $\{0, 1\}$ gelesen wird,
  - überschreibe das Symbol mit B,
  - bewege den Kopf nach rechts, und
  - gehe in den Zustand q_0, wenn das Symbol eine 0 war, sonst in den Zustand q_1
- Sobald ein Blank gelesen wird,
  - akzeptiere die Eingabe, falls der aktuelle Zustand q_1 ist, und
  - verwirf die Eingabe ansonsten.

Veranschaulichung des Algorithmus

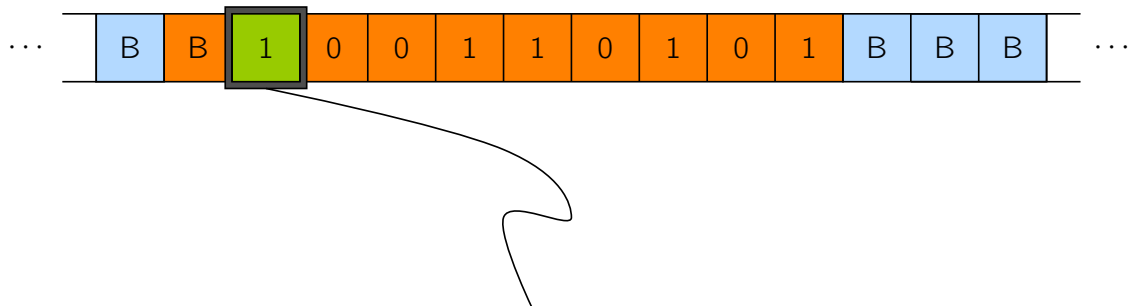
Eingabe $x = 0100110101$.



δ	0	1	B
q_0	(q_0, B, R)	(q_1, B, R)	reject
q_1	(q_0, B, R)	(q_1, B, R)	accept

Veranschaulichung des Algorithmus

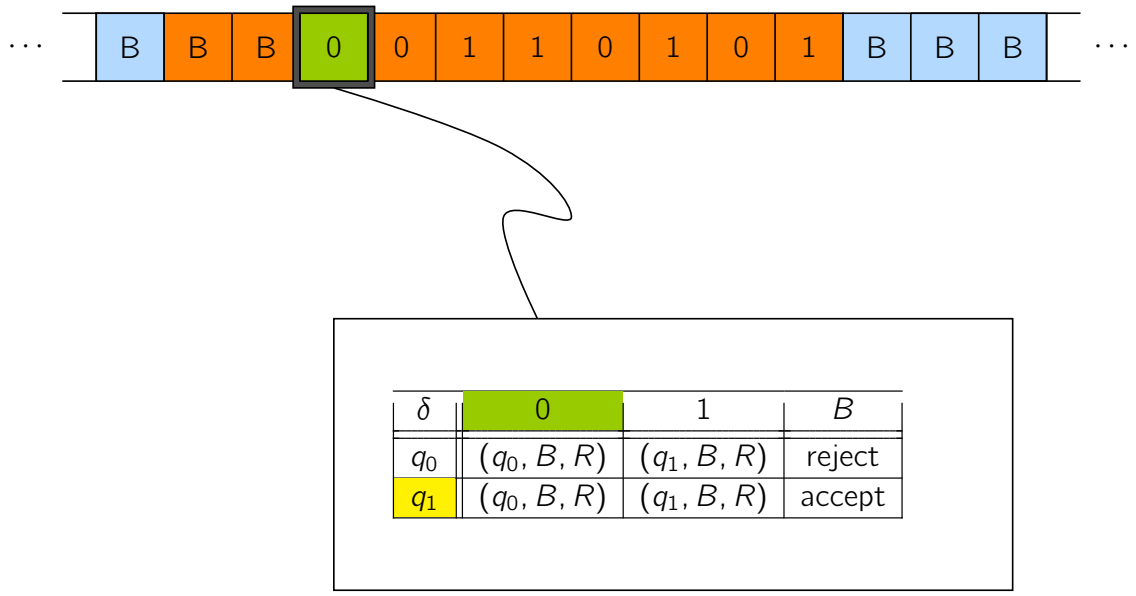
Eingabe $x = 0100110101$.



δ	0	1	B
q_0	(q_0, B, R)	(q_1, B, R)	reject
q_1	(q_0, B, R)	(q_1, B, R)	accept

Veranschaulichung des Algorithmus

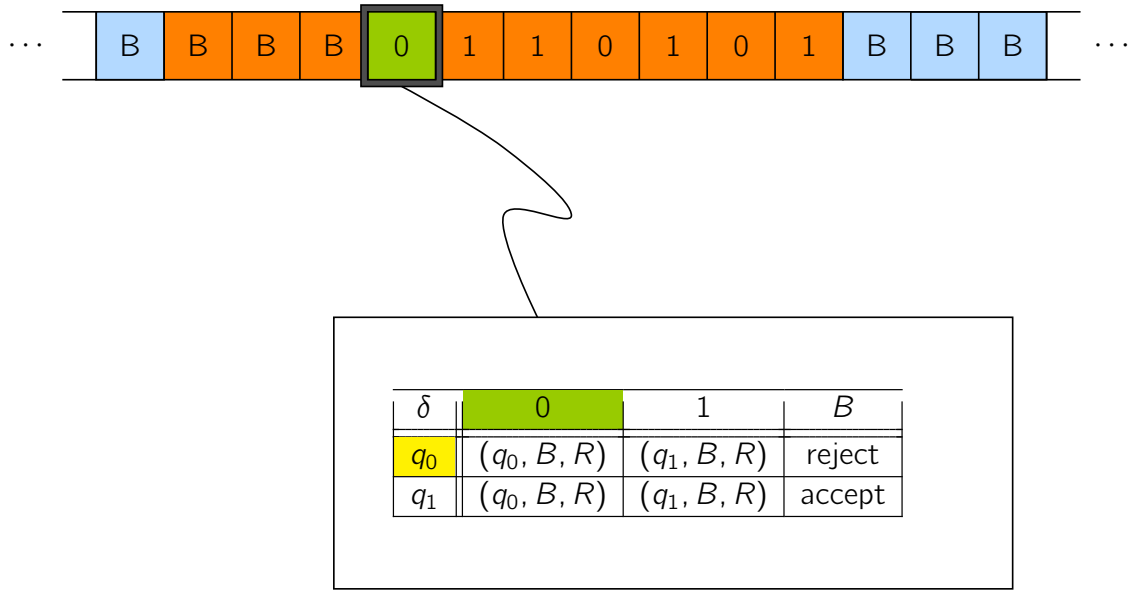
Eingabe $x = 0100110101$.



δ	0	1	B
q_0	(q_0, B, R)	(q_1, B, R)	reject
q_1	(q_0, B, R)	(q_1, B, R)	accept

Veranschaulichung des Algorithmus

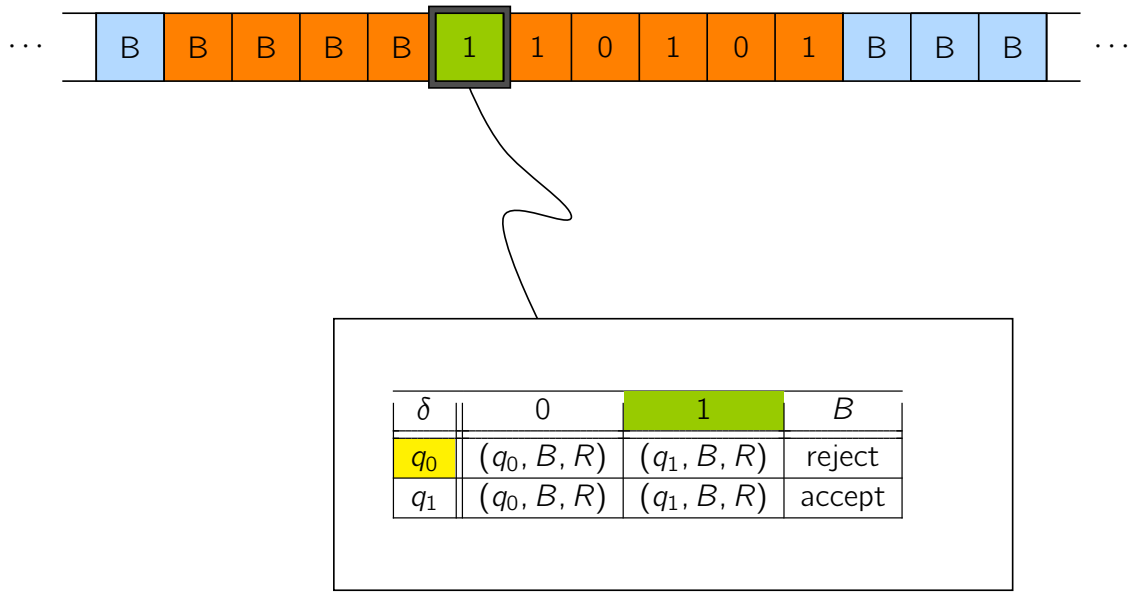
Eingabe $x = 0100110101$.



δ	0	1	B
q_0	(q_0, B, R)	(q_1, B, R)	reject
q_1	(q_0, B, R)	(q_1, B, R)	accept

Veranschaulichung des Algorithmus

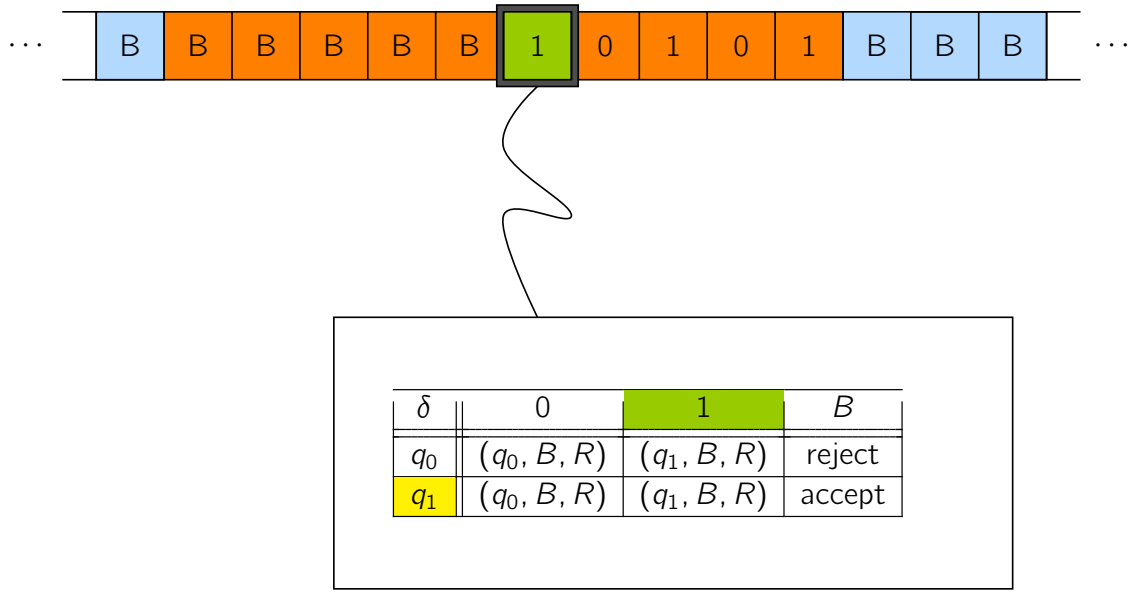
Eingabe $x = 0100110101$.



δ	0	1	B
q_0	(q_0, B, R)	(q_1, B, R)	reject
q_1	(q_0, B, R)	(q_1, B, R)	accept

Veranschaulichung des Algorithmus

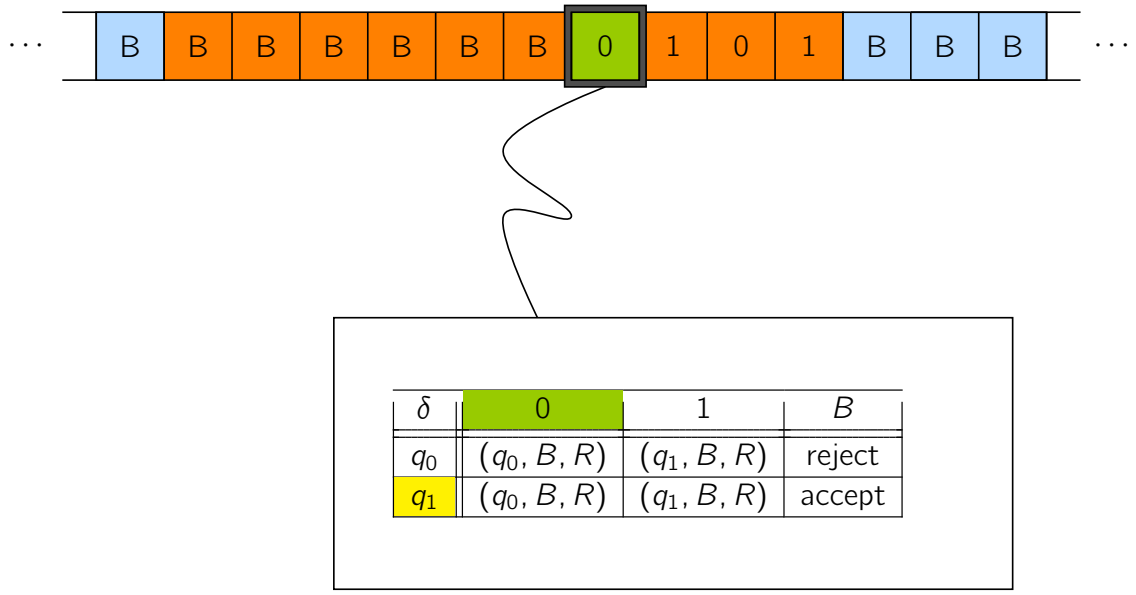
Eingabe $x = 0100110101$.



δ	0	1	B
q_0	(q_0, B, R)	(q_1, B, R)	reject
q_1	(q_0, B, R)	(q_1, B, R)	accept

Veranschaulichung des Algorithmus

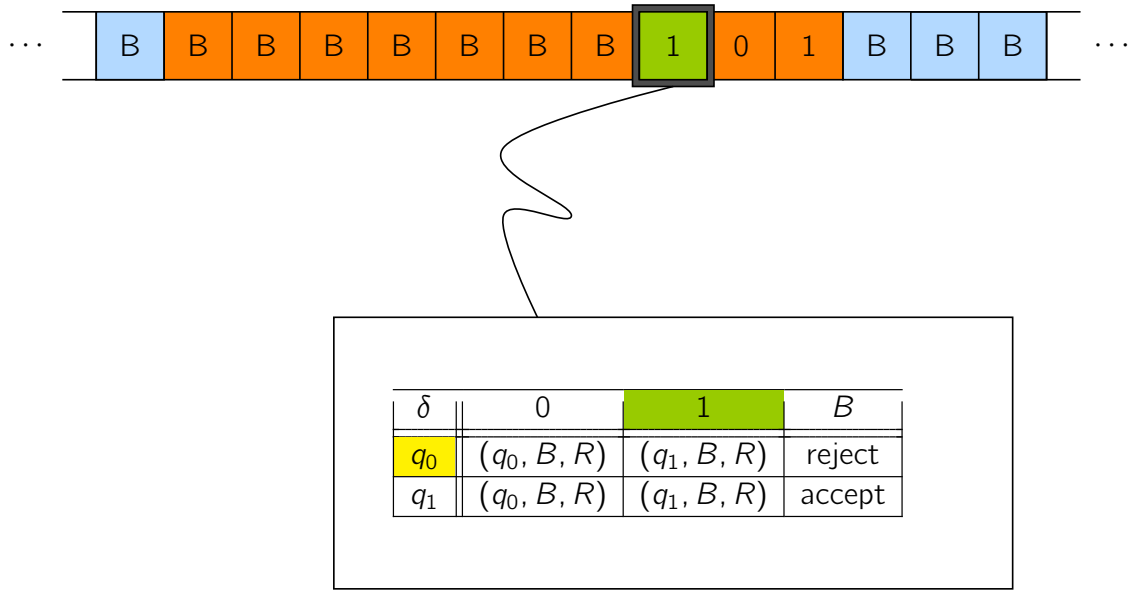
Eingabe $x = 0100110101$.



δ	0	1	B
q_0	(q_0, B, R)	(q_1, B, R)	reject
q_1	(q_0, B, R)	(q_1, B, R)	accept

Veranschaulichung des Algorithmus

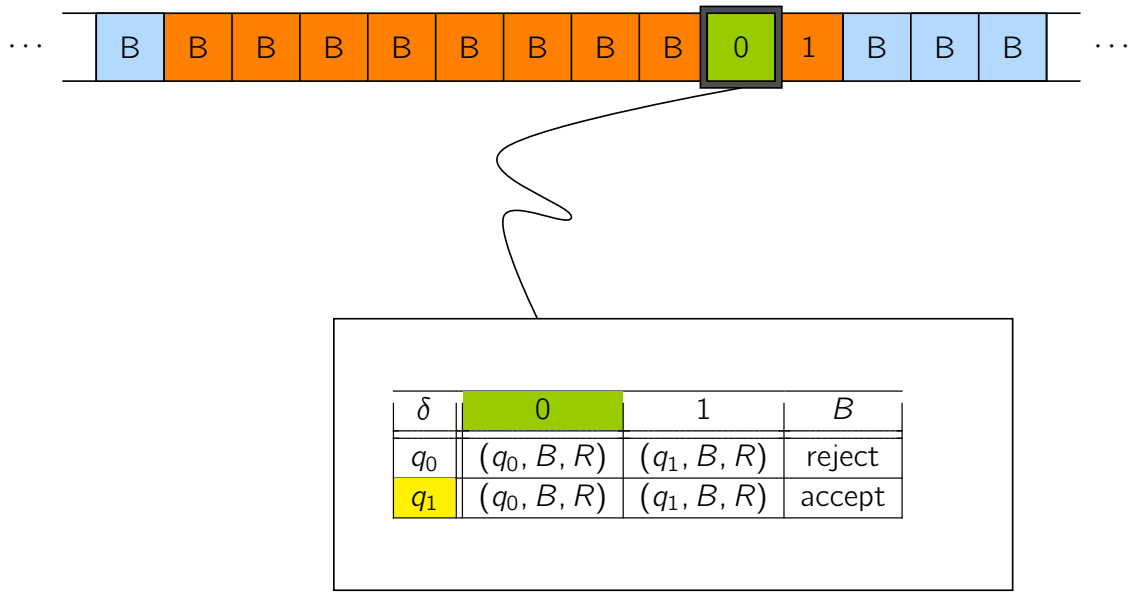
Eingabe $x = 0100110101$.



δ	0	1	B
q_0	(q_0, B, R)	(q_1, B, R)	reject
q_1	(q_0, B, R)	(q_1, B, R)	accept

Veranschaulichung des Algorithmus

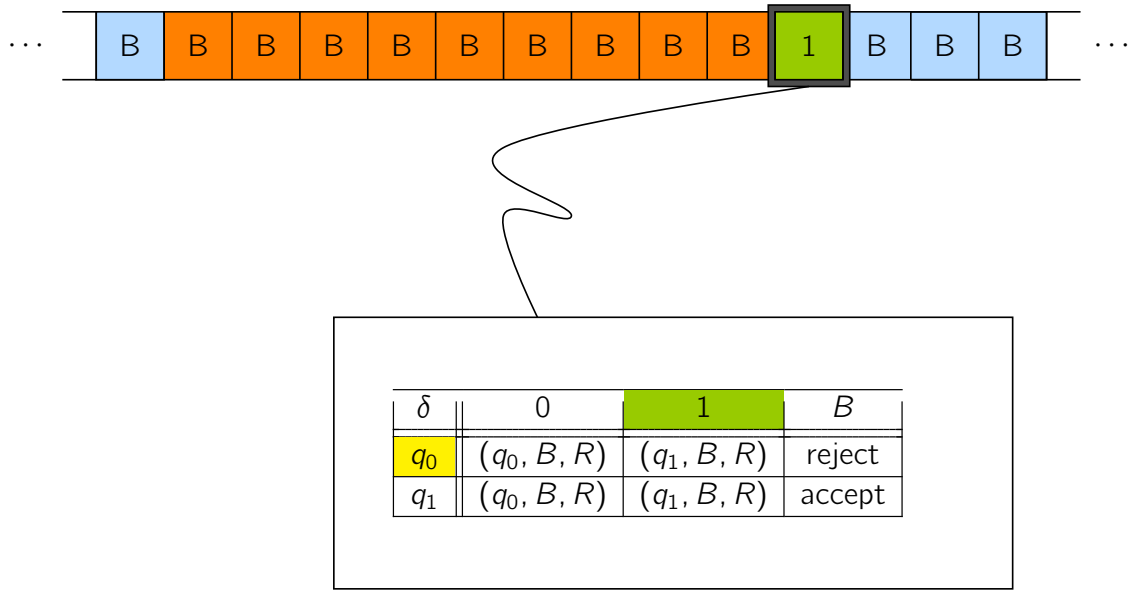
Eingabe $x = 0100110101$.



δ	0	1	B
q_0	(q_0, B, R)	(q_1, B, R)	reject
q_1	(q_0, B, R)	(q_1, B, R)	accept

Veranschaulichung des Algorithmus

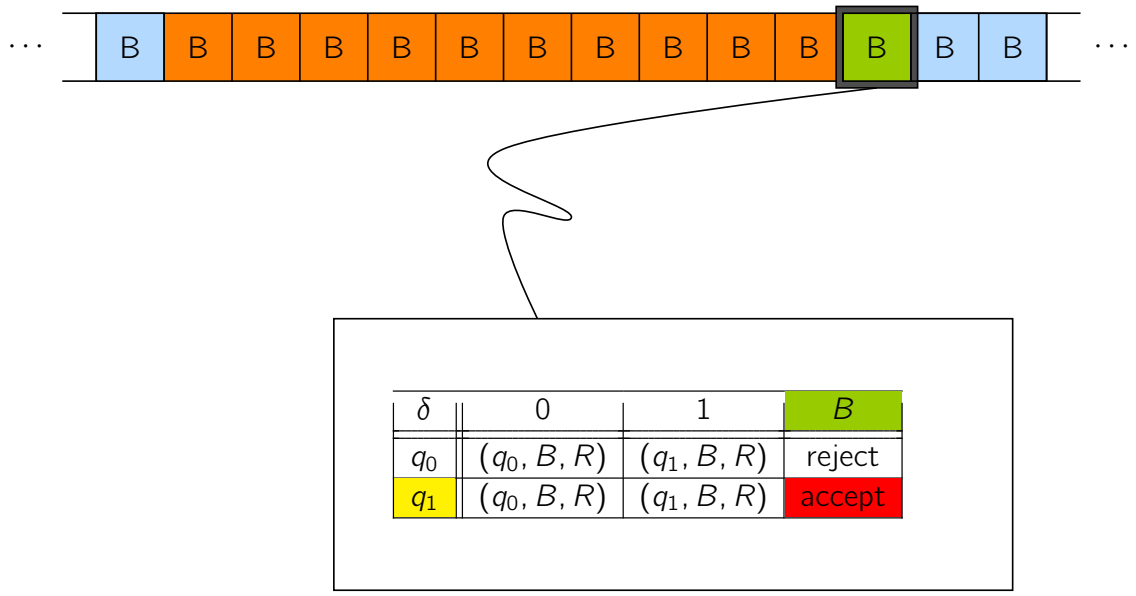
Eingabe $x = 0100110101$.



δ	0	1	B
q_0	(q_0, B, R)	(q_1, B, R)	reject
q_1	(q_0, B, R)	(q_1, B, R)	accept

Veranschaulichung des Algorithmus

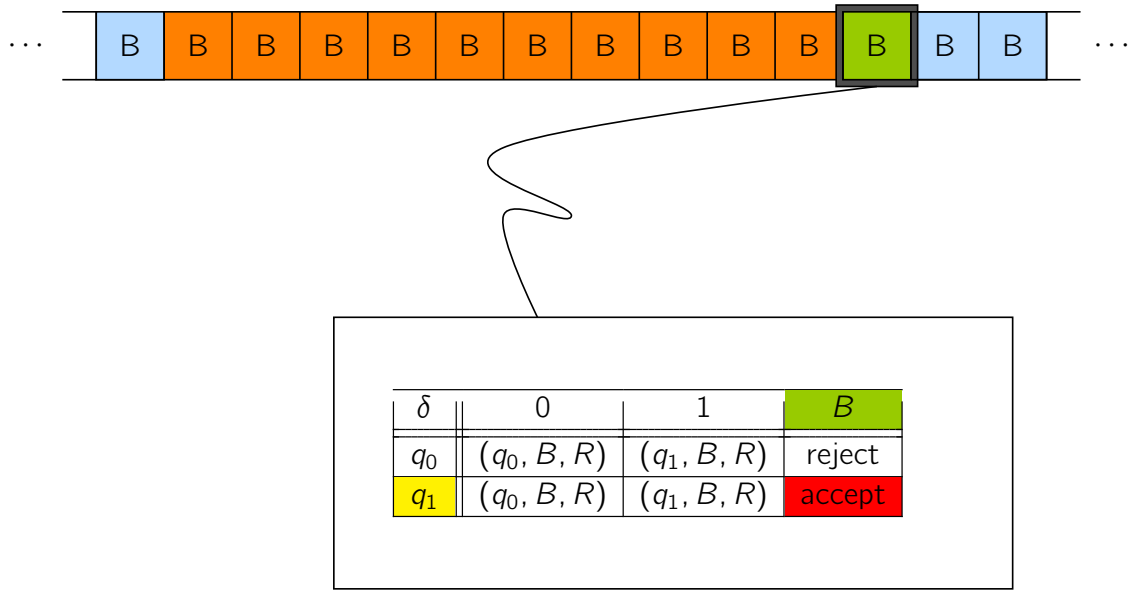
Eingabe $x = 0100110101$.



δ	0	1	B
q_0	(q_0, B, R)	(q_1, B, R)	reject
q_1	(q_0, B, R)	(q_1, B, R)	accept

Veranschaulichung des Algorithmus

Eingabe $x = 0100110101$.



δ	0	1	B
q_0	(q_0, B, R)	(q_1, B, R)	reject
q_1	(q_0, B, R)	(q_1, B, R)	accept

Zum Sinn und Zweck des TM-Modells

Die TM dient als formales (bzw. mathematisches) Modell zur Beschreibung von Algorithmen.

Die Frage, *ob es für ein Problem einen Algorithmus gibt*, setzen wir gleich mit der Frage, *ob es eine TM gibt, die dieses Problem löst*.

- Es stellt sich die Frage, ob das TM-Modell sinnvoll oder allgemein genug ist.
  Kann es alle denkbaren Algorithmen abdecken bzw. alle Fähigkeiten abdecken, die ein moderner oder zukünftiger Computer bzw. haben könnte?
- Auf diese Problematik kommen wir später noch zurück.

Formale Definition des Begriffes *TM-berechenbar*

Bzgl. der Berechnungsprobleme beschränken wir uns in dieser Vorlesung auf Funktionen und Entscheidungsprobleme (Sprachen).

Definition

Eine Funktion $f : \Sigma^* \to \Sigma^*$ heißt TM-berechenbar, wenn es eine TM gibt, die aus der Eingabe x den Funktionswert $f(x)$ berechnet.

Definition

Eine Sprache $L \subseteq \Sigma^*$ heißt TM-entscheidbar, wenn es eine TM gibt, die für alle Eingaben terminiert und die Eingabe w genau dann akzeptiert, wenn $w \in L$ ist.

TM-berechenbare Funktionen und TM-entscheidbare Sprachen werden auch rekursive Funktionen bzw. Sprachen genannt.

Programmierung der TM am Beispiel

Wir entwickeln eine TM für die Sprache

$$L = \{0^n 1^n \mid n \geq 1\} \ .$$

Sei $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, B\}$, $Q = \{q_0, \ldots, q_6, \bar{q}\}$.

Unsere TM arbeitet in zwei Phasen:

- **Phase 1:** Teste, ob das Eingabewort von der Form $0^i 1^j$ für $i \geq 0$ und $j \geq 1$ ist.
- **Phase 2:** Teste, ob $i = j$ gilt.

Phase 1 verwendet $\{q_0, q_1\}$ und wechselt bei Erfolg zu q_2.
Phase 2 verwendet $\{q_2, \ldots, q_6\}$ und akzeptiert bei Erfolg.

Programmierung der TM am Beispiel - Phase 1

δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)

q_0 : Laufe von links nach rechts über die Eingabe, bis ein Zeichen ungleich 0 gefunden wird.

- Falls dieses Zeichen eine 1 ist, gehe über in Zustand q_1.
- Sonst ist dieses Zeichen ein Blank. Verwirf die Eingabe.

q_1 : Gehe weiter nach rechts bis zum ersten Zeichen ungleich 1.

- Falls dieses Zeichen eine 0 ist, verwirf die Eingabe.
- Sonst ist das gefundene Zeichen ein Blank. Bewege den Kopf um eine Position nach links auf die letzte gelesene 1. Wechsele in den Zustand q_2, Phase 2 beginnt.

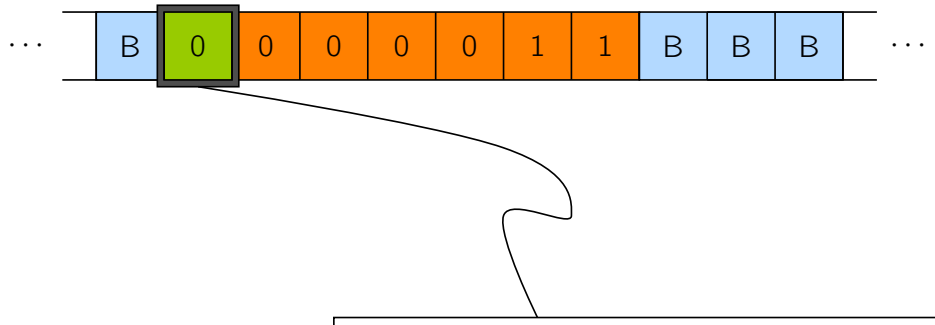
Programmierung der TM am Beispiel - Phase 2

δ	0	1	B
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

- q_2 : Kopf steht auf dem letzten Nichtblank. Falls dieses Zeichen eine 1 ist, so lösche es, gehe nach links, und wechsele in den Zustand q_3. Sonst verwirf die Eingabe.
- q_3 : Bewege den Kopf auf das erste Nichtblank. Dann q_4.
- q_4 : Falls das gelesene Zeichen eine 0 ist, ersetze es durch ein Blank und gehe nach q_5, sonst verwirf die Eingabe.
- q_5 : Wir haben jetzt die linkeste 0 und die rechteste 1 gelöscht. Falls Restwort leer, dann akzeptiere, sonst q_6.
- q_6 : Laufe wieder zum letzten Nichtblank und starte erneut in q_2.

Veranschaulichung der TM

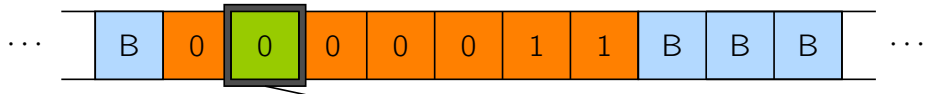
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

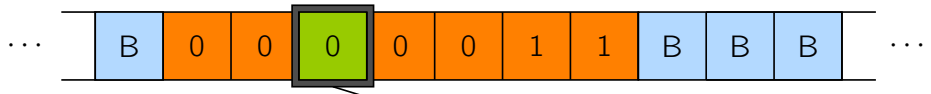
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

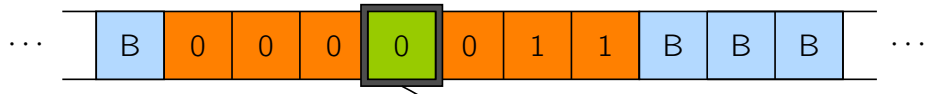
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

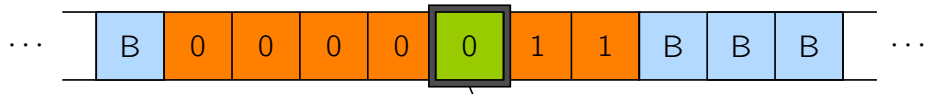
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

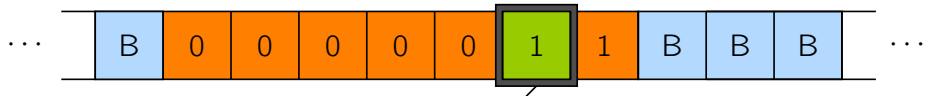
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

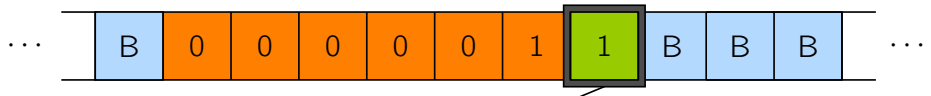
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

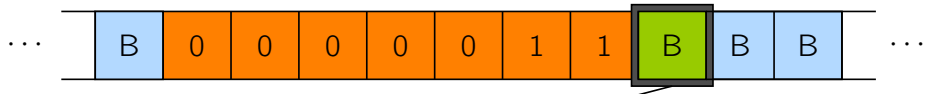
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

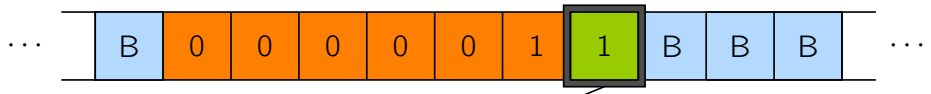
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

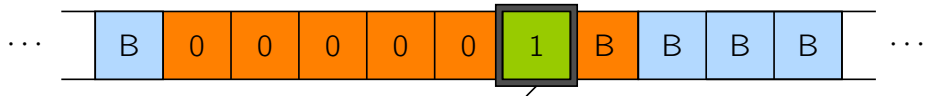
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

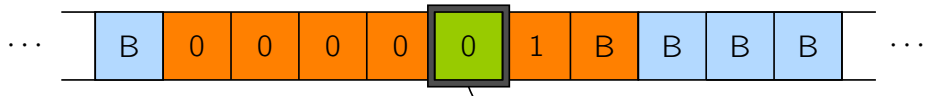
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

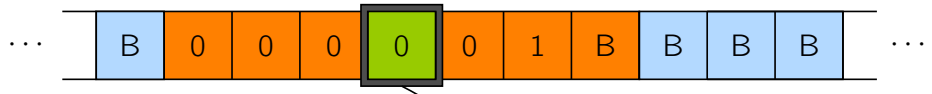
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

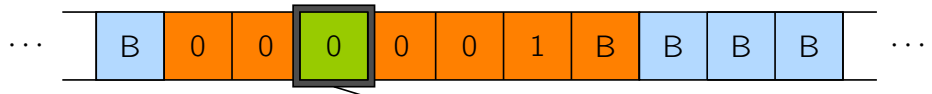
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

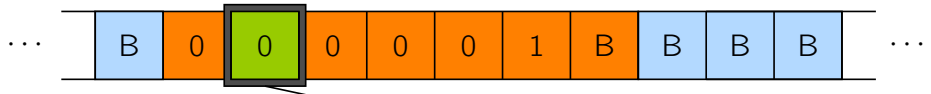
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

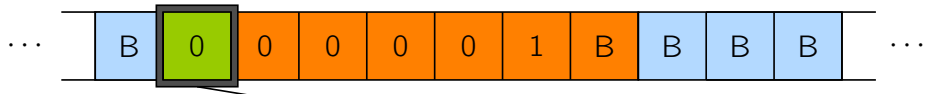
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

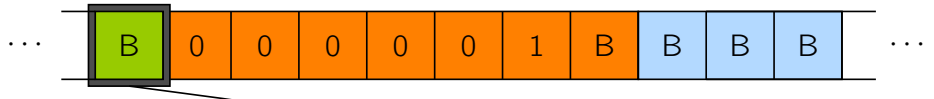
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

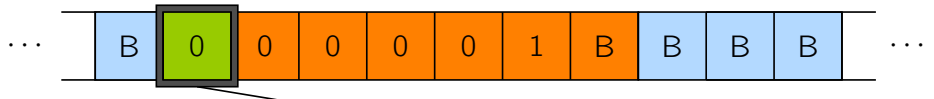
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

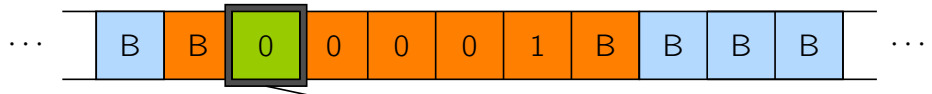
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

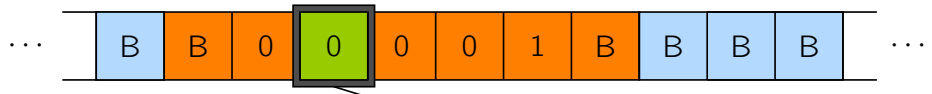
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

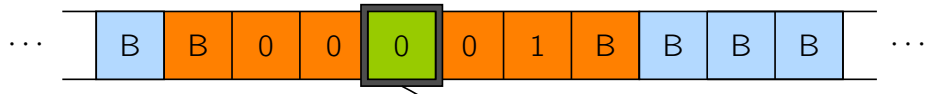
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

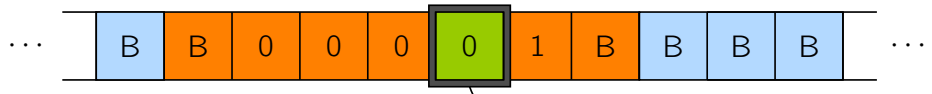
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

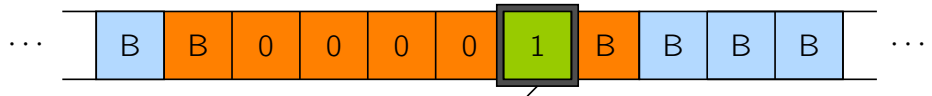
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

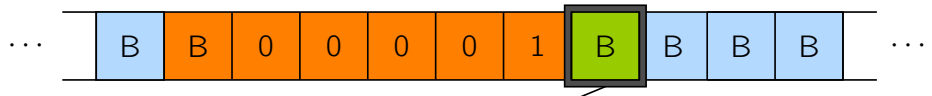
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

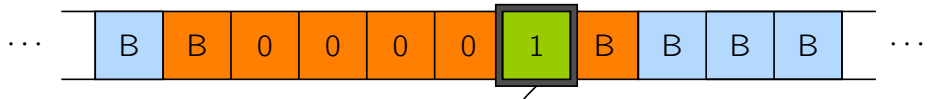
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

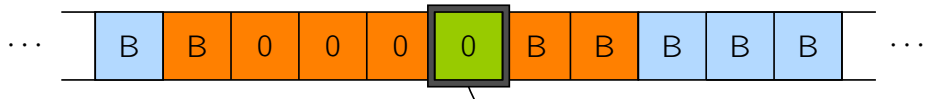
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

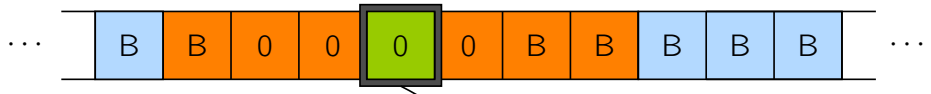
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

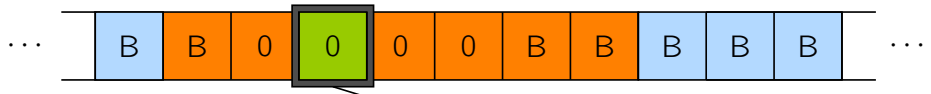
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

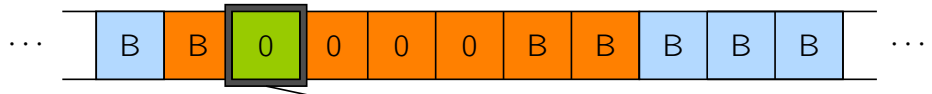
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

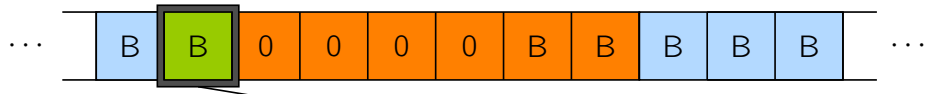
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

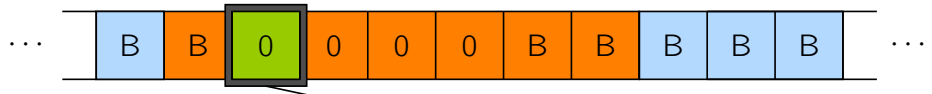
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

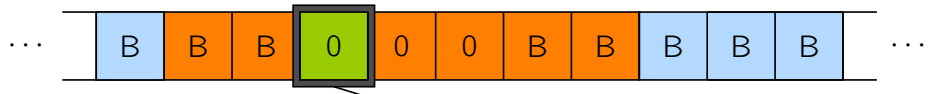
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

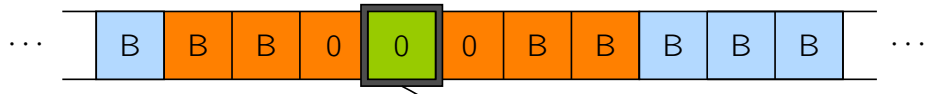
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

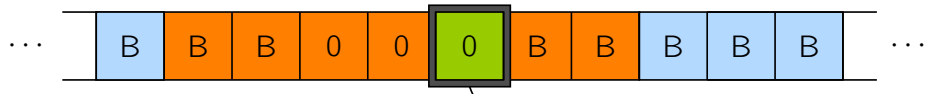
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

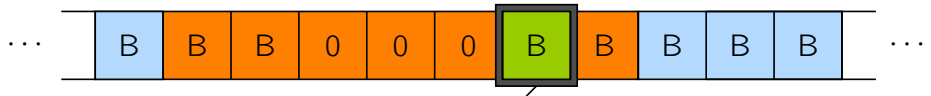
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

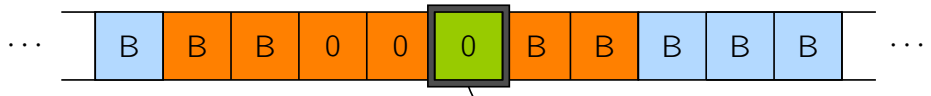
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

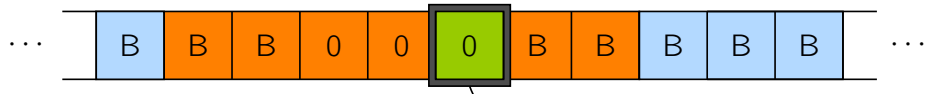
Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Veranschaulichung der TM

Eingabe $x = 0000011$.



δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

Konfigurationen und (direkte) Nachfolgekonfigurationen

Definition

i) Eine Konfiguration einer TM ist ein String $\alpha q \beta$, für $q \in Q$ und $\alpha, \beta \in \Gamma^*$.
Bedeutung: auf dem Band steht $\alpha\beta$ eingerahmt von Blanks, der Zustand ist q, und der Kopf steht über dem ersten Zeichen von β.

ii) $\alpha' q' \beta'$ ist direkte Nachfolgekonfiguration von $\alpha q \beta$, falls $\alpha' q' \beta'$ in einem Rechenschritt aus $\alpha q \beta$ entsteht. Wir schreiben

$$\alpha q \beta \vdash \alpha' q' \beta'.$$

iii) $\alpha'' q'' \beta''$ ist Nachfolgekonfiguration von $\alpha q \beta$, falls $\alpha'' q'' \beta''$ in endlich vielen Rechenschritten aus $\alpha q \beta$ entsteht. Wir schreiben

$$\alpha q \beta \vdash^* \alpha'' q'' \beta''.$$

Bemerkung: insbesondere gilt $\alpha q \beta \vdash^* \alpha q \beta$.

Beispiel zum Umgang mit Konfigurationen

Die für die Sprache $L = \{0^n 1^n | n \geq 1\}$ beschriebene TM liefert in Phase 1 auf der Eingabe 0011 die folgende Konfigurationsfolge.

Phase 1:

$$q_0 0011 \vdash 0 q_0 011 \vdash 00 q_0 11 \vdash 001 q_1 1 \vdash 0011 q_1 B \vdash 001 q_2 1$$

Beobachtung: abgesehen von Blanks am Anfang und Ende des Strings sind die Konfigurationen eindeutig.

Phase 2:

$$001 q_2 1 \vdash 00 q_3 1 \vdash 0 q_3 01 \vdash q_3 001 \vdash q_3 B001 \vdash q_4 001 \vdash q_5 01 \vdash$$

$$0 q_6 1 \vdash 01 q_6 \vdash 0 q_2 1 \vdash \ldots$$

Techniken zur Programmierung von Turingmaschinen

Technik 1: Speicher im Zustandsraum

Für beliebiges festes $k \in \mathbb{N}$ können wir k Zeichen unseres Bandalphabets im Zustand abspeichern, indem wir den Zustandsraum um den Faktor $|\Gamma|^k$ vergrößern, d.h. wir setzen

$$Q_{\text{neu}} := Q \times \Gamma^k \quad .$$

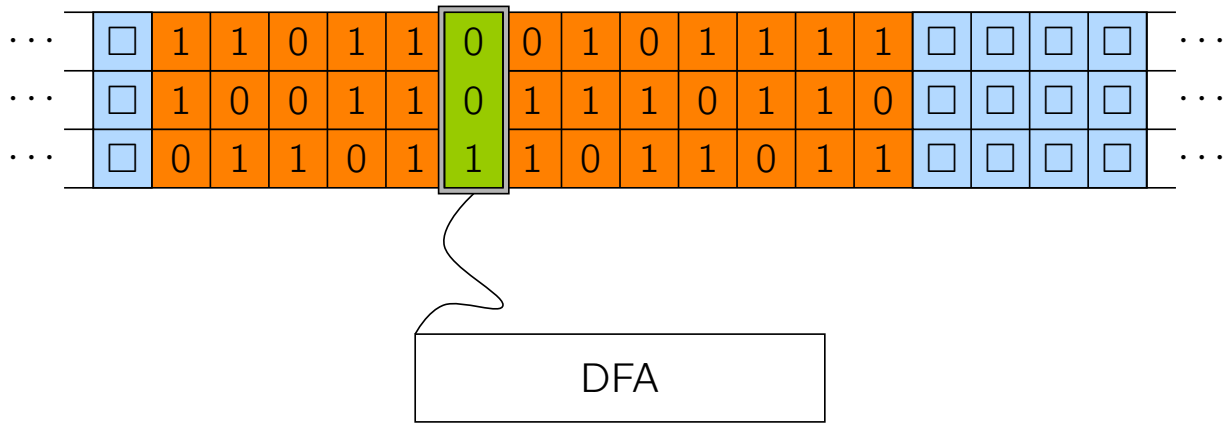
Neue Zustände sind dann zum Beispiel $(q_0, B), (q_1, 1)$.

Techniken zur Programmierung von Turingmaschinen

Technik 2: Mehrspurmaschinen

- k-spurige TM: eine TM, bei der das Band in k sogenannte Spuren eingeteilt ist. D.h. in jeder Bandzelle stehen k Zeichen, die der Kopf gleichzeitig einlesen kann.
- Das können wir erreichen, indem wir das Bandalphabet um k-dimensionale Vektoren erweitern, z.B.

$$\Gamma_{\text{neu}} := \Gamma \cup \Gamma^k \quad .$$

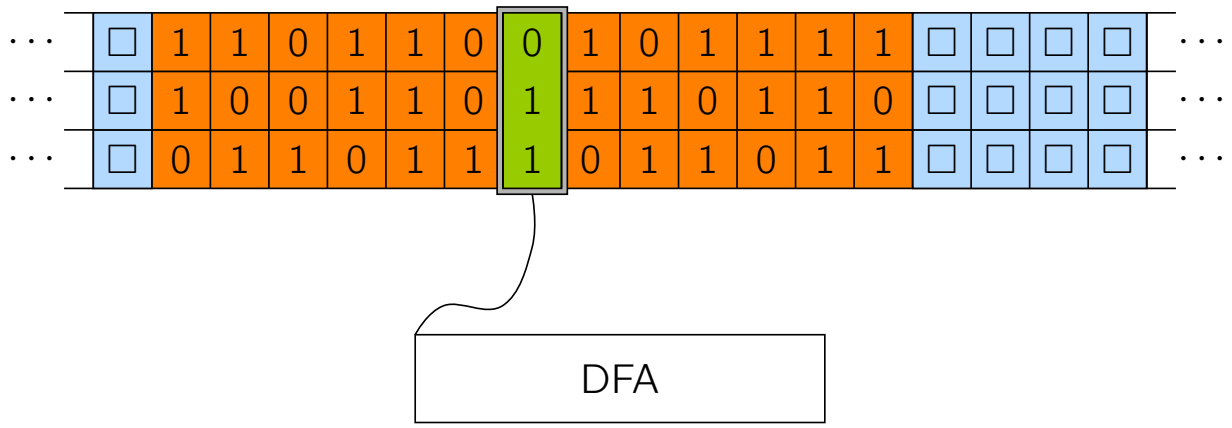


Techniken zur Programmierung von Turingmaschinen

Technik 2: Mehrspurmaschinen

- k-spurige TM: eine TM, bei der das Band in k sogenannte Spuren eingeteilt ist. D.h. in jeder Bandzelle stehen k Zeichen, die der Kopf gleichzeitig einlesen kann.
- Das können wir erreichen, indem wir das Bandalphabet um k-dimensionale Vektoren erweitern, z.B.

$$\Gamma_{\text{neu}} := \Gamma \cup \Gamma^k \quad .$$

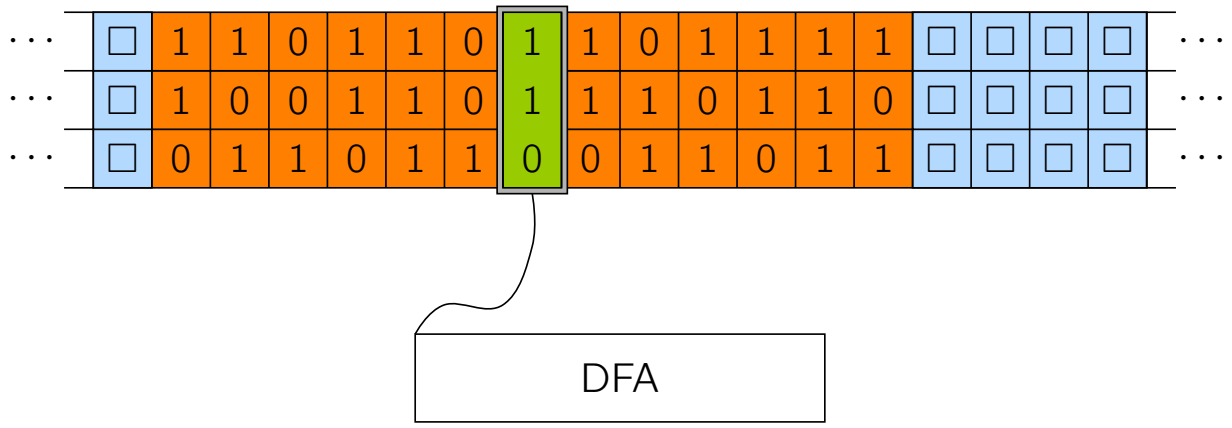


Techniken zur Programmierung von Turingmaschinen

Technik 2: Mehrspurmaschinen

- k-spurige TM: eine TM, bei der das Band in k sogenannte Spuren eingeteilt ist. D.h. in jeder Bandzelle stehen k Zeichen, die der Kopf gleichzeitig einlesen kann.
- Das können wir erreichen, indem wir das Bandalphabet um k-dimensionale Vektoren erweitern, z.B.

$$\Gamma_{\text{neu}} := \Gamma \cup \Gamma^k \quad .$$

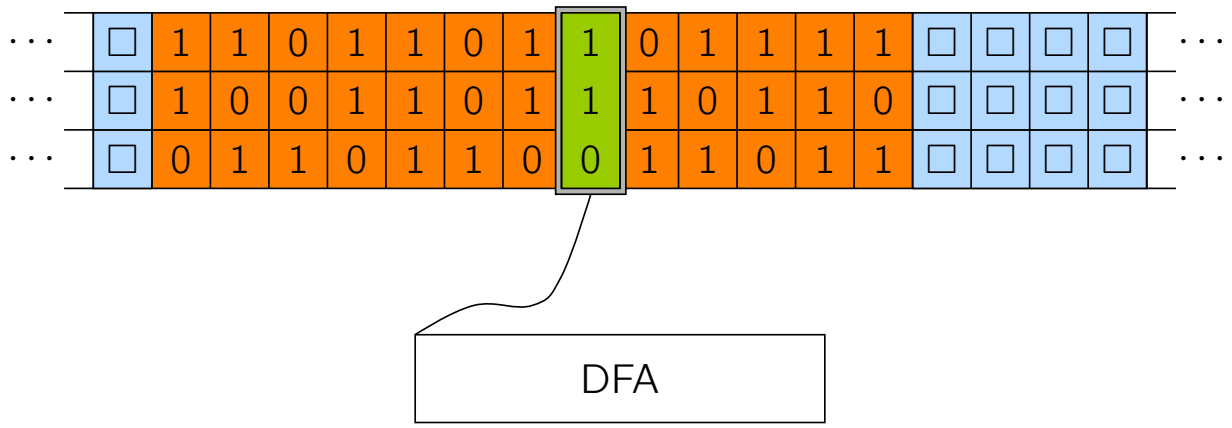


Techniken zur Programmierung von Turingmaschinen

Technik 2: Mehrspurmaschinen

- k-spurige TM: eine TM, bei der das Band in k sogenannte Spuren eingeteilt ist. D.h. in jeder Bandzelle stehen k Zeichen, die der Kopf gleichzeitig einlesen kann.
- Das können wir erreichen, indem wir das Bandalphabet um k-dimensionale Vektoren erweitern, z.B.

$$\Gamma_{\text{neu}} := \Gamma \cup \Gamma^k \quad .$$

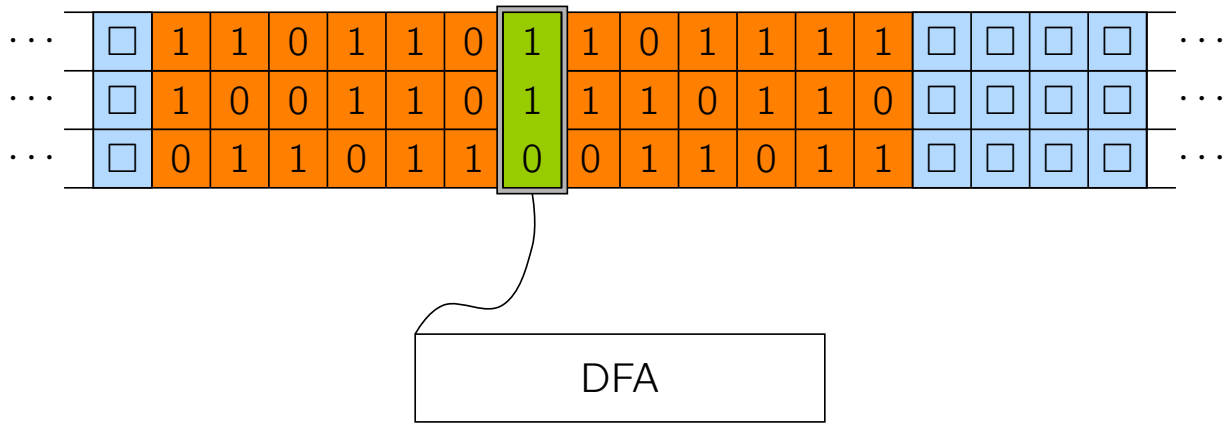


Techniken zur Programmierung von Turingmaschinen

Technik 2: Mehrspurmaschinen

- k-spurige TM: eine TM, bei der das Band in k sogenannte Spuren eingeteilt ist. D.h. in jeder Bandzelle stehen k Zeichen, die der Kopf gleichzeitig einlesen kann.
- Das können wir erreichen, indem wir das Bandalphabet um k-dimensionale Vektoren erweitern, z.B.

$$\Gamma_{\text{neu}} := \Gamma \cup \Gamma^k \quad .$$



Beispiel: Addition mittels 3-spuriger TM

Die Verwendung einer mehrspurigen TM erlaubt es, Algorithmen einfacher zu beschreiben.

Wir verdeutlichen dies am Beispiel der Addition. Aus der Eingabe $\mathsf{bin}(i_1)\#\,\mathsf{bin}(i_2)$ für $i_1, i_2 \in \mathbb{N}$ soll $\mathsf{bin}(i_1 + i_2)$ berechnet werden. Wir verwenden eine 3-spurige TM mit den Alphabeten $\Sigma = \{0, 1, \#\}$ und

$$\Gamma = \left\{ 0, 1, \#, \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \ldots, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, B \right\} \quad .$$

Beispiel: Addition mittels 3-spuriger TM

- *Schritt 1:* Transformation in Spurendarstellung: Schiebe die Eingabe so zusammen, dass die Binärkodierungen von i_1 und i_2 in der ersten und zweiten Spur rechtsbündig übereinander stehen.
  Aus der Eingabe 0011#0110 wird beispielsweise

$$B^* \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} B^* \ .$$

- *Schritt 2:* Addition nach der Schulmethode, indem der Kopf das Band von rechts nach links abläuft. Überträge werden im Zustand gespeichert. Als Ergebnis auf Spur 3 ergibt sich

$$B^* \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} B^* \ .$$

- *Schritt 3:* Rücktransformation von Spur 3 ins Einspur-Format: Ausgabe 1001.

Techniken zur Programmierung von TMen

Standardtechniken aus der Programmierung können auch auf TMen implementiert werden.

- *Schleifen* haben wir bereits an Beispielen gesehen.
- *Variablen* können realisiert werden, indem wir pro Variable eine Spur reservieren.
- *Felder (Arrays)* können ebenfalls auf einer Spur abgespeichert werden.
- *Unterprogramme* können implementiert werden, indem wir eine Spur des Bandes als Prozedurstack verwenden.

Basierend auf diesen Techniken können wir uns klar machen, dass bekannte Algorithmen, z.B. zum Sortieren von Daten, ohne Weiteres auf einer TM ausgeführt werden können.

Vorlesung 3
Mehrband-Turingmaschinen und die universelle Turingmaschine

Wdh.: Kodierung von Berechnungsproblemen

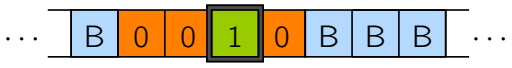
3 mögliche formale Definitionen.

Als Relation:	Als Funktion	Als Sprache
▶ Primfaktor:		
$(110, 11) \in R$ $(101, 11) \notin R$ $(00110, 11) \notin R$		
▶ Multiplikation		
$(11\#10, 110) \in R$	$f(11\#10) = 110$	
$(11\#10, 11) \notin R$		
$(1\#1\#0, 110) \notin R$	$(f(1\#1\#0) = \perp)$	
▶ Wörter die auf 1 enden.		
$(11, 1) \in R$ $(110, 1) \notin R$ $(10, 0) \in R$	$f(11) = 1$ $f(110) = 0$	$11 \in L$ $110 \notin L$

Wdh.: Turingmaschinen

Wdh.: Turingmaschinen

Anschauliche Definition:



δ	0	1	B
q_1	$(q_1, 1, L)$	$(q_2, 1, R)$	(q_1, B, N)
q_2	(q_3, B, R)	$(q_1, 0, L)$	(q_3, B, R)
q_3	$(q_2, 0, N)$	$(q_2, 0, R)$	(q_3, B, R)

Formale Definition:

Eine Turingmaschine ist ein 7-Tupel $(Q, \Sigma, \Gamma, B, q_0, \bar{q}, \delta)$, wobei

- Q, Σ, Γ endliche Mengen sind,
- $\Sigma \subseteq \Gamma$,
- $B \in \Gamma \setminus \Sigma$,
- $q_0, \bar{q} \in Q$ und
- $\delta \colon (Q \setminus \{\bar{q}\}) \times \Gamma \to Q \times \Gamma \times \{R, L, N\}$.

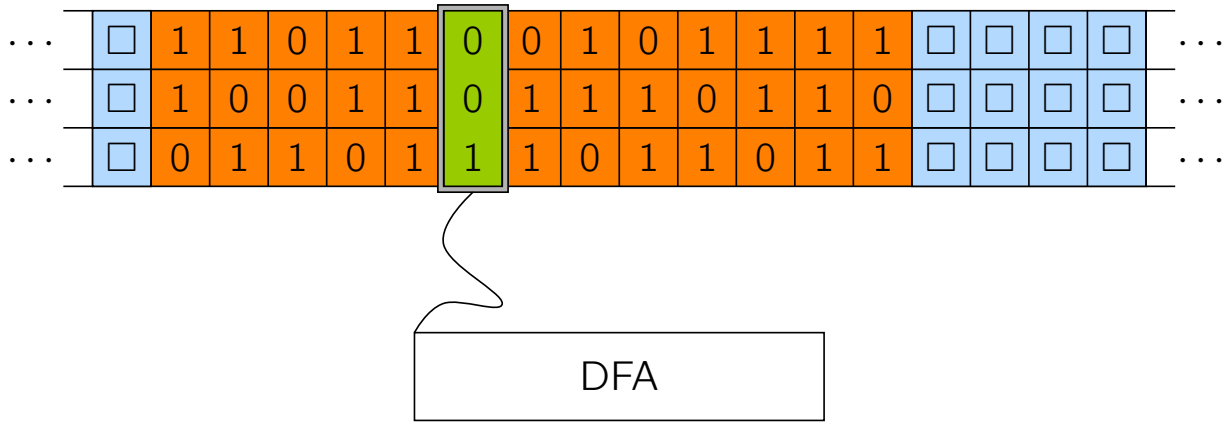
Wdh.: TM-Techniken

- Speicher im Zustandsraum:

$$Q_{\text{neu}} := Q \times \Gamma^k$$

- Mehrspurmaschinen:

$$\Gamma_{\text{neu}} := \Gamma \cup \Gamma^k$$



- Schleifen, Variablen, Felder (Arrays), Unterprogramme

Turingmaschinen mit mehreren Bändern

Turingmaschinen mit mehreren Bändern

k-Band-TM

Eine k-Band-TM ist eine Verallgemeinerung der Turingmaschine und verfügt über k Arbeitsbänder mit jeweils einem unabhängigen Kopf.
Die Zustandsübergangsfunktion ist entsprechend von der Form

$$\delta : (Q \setminus \{\bar{q}\}) \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, N\}^k \quad .$$

- Band 1 fungiert als Ein-/Ausgabeband wie bei der (1-Band-)TM.
- Die Zellen der Bänder $2, \ldots, k$ sind initial leer (ausschließlich B).

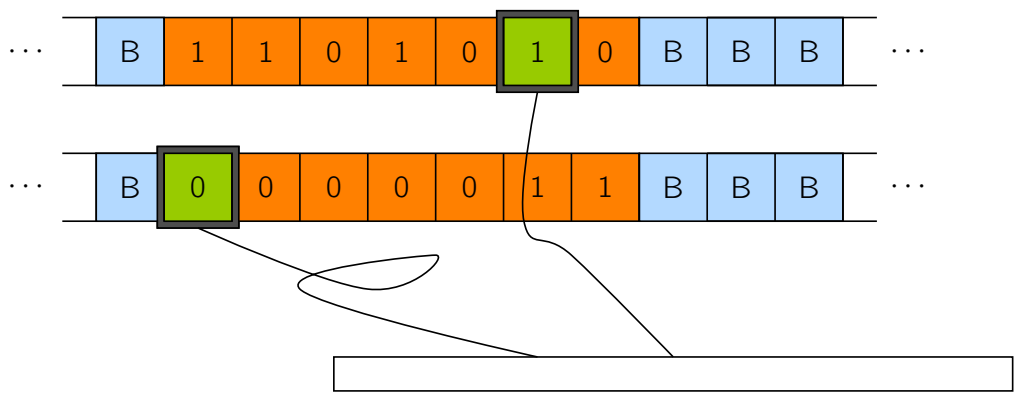
Turingmaschinen mit mehreren Bändern

k-Band-TM

Eine k-Band-TM ist eine Verallgemeinerung der Turingmaschine und verfügt über k Arbeitsbänder mit jeweils einem unabhängigen Kopf.
Die Zustandsübergangsfunktion ist entsprechend von der Form

$$\delta : (Q \setminus \{\bar{q}\}) \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, N\}^k \quad .$$

- Band 1 fungiert als Ein-/Ausgabeband wie bei der (1-Band-)TM.
- Die Zellen der Bänder $2, \ldots, k$ sind initial leer (ausschließlich B).



Simulation k-Band-TM durch 1-Band-TM

Satz

Eine k-Band-TM M, die mit Rechenzeit $t(n)$ und Platz $s(n)$ auskommt, kann von einer (1-Band-)TM M' mit Zeitbedarf $O(t^2(n))$ und Platzbedarf $O(s(n))$ simuliert werden.

Simulation k-Band-TM durch 1-Band-TM

Satz

Eine k-Band-TM M, die mit Rechenzeit $t(n)$ und Platz $s(n)$ auskommt, kann von einer (1-Band-)TM M' mit Zeitbedarf $O(t^2(n))$ und Platzbedarf $O(s(n))$ simuliert werden.

Beweisskizze

Die TM M' verwendet $2k$ Spuren. Nach Simulation des t-ten Schrittes für $0 \leq t \leq t(n)$ gilt

- Die ungeraden Spuren $1, 3, \ldots, 2k-1$ enthalten den Inhalt der Bänder $1, \ldots, k$ von M.

Simulation k-Band-TM durch 1-Band-TM

Satz

Eine k-Band-TM M, die mit Rechenzeit $t(n)$ und Platz $s(n)$ auskommt, kann von einer (1-Band-)TM M' mit Zeitbedarf $O(t^2(n))$ und Platzbedarf $O(s(n))$ simuliert werden.

Beweisskizze

Die TM M' verwendet $2k$ Spuren. Nach Simulation des t-ten Schrittes für $0 \leq t \leq t(n)$ gilt

- Die ungeraden Spuren $1, 3, \ldots, 2k-1$ enthalten den Inhalt der Bänder $1, \ldots, k$ von M.
- Auf den geraden Spuren $2, 4, \ldots, 2k$ sind die Kopfpositionen auf diesen Bändern mit dem Zeichen $\#$ markiert.

Simulation k-Band-TM durch 1-Band-TM

Satz

Eine k-Band-TM M, die mit Rechenzeit $t(n)$ und Platz $s(n)$ auskommt, kann von einer (1-Band-)TM M' mit Zeitbedarf $O(t^2(n))$ und Platzbedarf $O(s(n))$ simuliert werden.

Beweisskizze

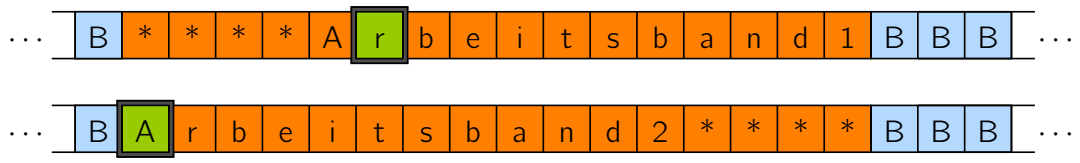
Die TM M' verwendet $2k$ Spuren. Nach Simulation des t-ten Schrittes für $0 \leq t \leq t(n)$ gilt

- Die ungeraden Spuren $1, 3, \ldots, 2k-1$ enthalten den Inhalt der Bänder $1, \ldots, k$ von M.
- Auf den geraden Spuren $2, 4, \ldots, 2k$ sind die Kopfpositionen auf diesen Bändern mit dem Zeichen $\#$ markiert.

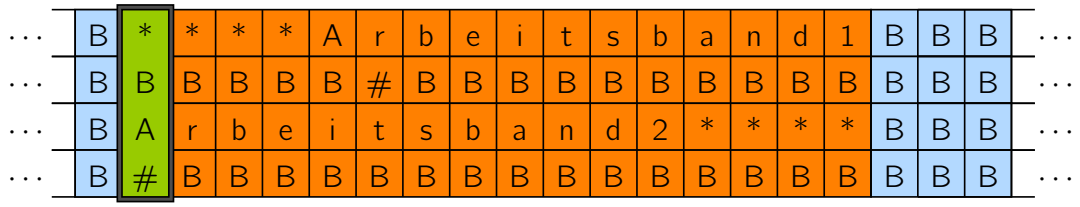
Diese Initialisierung der Spuren ist in Zeit $O(1)$ möglich.

Simulation k-Band-TM durch 1-Band-TM – Illustration

simulierte 2-Band-Turingmaschine M



simulierende 4-Spur-Turingmaschine M' (zu Beginn des Simulationsschrittes)



Simulation k-Band-TM durch 1-Band-TM – Beweis

Jeder Rechenschritt von M wird durch M' wie folgt simuliert.

- Am Anfang stehe der Kopf von M' auf der linkesten Zelle, die $\#$ enthält, und M' kenne den Zustand von M.

Simulation k-Band-TM durch 1-Band-TM – Beweis

Jeder Rechenschritt von M wird durch M' wie folgt simuliert.

- Am Anfang stehe der Kopf von M' auf der linkesten Zelle, die $\#$ enthält, und M' kenne den Zustand von M.
- Der Kopf von M' läuft nach rechts bis zum rechtesten $\#$, wobei die k Zeichen an den mit $\#$ markierten Spurpositionen im Zustand abgespeichert werden.

Simulation k-Band-TM durch 1-Band-TM – Beweis

Jeder Rechenschritt von M wird durch M' wie folgt simuliert.

- Am Anfang stehe der Kopf von M' auf der linkesten Zelle, die $\#$ enthält, und M' kenne den Zustand von M.
- Der Kopf von M' läuft nach rechts bis zum rechtesten $\#$, wobei die k Zeichen an den mit $\#$ markierten Spurpositionen im Zustand abgespeichert werden.
- An der Zelle mit dem rechtesten $\#$-Zeichen angekommen, kann M' die Übergangsfunktion von M auswerten und kennt den neuen Zustand von M sowie die erforderlichen Übergänge auf den k Bändern.

Simulation k-Band-TM durch 1-Band-TM – Beweis

Jeder Rechenschritt von M wird durch M' wie folgt simuliert.

- Am Anfang stehe der Kopf von M' auf der linkesten Zelle, die $\#$ enthält, und M' kenne den Zustand von M.
- Der Kopf von M' läuft nach rechts bis zum rechtesten $\#$, wobei die k Zeichen an den mit $\#$ markierten Spurpositionen im Zustand abgespeichert werden.
- An der Zelle mit dem rechtesten $\#$-Zeichen angekommen, kann M' die Übergangsfunktion von M auswerten und kennt den neuen Zustand von M sowie die erforderlichen Übergänge auf den k Bändern.
- Nun läuft der Kopf von M' zurück, verändert dabei die Bandinschriften an den mit $\#$ markierten Stellen und verschiebt, falls erforderlich, auch die $\#$-Markierungen um eine Position nach links oder rechts.

Simulation k-Band-TM durch 1-Band-TM – Beweis

Laufzeitanalyse:

Wieviele Bandpositionen können zwischen dem linkesten und dem rechtesten $\#$ liegen?

Simulation k-Band-TM durch 1-Band-TM – Beweis

Laufzeitanalyse:

Wieviele Bandpositionen können zwischen dem linkesten und dem rechtesten $\#$ liegen?

Nach t Schritten können diese Markierungen höchstens $2t$ Positionen auseinanderliegen.

Also ist der Abstand zwischen diesen Zeichen und somit auch die Laufzeit zur Simulation eines Schrittes durch $O(t(n))$ beschränkt.

Simulation k-Band-TM durch 1-Band-TM – Beweis

Laufzeitanalyse:

Wieviele Bandpositionen können zwischen dem linkesten und dem rechtesten $\#$ liegen?

Nach t Schritten können diese Markierungen höchstens $2t$ Positionen auseinanderliegen.

Also ist der Abstand zwischen diesen Zeichen und somit auch die Laufzeit zur Simulation eines Schrittes durch $O(t(n))$ beschränkt.

Insgesamt ergibt das zur Simulation von $t(n)$ Schritten eine Laufzeitschranke von $O(t(n)^2)$. □

Special versus General Purpose Rechner

- Bisher haben wir für jedes Problem eine eigene TM entworfen, einen special purpose Rechner.

Special versus General Purpose Rechner

- Bisher haben wir für jedes Problem eine eigene TM entworfen, einen special purpose Rechner.
- Real existierende Maschinen sind jedoch programmierbare general purpose Rechner.

Special versus General Purpose Rechner

- Bisher haben wir für jedes Problem eine eigene TM entworfen, einen special purpose Rechner.
- Real existierende Maschinen sind jedoch programmierbare general purpose Rechner.
- Wir konstruieren jetzt eine programmierbare Variante der TM, die sogenannte universelle TM.

Ein-/Ausgabeverhalten der universellen TM

- Das Programm der universellen TM U ist die Kodierung einer beliebigen TM M.

Ein-/Ausgabeverhalten der universellen TM

- Das Programm der universellen TM U ist die Kodierung einer beliebigen TM M.
- Mit $\langle M \rangle$ bezeichnen wir diese Kodierung der TM M.
- Als Eingabe erhält U einen String der Form $\langle M \rangle w$ bestehend aus einer TM-Kodierung $\langle M \rangle$ und einem Wort w.

Ein-/Ausgabeverhalten der universellen TM

- Das Programm der universellen TM U ist die Kodierung einer beliebigen TM M.
- Mit $\langle M \rangle$ bezeichnen wir diese Kodierung der TM M.
- Als Eingabe erhält U einen String der Form $\langle M \rangle w$ bestehend aus einer TM-Kodierung $\langle M \rangle$ und einem Wort w.
- Die universelle TM simuliert das Verhalten der TM M auf der Eingabe w.

Ein-/Ausgabeverhalten der universellen TM

- Das Programm der universellen TM U ist die Kodierung einer beliebigen TM M.
- Mit $\langle M \rangle$ bezeichnen wir diese Kodierung der TM M.
- Als Eingabe erhält U einen String der Form $\langle M \rangle w$ bestehend aus einer TM-Kodierung $\langle M \rangle$ und einem Wort w.
- Die universelle TM simuliert das Verhalten der TM M auf der Eingabe w.
- Bei inkorrekter Eingabe (d.h., die Eingabe beginnt nicht mit einer TM-Kodierung) gibt U eine Fehlermeldung aus.

Gödelnummern

Wir entwickeln nun eine eindeutige präfixfreie Kodierung, die einer Turingmaschine M ein Wort $\langle M \rangle$ über dem Alphabet $\{0, 1\}$ zuordnet.

Definition

Wir nennen die Kodierung $\langle M \rangle$ die Gödelnummer der Turingmaschine M.

- *Präfixfrei* bedeutet, dass keine Gödelnummer Präfix (Anfangsteilwort) einer anderen Gödelnummer sein darf.

Gödelnummern

Wir entwickeln nun eine eindeutige präfixfreie Kodierung, die einer Turingmaschine M ein Wort $\langle M \rangle$ über dem Alphabet $\{0, 1\}$ zuordnet.

Definition

Wir nennen die Kodierung $\langle M \rangle$ die Gödelnummer der Turingmaschine M.

- *Präfixfrei* bedeutet, dass keine Gödelnummer Präfix (Anfangsteilwort) einer anderen Gödelnummer sein darf.
- Um Präfixfreiheit zu erreichen, vereinbaren wir, dass alle Gödelnummern mit 111 beginnen und auf 111 enden und ansonsten der Teilstring 111 nicht in der Kodierung vorkommt.

Realisierung von Gödelnummern

Zur präfixfreien Kodierung von TMen gibt es viele Möglichkeiten. Wir stellen jetzt eine mögliche Definition der Gödelnummer vor.

Realisierung von Gödelnummern

Zur präfixfreien Kodierung von TMen gibt es viele Möglichkeiten. Wir stellen jetzt eine mögliche Definition der Gödelnummer vor.

Wir beschränken uns (O.B.d.A.) auf TMen der folgenden Form:

- $Q = \{q_1, \ldots, q_t\}$ für ein $t \geq 2$.
- Der Anfangszustand ist q_1 und der Endzustand ist q_2.
- $\Gamma = \{0, 1, B\}$.

Zur Beschreibung von TMen dieser Form müssen wir nur die Übergangsfunktion als Binärstring kodieren.

Realisierung von Gödelnummern

Zur präfixfreien Kodierung von TMen gibt es viele Möglichkeiten. Wir stellen jetzt eine mögliche Definition der Gödelnummer vor.

Wir beschränken uns (O.B.d.A.) auf TMen der folgenden Form:

- $Q = \{q_1, \ldots, q_t\}$ für ein $t \geq 2$.
- Der Anfangszustand ist q_1 und der Endzustand ist q_2.
- $\Gamma = \{0, 1, B\}$.

Zur Beschreibung von TMen dieser Form müssen wir nur die Übergangsfunktion als Binärstring kodieren.

- Wir nummerieren das Alphabet durch, indem wir $X_1 = 0$, $X_2 = 1$ und $X_3 = B$ setzen.
- Auch die möglichen Kopfbewegungen nummerieren wir, indem wir $D_1 = L$, $D_2 = N$ und $D_3 = R$ setzen.

Realisierung von Gödelnummern

Kodierung der Übergangsfunktion

- Der Übergang $\delta(q_i, X_j) = (q_k, X_\ell, D_m)$ wird kodiert durch den Binärstring

$$0^i 1 0^j 1 0^k 1 0^\ell 1 0^m \quad .$$

Realisierung von Gödelnummern

Kodierung der Übergangsfunktion

- Der Übergang $\delta(q_i, X_j) = (q_k, X_\ell, D_m)$ wird kodiert durch den Binärstring

$$0^i 1 0^j 1 0^k 1 0^\ell 1 0^m \quad .$$

- Die Kodierung des t-ten Übergangs bezeichnen wir mit $\mathrm{code}(t)$.

Realisierung von Gödelnummern

Kodierung der Übergangsfunktion

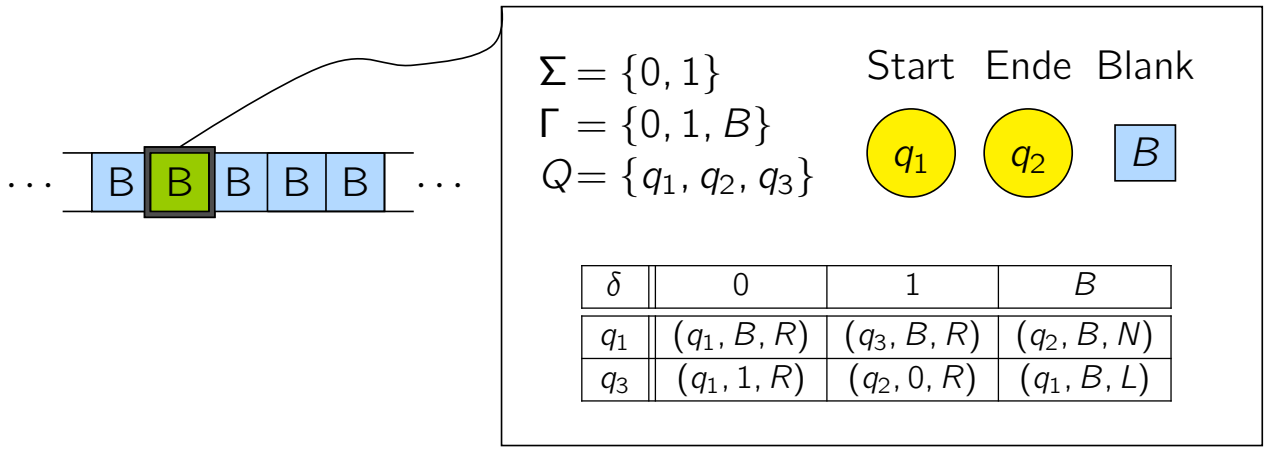
- Der Übergang $\delta(q_i, X_j) = (q_k, X_\ell, D_m)$ wird kodiert durch den Binärstring

$$0^i 1 0^j 1 0^k 1 0^\ell 1 0^m \quad .$$

- Die Kodierung des t-ten Übergangs bezeichnen wir mit $\text{code}(t)$.
- Die Gödelnummer einer TM M mit s Übergängen ist dann

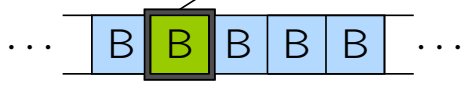
$$\langle M \rangle \;=\; 111\ \text{code}(1)\ 11\ \text{code}(2)\ 11 \ldots 11\ \text{code}(s)\ 111 \quad .$$

Beispielkodierung



δ	0	1	B
q_1	(q_1, B, R)	(q_3, B, R)	(q_2, B, N)
q_3	$(q_1, 1, R)$	$(q_2, 0, R)$	(q_1, B, L)

Beispielkodierung



$\Sigma = \{0, 1\}$
$\Gamma = \{0, 1, B\}$
$Q = \{q_1, q_2, q_3\}$

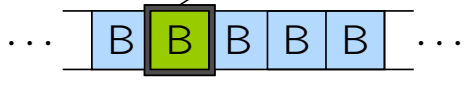
Start q_1 Ende q_2 Blank B

δ	0	1	B
q_1	(q_1, B, R)	(q_3, B, R)	(q_2, B, N)
q_3	$(q_1, 1, R)$	$(q_2, 0, R)$	(q_1, B, L)

111

111

Beispielkodierung



$\Sigma = \{0, 1\}$
$\Gamma = \{0, 1, B\}$
$Q = \{q_1, q_2, q_3\}$

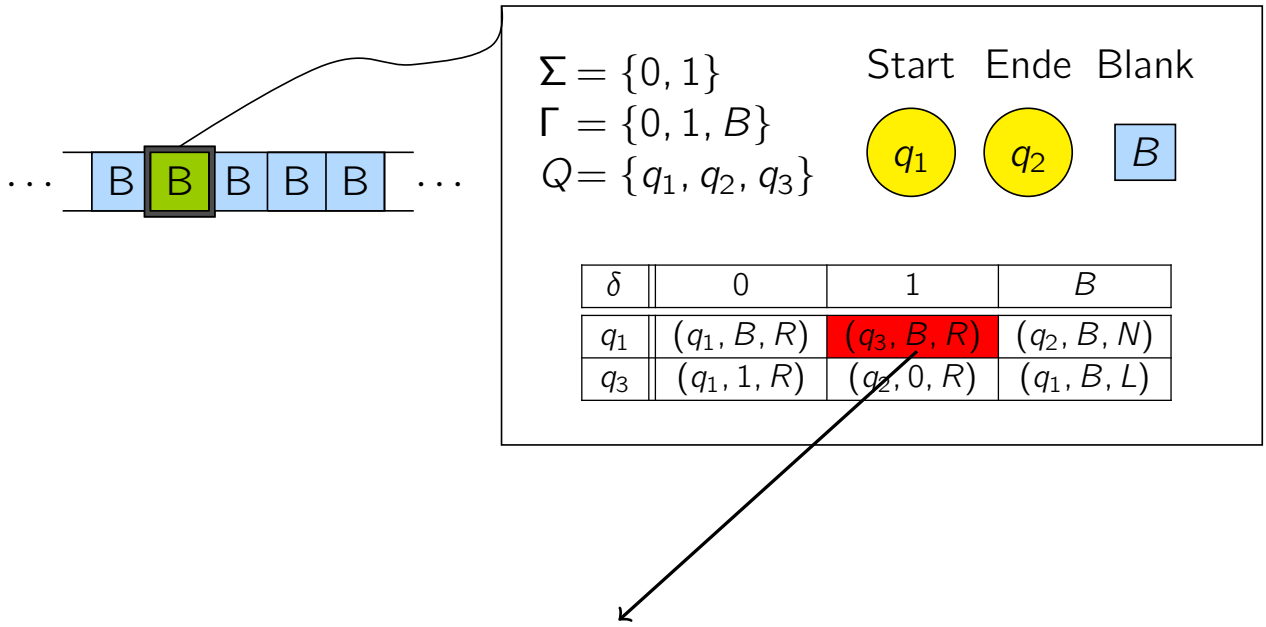
Start q_1 Ende q_2 Blank B

δ	0	1	B
q_1	(q_1, B, R)	(q_3, B, R)	(q_2, B, N)
q_3	$(q_1, 1, R)$	$(q_2, 0, R)$	(q_1, B, L)

111 0101010001000

111

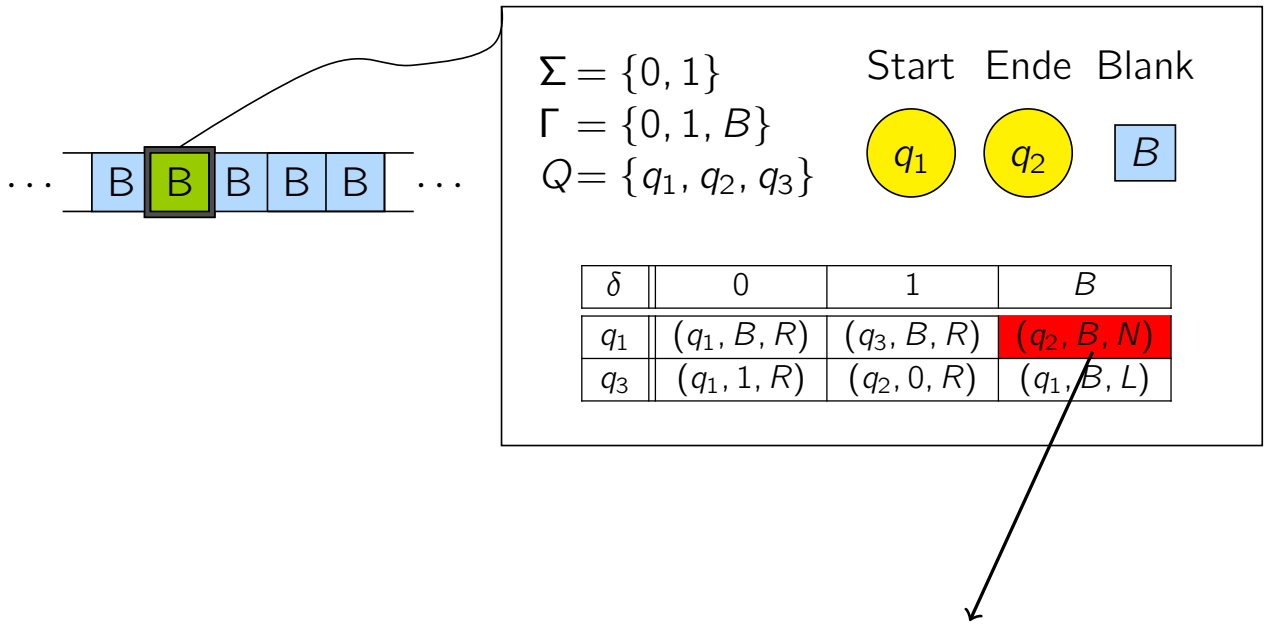
Beispielkodierung



111 0101010001000 11 0100100010001000

111

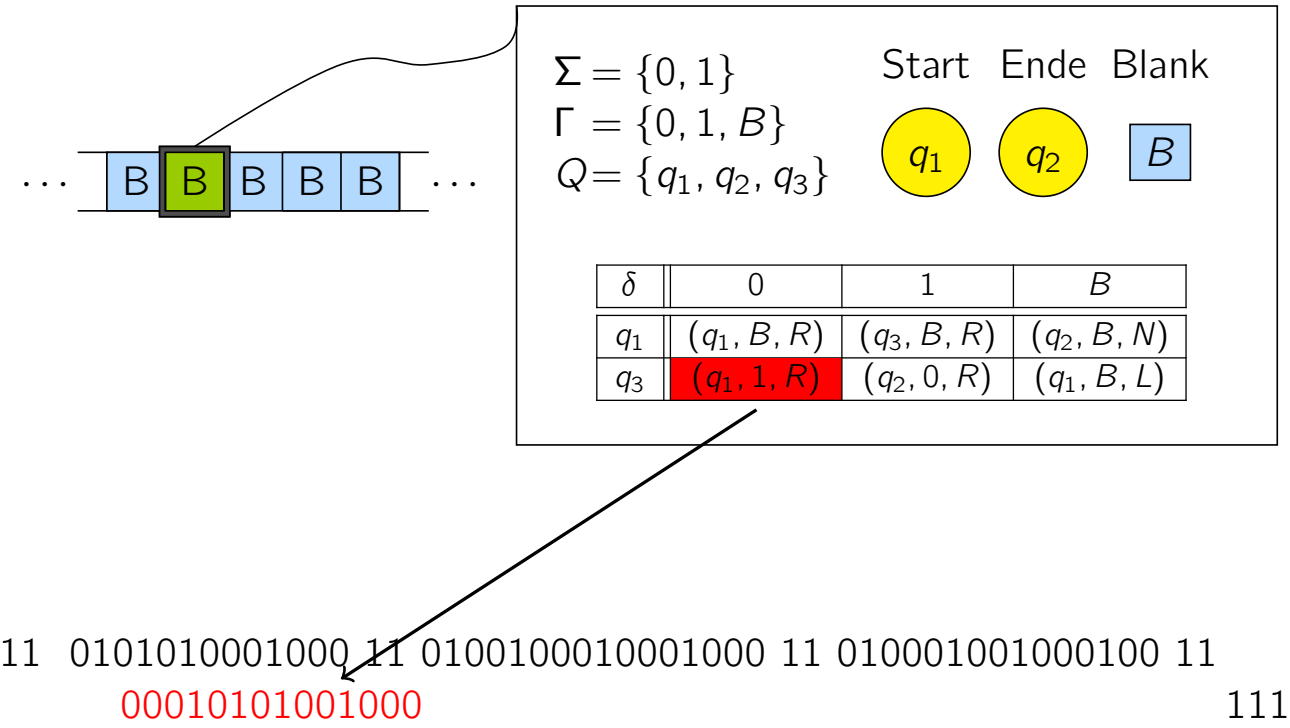
Beispielkodierung



111 0101010001000 11 0100100010001000 11 0100010010001000

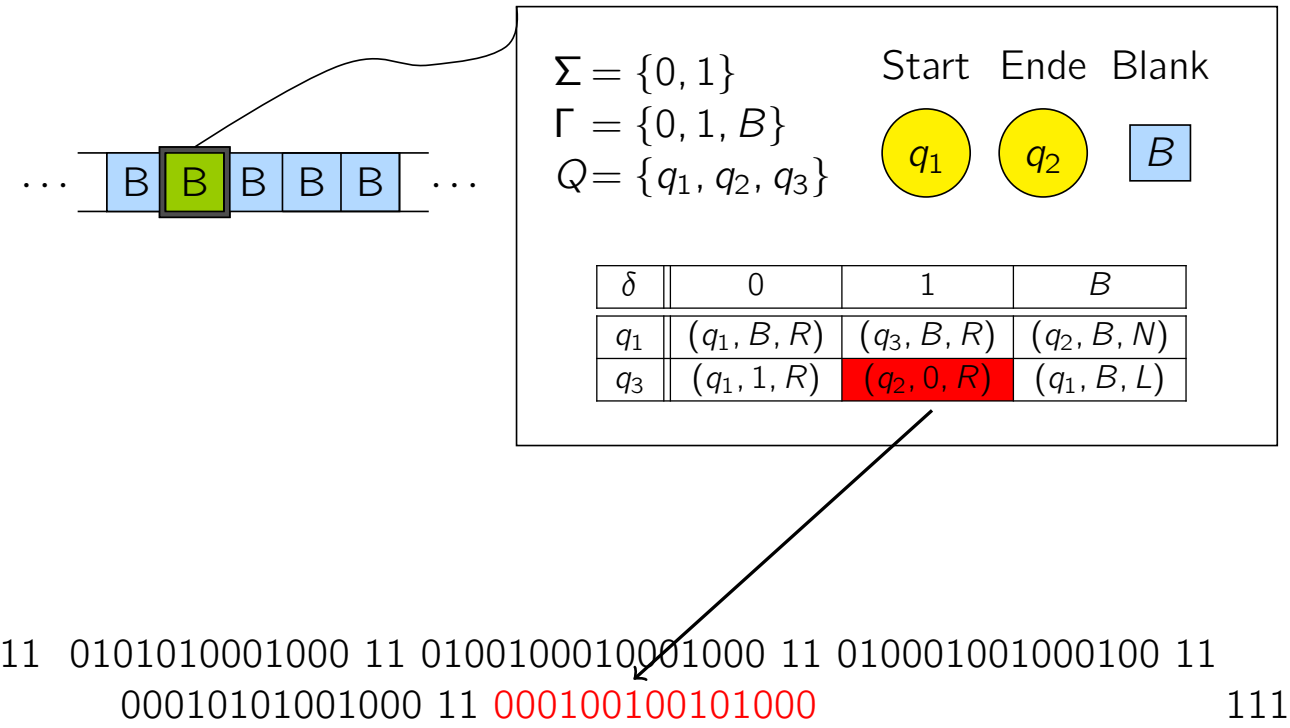
111

Beispielkodierung



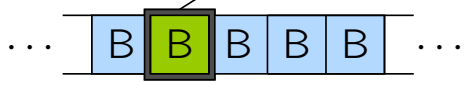
111 0101010001000 11 0100100010001000 11 010001001000100 11
00010101001000 111

Beispielkodierung



111 0101010001000 11 0100100010001000 11 010001001000100 11
00010101001000 11 000100100101000 111

Beispielkodierung



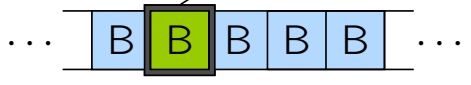
$\Sigma = \{0, 1\}$
$\Gamma = \{0, 1, B\}$
$Q = \{q_1, q_2, q_3\}$

Start q_1 Ende q_2 Blank B

δ	0	1	B
q_1	(q_1, B, R)	(q_3, B, R)	(q_2, B, N)
q_3	$(q_1, 1, R)$	$(q_2, 0, R)$	(q_1, B, L)

111 0101010001000 11 0100100010001000 11 0100010010001000 11
00010101001000 11 000100100101000 11 000100010100010 111

Beispielkodierung



$\Sigma = \{0, 1\}$
$\Gamma = \{0, 1, B\}$
$Q = \{q_1, q_2, q_3\}$

Start q_1 Ende q_2 Blank B

δ	0	1	B
q_1	(q_1, B, R)	(q_3, B, R)	(q_2, B, N)
q_3	$(q_1, 1, R)$	$(q_2, 0, R)$	(q_1, B, L)

111 0101010001000 11 0100100010001000 11 0100010010001000 11
00010101001000 11 000100100101000 11 000100010100010 111

Implementierung der universellen TM

Als Eingabe erhält die universelle TM U ein Wort der Form $\langle M \rangle w$ für beliebiges $w \in \{0,1\}^*$.

Implementierung der universellen TM

Als Eingabe erhält die universelle TM U ein Wort der Form $\langle M \rangle w$ für beliebiges $w \in \{0,1\}^*$.

Wir implementieren U zunächst in Form einer 3-Band-TM:

- Band 1 von U simuliert das Band der TM M.

Implementierung der universellen TM

Als Eingabe erhält die universelle TM U ein Wort der Form $\langle M \rangle w$ für beliebiges $w \in \{0,1\}^*$.

Wir implementieren U zunächst in Form einer 3-Band-TM:

- Band 1 von U simuliert das Band der TM M.
- Band 2 von U enthält die Gödelnummer von M.

Implementierung der universellen TM

Als Eingabe erhält die universelle TM U ein Wort der Form $\langle M \rangle w$ für beliebiges $w \in \{0,1\}^*$.

Wir implementieren U zunächst in Form einer 3-Band-TM:

- Band 1 von U simuliert das Band der TM M.
- Band 2 von U enthält die Gödelnummer von M.
- Auf Band 3 speichert U den jeweils aktuellen Zustand von M.

Implementierung der universellen TM

Initialisierung:

- U überprüft, ob die Eingabe eine korrekte Gödelnummer enthält. Falls nein, Fehlerausgabe.

Implementierung der universellen TM

Initialisierung:

- U überprüft, ob die Eingabe eine korrekte Gödelnummer enthält. Falls nein, Fehlerausgabe.
- U kopiert die Gödelnummer auf Band 2 und schreibt die Kodierung des Anfangszustands auf Band 3.

Implementierung der universellen TM

Initialisierung:

- U überprüft, ob die Eingabe eine korrekte Gödelnummer enthält. Falls nein, Fehlerausgabe.
- U kopiert die Gödelnummer auf Band 2 und schreibt die Kodierung des Anfangszustands auf Band 3.
- U bereitet Band 1 so vor, dass es nur das Wort w enthält. Der Kopf steht auf dem ersten Zeichen von w.

Implementierung der universellen TM

Initialisierung:

- U überprüft, ob die Eingabe eine korrekte Gödelnummer enthält. Falls nein, Fehlerausgabe.
- U kopiert die Gödelnummer auf Band 2 und schreibt die Kodierung des Anfangszustands auf Band 3.
- U bereitet Band 1 so vor, dass es nur das Wort w enthält. Der Kopf steht auf dem ersten Zeichen von w.

Laufzeit?

Implementierung der universellen TM

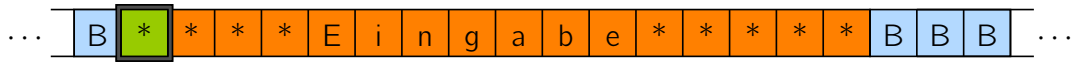
Initialisierung:

- U überprüft, ob die Eingabe eine korrekte Gödelnummer enthält. Falls nein, Fehlerausgabe.
- U kopiert die Gödelnummer auf Band 2 und schreibt die Kodierung des Anfangszustands auf Band 3.
- U bereitet Band 1 so vor, dass es nur das Wort w enthält. Der Kopf steht auf dem ersten Zeichen von w.

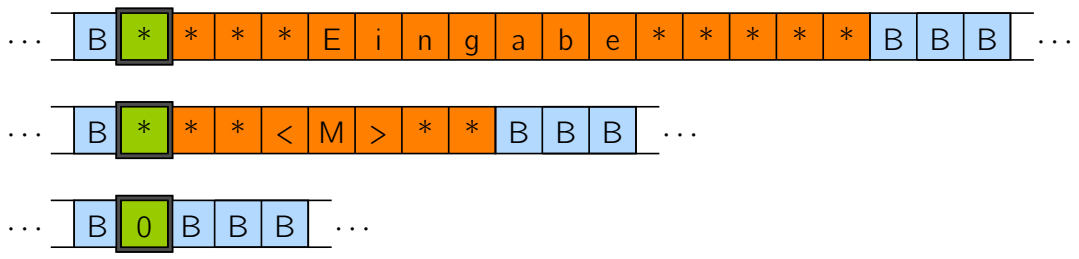
Laufzeit? – Die Laufzeit ist $O(1)$, wenn wir die Kodierungslänge von M als Konstante ansehen.

Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M



Initialisierung der universellen Maschine U



Implementierung der universellen TM

Simulation eines Schritts von M:

U sucht zu dem Zeichen an der Kopfposition auf Band 1 und dem Zustand auf Band 3 die Kodierung des entsprechenden Übergangs von M auf Band 2.

Implementierung der universellen TM

Simulation eines Schritts von M:

U sucht zu dem Zeichen an der Kopfposition auf Band 1 und dem Zustand auf Band 3 die Kodierung des entsprechenden Übergangs von M auf Band 2.

Wie in der Übergangsfunktion beschrieben

- aktualisiert U die Inschrift auf Band 1,
- bewegt U den Kopf auf Band 1, und
- verändert U den auf Band 3 abgespeicherten Zustand von M.

Implementierung der universellen TM

Simulation eines Schritts von M:

U sucht zu dem Zeichen an der Kopfposition auf Band 1 und dem Zustand auf Band 3 die Kodierung des entsprechenden Übergangs von M auf Band 2.

Wie in der Übergangsfunktion beschrieben

- aktualisiert U die Inschrift auf Band 1,
- bewegt U den Kopf auf Band 1, und
- verändert U den auf Band 3 abgespeicherten Zustand von M.

Laufzeit eines Simulationsschrittes: $O(1)$.

Implementierung der universellen TM

Simulation eines Schritts von M:

U sucht zu dem Zeichen an der Kopfposition auf Band 1 und dem Zustand auf Band 3 die Kodierung des entsprechenden Übergangs von M auf Band 2.

Wie in der Übergangsfunktion beschrieben

- aktualisiert U die Inschrift auf Band 1,
- bewegt U den Kopf auf Band 1, und
- verändert U den auf Band 3 abgespeicherten Zustand von M.

Laufzeit eines Simulationsschrittes: $O(1)$.

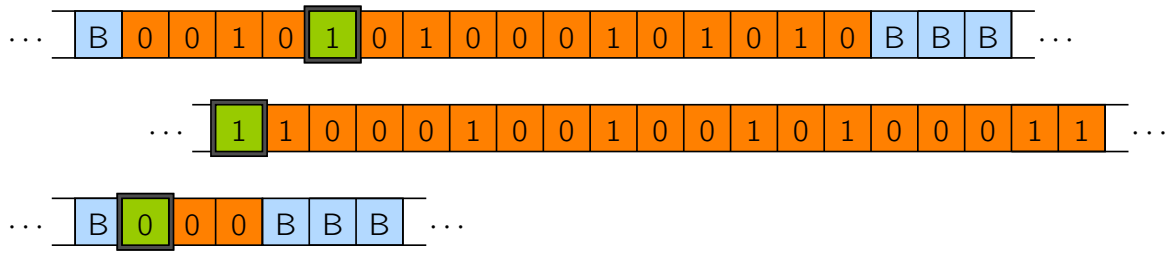
Das bedeutet, U simuliert M mit konstantem Zeitverlust!

Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M

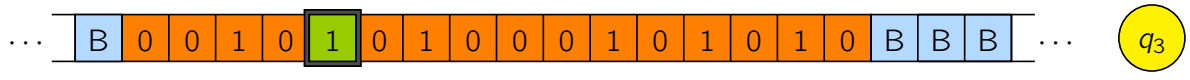
δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

simulierende universelle Maschine U



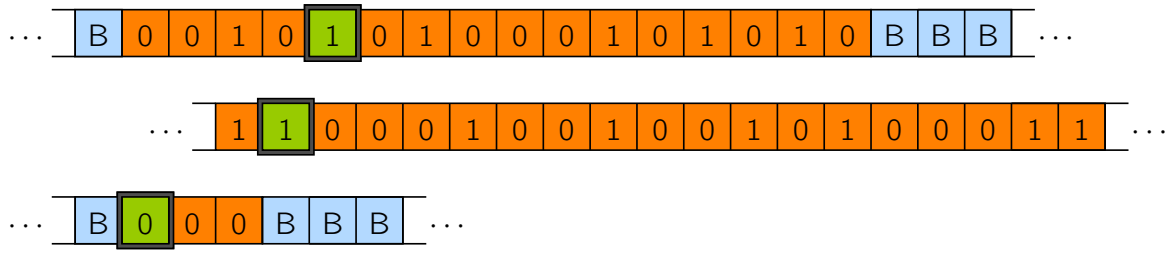
Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M



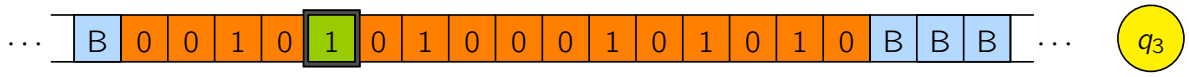
δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

simulierende universelle Maschine U



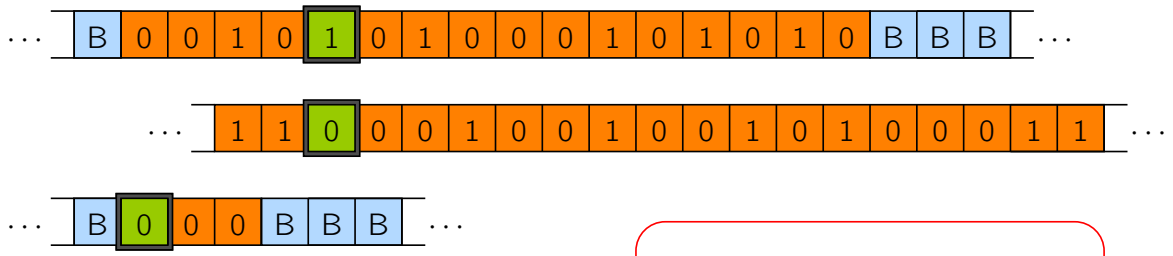
Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M



δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

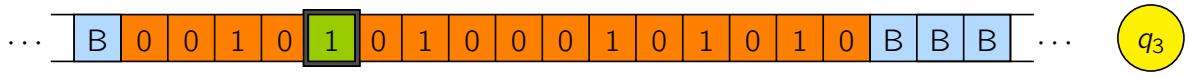
simulierende universelle Maschine U



Vergleiche Band 3 mit Band 2

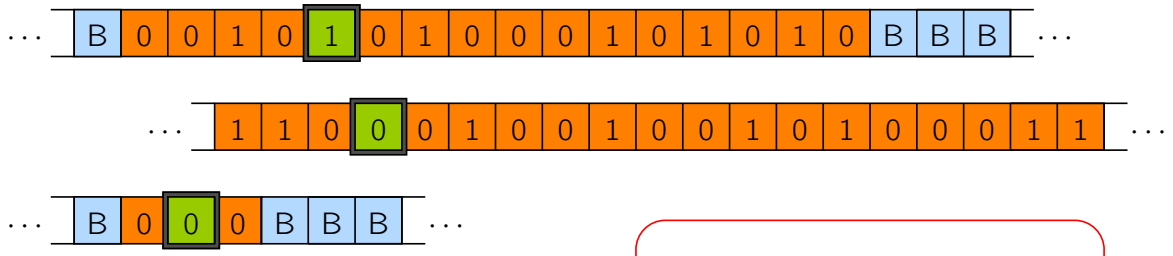
Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M



δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

simulierende universelle Maschine U



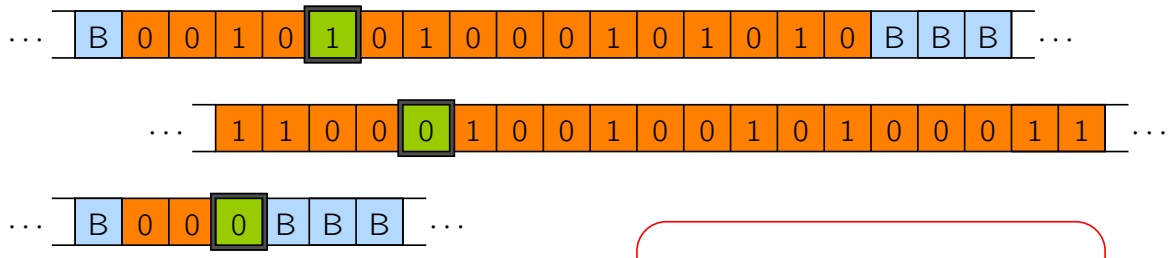
Vergleiche Band 3 mit Band 2

Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M

δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

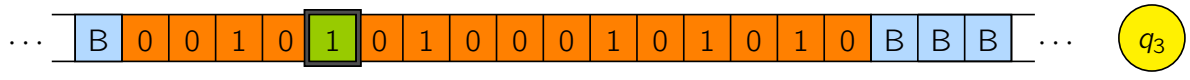
simulierende universelle Maschine U



Vergleiche Band 3 mit Band 2

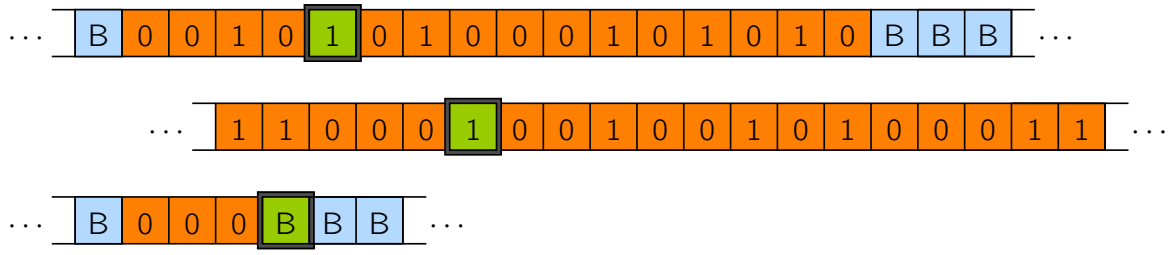
Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M



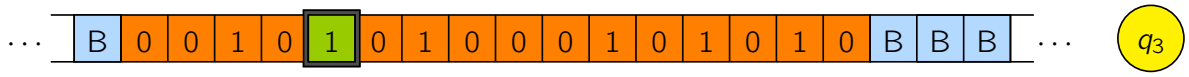
δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

simulierende universelle Maschine U



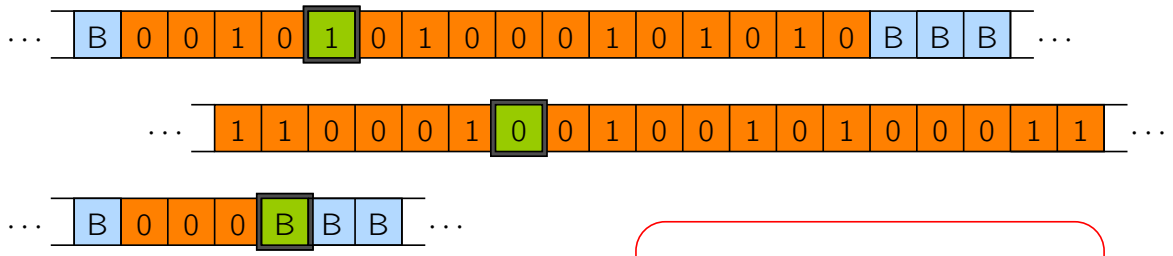
Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M



δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

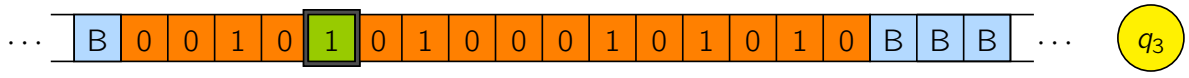
simulierende universelle Maschine U



Vergleiche Band 2 mit Band 1

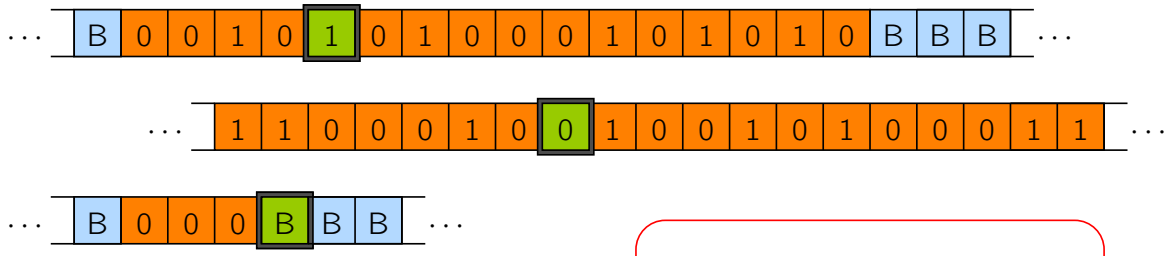
Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M



δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

simulierende universelle Maschine U



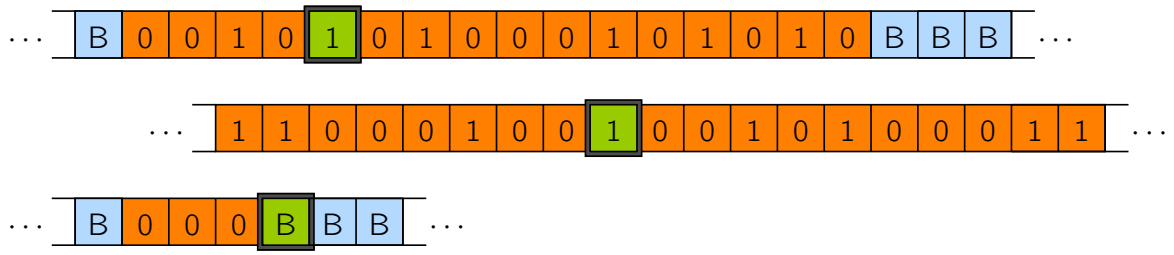
Vergleiche Band 2 mit Band 1

Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M

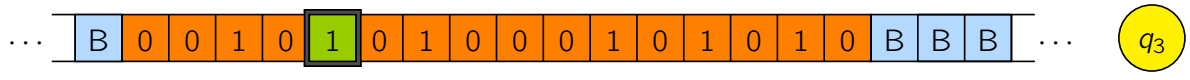
δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

simulierende universelle Maschine U



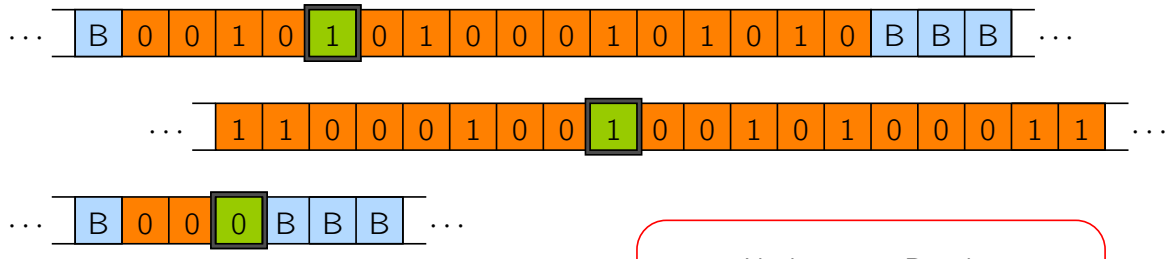
Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M



δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

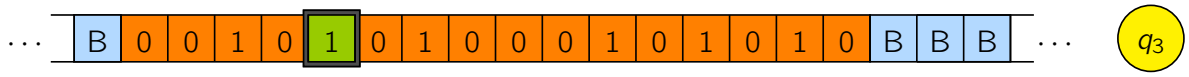
simulierende universelle Maschine U



Update von Band 3

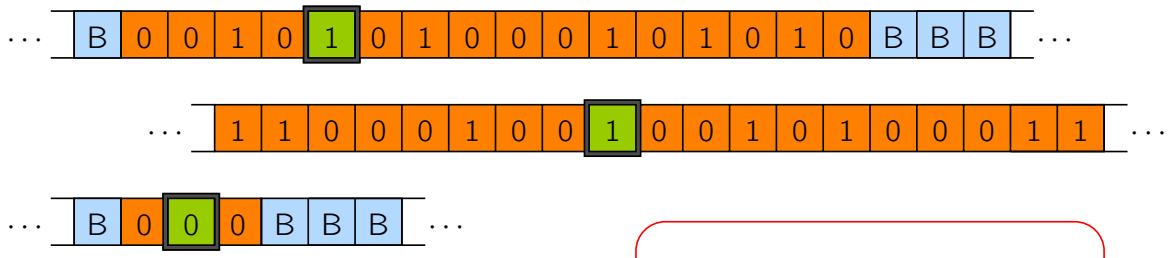
Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M



δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

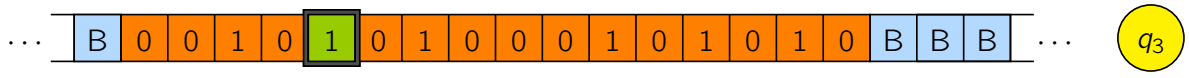
simulierende universelle Maschine U



Update von Band 3

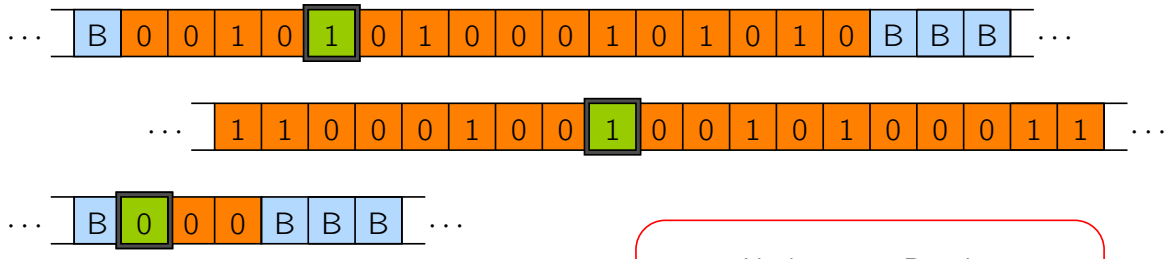
Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M



δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

simulierende universelle Maschine U



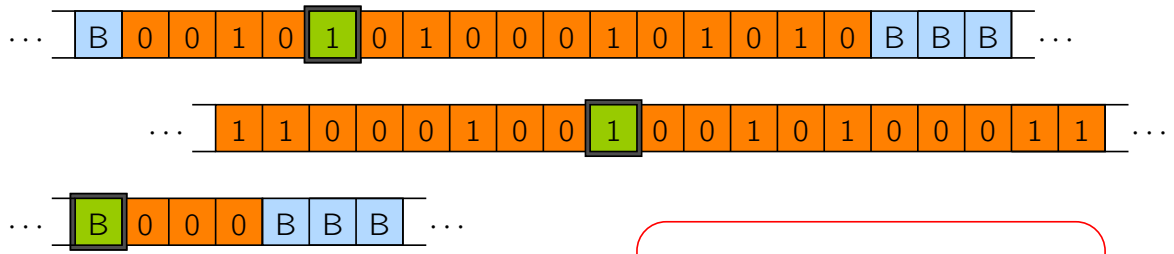
Update von Band 3

Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M

δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

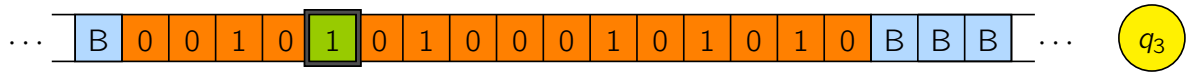
simulierende universelle Maschine U



Update von Band 3

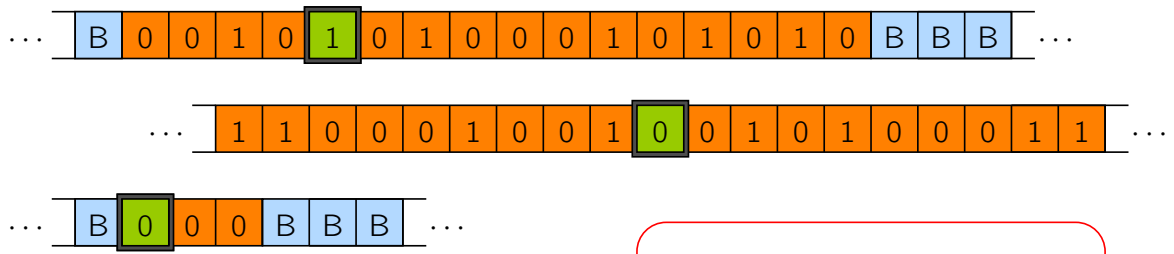
Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M



δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

simulierende universelle Maschine U



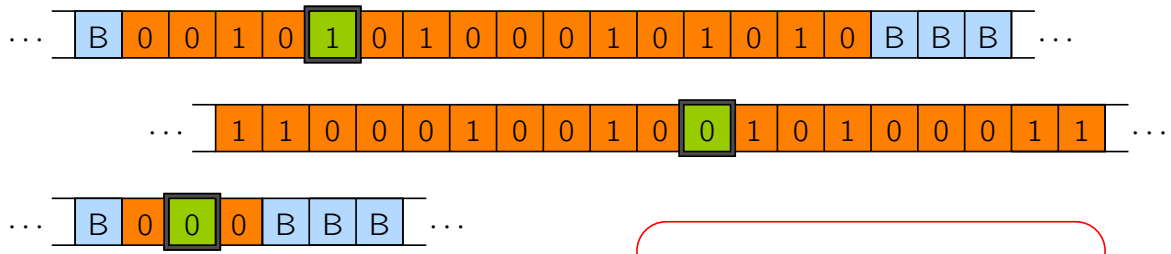
Update von Band 3

Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M

δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

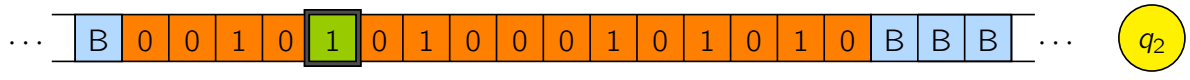
simulierende universelle Maschine U



Update von Band 3

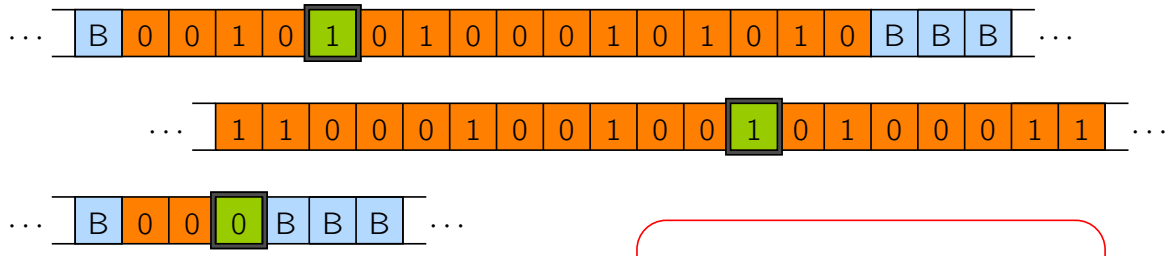
Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M



δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

simulierende universelle Maschine U



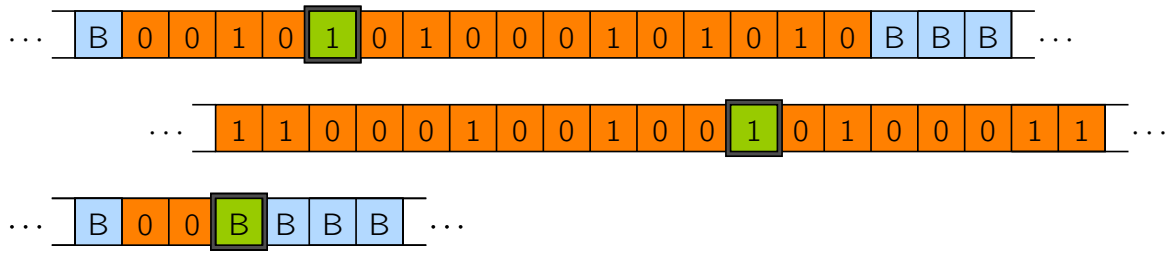
Update von Band 3

Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M

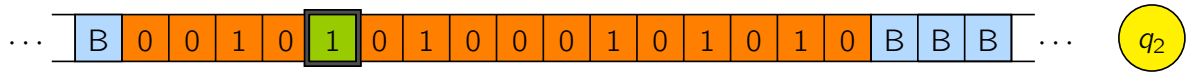
δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

simulierende universelle Maschine U



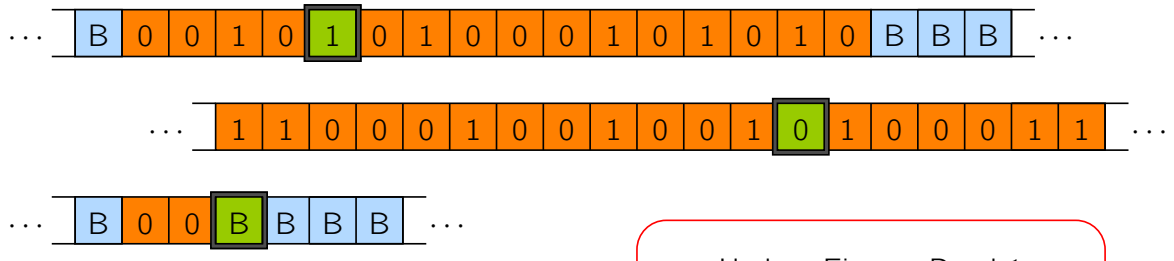
Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M



δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

simulierende universelle Maschine U



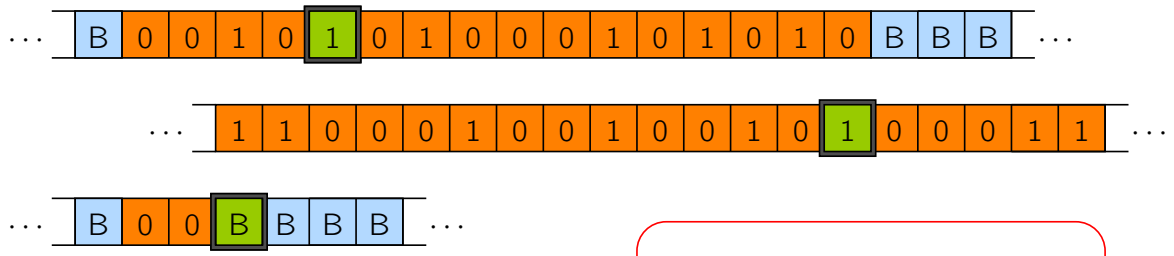
Update Eintrag Band 1

Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M

δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

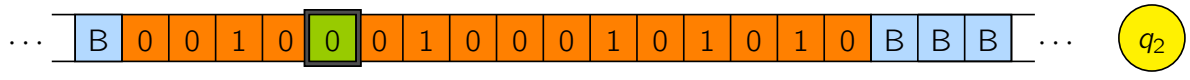
simulierende universelle Maschine U



Update Eintrag Band 1

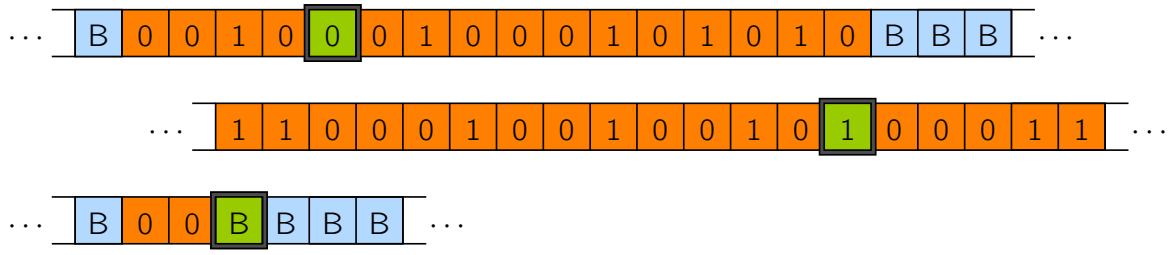
Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M



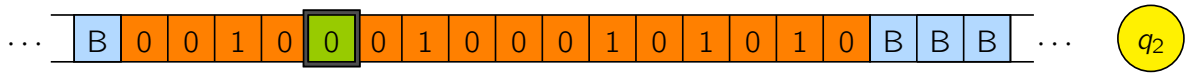
δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

simulierende universelle Maschine U



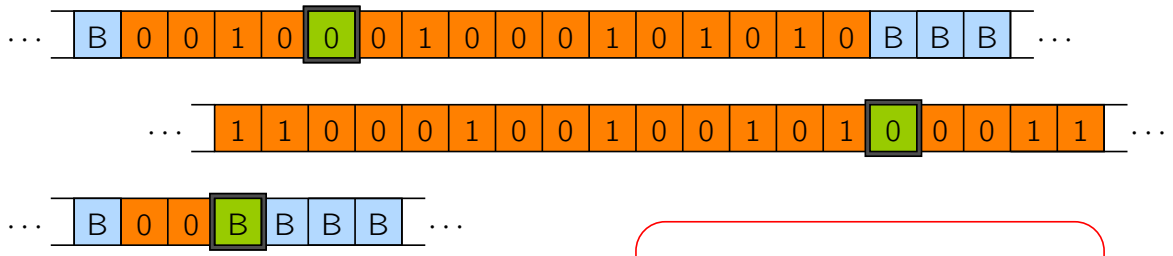
Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M



δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

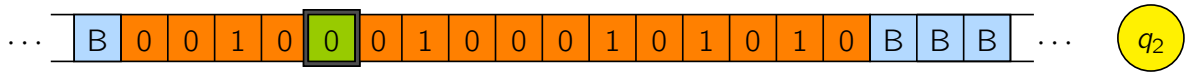
simulierende universelle Maschine U



Update Kopfposition Band 1

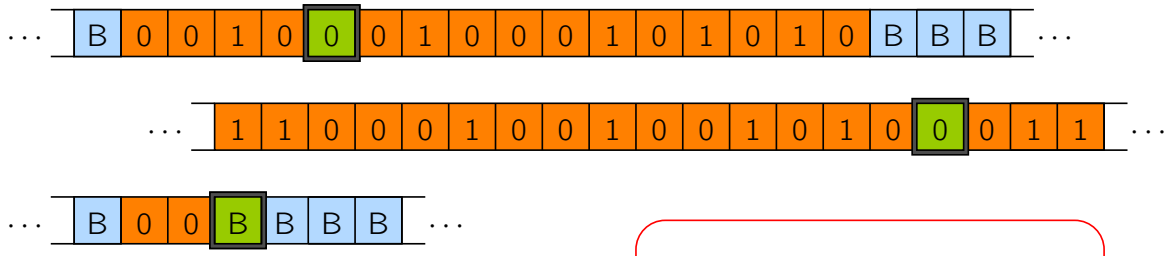
Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M



δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

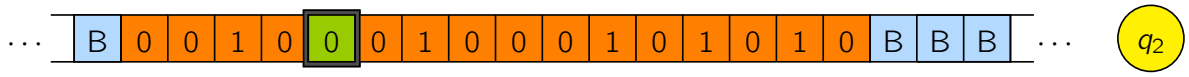
simulierende universelle Maschine U



Update Kopfposition Band 1

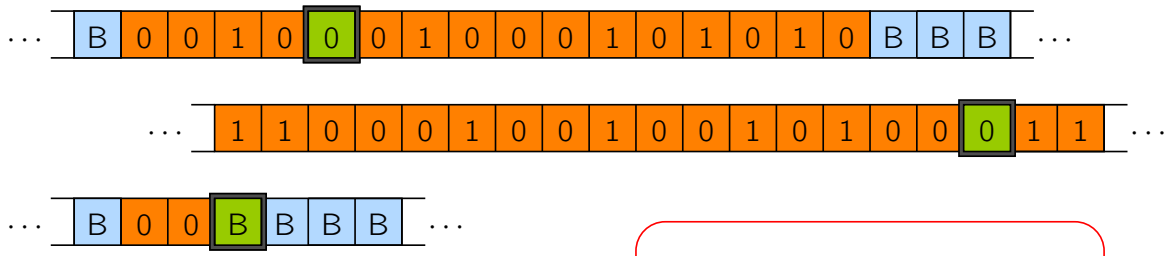
Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M



δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

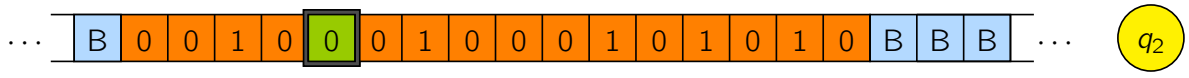
simulierende universelle Maschine U



Update Kopfposition Band 1

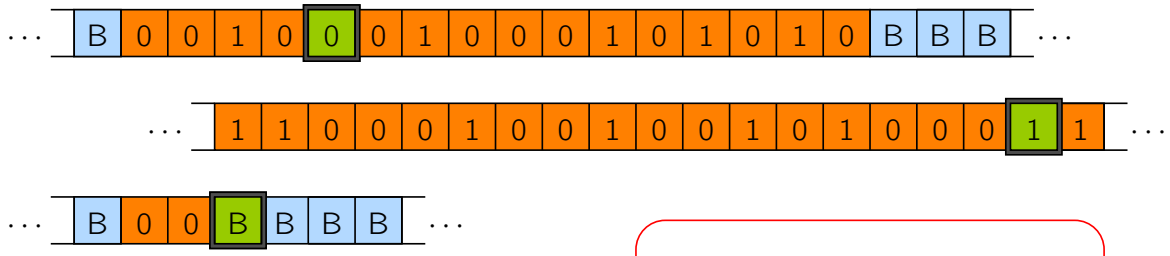
Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M



δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

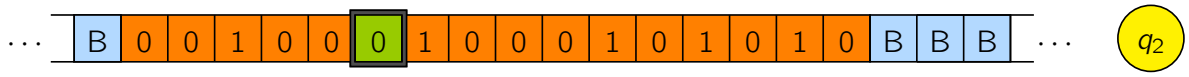
simulierende universelle Maschine U



Update Kopfposition Band 1

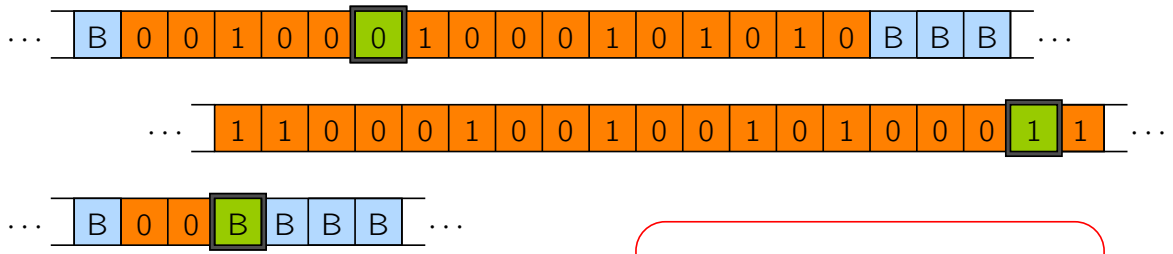
Simulation durch universelle Maschine – Illustration

simulierte Turingmaschine M



δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

simulierende universelle Maschine U



Update Kopfposition Band 1

Implementierung der universellen TM

Können wir dieses Ergebnis auch mit einer (1-Band-)TM erreichen?

Implementierung der universellen TM

Können wir dieses Ergebnis auch mit einer (1-Band-)TM erreichen?

Natürlich können wir die beschriebene 3-Band-TM auf der 1-Band TM mit mehreren Spuren simulieren.

Implementierung der universellen TM

Können wir dieses Ergebnis auch mit einer (1-Band-)TM erreichen?

Natürlich können wir die beschriebene 3-Band-TM auf der 1-Band TM mit mehreren Spuren simulieren.

Aber bei Verwendung dieser Simulation handeln wir uns einen quadratischen Zeitverlust ein.

Implementierung der universellen TM

Können wir dieses Ergebnis auch mit einer (1-Band-)TM erreichen?

Natürlich können wir die beschriebene 3-Band-TM auf der 1-Band TM mit mehreren Spuren simulieren.

Aber bei Verwendung dieser Simulation handeln wir uns einen quadratischen Zeitverlust ein.

Wir erhalten eine universelle 1-Band-TM mit konstantem Zeitverlust, wenn wir ...

Implementierung der universellen TM

Können wir dieses Ergebnis auch mit einer (1-Band-)TM erreichen?

Natürlich können wir die beschriebene 3-Band-TM auf der 1-Band TM mit mehreren Spuren simulieren.

Aber bei Verwendung dieser Simulation handeln wir uns einen quadratischen Zeitverlust ein.

Wir erhalten eine universelle 1-Band-TM mit konstantem Zeitverlust, wenn wir ... die Gödelnummer auf Spur 2 und den Zustand auf Spur 3 mit dem Kopf der TM M mitführen.

Vorlesung 4
Registermaschine (RAM), Church-Turing-These

Wdh.: k-Band- vs 1-Band-TM

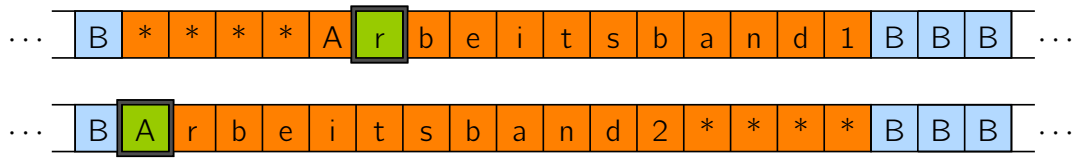
Satz

Eine k-Band-TM M, die mit Rechenzeit $t(n)$ und Platz $s(n)$ auskommt, kann von einer (1-Band-)TM M' mit Zeitbedarf $O(t^2(n))$ und Platzbedarf $O(s(n))$ simuliert werden.

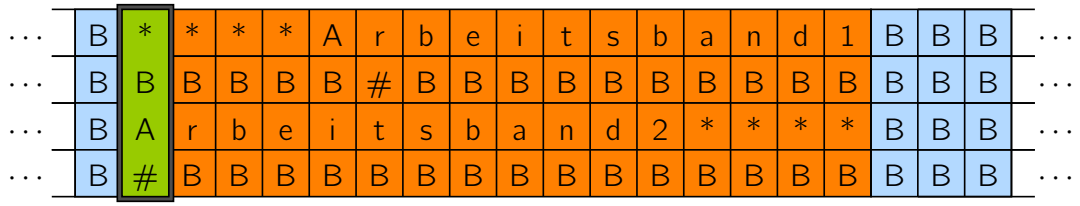
Wdh.: k-Band- vs 1-Band-TM

Satz

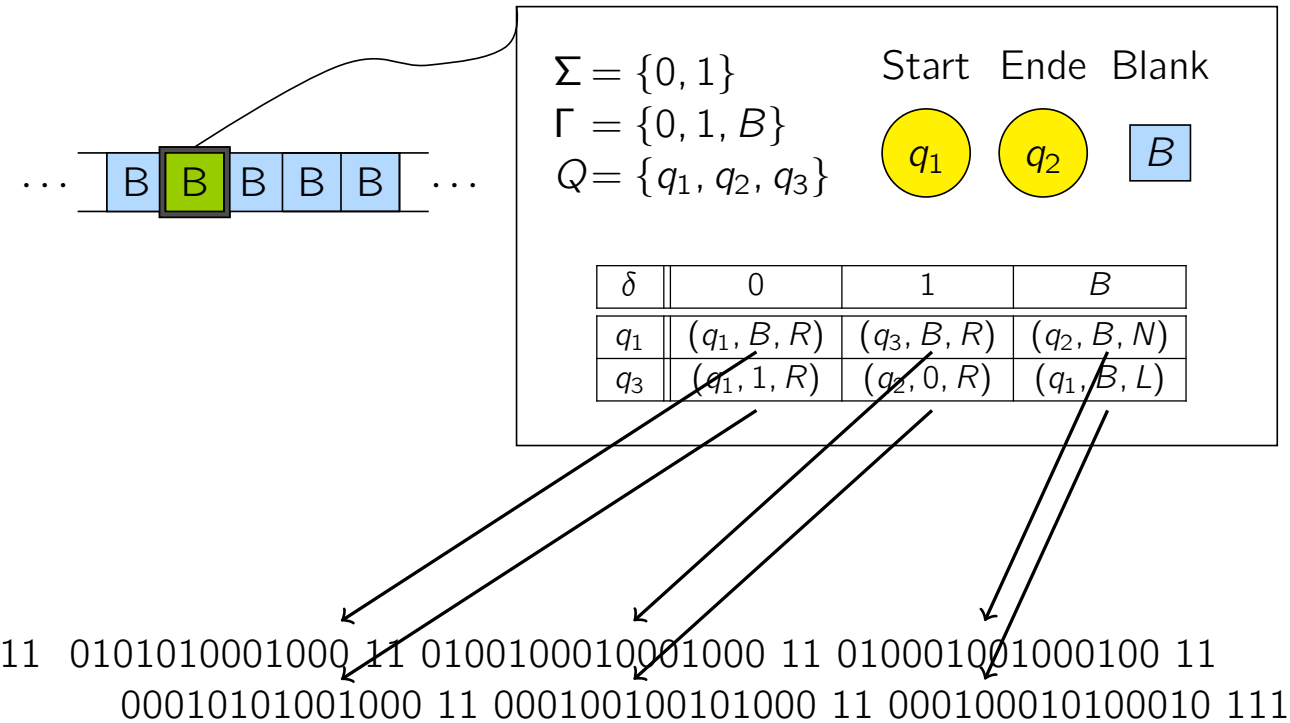
Eine k-Band-TM M, die mit Rechenzeit $t(n)$ und Platz $s(n)$ auskommt, kann von einer (1-Band-)TM M' mit Zeitbedarf $O(t^2(n))$ und Platzbedarf $O(s(n))$ simuliert werden.



Simuliert durch

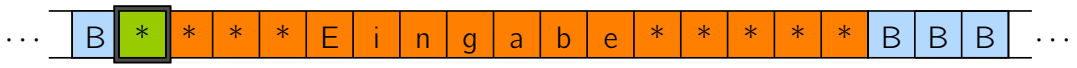


Wdh.: Gödelnummer $\langle M \rangle$

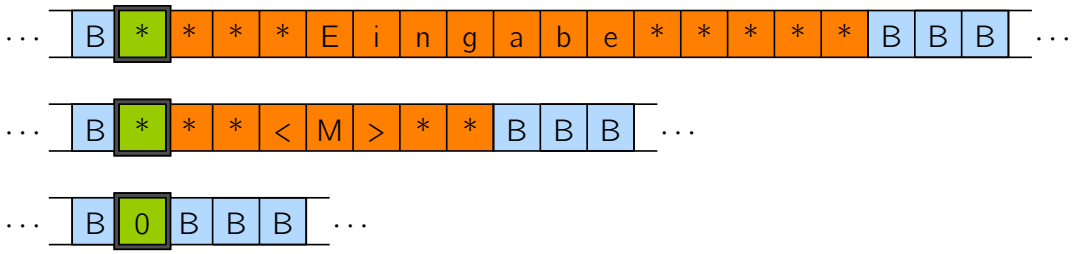


Wdh.: Universelle TM

simulierte Turingmaschine M



Initialisierung der universellen Maschine U



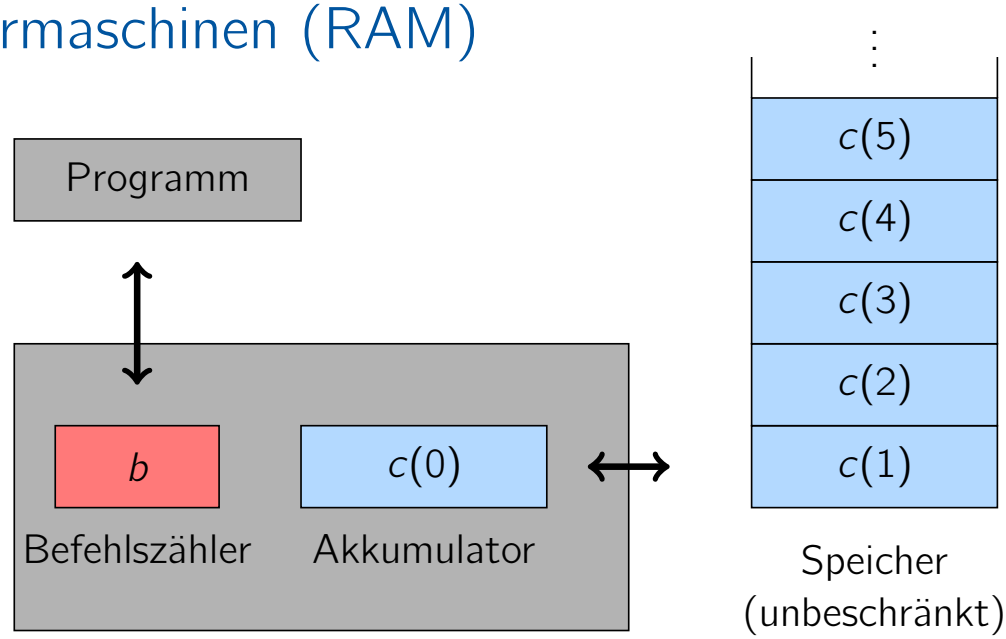
Wdh.: Universelle TM

Laufzeit der universellen TM

- Bei Eingabe $\langle M \rangle w$ simuliert U die TM M auf Wort w.
- Jeder Schritt von M wird dabei von U in $f(|\langle M \rangle|)$ Zeit simuliert.
- Wenn $|\langle M \rangle|$ als Konstante angesehen wird, so simuliert U die TM M mit einem konstanten Zeit- und Platzverlust.

Registermaschinen (RAM)

Registermaschinen (RAM)



Befehlssatz:
LOAD, STORE, ADD, SUB, MULT, DIV
INDLOAD, INDSTORE, INDADD, INDSUB, INDMULT, INDDIV
CLOAD, CADD, CSUB, CMULT, CDIV
GOTO, IF c(0)?x THEN GOTO j (wobei ? aus {=,<,<=,>,>=} ist),
END

Erläuterung einiger ausgewählter RAM-Befehle

LOAD i: $c(0) := c(i)$, $b := b + 1$;

Erläuterung einiger ausgewählter RAM-Befehle

LOAD i: $c(0) := c(i)$, $b := b + 1$;

INDLOAD i: $c(0) := c(c(i))$, $b := b + 1$;

Vorlesung BuK im WS 22/23, M. Grohe **Seite 94** Version 12. Oktober 2022

Erläuterung einiger ausgewählter RAM-Befehle

LOAD i:	$c(0) := c(i)$,	$b := b + 1$;
INDLOAD i:	$c(0) := c(c(i))$,	$b := b + 1$;
CLOAD i:	$c(0) := i$,	$b := b + 1$;

Erläuterung einiger ausgewählter RAM-Befehle

LOAD i:	$c(0) := c(i)$,	$b := b + 1$;
INDLOAD i:	$c(0) := c(c(i))$,	$b := b + 1$;
CLOAD i:	$c(0) := i$,	$b := b + 1$;
STORE i:	$c(i) := c(0)$,	$b := b + 1$;

Erläuterung einiger ausgewählter RAM-Befehle

LOAD i:	$c(0) := c(i)$,	$b := b + 1$;
INDLOAD i:	$c(0) := c(c(i))$,	$b := b + 1$;
CLOAD i:	$c(0) := i$,	$b := b + 1$;
STORE i:	$c(i) := c(0)$,	$b := b + 1$;
INDSTORE i:	$c(c(i)) := c(0)$,	$b := b + 1$;

Erläuterung einiger ausgewählter RAM-Befehle

LOAD i:	$c(0) := c(i)$,	$b := b + 1$;
INDLOAD i:	$c(0) := c(c(i))$,	$b := b + 1$;
CLOAD i:	$c(0) := i$,	$b := b + 1$;
STORE i:	$c(i) := c(0)$,	$b := b + 1$;
INDSTORE i:	$c(c(i)) := c(0)$,	$b := b + 1$;
ADD i:	$c(0) := c(0) + c(i)$,	$b := b + 1$;

Vorlesung BuK im WS 22/23, M. Grohe Seite 94 Version 12. Oktober 2022

Erläuterung einiger ausgewählter RAM-Befehle

LOAD i:	$c(0) := c(i)$,	$b := b + 1$;
INDLOAD i:	$c(0) := c(c(i))$,	$b := b + 1$;
CLOAD i:	$c(0) := i$,	$b := b + 1$;
STORE i:	$c(i) := c(0)$,	$b := b + 1$;
INDSTORE i:	$c(c(i)) := c(0)$,	$b := b + 1$;
ADD i:	$c(0) := c(0) + c(i)$,	$b := b + 1$;
CADD i:	$c(0) := c(0) + i$,	$b := b + 1$;

Erläuterung einiger ausgewählter RAM-Befehle

LOAD i:	$c(0) := c(i)$,	$b := b + 1$;
INDLOAD i:	$c(0) := c(c(i))$,	$b := b + 1$;
CLOAD i:	$c(0) := i$,	$b := b + 1$;
STORE i:	$c(i) := c(0)$,	$b := b + 1$;
INDSTORE i:	$c(c(i)) := c(0)$,	$b := b + 1$;
ADD i:	$c(0) := c(0) + c(i)$,	$b := b + 1$;
CADD i:	$c(0) := c(0) + i$,	$b := b + 1$;
INDADD i:	$c(0) := c(0) + c(c(i))$,	$b := b + 1$;

Erläuterung einiger ausgewählter RAM-Befehle

LOAD i:	$c(0) := c(i)$,	$b := b + 1$;
INDLOAD i:	$c(0) := c(c(i))$,	$b := b + 1$;
CLOAD i:	$c(0) := i$,	$b := b + 1$;
STORE i:	$c(i) := c(0)$,	$b := b + 1$;
INDSTORE i:	$c(c(i)) := c(0)$,	$b := b + 1$;
ADD i:	$c(0) := c(0) + c(i)$,	$b := b + 1$;
CADD i:	$c(0) := c(0) + i$,	$b := b + 1$;
INDADD i:	$c(0) := c(0) + c(c(i))$,	$b := b + 1$;
	$\vdots$	

Erläuterung einiger ausgewählter RAM-Befehle

LOAD i:	$c(0) := c(i)$,	$b := b + 1$;
INDLOAD i:	$c(0) := c(c(i))$,	$b := b + 1$;
CLOAD i:	$c(0) := i$,	$b := b + 1$;
STORE i:	$c(i) := c(0)$,	$b := b + 1$;
INDSTORE i:	$c(c(i)) := c(0)$,	$b := b + 1$;
ADD i:	$c(0) := c(0) + c(i)$,	$b := b + 1$;
CADD i:	$c(0) := c(0) + i$,	$b := b + 1$;
INDADD i:	$c(0) := c(0) + c(c(i))$,	$b := b + 1$;
	$\vdots$	
SUB i:	$c(0) := \max\{0, c(0) - c(i)\}$	$b := b + 1$;

Vorlesung BuK im WS 22/23, M. Grohe **Seite 94** Version 12. Oktober 2022

Erläuterung einiger ausgewählter RAM-Befehle

LOAD i:	$c(0) := c(i)$,	$b := b+1$;
INDLOAD i:	$c(0) := c(c(i))$,	$b := b+1$;
CLOAD i:	$c(0) := i$,	$b := b+1$;
STORE i:	$c(i) := c(0)$,	$b := b+1$;
INDSTORE i:	$c(c(i)) := c(0)$,	$b := b+1$;
ADD i:	$c(0) := c(0) + c(i)$,	$b := b+1$;
CADD i:	$c(0) := c(0) + i$,	$b := b+1$;
INDADD i:	$c(0) := c(0) + c(c(i))$,	$b := b+1$;
	$\vdots$	
SUB i:	$c(0) := \max\{0, c(0) - c(i)\}$	$b := b+1$;
	$\vdots$	

Erläuterung einiger ausgewählter RAM-Befehle

LOAD i:	$c(0) := c(i)$,	$b := b+1$;
INDLOAD i:	$c(0) := c(c(i))$,	$b := b+1$;
CLOAD i:	$c(0) := i$,	$b := b+1$;
STORE i:	$c(i) := c(0)$,	$b := b+1$;
INDSTORE i:	$c(c(i)) := c(0)$,	$b := b+1$;
ADD i:	$c(0) := c(0) + c(i)$,	$b := b+1$;
CADD i:	$c(0) := c(0) + i$,	$b := b+1$;
INDADD i:	$c(0) := c(0) + c(c(i))$,	$b := b+1$;
	$\vdots$	
SUB i:	$c(0) := \max\{0, c(0) - c(i)\}$	$b := b+1$;
	$\vdots$	
DIV i:	$c(0) := \begin{cases} \lfloor c(0)/c(i) \rfloor & \text{falls } c(i) \neq 0, \\ 0 & \text{sonst} \end{cases}$,	$b := b+1$;
	$\vdots$	

Erläuterung einiger ausgewählter RAM-Befehle

LOAD i:	$c(0) := c(i)$,	$b := b+1$;
INDLOAD i:	$c(0) := c(c(i))$,	$b := b+1$;
CLOAD i:	$c(0) := i$,	$b := b+1$;
STORE i:	$c(i) := c(0)$,	$b := b+1$;
INDSTORE i:	$c(c(i)) := c(0)$,	$b := b+1$;
ADD i:	$c(0) := c(0) + c(i)$,	$b := b+1$;
CADD i:	$c(0) := c(0) + i$,	$b := b+1$;
INDADD i:	$c(0) := c(0) + c(c(i))$,	$b := b+1$;
	$\vdots$	
SUB i:	$c(0) := \max\{0, c(0) - c(i)\}$	$b := b+1$;
	$\vdots$	
DIV i:	$c(0) := \begin{cases} \lfloor c(0)/c(i) \rfloor & \text{falls } c(i) \neq 0, \\ 0 & \text{sonst} \end{cases}$,	$b := b+1$;
	$\vdots$	
GOTO j:		$b := j$

Erläuterung einiger ausgewählter RAM-Befehle

LOAD i:	$c(0) := c(i)$,	$b := b+1$;
INDLOAD i:	$c(0) := c(c(i))$,	$b := b+1$;
CLOAD i:	$c(0) := i$,	$b := b+1$;
STORE i:	$c(i) := c(0)$,	$b := b+1$;
INDSTORE i:	$c(c(i)) := c(0)$,	$b := b+1$;
ADD i:	$c(0) := c(0) + c(i)$,	$b := b+1$;
CADD i:	$c(0) := c(0) + i$,	$b := b+1$;
INDADD i:	$c(0) := c(0) + c(c(i))$,	$b := b+1$;
	$\vdots$	
SUB i:	$c(0) := \max\{0, c(0) - c(i)\}$	$b := b+1$;
	$\vdots$	
DIV i:	$c(0) := \begin{cases} \lfloor c(0)/c(i) \rfloor & \text{falls } c(i) \neq 0, \\ 0 & \text{sonst} \end{cases}$,	$b := b+1$;
	$\vdots$	
GOTO j:		$b := j$
IF $c(0) = x$ GOTO j:	$b := j$ falls $c(0) = x$, sonst $b := b+1$;	

Erläuterung einiger ausgewählter RAM-Befehle

LOAD i:	$c(0) := c(i)$,	$b := b + 1$;
INDLOAD i:	$c(0) := c(c(i))$,	$b := b + 1$;
CLOAD i:	$c(0) := i$,	$b := b + 1$;
STORE i:	$c(i) := c(0)$,	$b := b + 1$;
INDSTORE i:	$c(c(i)) := c(0)$,	$b := b + 1$;
ADD i:	$c(0) := c(0) + c(i)$,	$b := b + 1$;
CADD i:	$c(0) := c(0) + i$,	$b := b + 1$;
INDADD i:	$c(0) := c(0) + c(c(i))$,	$b := b + 1$;
	$\vdots$	
SUB i:	$c(0) := \max\{0, c(0) - c(i)\}$	$b := b + 1$;
	$\vdots$	
DIV i:	$c(0) := \begin{cases} \lfloor c(0)/c(i) \rfloor & \text{falls } c(i) \neq 0, \\ 0 & \text{sonst} \end{cases}$,	$b := b + 1$;
	$\vdots$	
GOTO j:		$b := j$

IF $c(0) = x$ GOTO j: $b := j$ falls $c(0) = x$, sonst $b := b + 1$;

END.

Funktionsweise der RAM

- Der Speicher der RAM ist unbeschränkt und besteht aus dem Akkumulator $c(0)$ und den Registern $c(1), c(2), c(3), \ldots$
- Die Inhalte der Register sind natürliche Zahlen, die beliebig groß sein können.

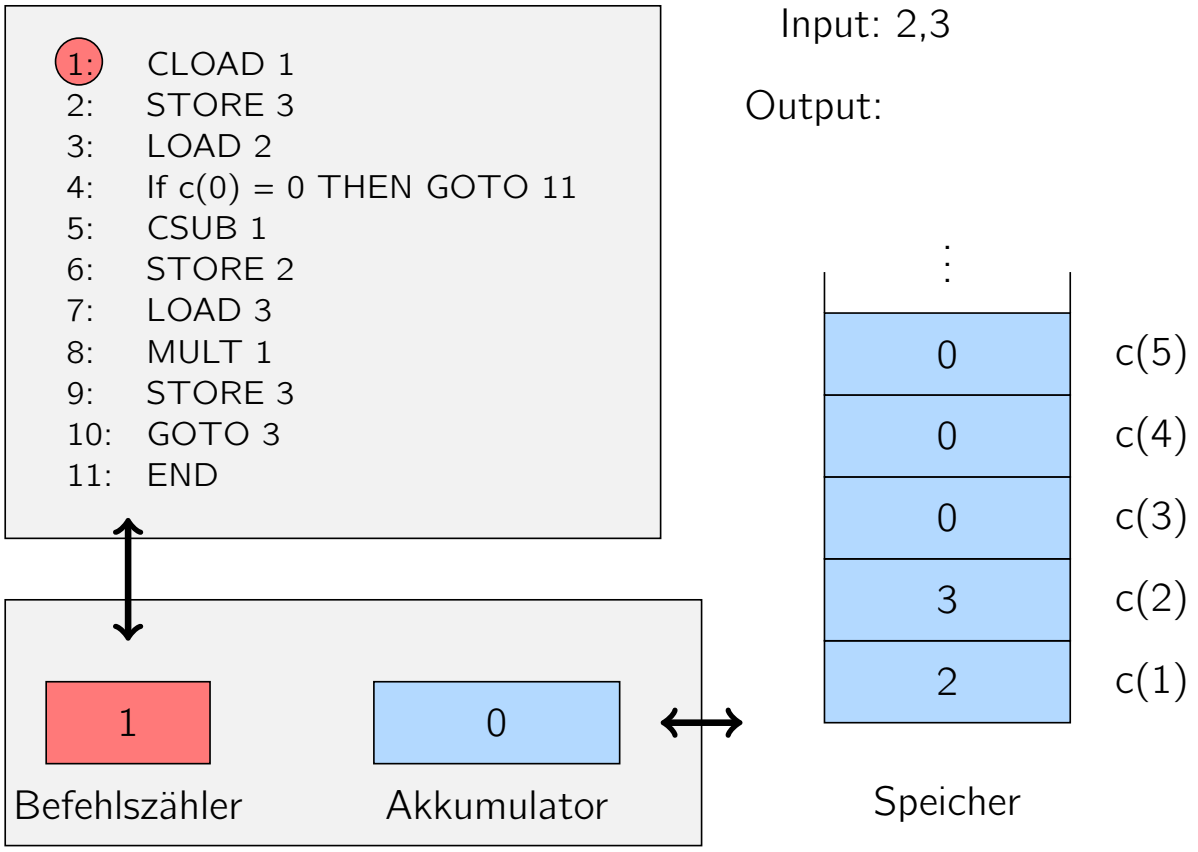
Funktionsweise der RAM

- Der Speicher der RAM ist unbeschränkt und besteht aus dem Akkumulator $c(0)$ und den Registern $c(1), c(2), c(3), \ldots$
- Die Inhalte der Register sind natürliche Zahlen, die beliebig groß sein können.
- Die Eingabe besteht ebenfalls aus natürlichen Zahlen, die initial in den ersten Registern abgespeichert sind.
- Der Befehlszähler startet mit dem Wert 1. Ausgeführt wird jeweils der Befehl in derjenigen Zeile, auf die der Befehlszähler verweist.

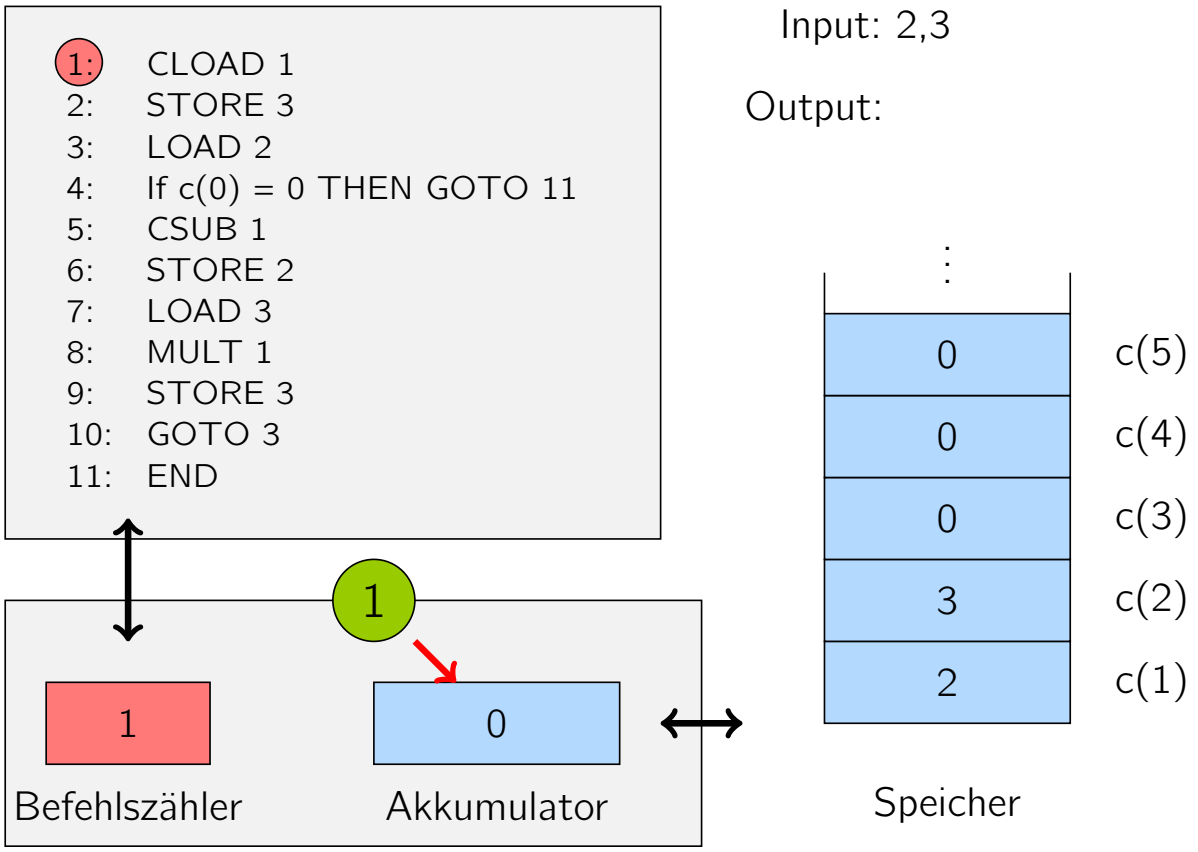
Funktionsweise der RAM

- Der Speicher der RAM ist unbeschränkt und besteht aus dem Akkumulator $c(0)$ und den Registern $c(1), c(2), c(3), \ldots$
- Die Inhalte der Register sind natürliche Zahlen, die beliebig groß sein können.
- Die Eingabe besteht ebenfalls aus natürlichen Zahlen, die initial in den ersten Registern abgespeichert sind.
- Der Befehlszähler startet mit dem Wert 1. Ausgeführt wird jeweils der Befehl in derjenigen Zeile, auf die der Befehlszähler verweist.
- Die Rechnung stoppt, sobald der Befehl END erreicht ist.
- Die Ausgabe befindet sich nach dem Stoppen in den ersten Registern.

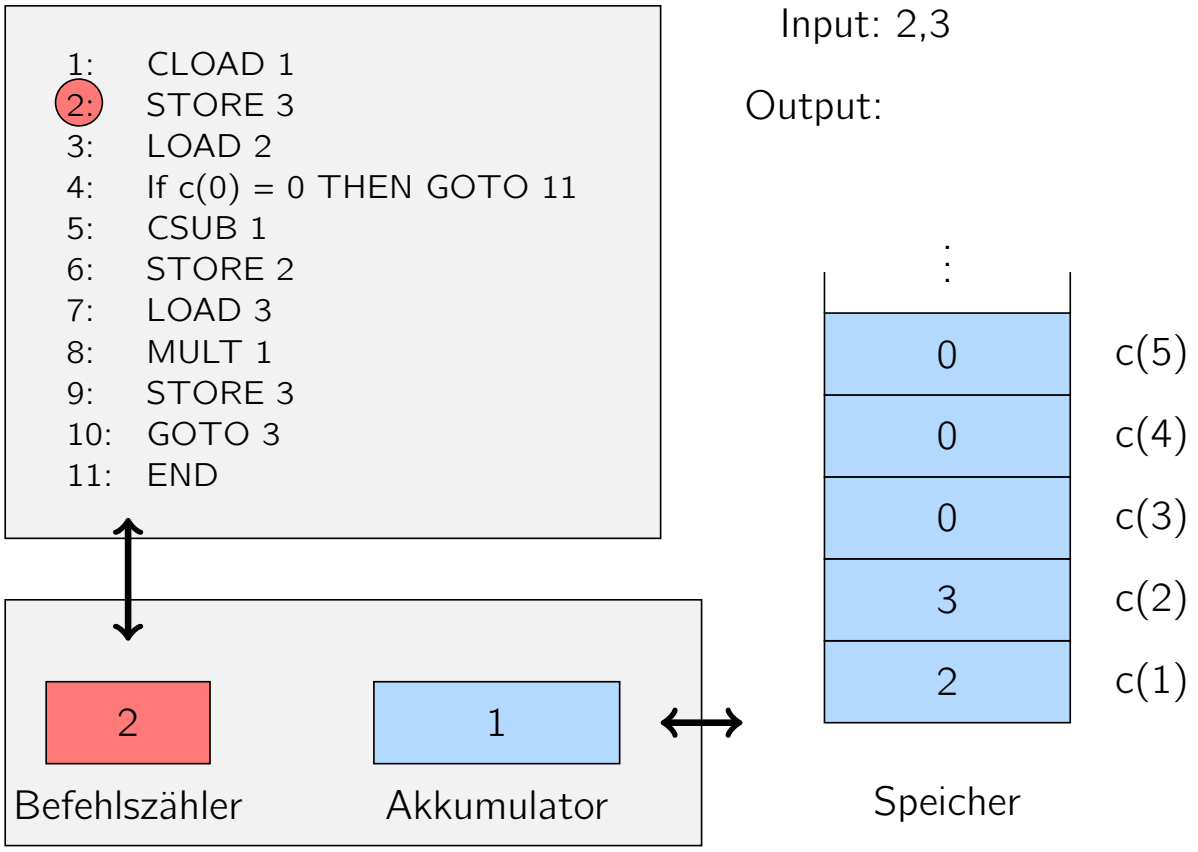
Beispielprogramm für die RAM



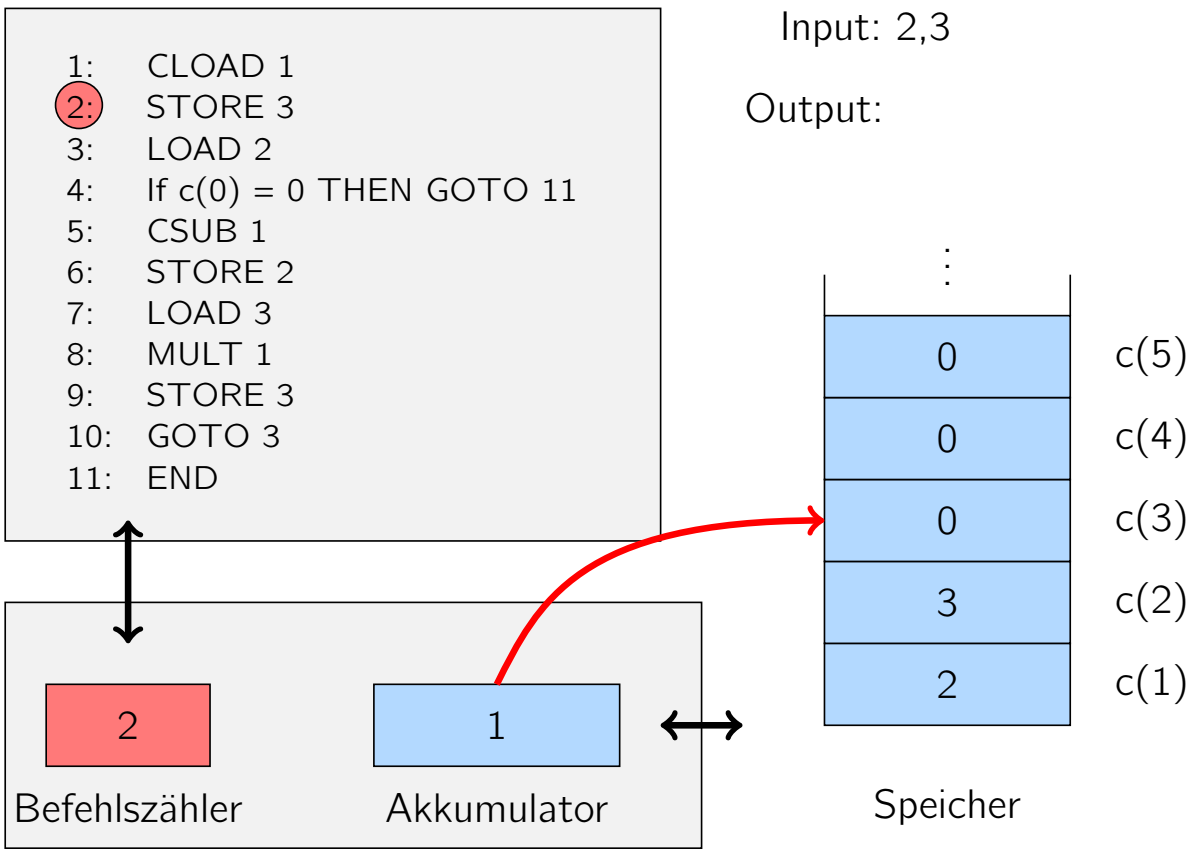
Beispielprogramm für die RAM



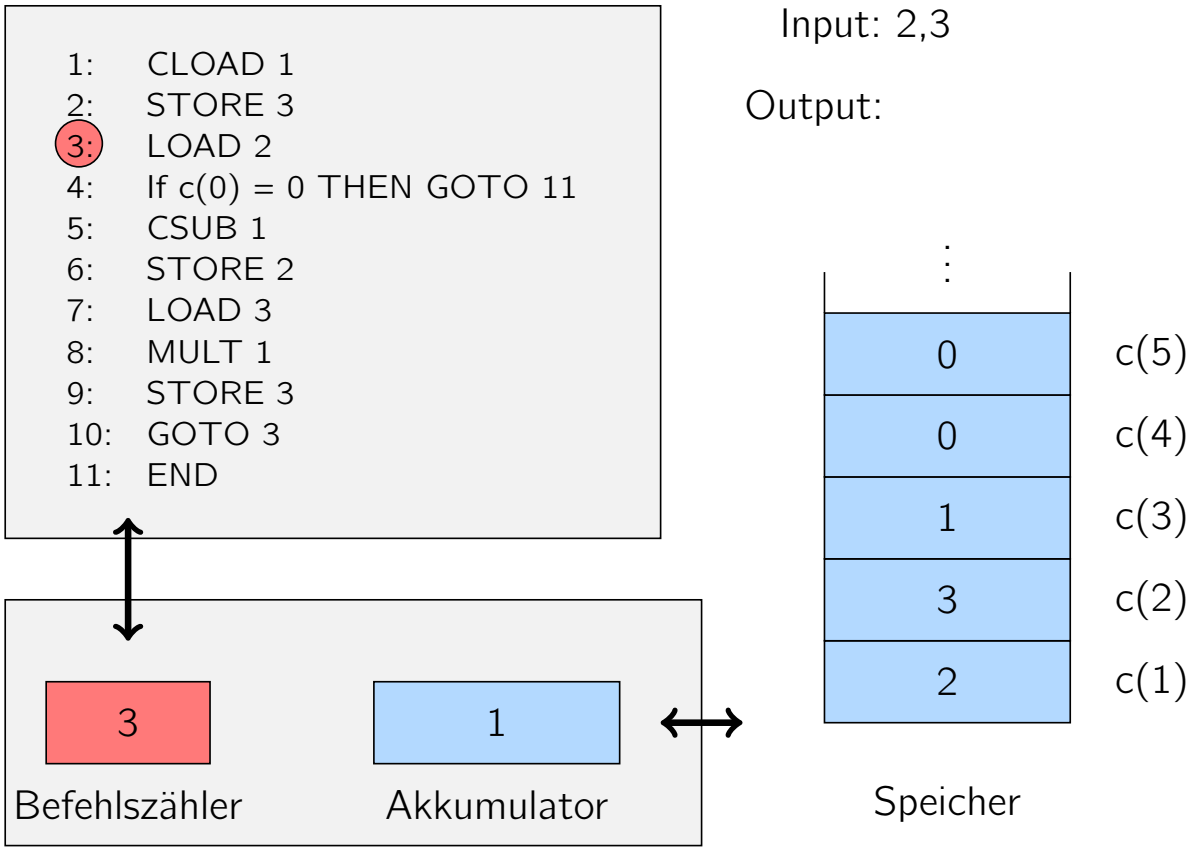
Beispielprogramm für die RAM



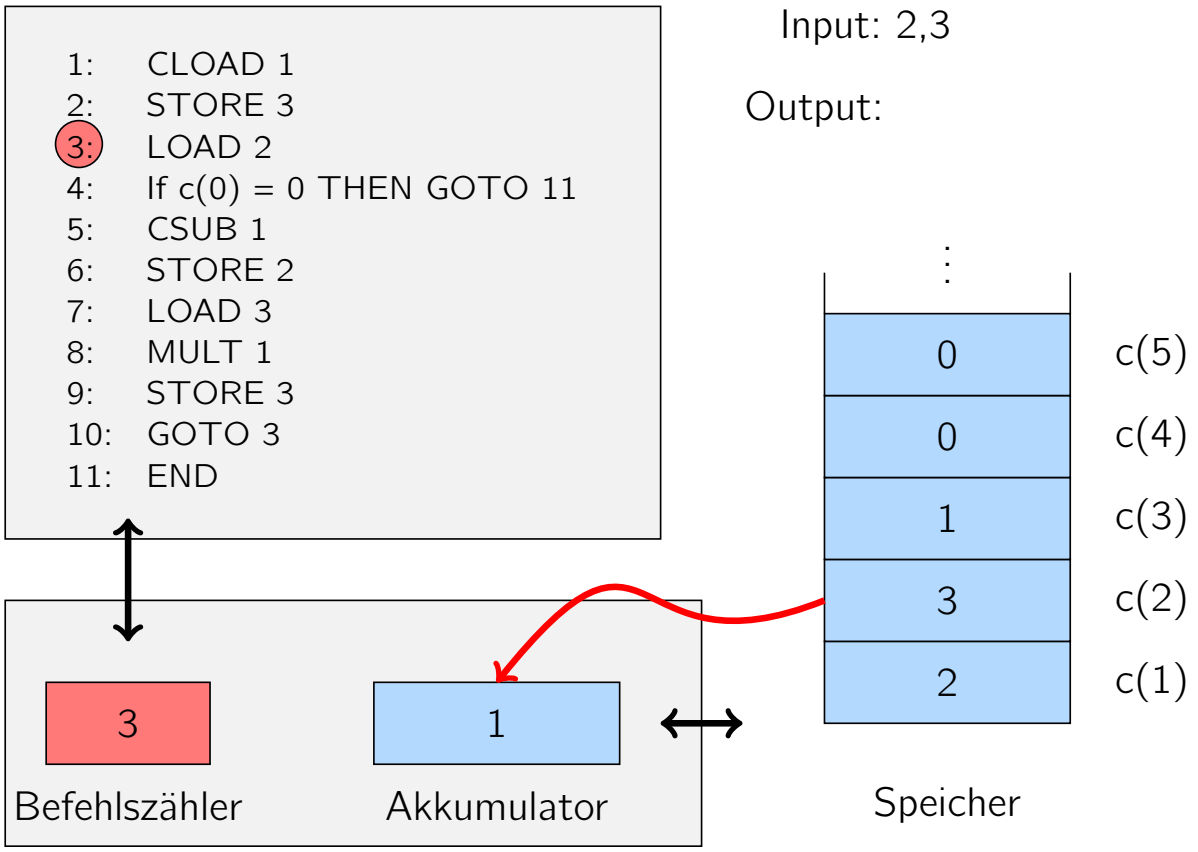
Beispielprogramm für die RAM



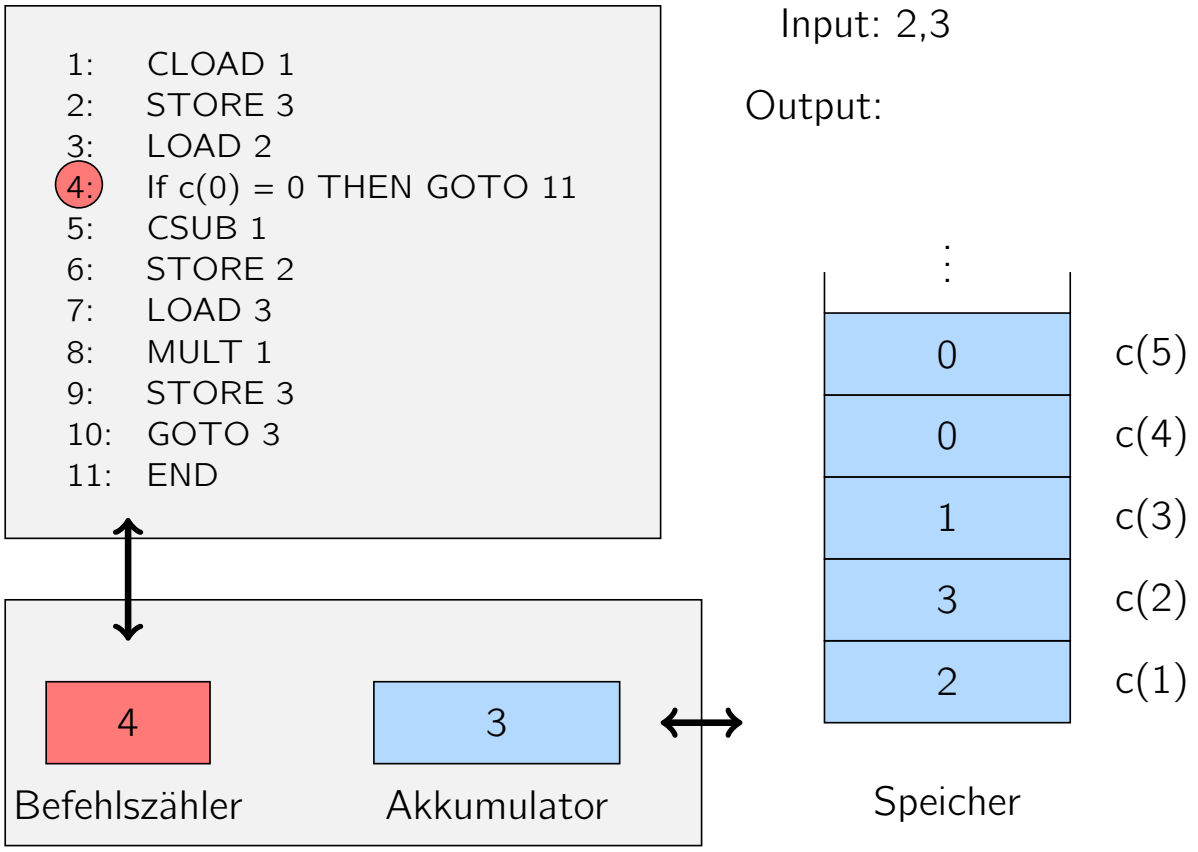
Beispielprogramm für die RAM



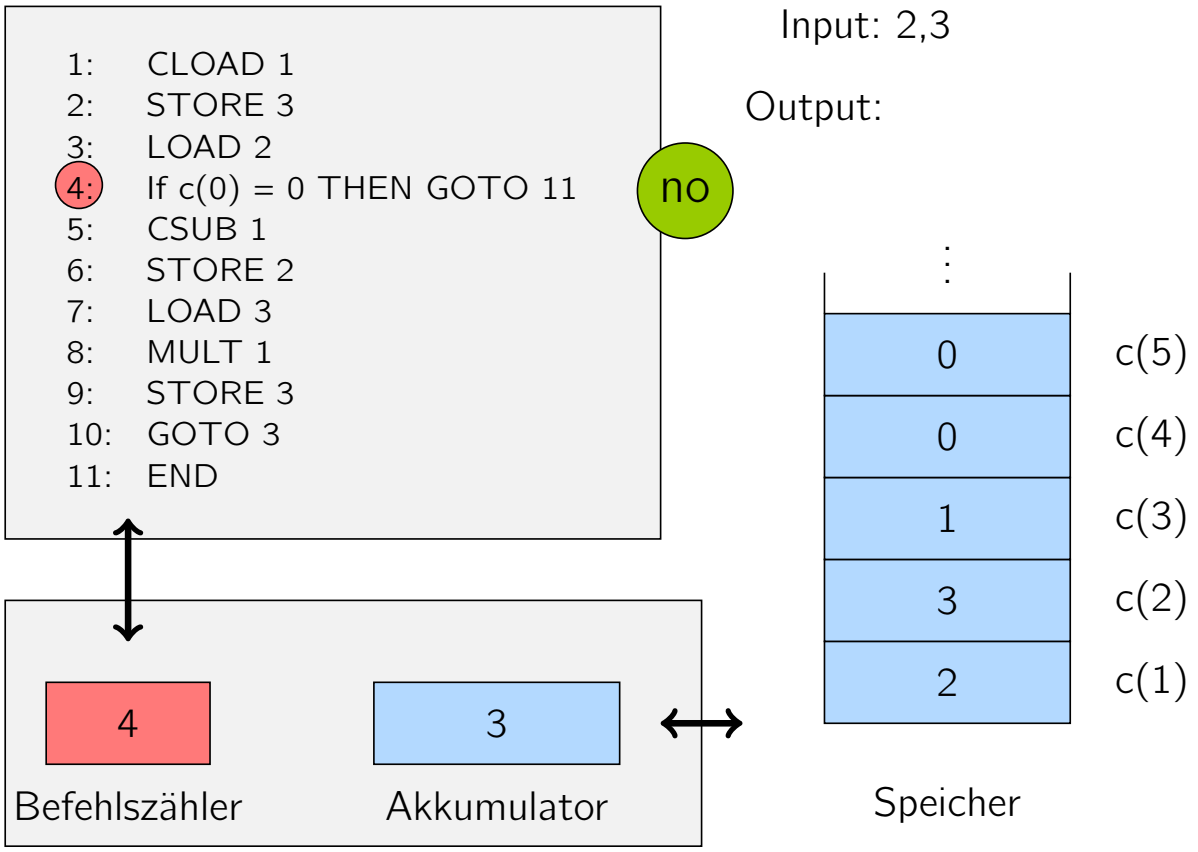
Beispielprogramm für die RAM



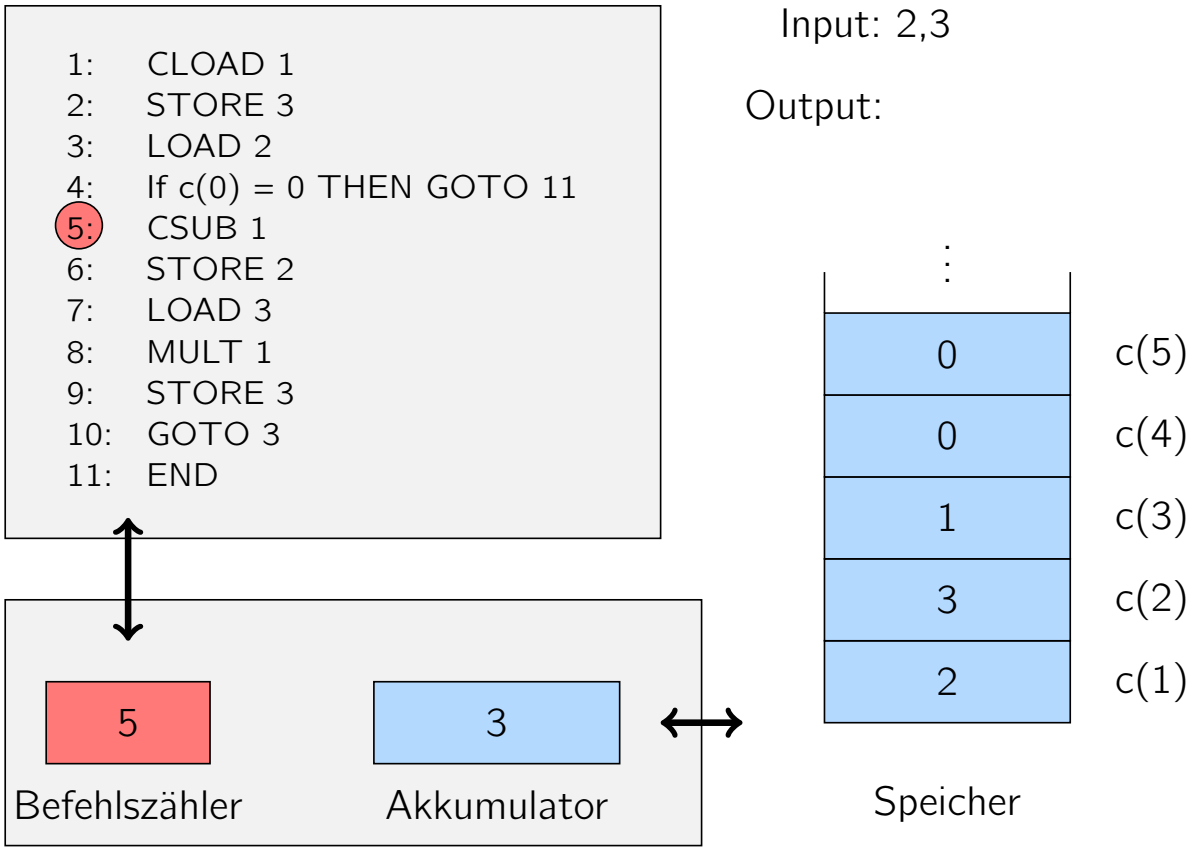
Beispielprogramm für die RAM



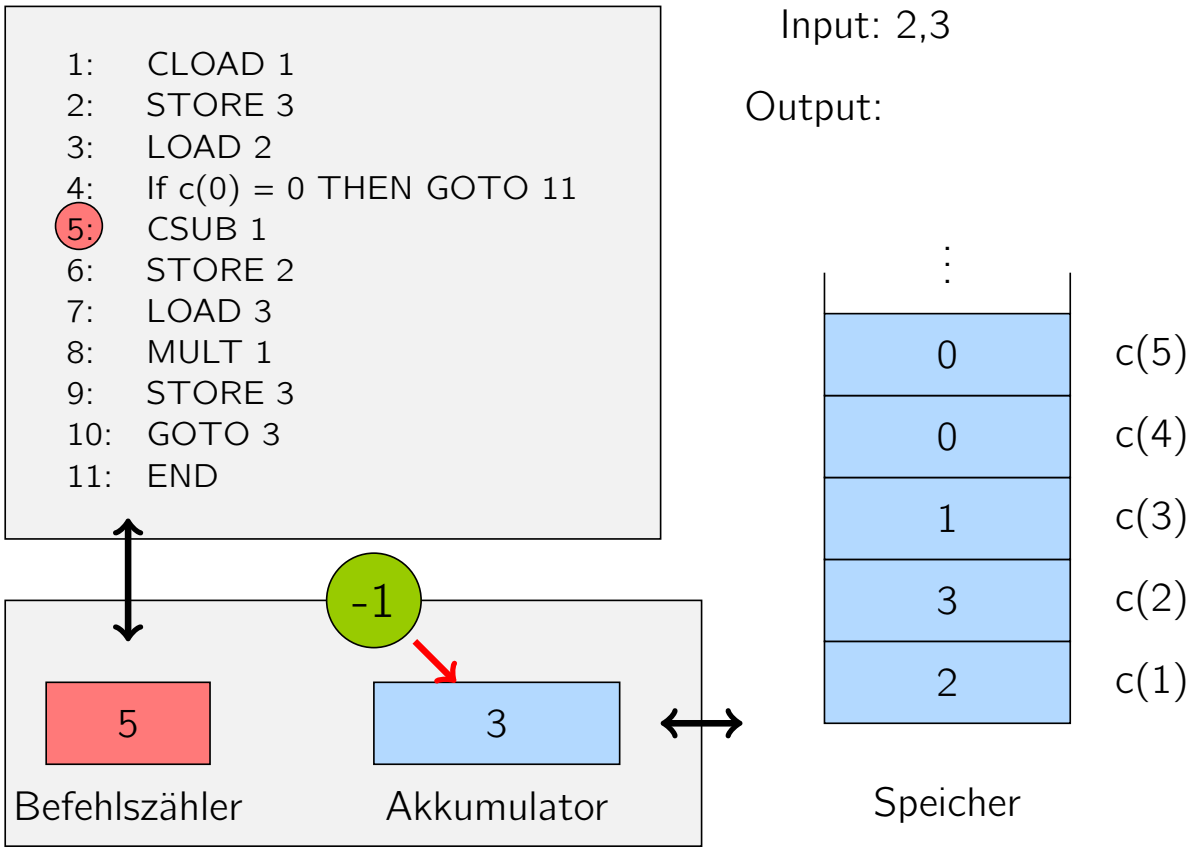
Beispielprogramm für die RAM



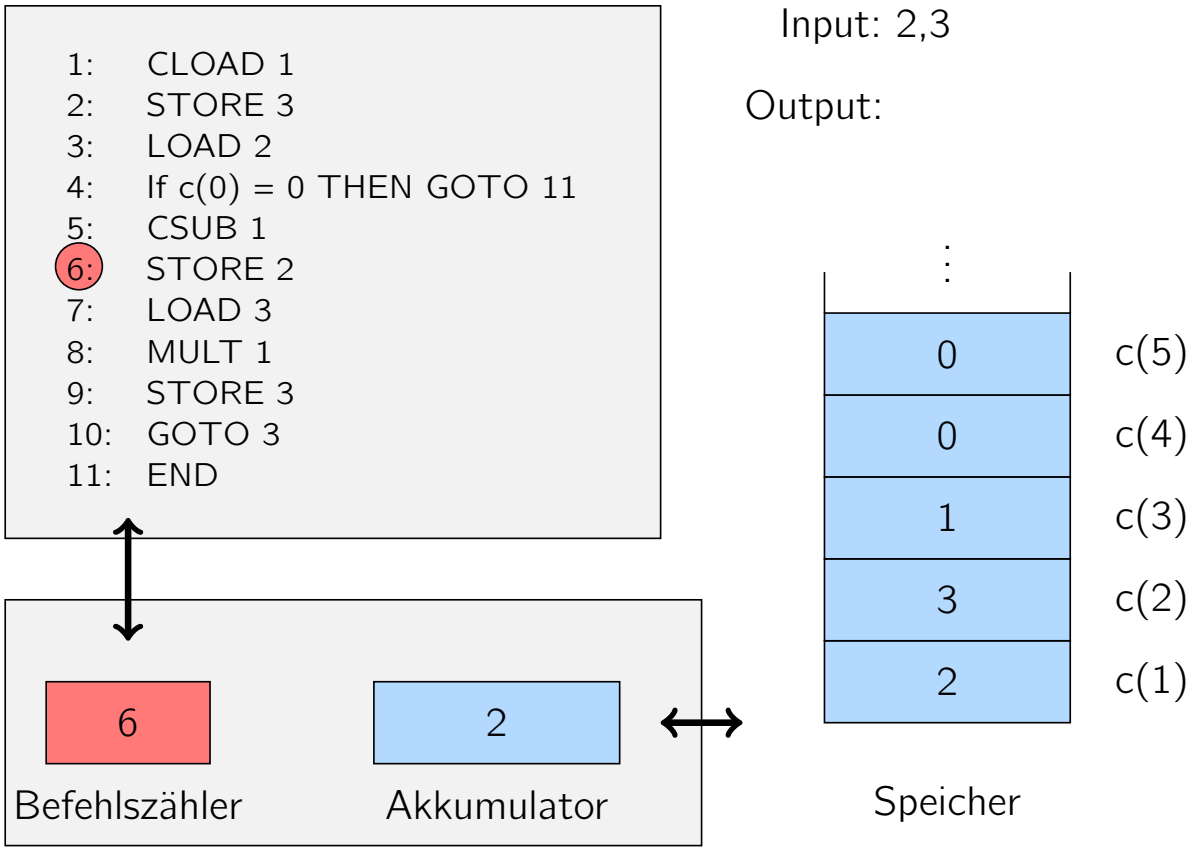
Beispielprogramm für die RAM



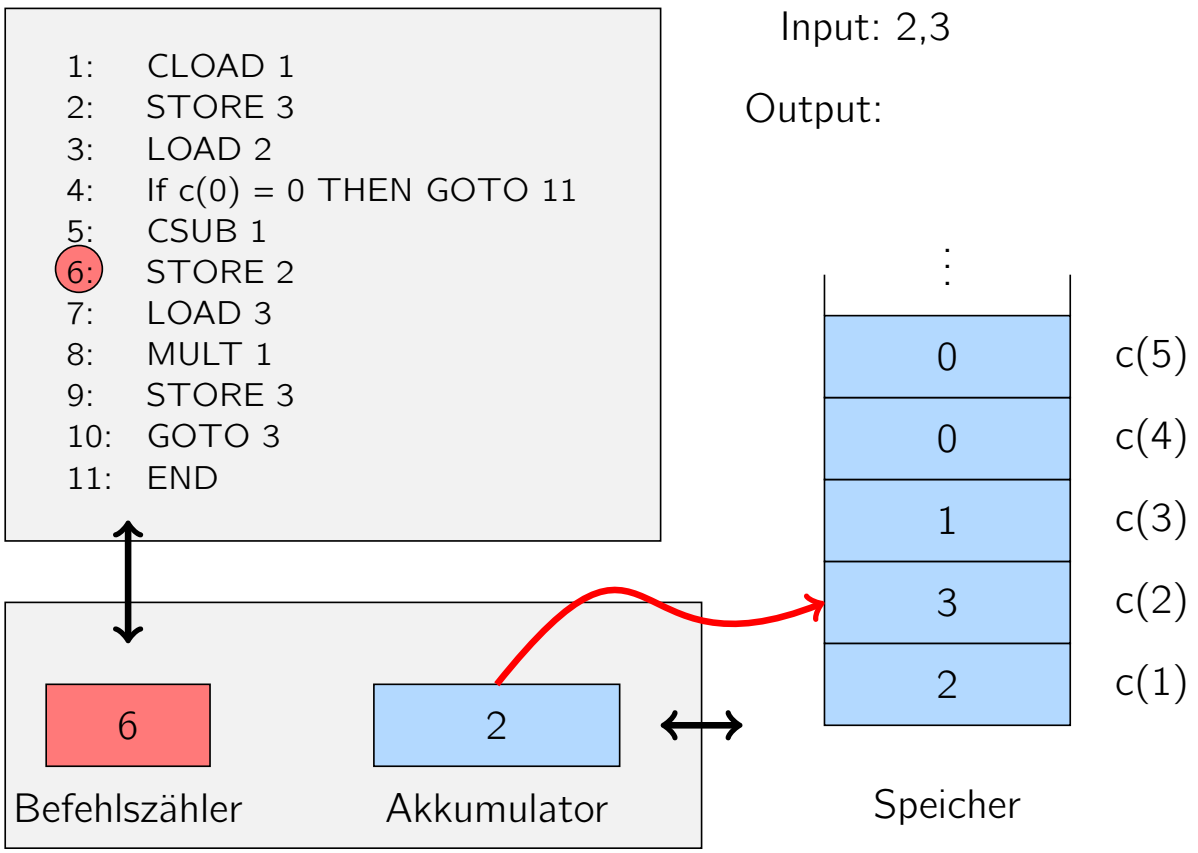
Beispielprogramm für die RAM



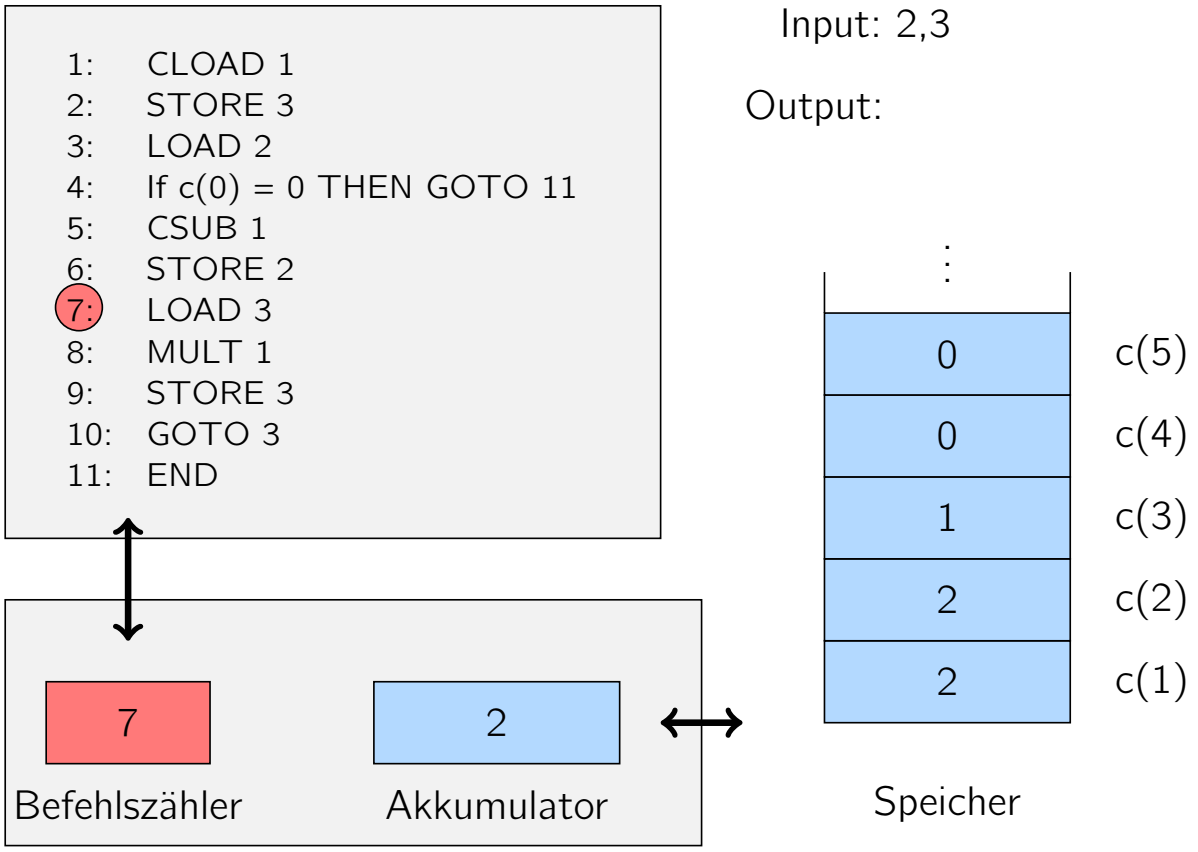
Beispielprogramm für die RAM



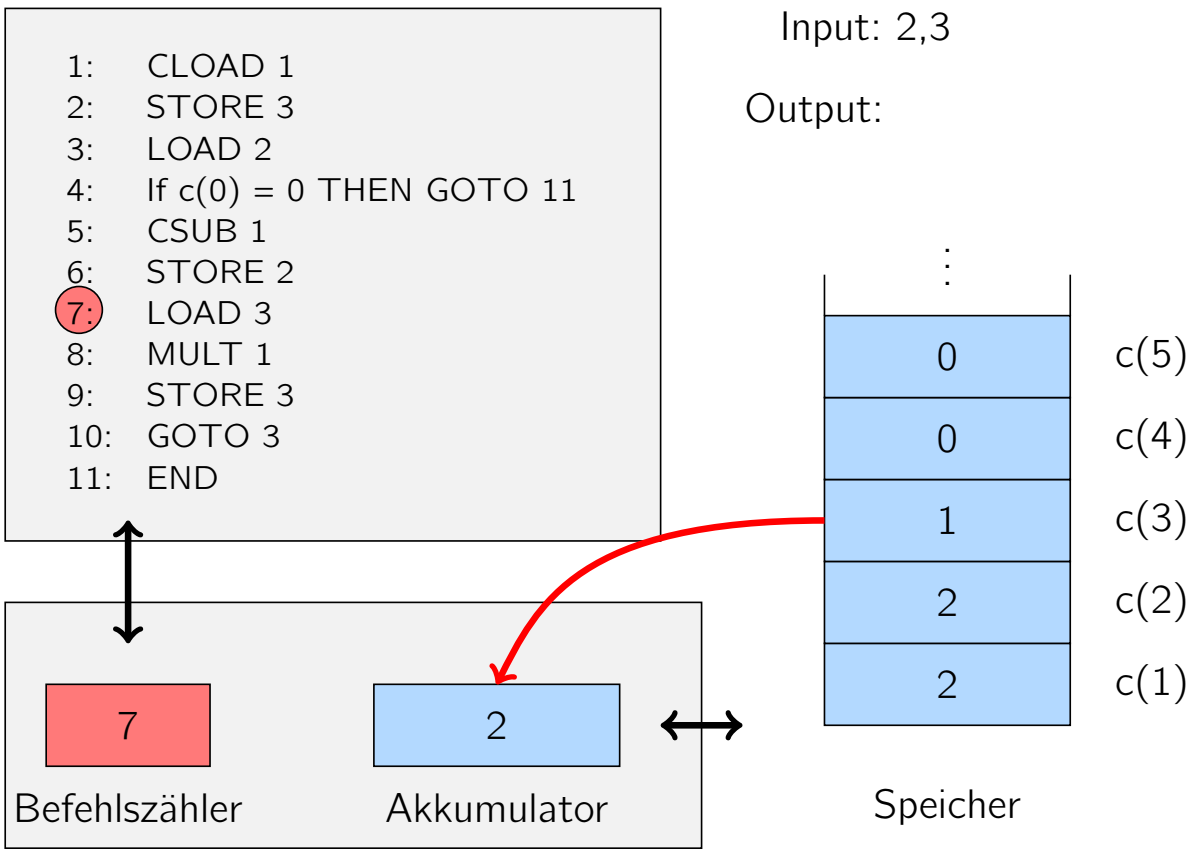
Beispielprogramm für die RAM



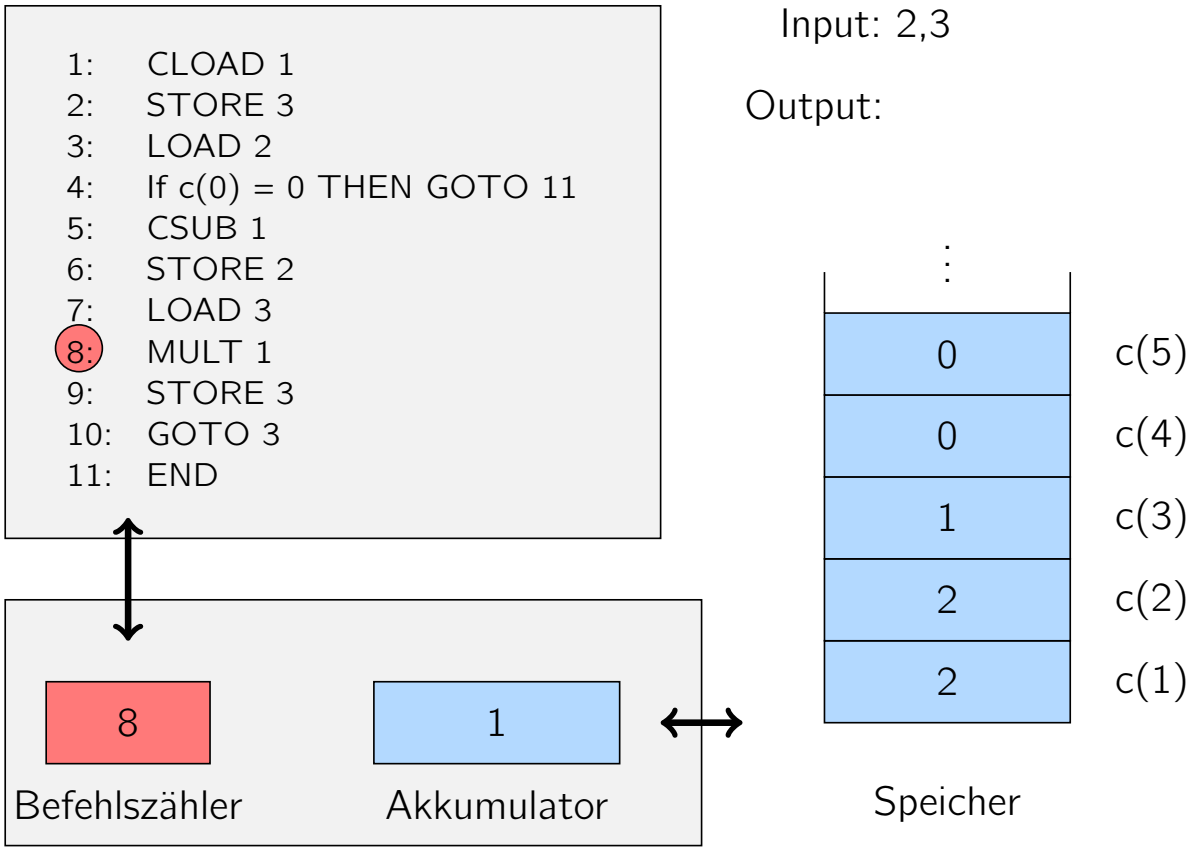
Beispielprogramm für die RAM



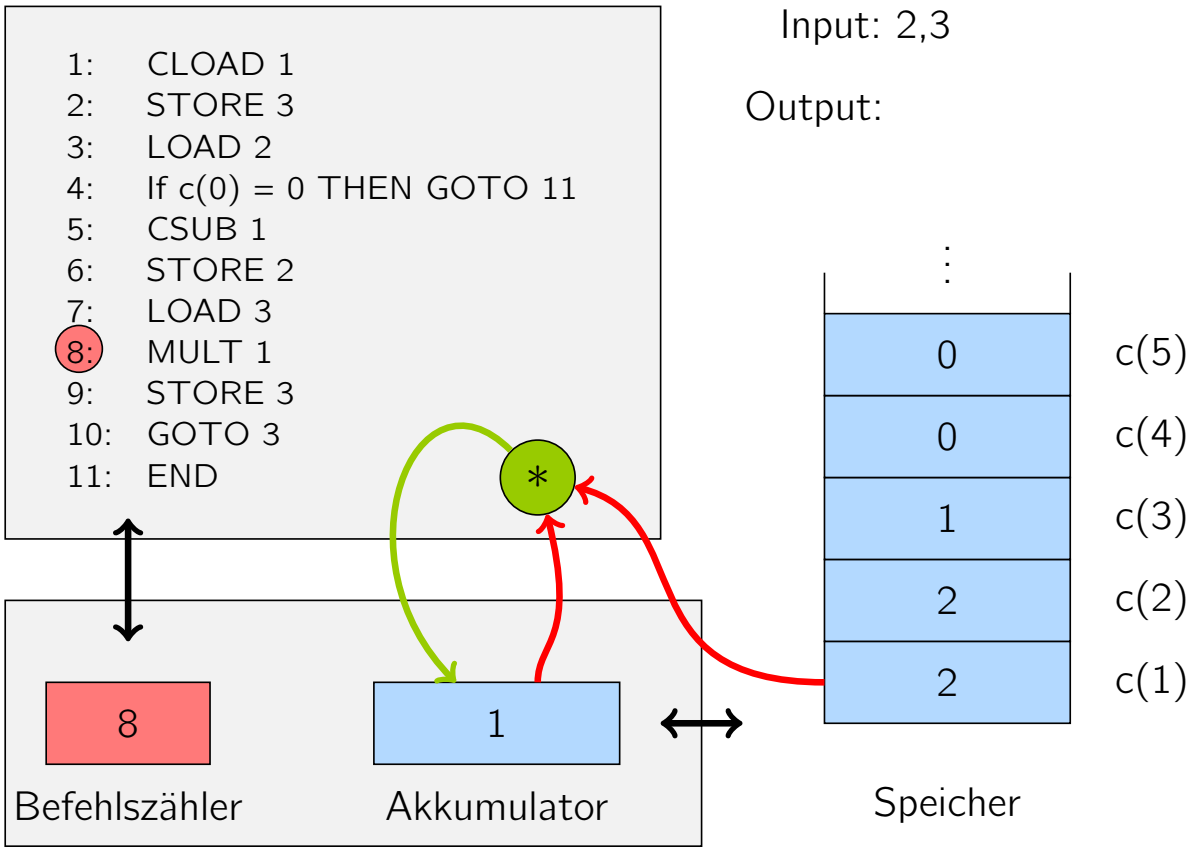
Beispielprogramm für die RAM



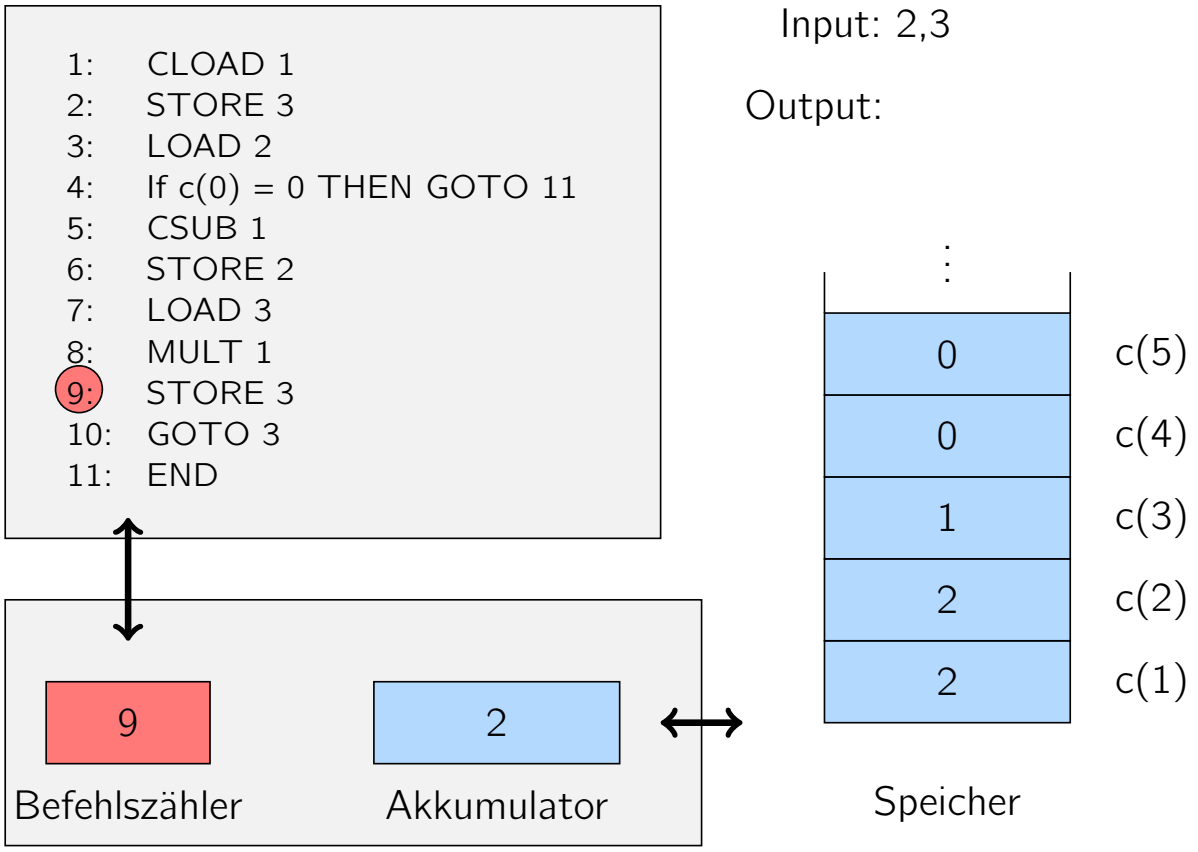
Beispielprogramm für die RAM



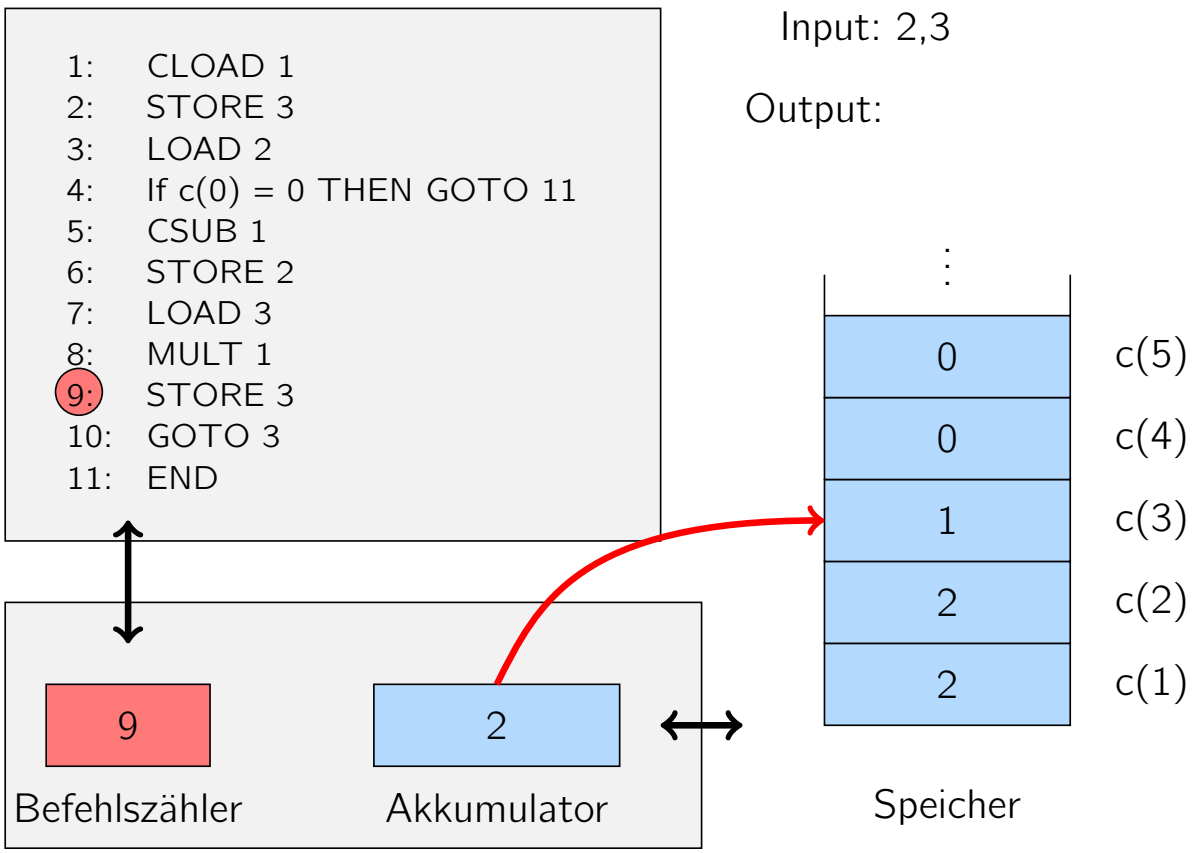
Beispielprogramm für die RAM



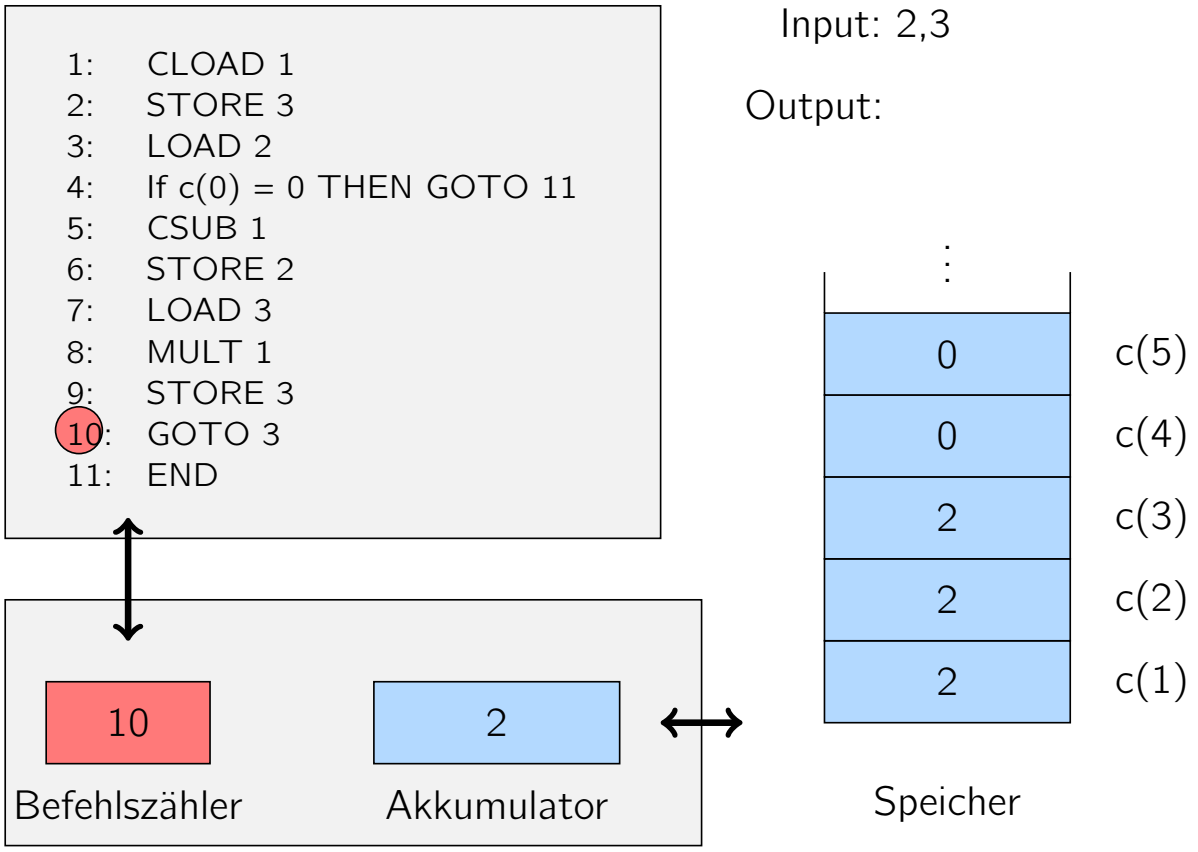
Beispielprogramm für die RAM



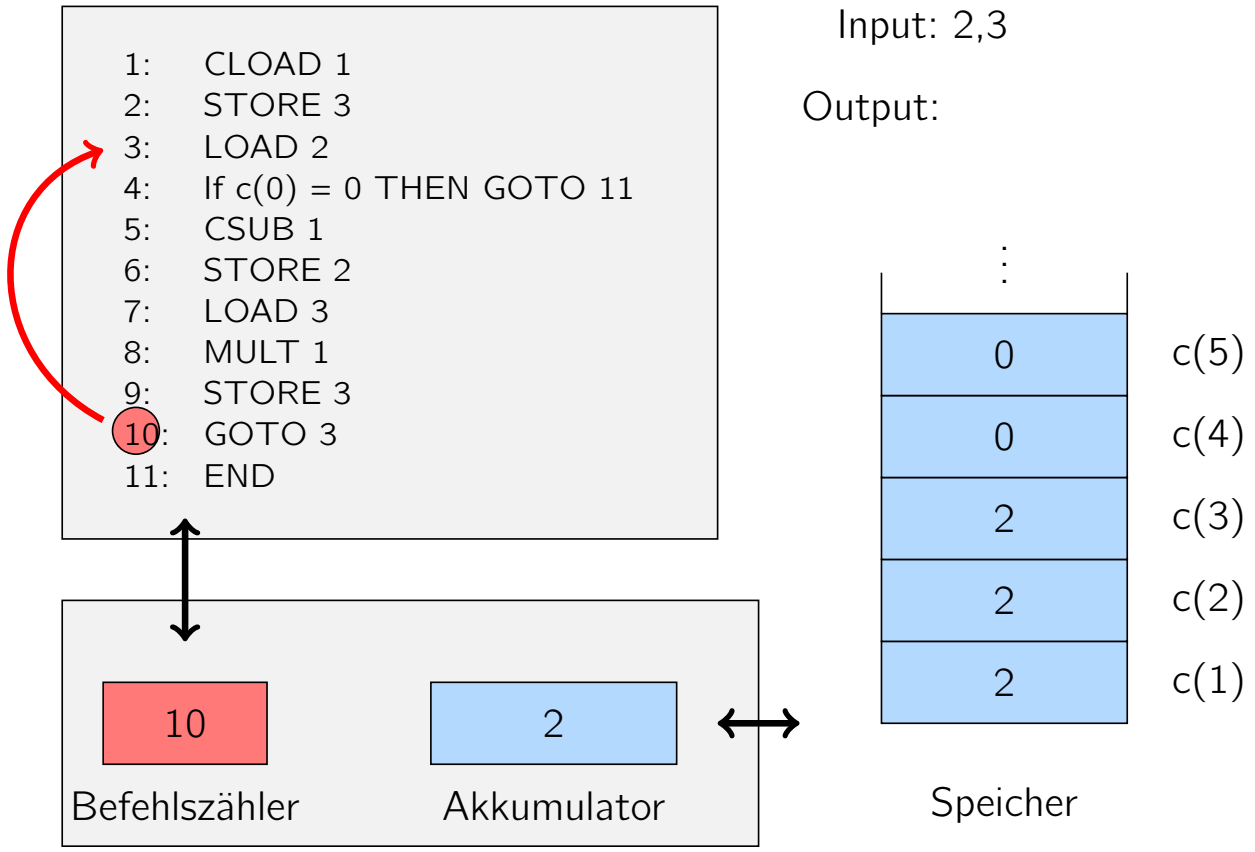
Beispielprogramm für die RAM



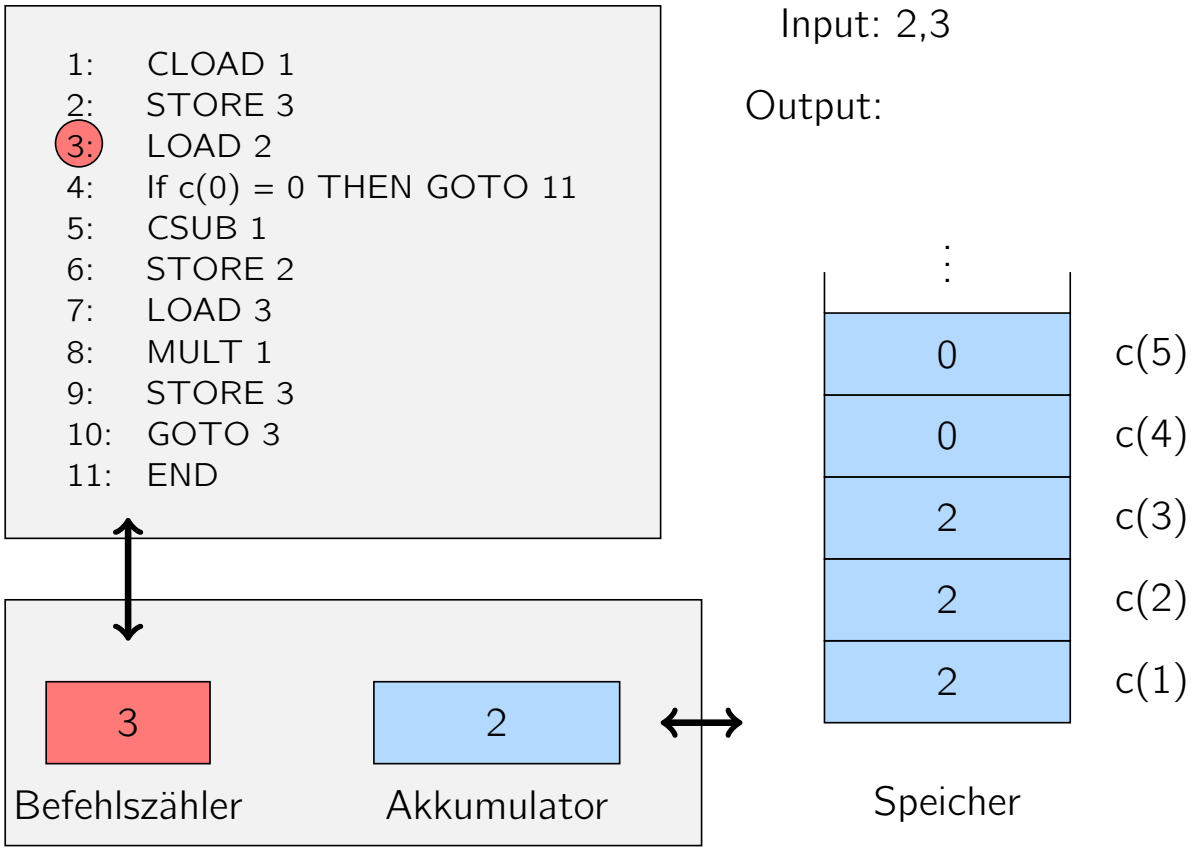
Beispielprogramm für die RAM



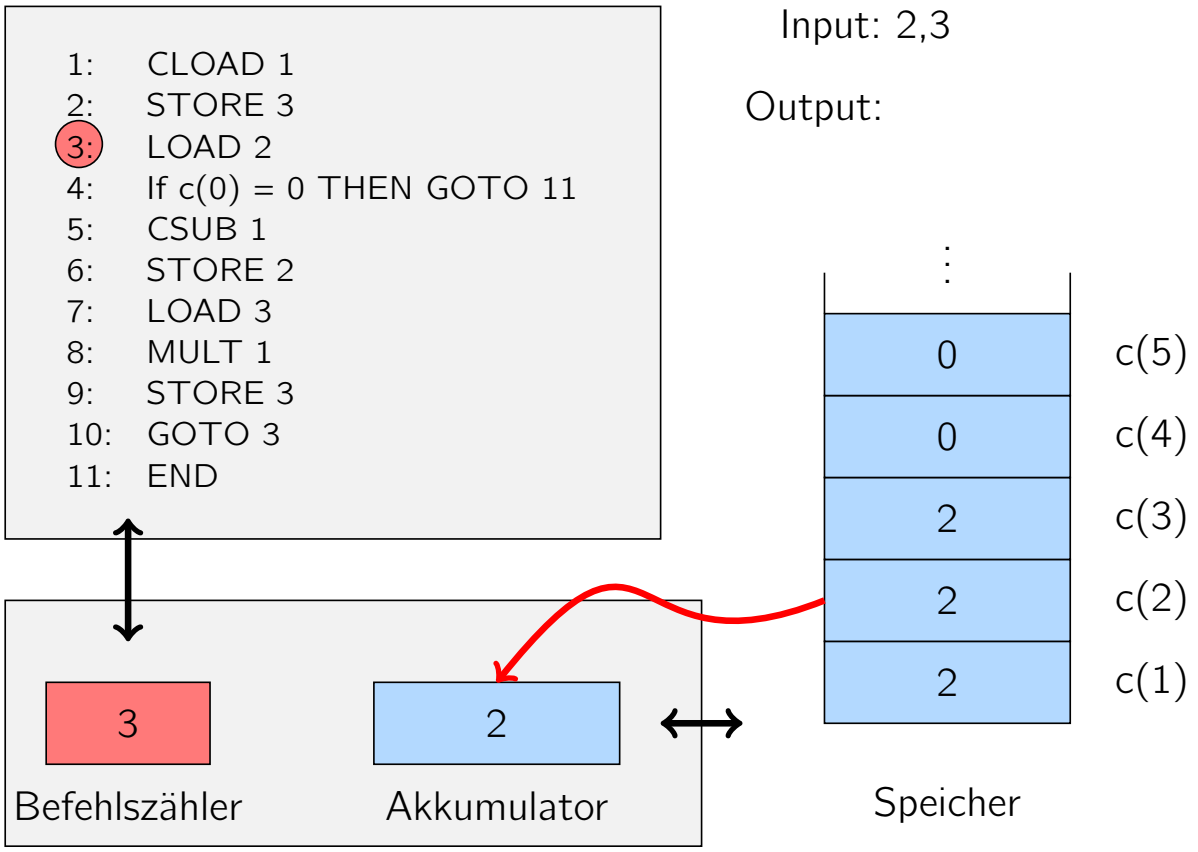
Beispielprogramm für die RAM



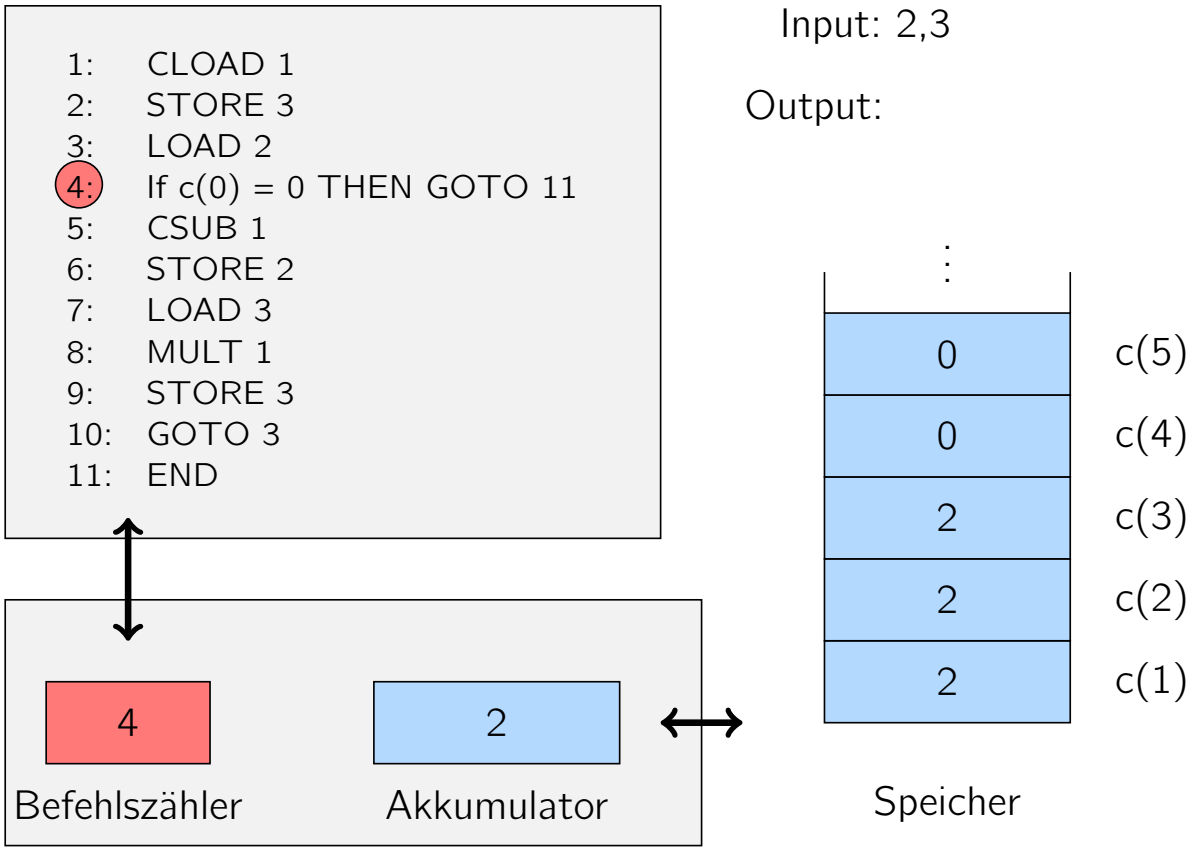
Beispielprogramm für die RAM



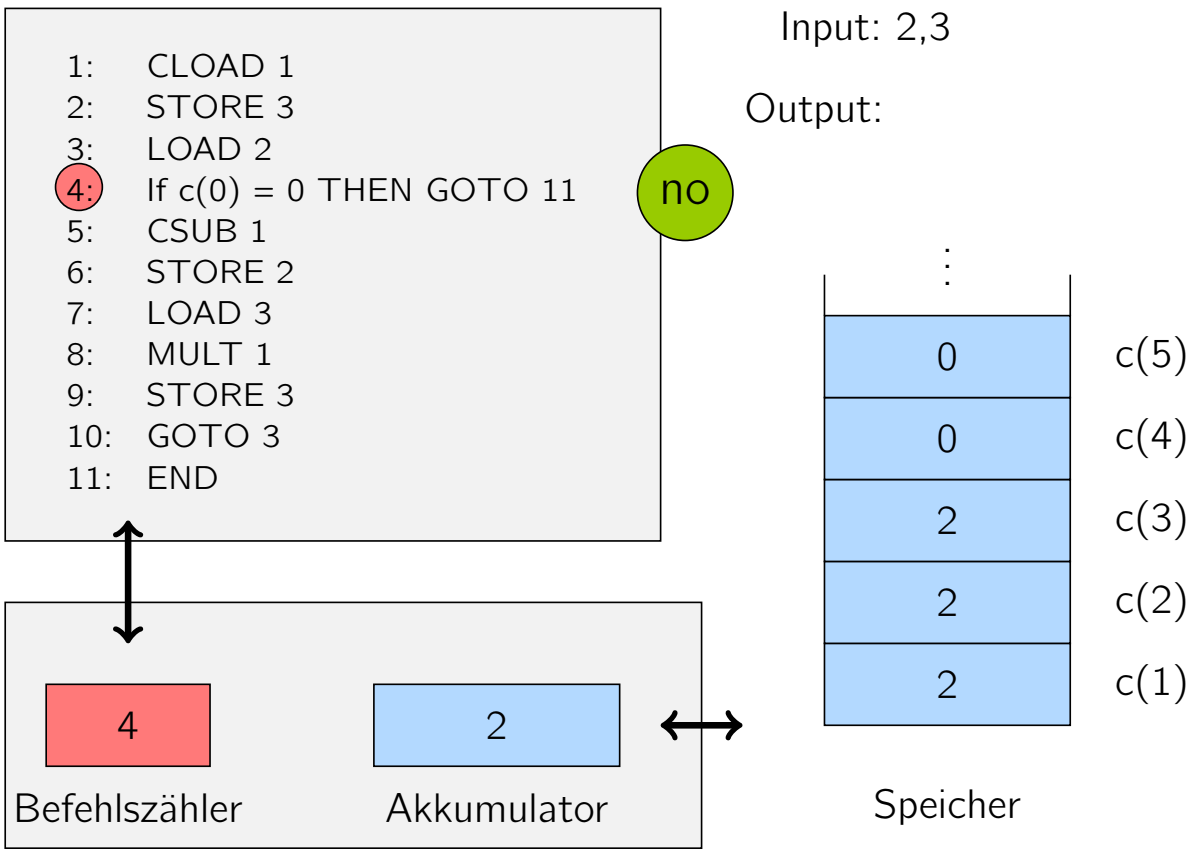
Beispielprogramm für die RAM



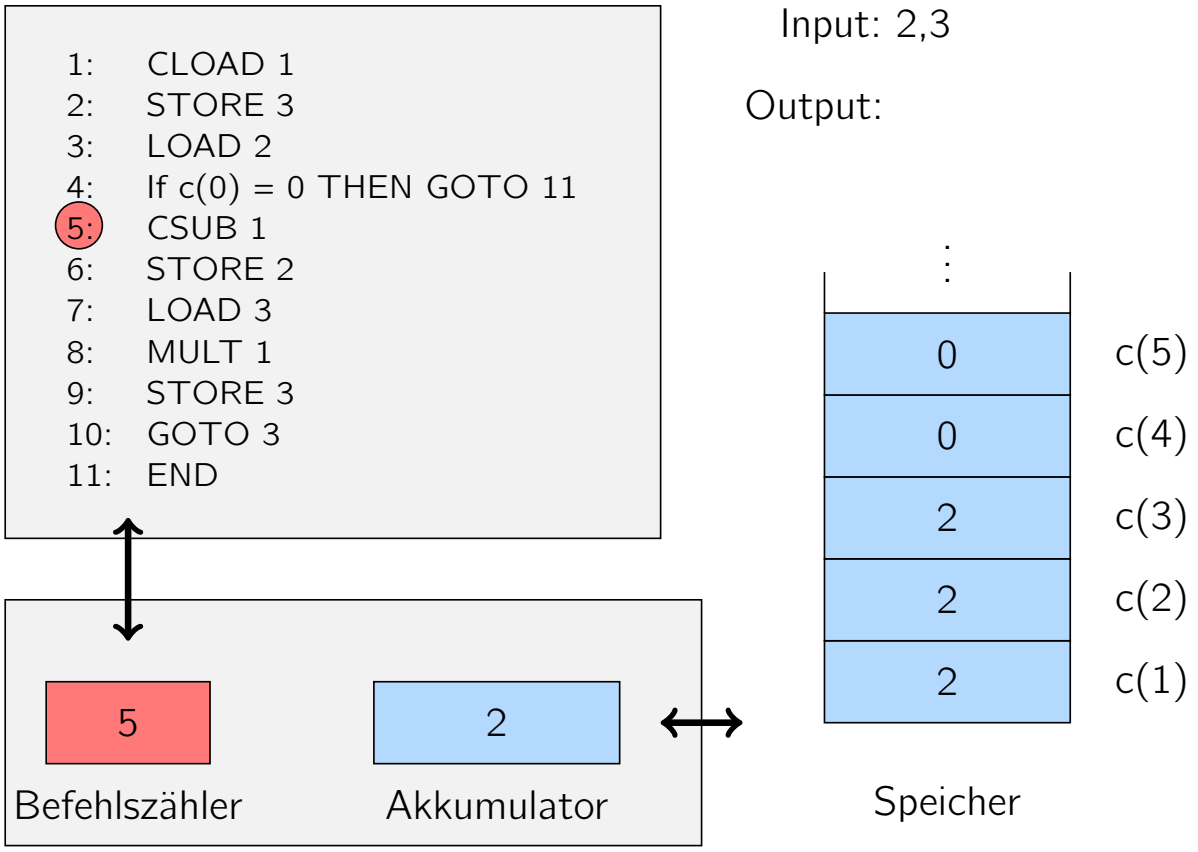
Beispielprogramm für die RAM



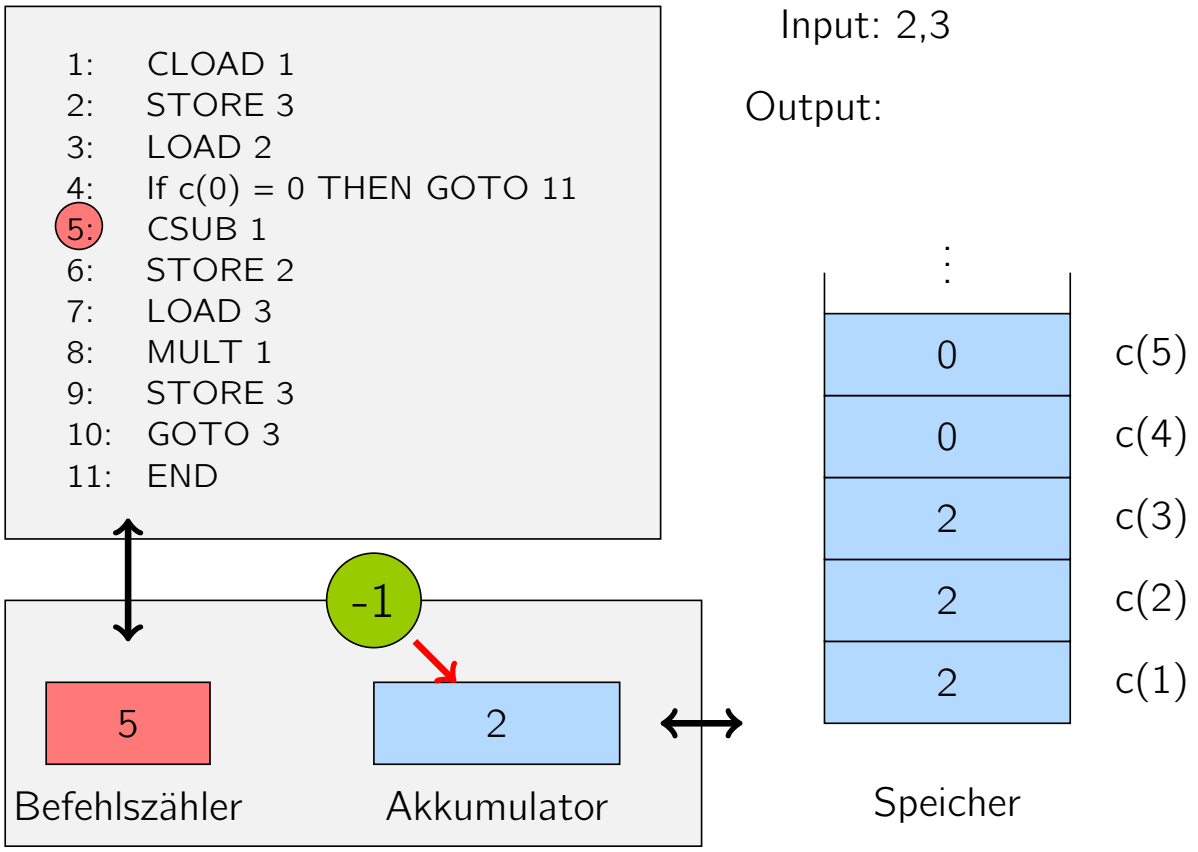
Beispielprogramm für die RAM



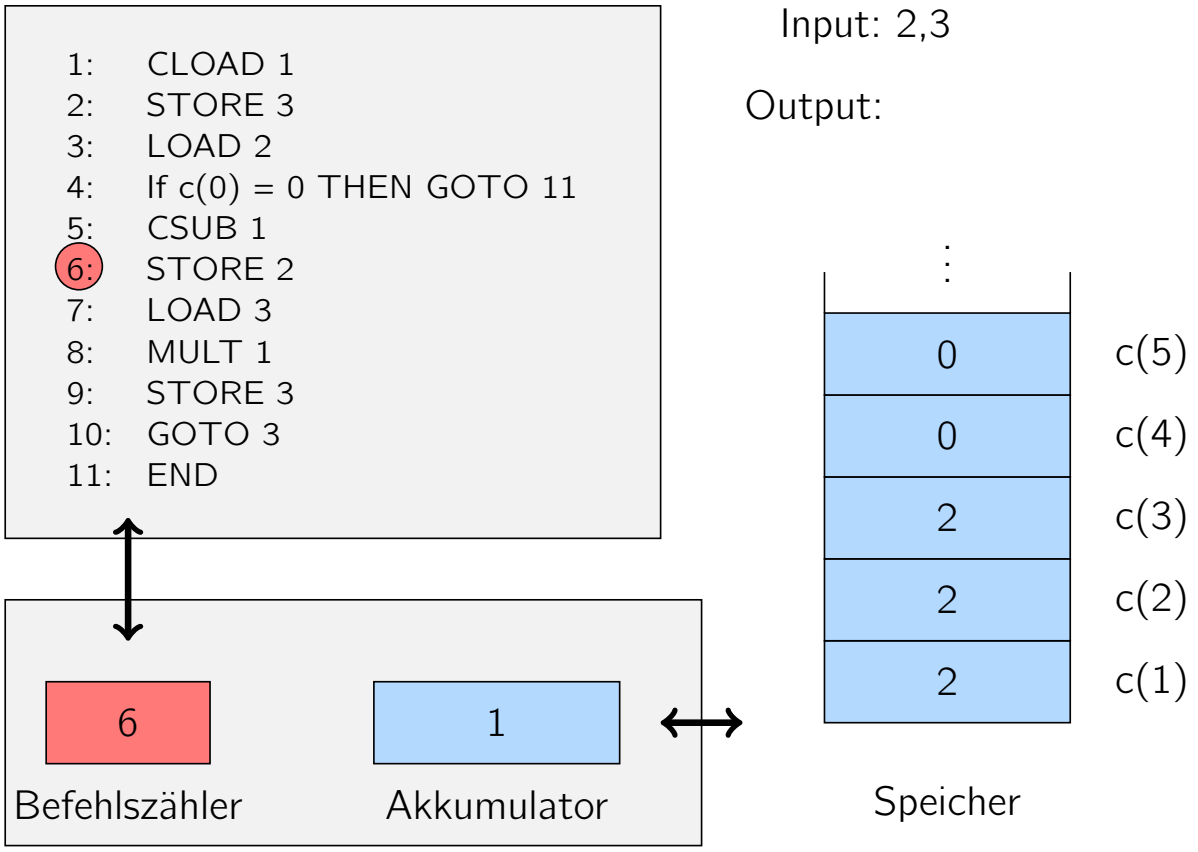
Beispielprogramm für die RAM



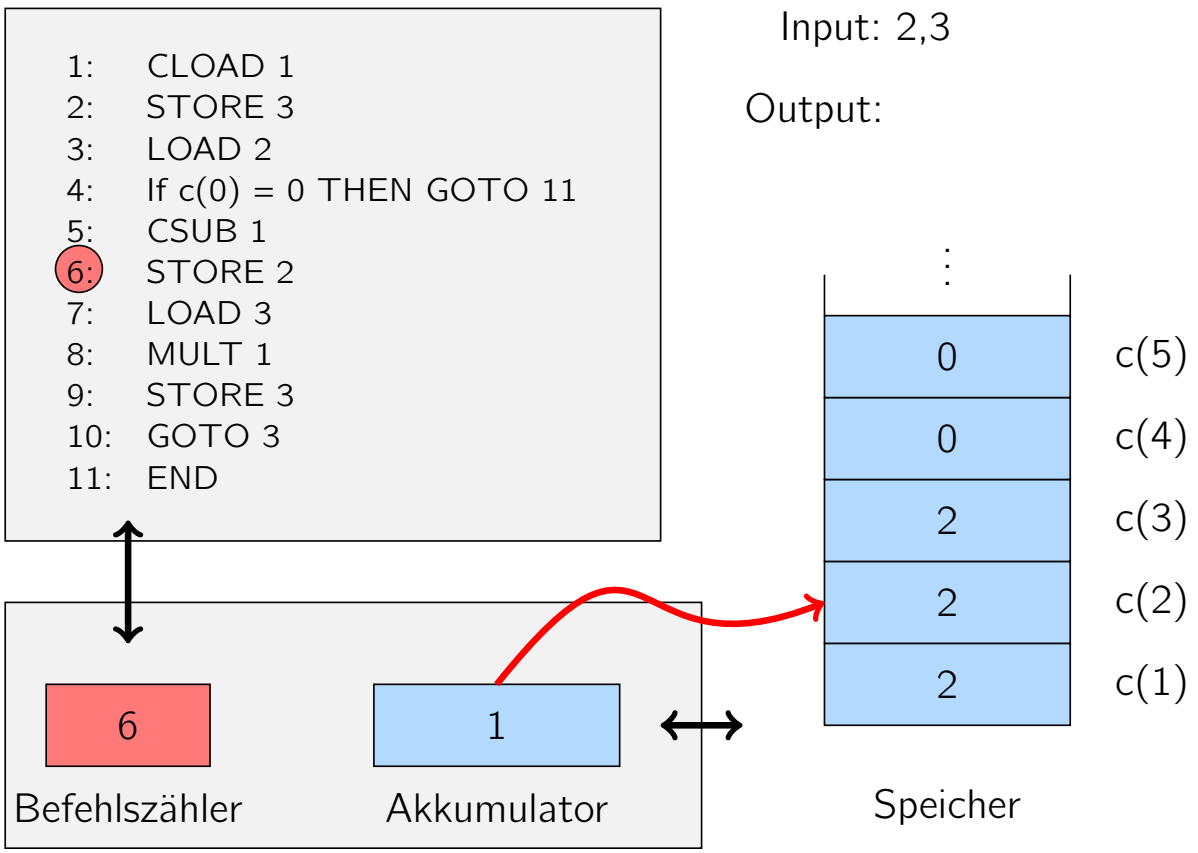
Beispielprogramm für die RAM



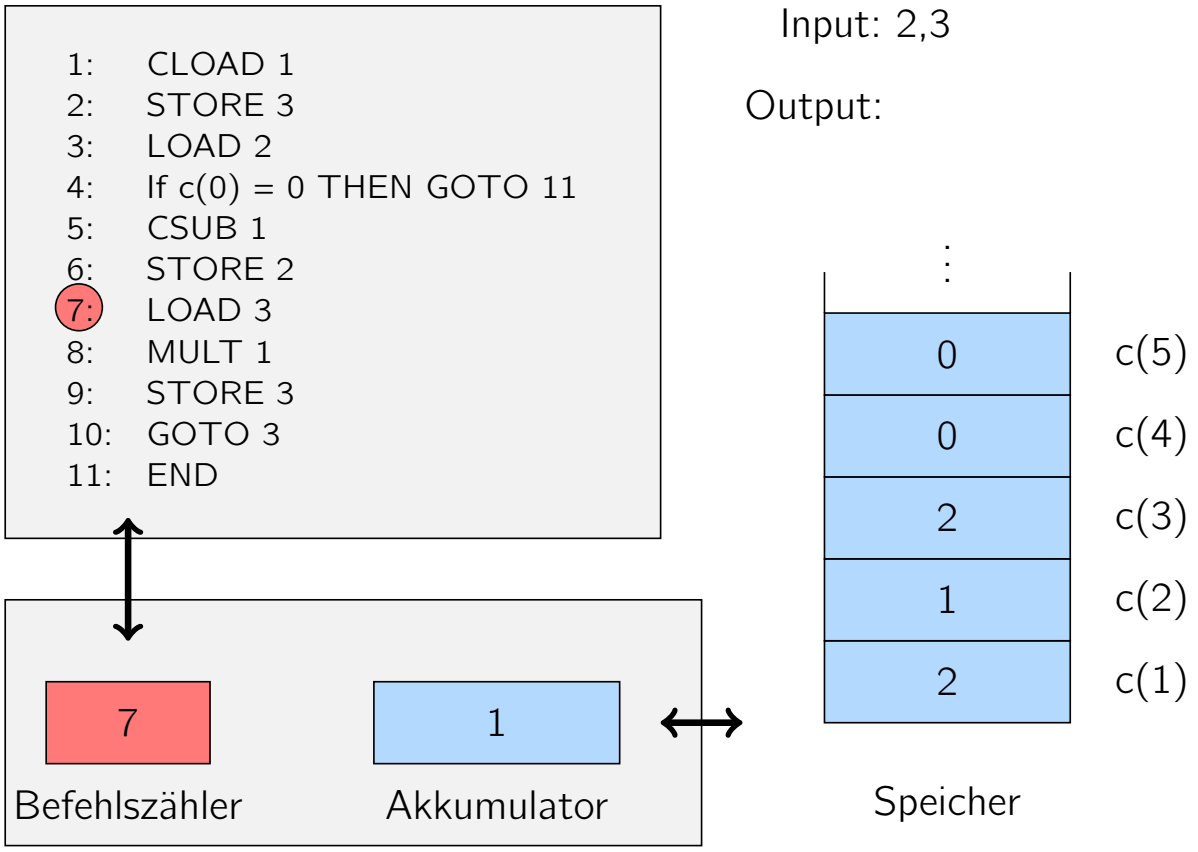
Beispielprogramm für die RAM



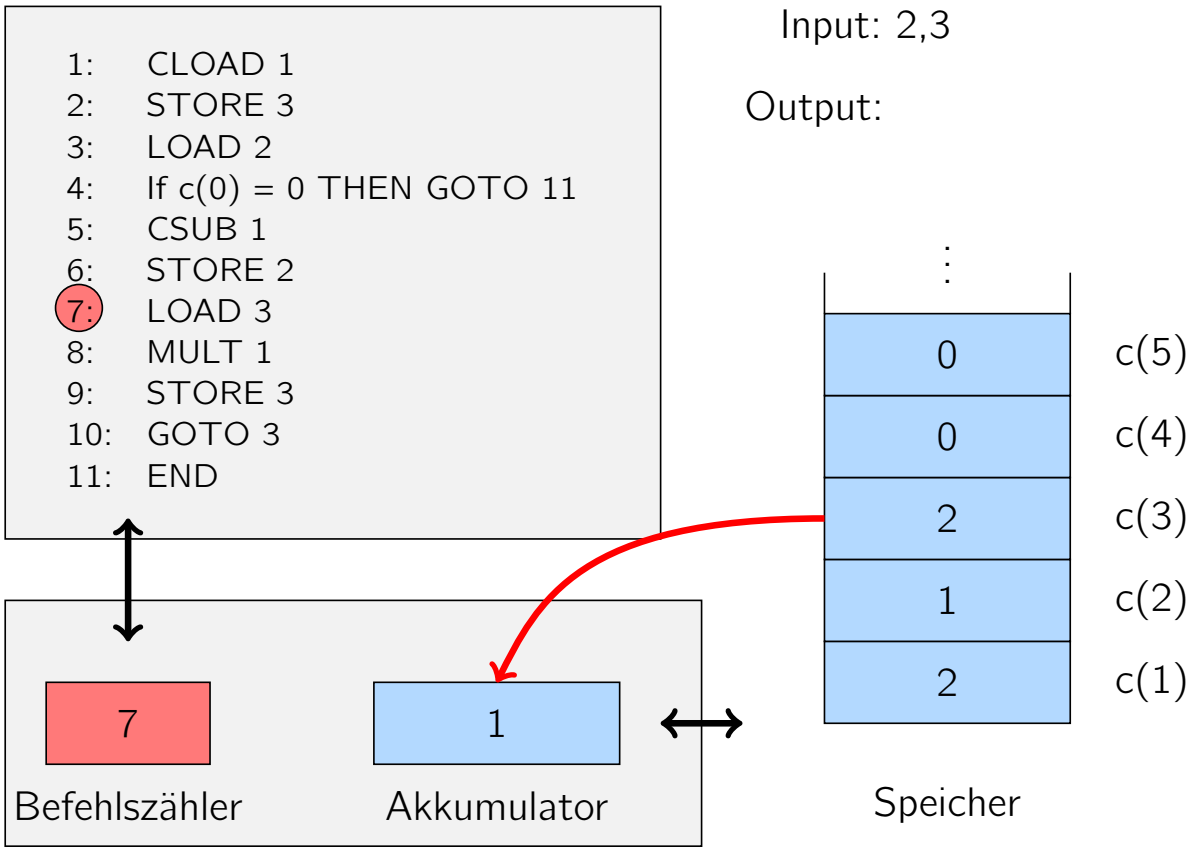
Beispielprogramm für die RAM



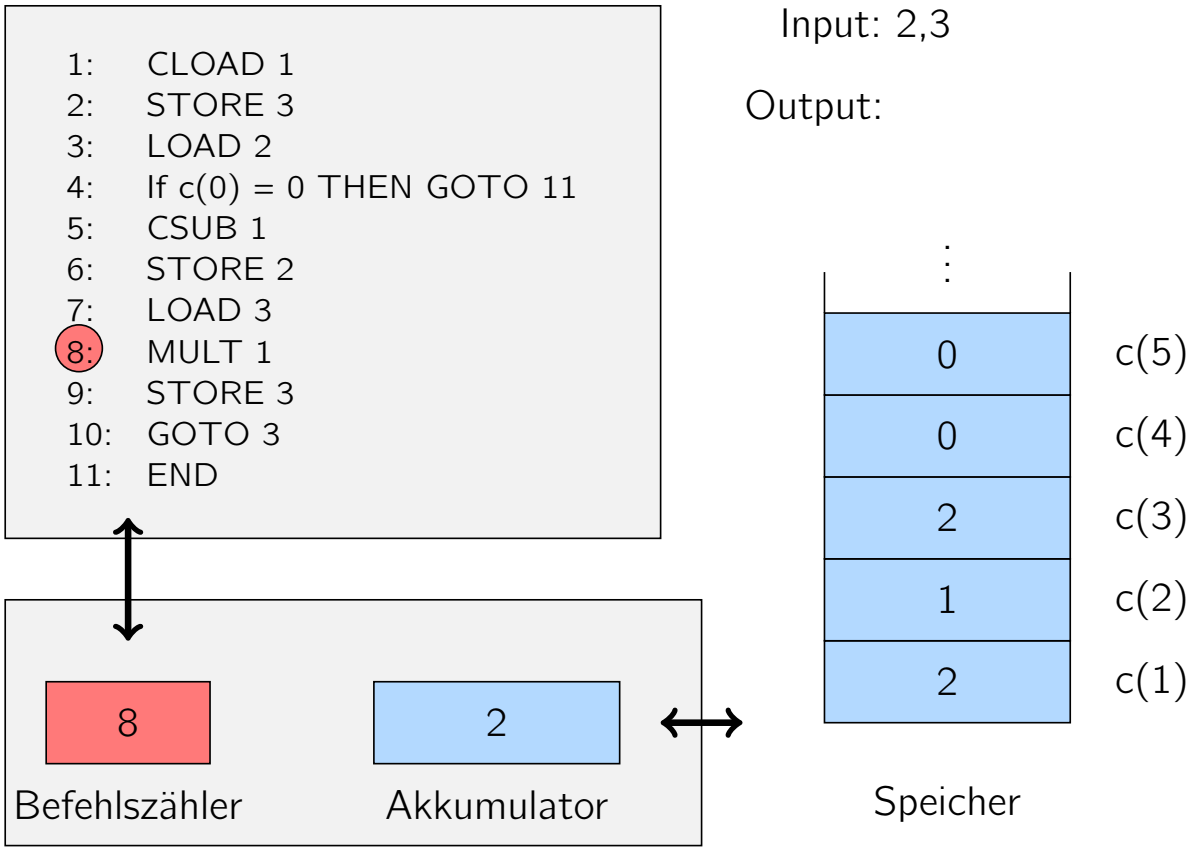
Beispielprogramm für die RAM



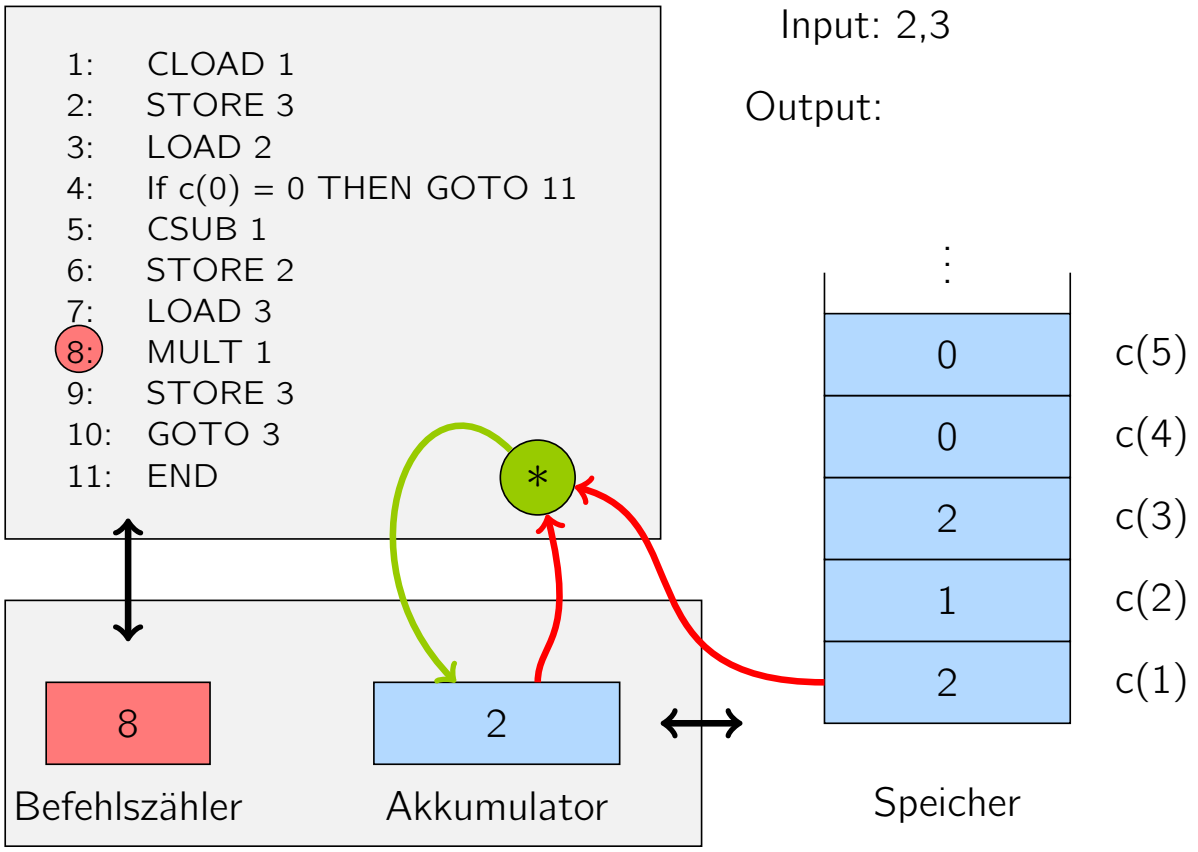
Beispielprogramm für die RAM



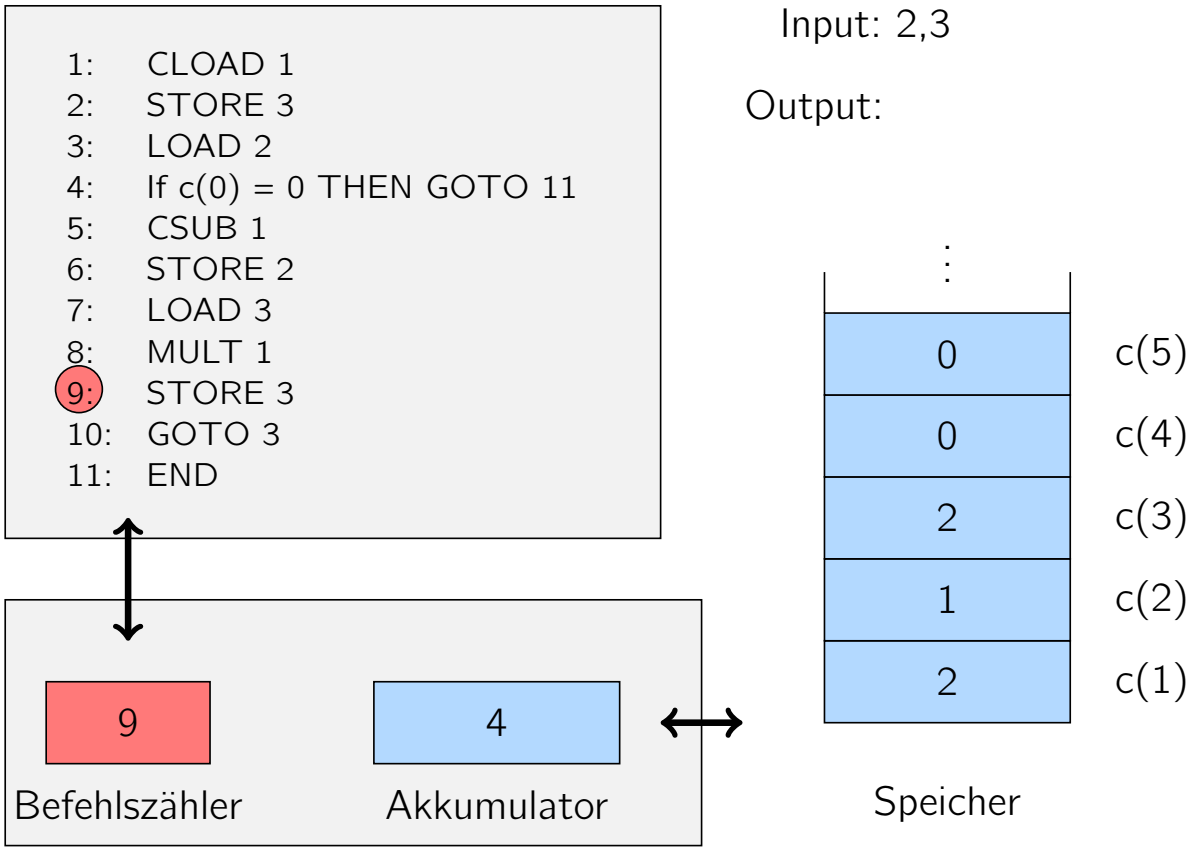
Beispielprogramm für die RAM



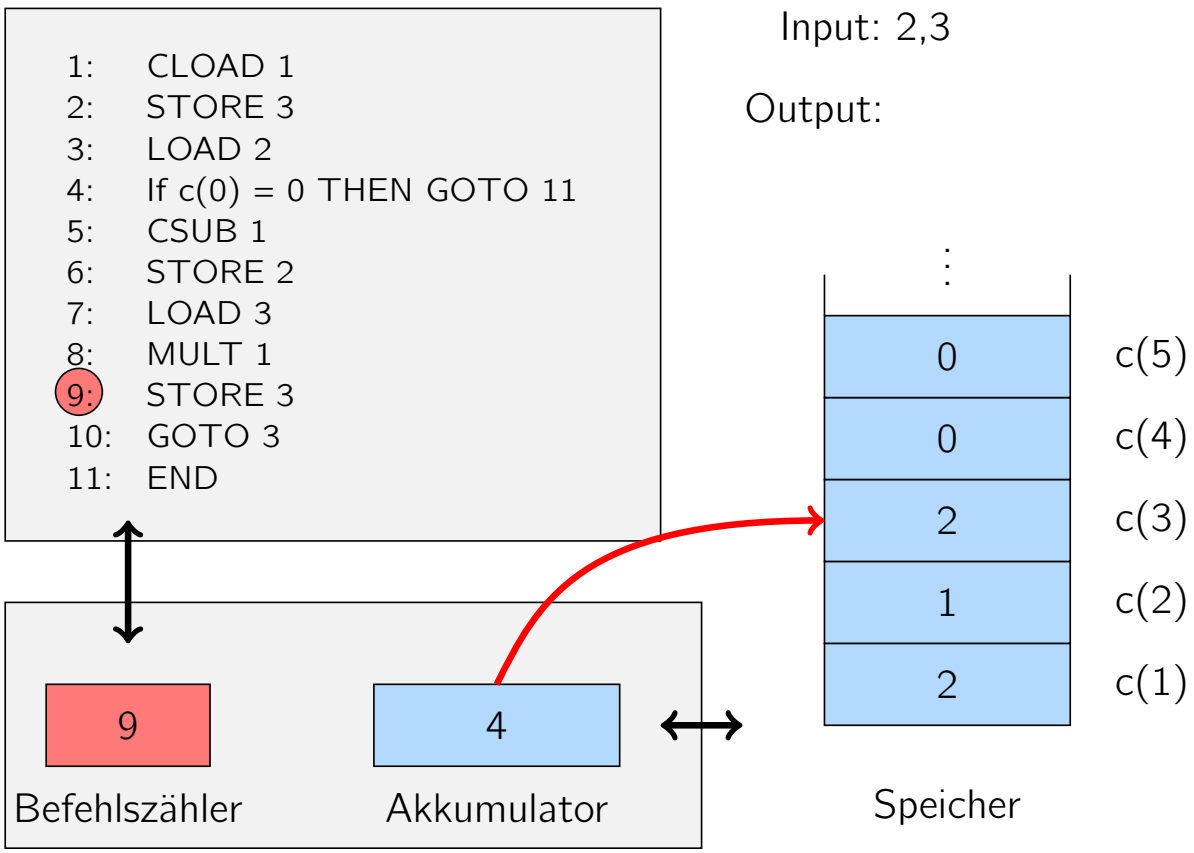
Beispielprogramm für die RAM



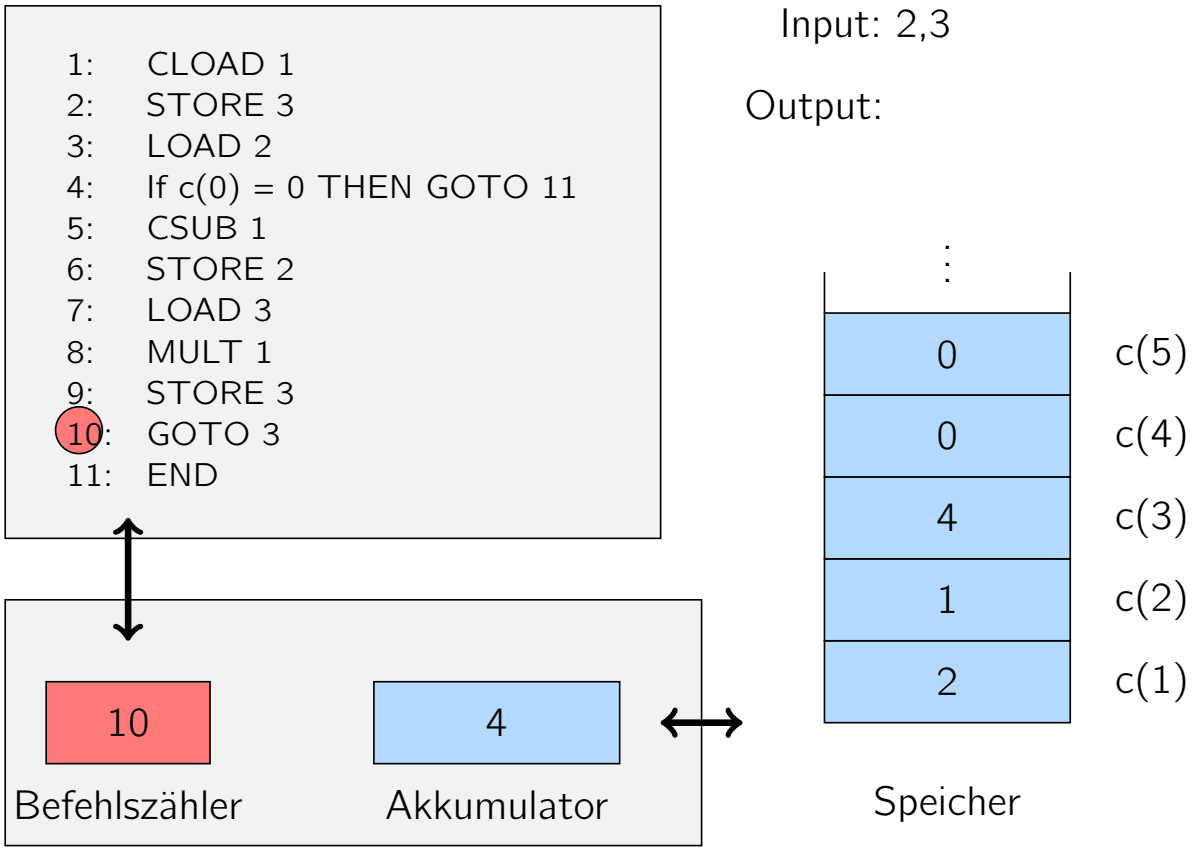
Beispielprogramm für die RAM



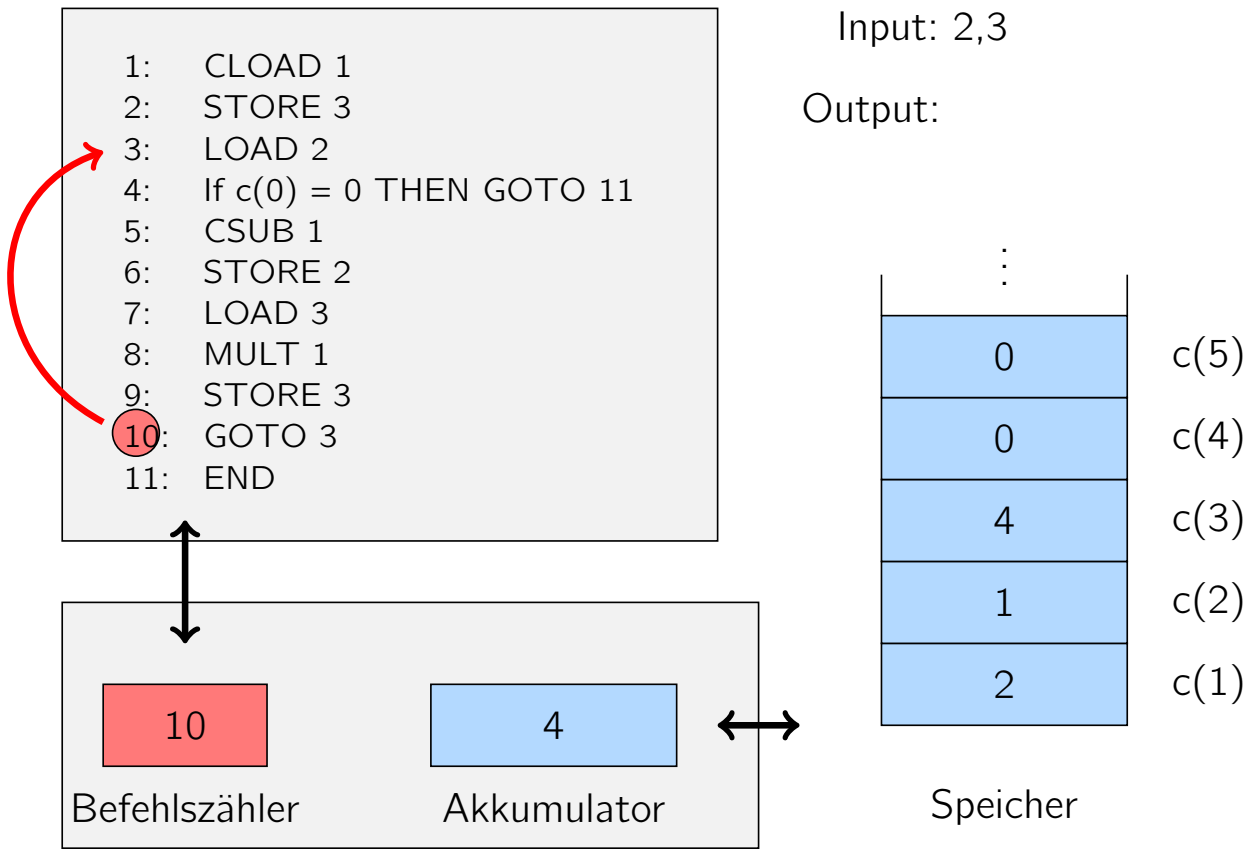
Beispielprogramm für die RAM



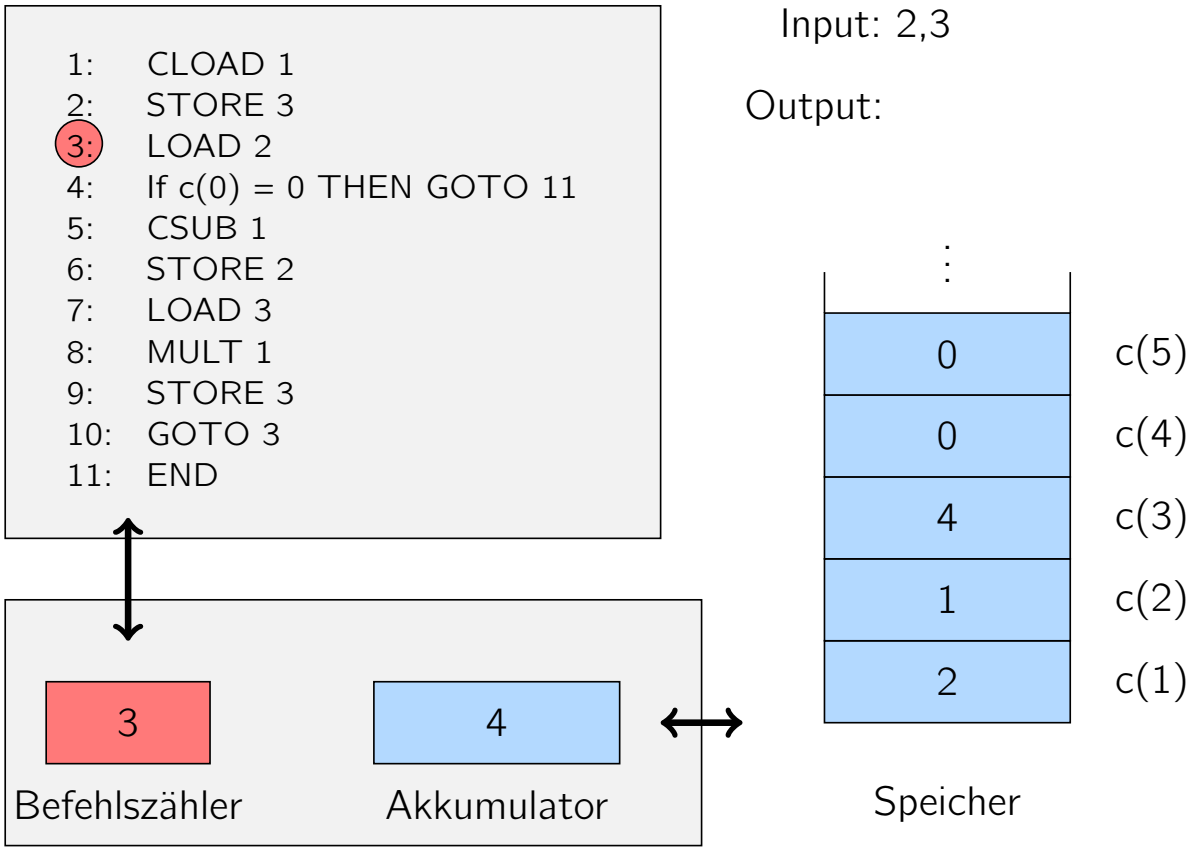
Beispielprogramm für die RAM



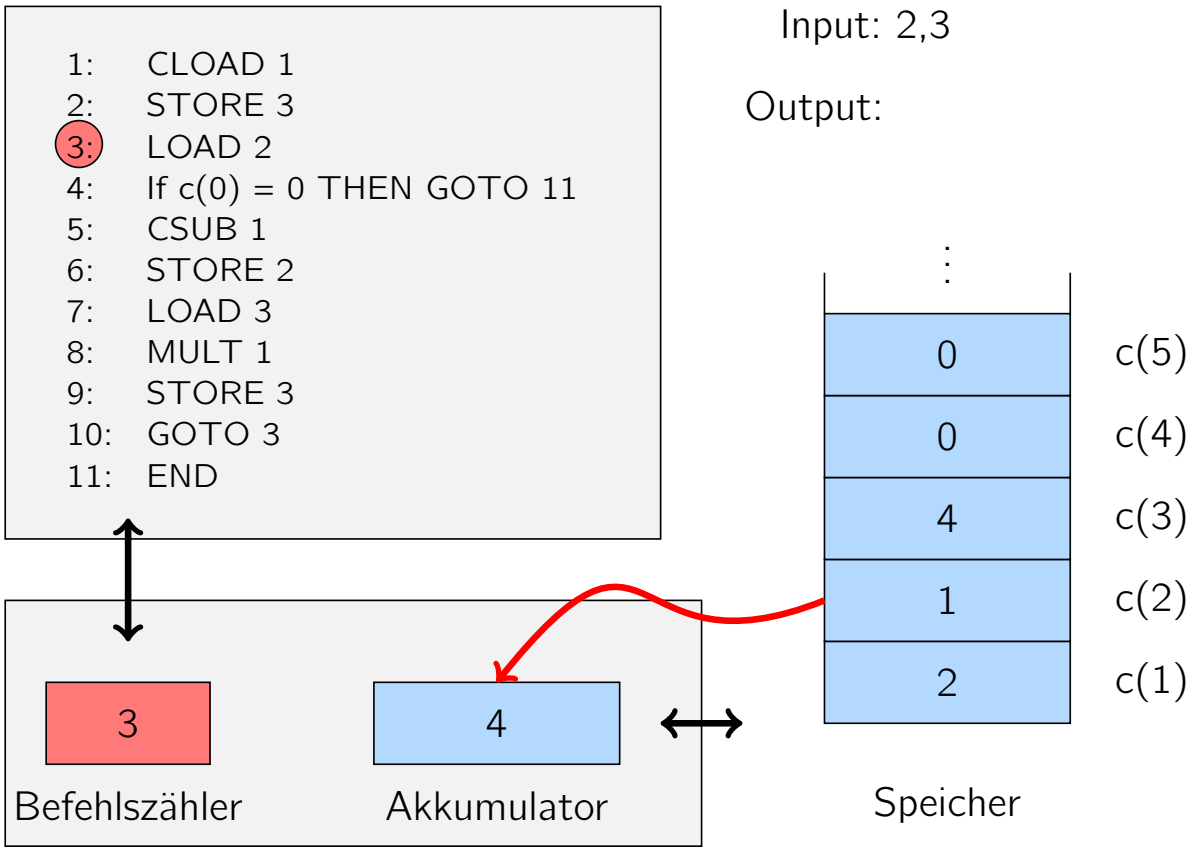
Beispielprogramm für die RAM



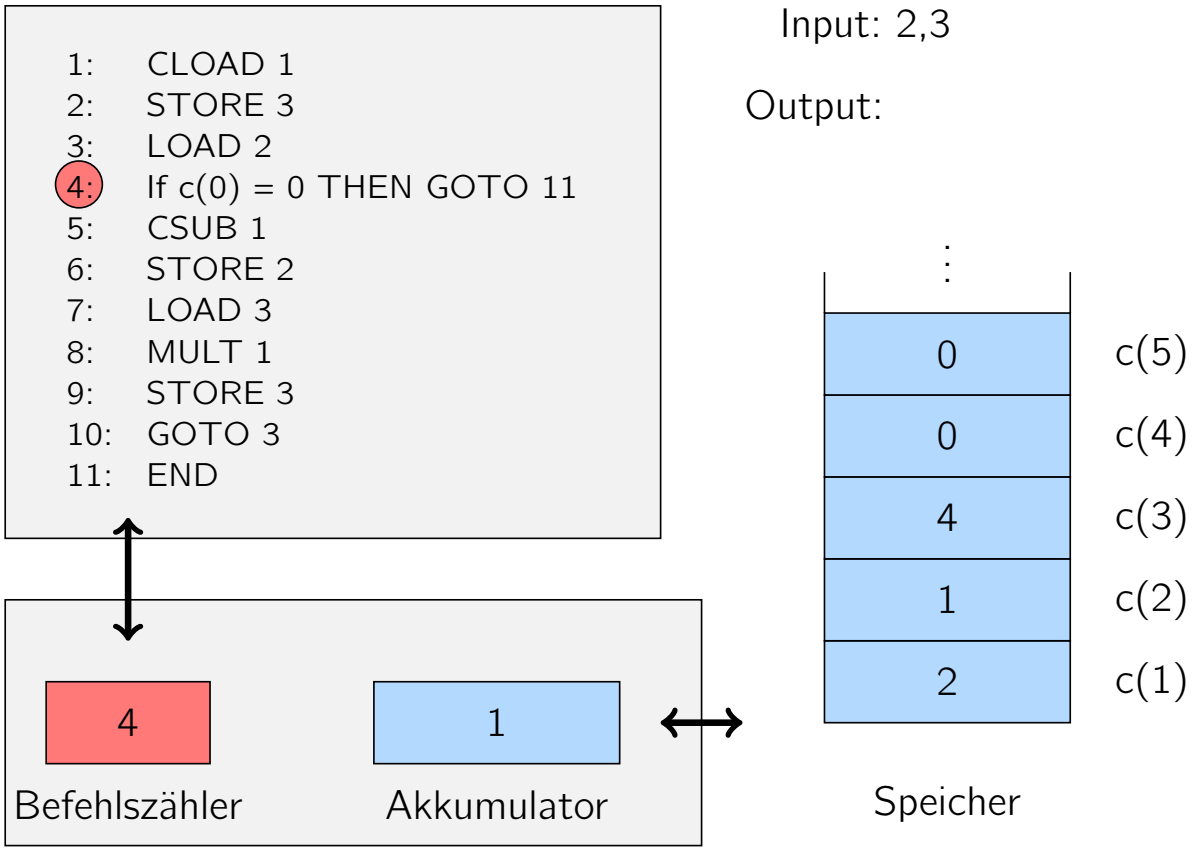
Beispielprogramm für die RAM



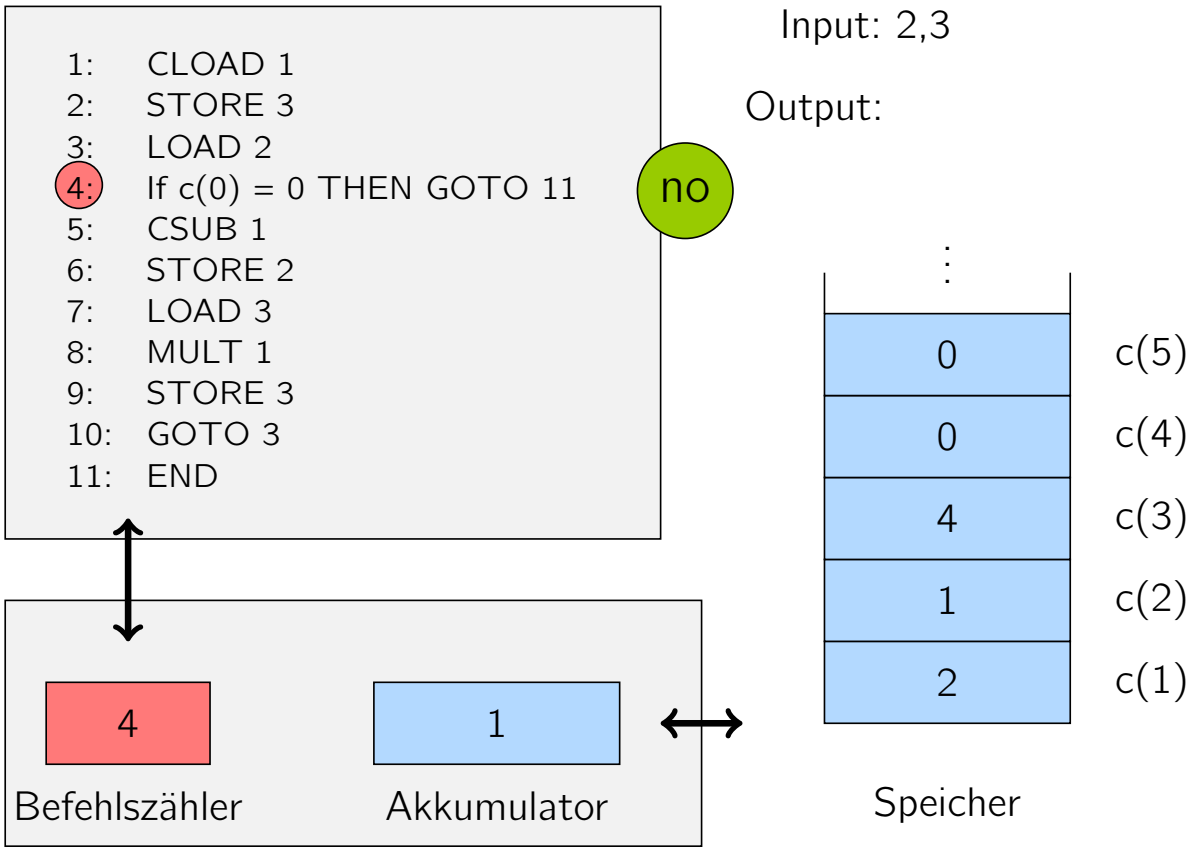
Beispielprogramm für die RAM



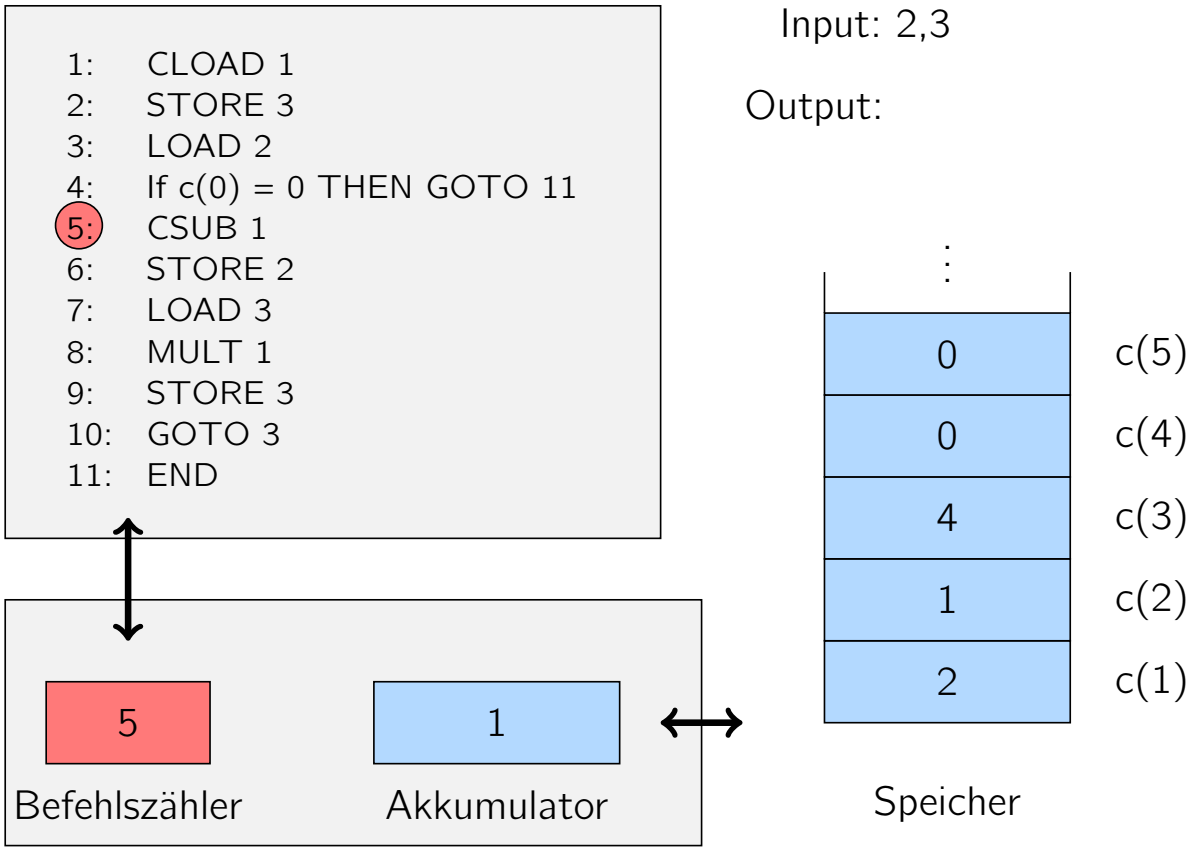
Beispielprogramm für die RAM



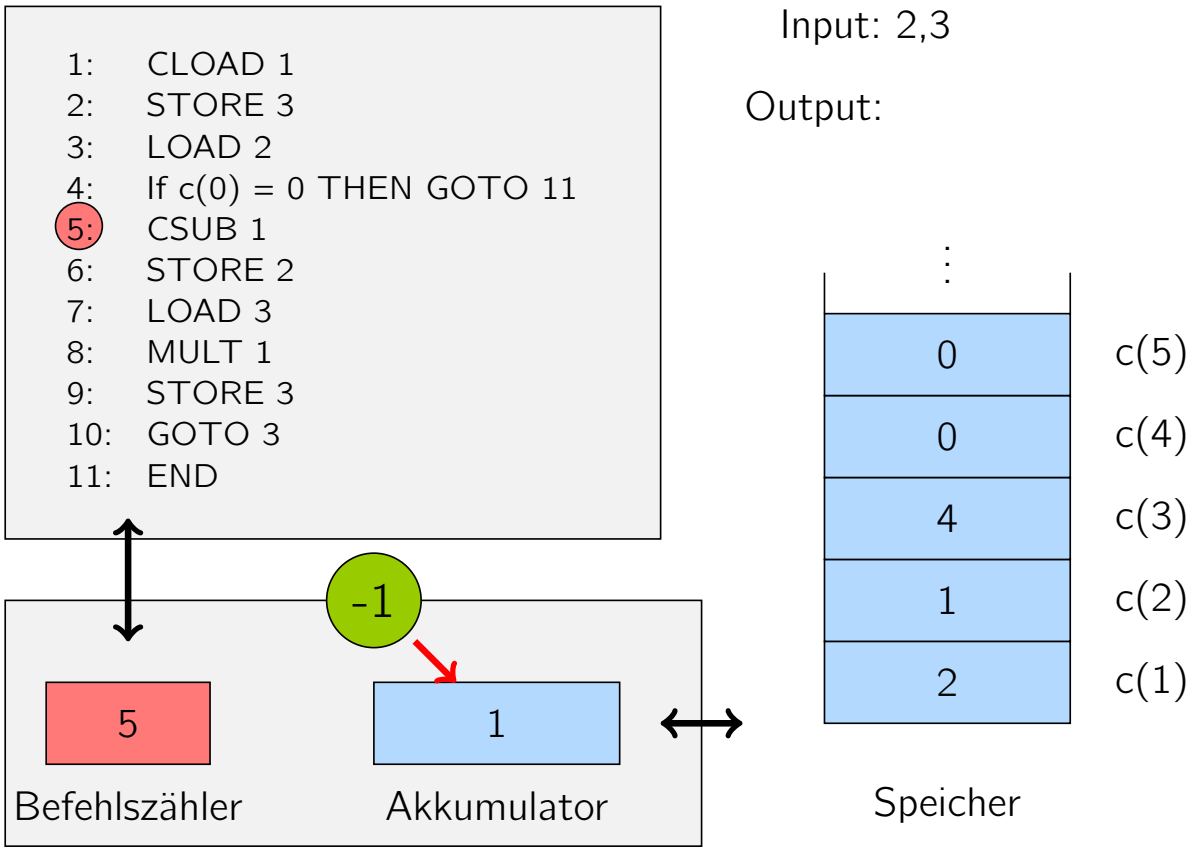
Beispielprogramm für die RAM



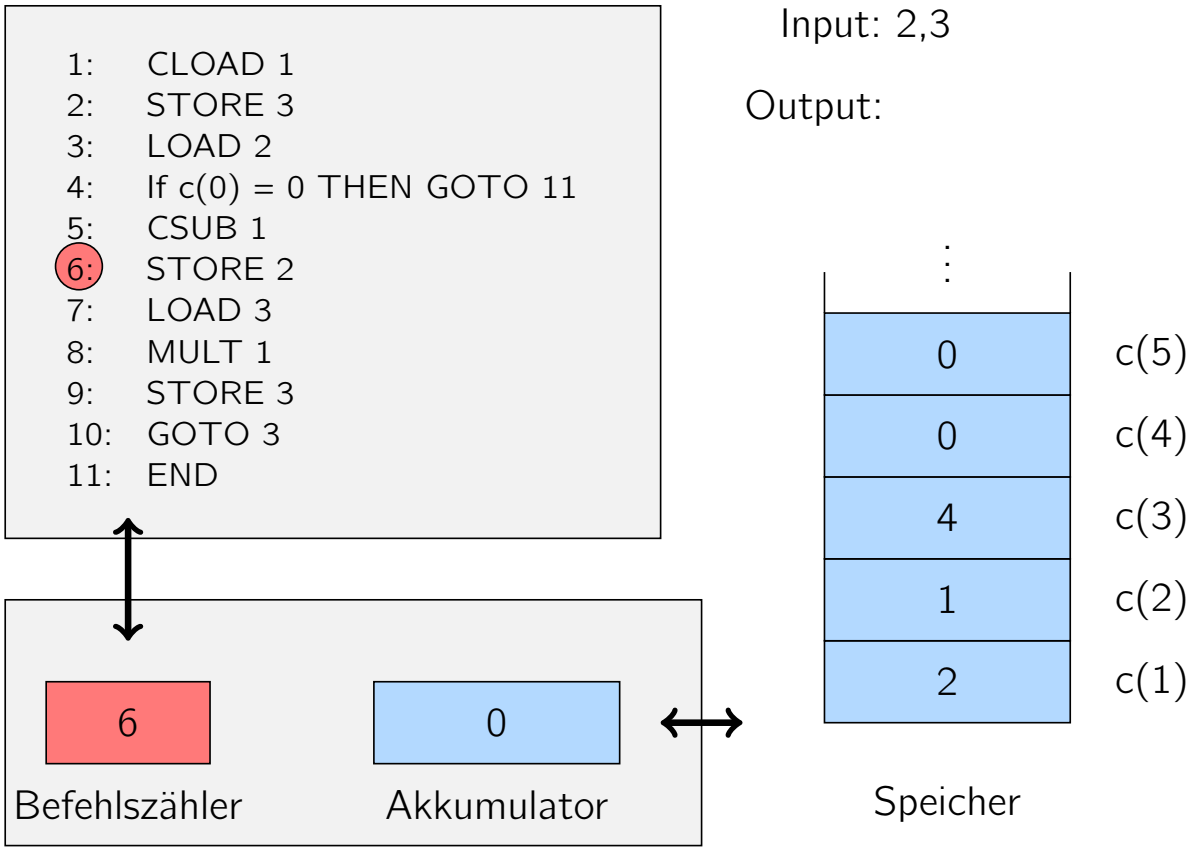
Beispielprogramm für die RAM



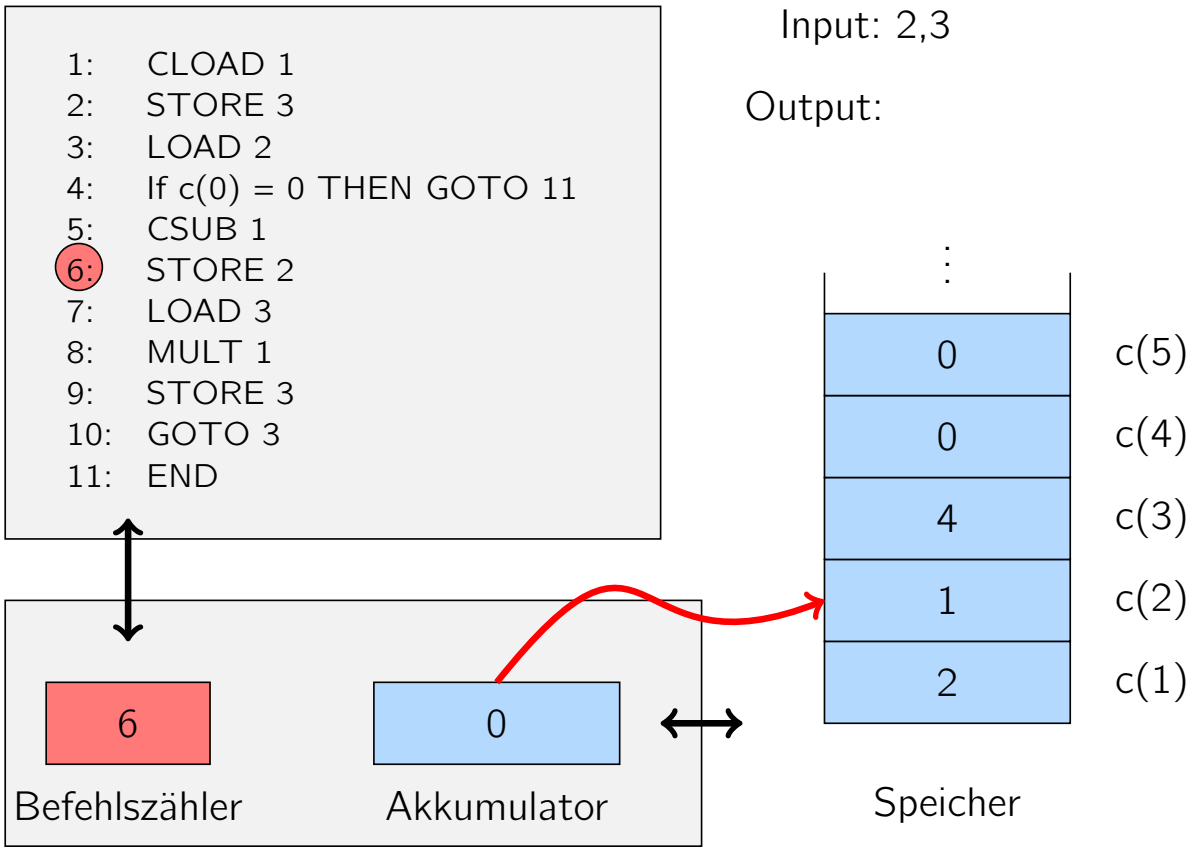
Beispielprogramm für die RAM



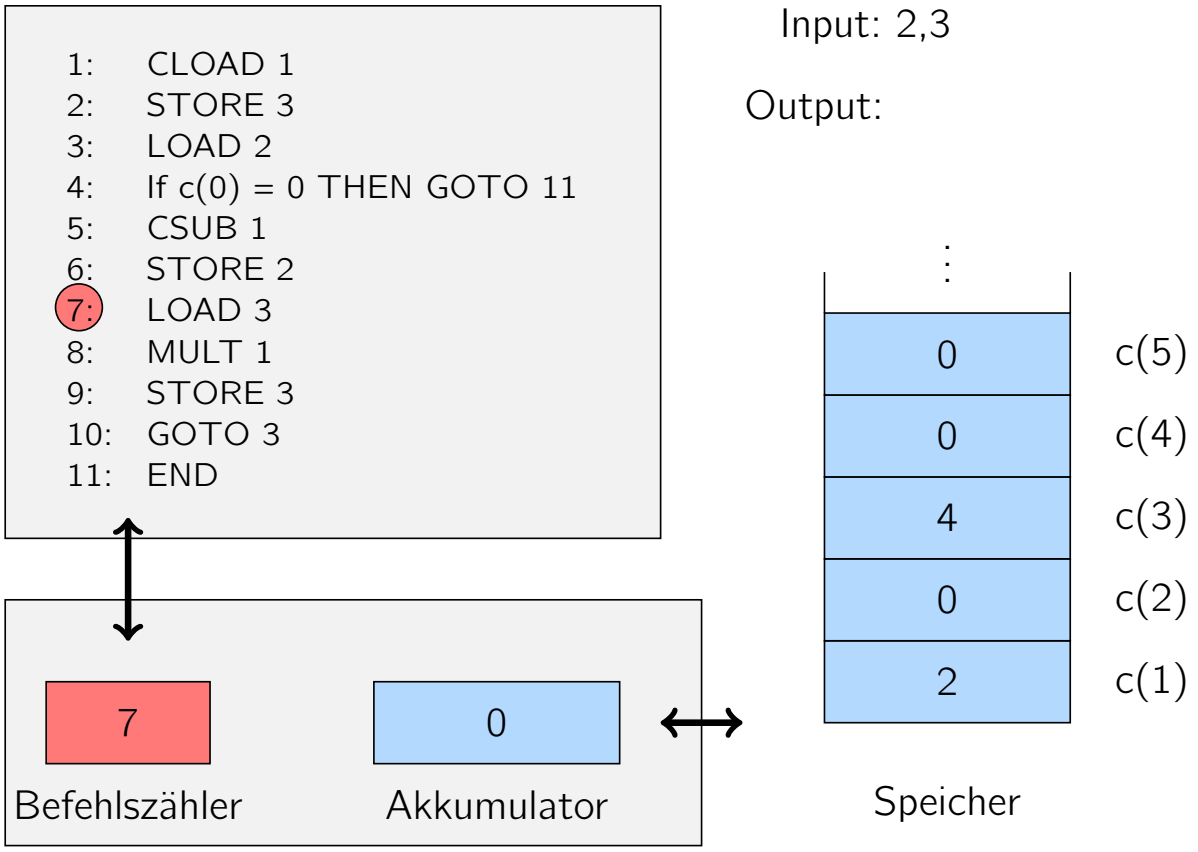
Beispielprogramm für die RAM



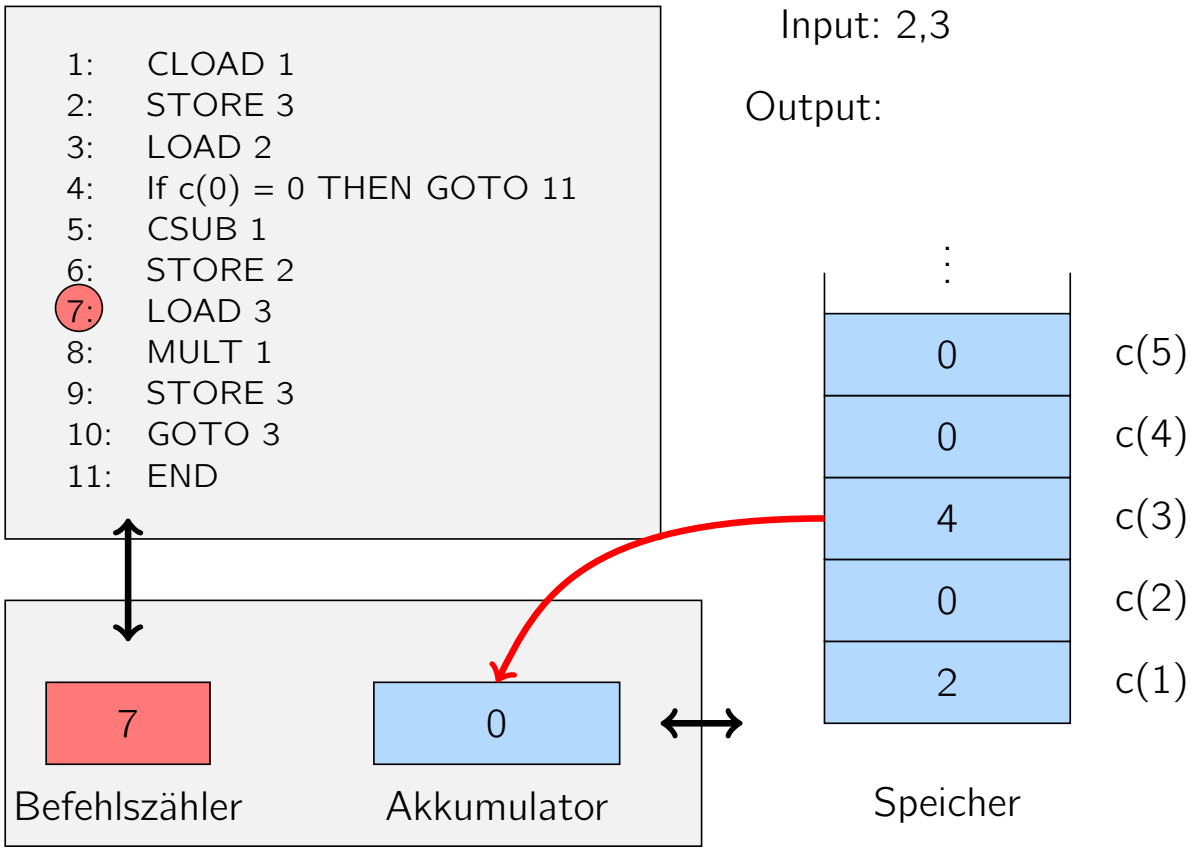
Beispielprogramm für die RAM



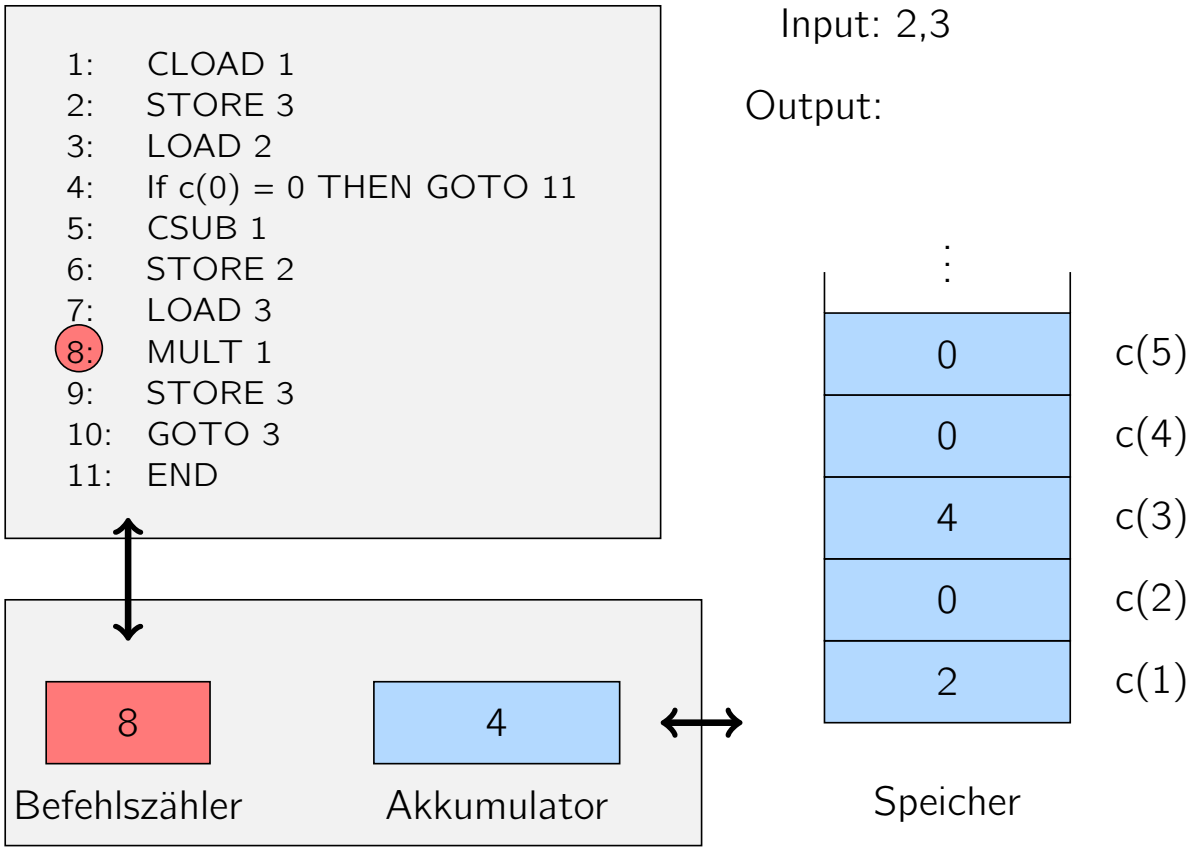
Beispielprogramm für die RAM



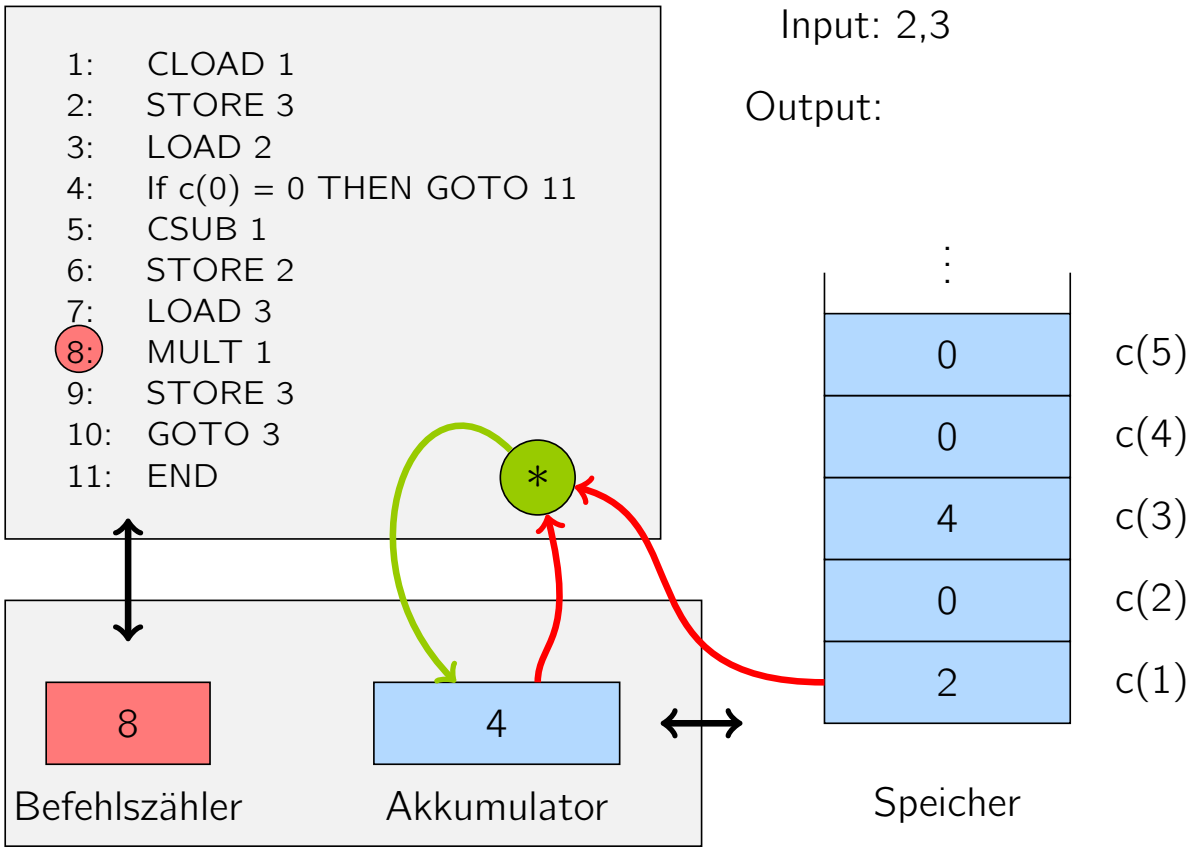
Beispielprogramm für die RAM



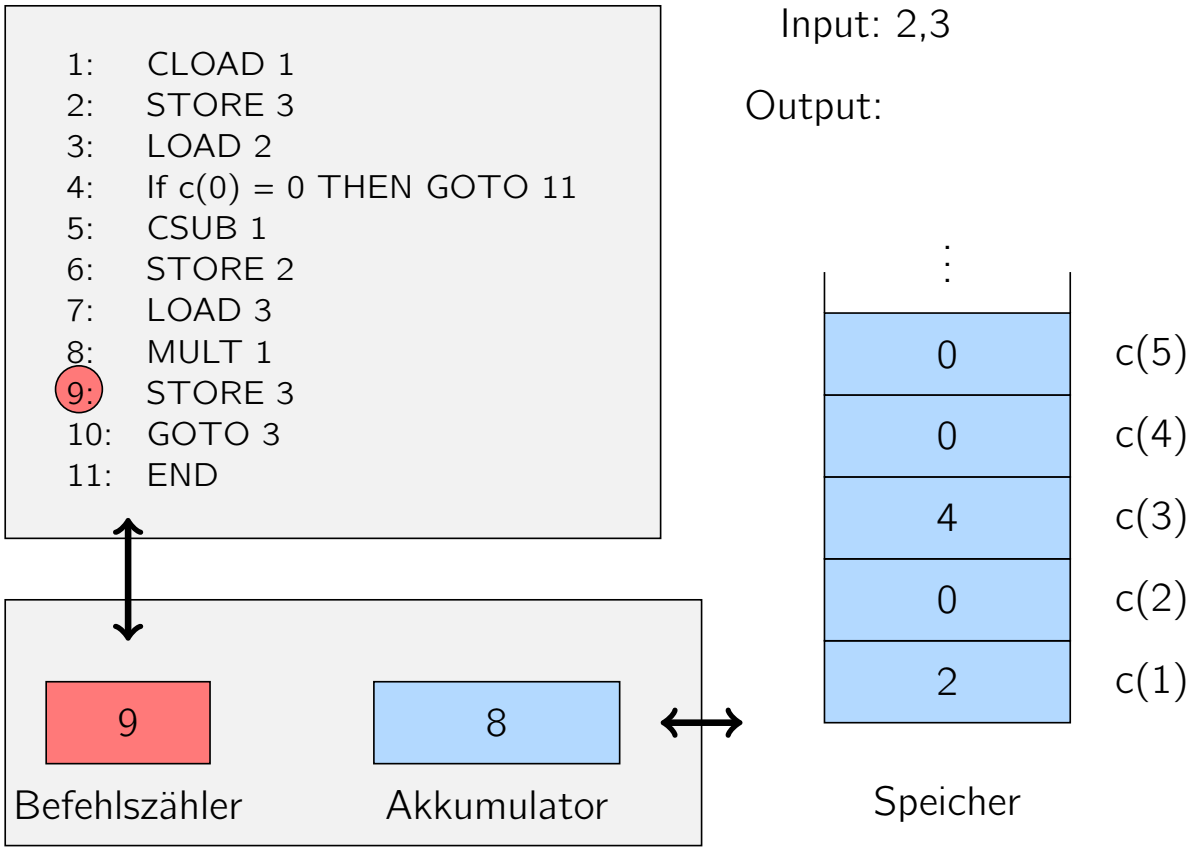
Beispielprogramm für die RAM



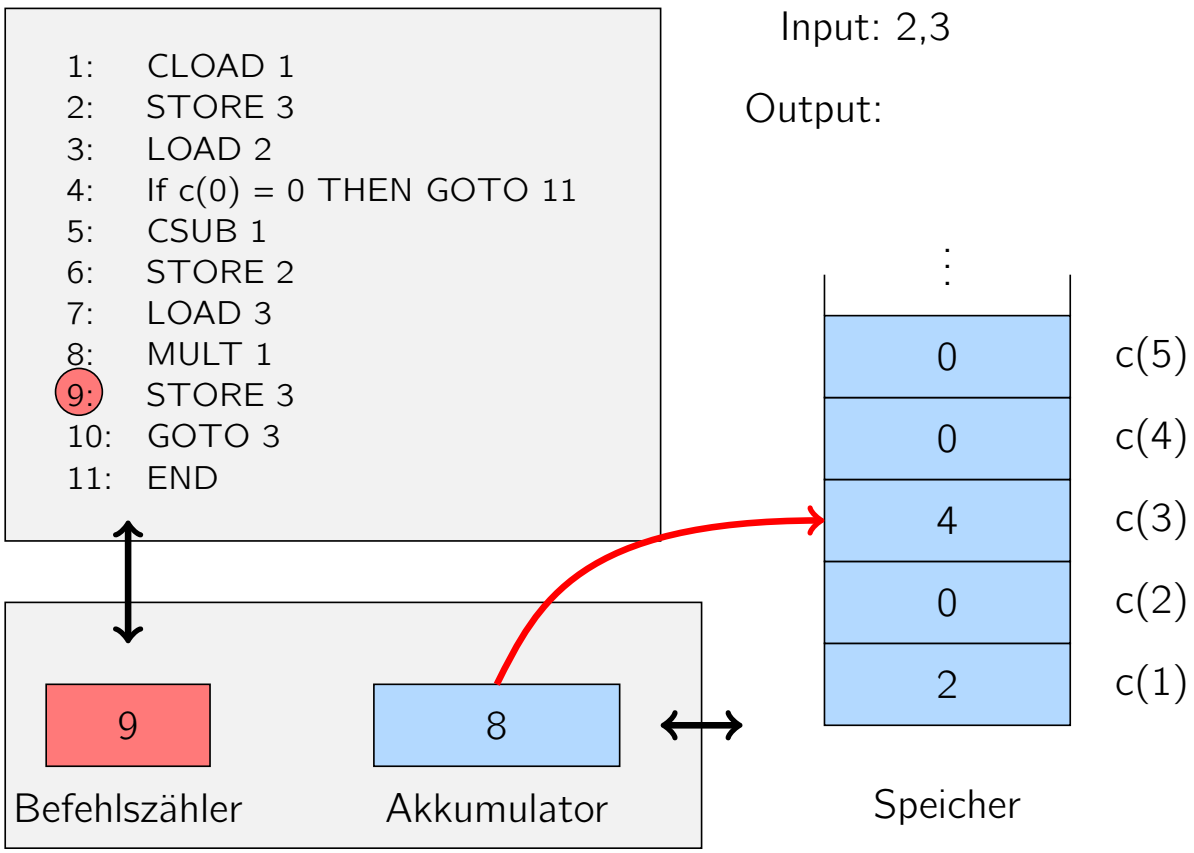
Beispielprogramm für die RAM



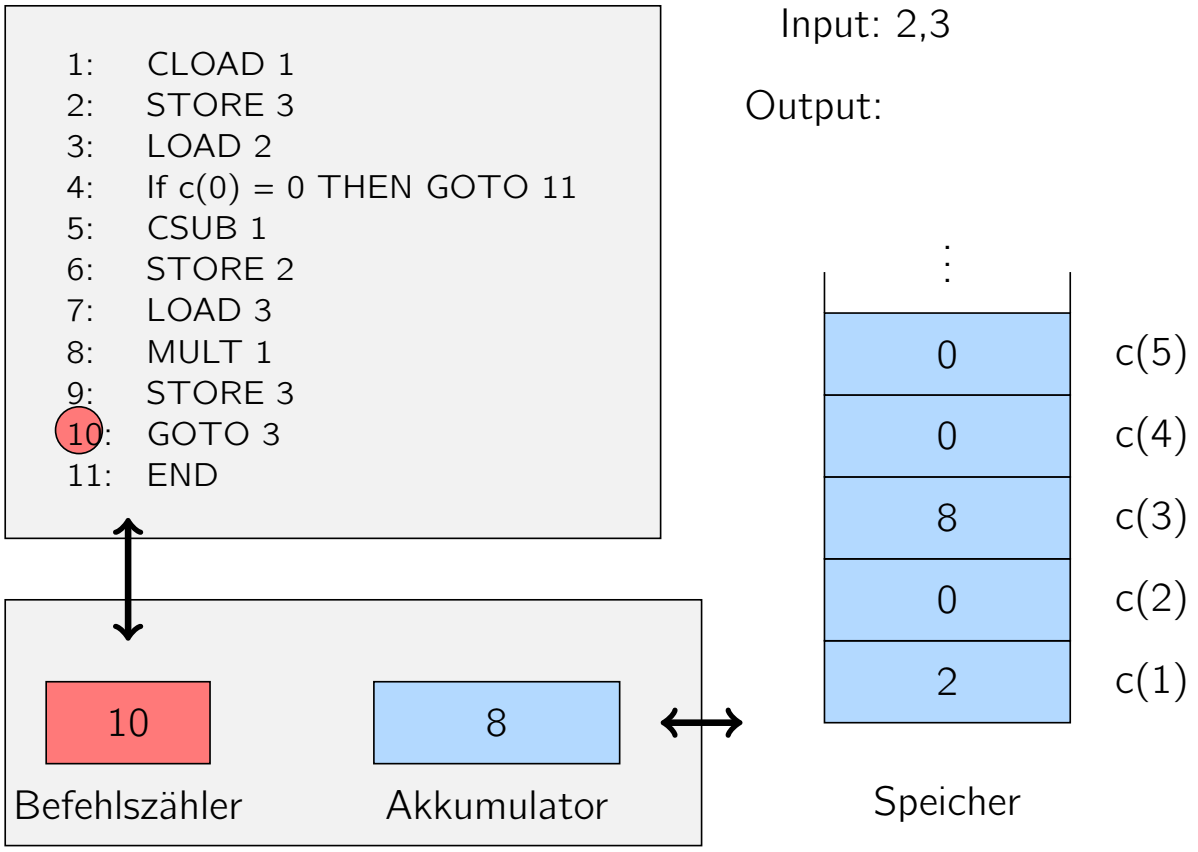
Beispielprogramm für die RAM



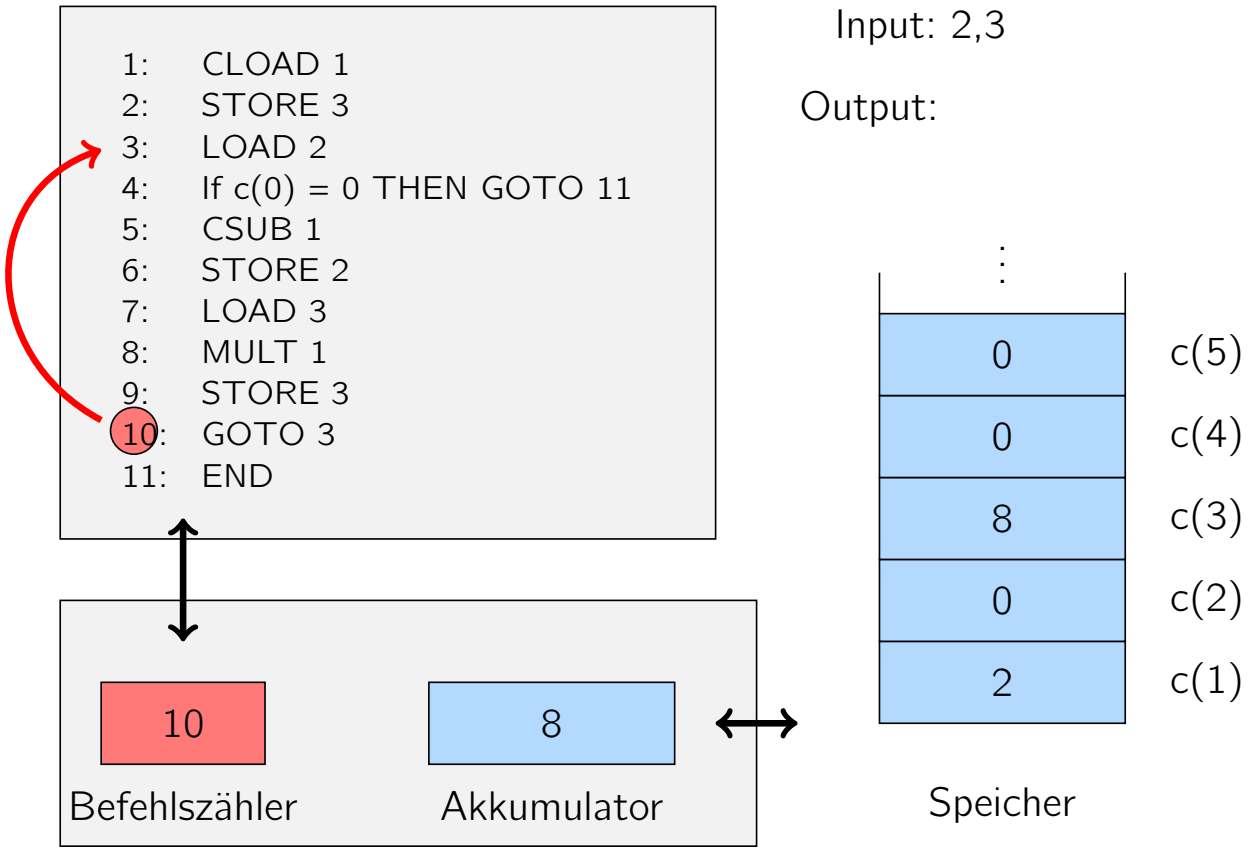
Beispielprogramm für die RAM



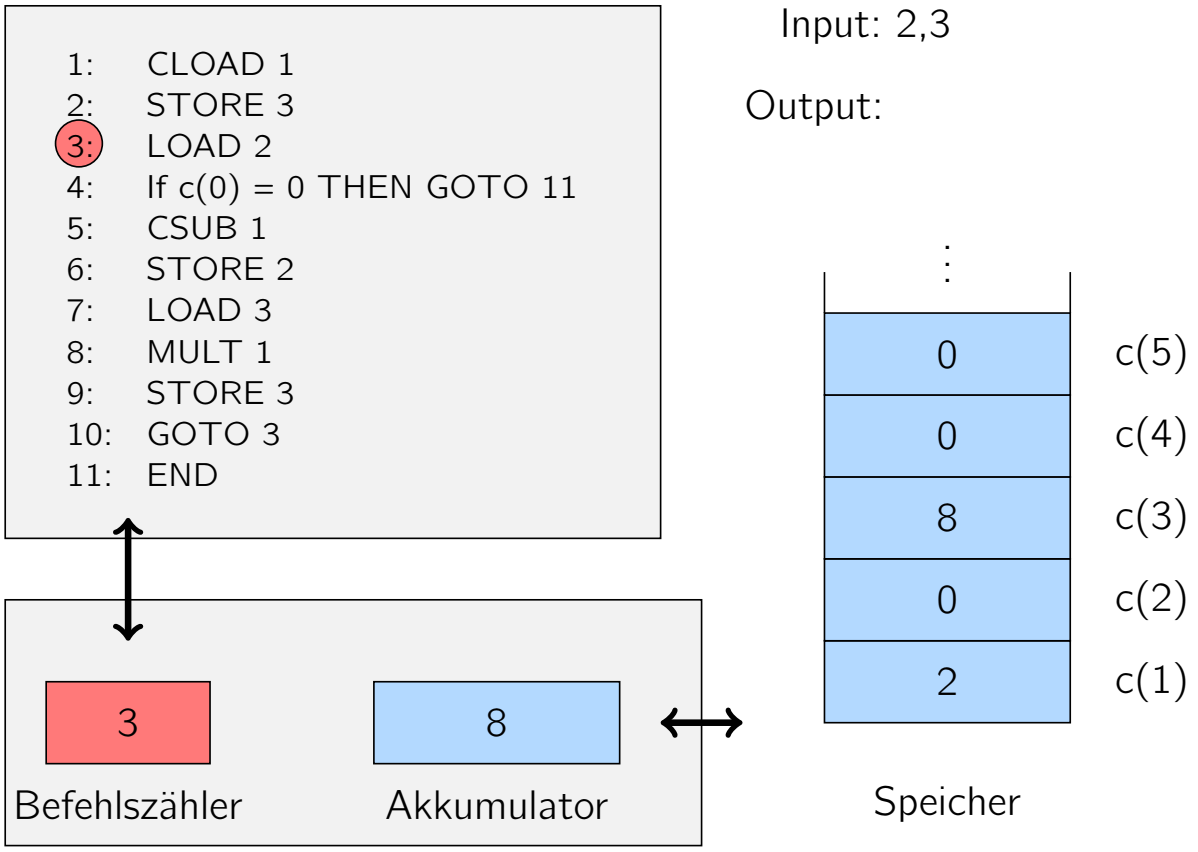
Beispielprogramm für die RAM



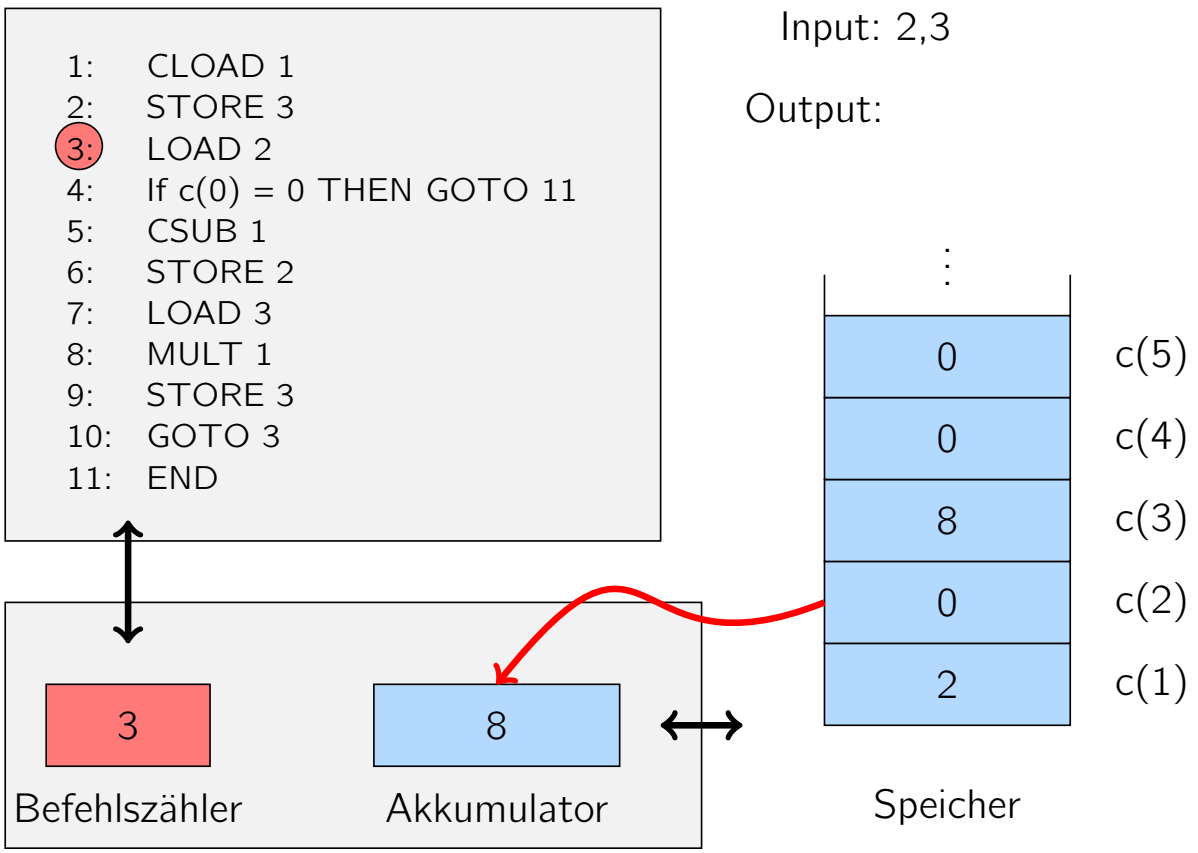
Beispielprogramm für die RAM



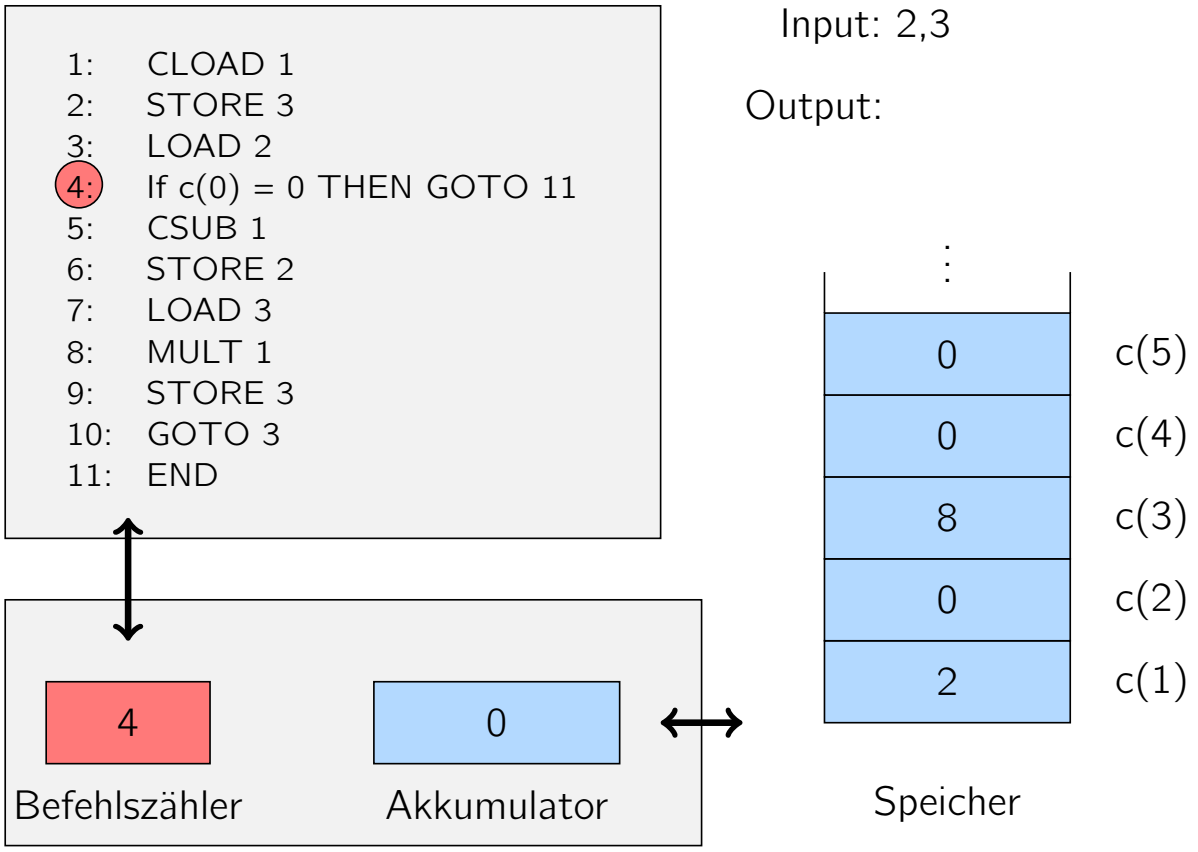
Beispielprogramm für die RAM



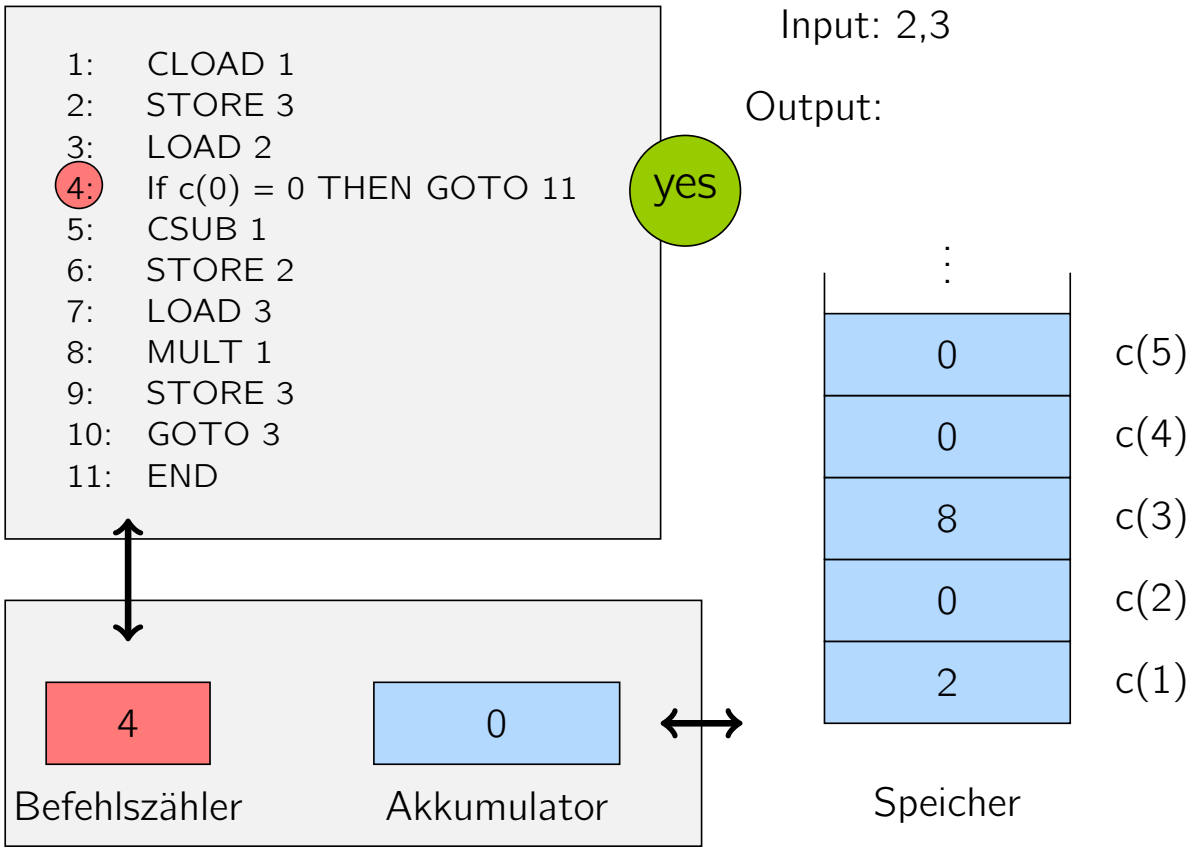
Beispielprogramm für die RAM



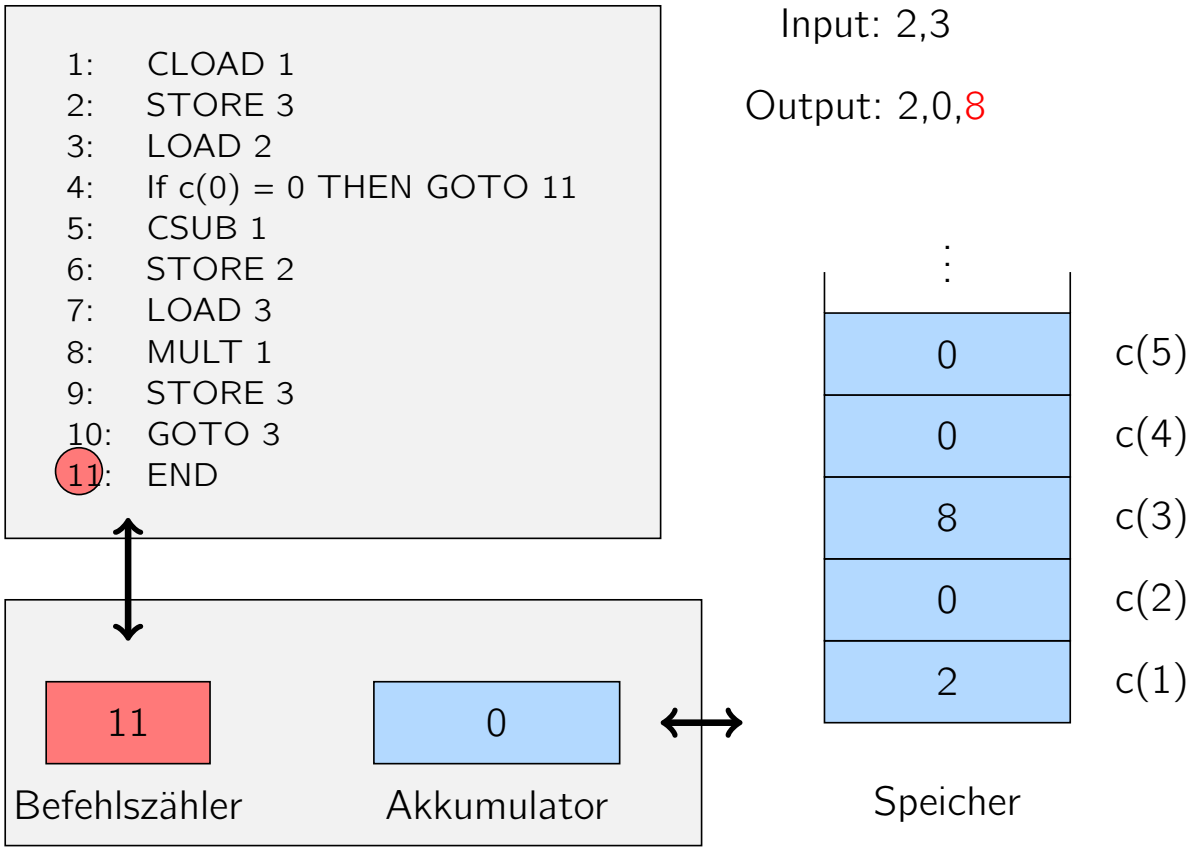
Beispielprogramm für die RAM



Beispielprogramm für die RAM



Beispielprogramm für die RAM



Bemerkungen zur RAM

Auf einer RAM können wir alle Befehle wie beispielsweise Schleifen und Rekursionen, die wir von höheren Programmiersprachen gewohnt sind, realisieren.

Bemerkungen zur RAM

Auf einer RAM können wir alle Befehle wie beispielsweise Schleifen und Rekursionen, die wir von höheren Programmiersprachen gewohnt sind, realisieren.

Modelle für die Rechenzeit

- Uniformes Kostenmaß: Jeder Schritt zählt eine Zeiteinheit.
- Logarithmisches Kostenmaß: Die Laufzeitkosten eines Schrittes sind proportional zur binären Länge der Zahlen in den angesprochenen Registern.

Simulation RAM durch TM

Satz

Für jede im logarithmischen Kostenmaß $t(n)$-zeitbeschränkte RAM R gibt es ein Polynom q und zu diesem eine $O\big(q(n + t(n))\big)$-TM M, die R simuliert.

Simulation RAM durch TM

Satz

Für jede im logarithmischen Kostenmaß $t(n)$-zeitbeschränkte RAM R gibt es ein Polynom q und zu diesem eine $O\big(q(n+t(n))\big)$-TM M, die R simuliert.

Im Beweis können wir für die Simulation eine 2-Band-TM statt einer (1-Band-)TM verwenden. Warum?

Simulation RAM durch TM – Vorbemerkung zum Beweis

- Seien $\alpha, \beta, \gamma \in \mathbb{N}$ geeignet gewählte Konstanten.

Simulation RAM durch TM – Vorbemerkung zum Beweis

- Seien $\alpha, \beta, \gamma \in \mathbb{N}$ geeignet gewählte Konstanten.
- Wir werden zeigen: Die Laufzeit der Simulation der RAM mit Laufzeitschranke $t(n)$ durch eine 2-Band-TM ist nach oben beschränkt durch $t'(n) = \alpha(n + t(n))^\beta$.

Simulation RAM durch TM – Vorbemerkung zum Beweis

- Seien $\alpha, \beta, \gamma \in \mathbb{N}$ geeignet gewählte Konstanten.
- Wir werden zeigen: Die Laufzeit der Simulation der RAM mit Laufzeitschranke $t(n)$ durch eine 2-Band-TM ist nach oben beschränkt durch $t'(n) = \alpha(n + t(n))^\beta$.
- Die 2-Band-TM mit Laufzeitschranke $t'(n)$ kann nun wiederum mit quadratischem Zeitverlust durch eine (1-Band-)TM simuliert werden, also mit einer Laufzeitschranke der Form $t''(n) = \gamma(t'(n))^2$.

Simulation RAM durch TM – Vorbemerkung zum Beweis

- Seien $\alpha, \beta, \gamma \in \mathbb{N}$ geeignet gewählte Konstanten.
- Wir werden zeigen: Die Laufzeit der Simulation der RAM mit Laufzeitschranke $t(n)$ durch eine 2-Band-TM ist nach oben beschränkt durch $t'(n) = \alpha(n + t(n))^\beta$.
- Die 2-Band-TM mit Laufzeitschranke $t'(n)$ kann nun wiederum mit quadratischem Zeitverlust durch eine (1-Band-)TM simuliert werden, also mit einer Laufzeitschranke der Form $t''(n) = \gamma(t'(n))^2$.
- Für die Simulation der RAM auf der (1-Band-)TM ergibt sich somit eine Laufzeitschranke von

$$t''(n) \;=\; \gamma(t'(n))^2$$

Simulation RAM durch TM – Vorbemerkung zum Beweis

- Seien $\alpha, \beta, \gamma \in \mathbb{N}$ geeignet gewählte Konstanten.
- Wir werden zeigen: Die Laufzeit der Simulation der RAM mit Laufzeitschranke $t(n)$ durch eine 2-Band-TM ist nach oben beschränkt durch $t'(n) = \alpha(n + t(n))^\beta$.
- Die 2-Band-TM mit Laufzeitschranke $t'(n)$ kann nun wiederum mit quadratischem Zeitverlust durch eine (1-Band-)TM simuliert werden, also mit einer Laufzeitschranke der Form $t''(n) = \gamma(t'(n))^2$.
- Für die Simulation der RAM auf der (1-Band-)TM ergibt sich somit eine Laufzeitschranke von

$$t''(n) \;=\; \gamma(t'(n))^2 \;=\; \gamma\left(\alpha(n + t(n))^\beta\right)^2$$

Simulation RAM durch TM – Vorbemerkung zum Beweis

- Seien $\alpha, \beta, \gamma \in \mathbb{N}$ geeignet gewählte Konstanten.
- Wir werden zeigen: Die Laufzeit der Simulation der RAM mit Laufzeitschranke $t(n)$ durch eine 2-Band-TM ist nach oben beschränkt durch $t'(n) = \alpha(n + t(n))^\beta$.
- Die 2-Band-TM mit Laufzeitschranke $t'(n)$ kann nun wiederum mit quadratischem Zeitverlust durch eine (1-Band-)TM simuliert werden, also mit einer Laufzeitschranke der Form $t''(n) = \gamma(t'(n))^2$.
- Für die Simulation der RAM auf der (1-Band-)TM ergibt sich somit eine Laufzeitschranke von

$$t''(n) \;=\; \gamma(t'(n))^2 \;=\; \gamma\left(\alpha(n + t(n))^\beta\right)^2 \;=\; \gamma\alpha^2 \cdot (n + t(n))^{2\beta}.$$

Simulation RAM durch TM – Vorbemerkung zum Beweis

- Seien $\alpha, \beta, \gamma \in \mathbb{N}$ geeignet gewählte Konstanten.
- Wir werden zeigen: Die Laufzeit der Simulation der RAM mit Laufzeitschranke $t(n)$ durch eine 2-Band-TM ist nach oben beschränkt durch $t'(n) = \alpha(n + t(n))^\beta$.
- Die 2-Band-TM mit Laufzeitschranke $t'(n)$ kann nun wiederum mit quadratischem Zeitverlust durch eine (1-Band-)TM simuliert werden, also mit einer Laufzeitschranke der Form $t''(n) = \gamma(t'(n))^2$.
- Für die Simulation der RAM auf der (1-Band-)TM ergibt sich somit eine Laufzeitschranke von

$$t''(n) \;=\; \gamma(t'(n))^2 \;=\; \gamma\left(\alpha(n + t(n))^\beta\right)^2 \;=\; \gamma\alpha^2 \cdot (n + t(n))^{2\beta}.$$

- Diese Laufzeitschranke ist polynomiell in $n + t(n)$, weil sowohl der Term $\gamma\alpha^2$ als auch der Term 2β konstant sind.

Simulation RAM durch TM – Vorbemerkung zum Beweis

Beobachtung

Die Klasse der Polynome ist unter Hintereinanderausführung abgeschlossen.

Mit anderen Worten:
Wenn sowohl die Abbildung $x \mapsto p(x)$ als auch die Abbildung $x \mapsto q(x)$ ein Polynom ist, dann ist auch die Abbildung $x \mapsto q(p(x))$ ein Polynom.

Simulation RAM durch TM – Vorbemerkung zum Beweis

Beobachtung

Die Klasse der Polynome ist unter Hintereinanderausführung abgeschlossen.

Mit anderen Worten:
Wenn sowohl die Abbildung $x \mapsto p(x)$ als auch die Abbildung $x \mapsto q(x)$ ein Polynom ist, dann ist auch die Abbildung $x \mapsto q(p(x))$ ein Polynom.

Deshalb können wir eine *konstante Anzahl* von Simulationen, deren Zeitverlust jeweils polynomiell nach oben beschränkt ist, ineinander schachteln und erhalten dadurch wiederum eine Simulation mit polynomiell beschränktem Zeitverlust.

Simulation RAM durch TM – Beweis

Beweis des Satzes

- Wir verwenden eine 2-Band-TM, die die RAM schrittweise simuliert.

Simulation RAM durch TM – Beweis

Beweis des Satzes

- Wir verwenden eine 2-Band-TM, die die RAM schrittweise simuliert.
- Das RAM-Programm P bestehe aus p Programmzeilen.

Simulation RAM durch TM – Beweis

Beweis des Satzes

- Wir verwenden eine 2-Band-TM, die die RAM schrittweise simuliert.
- Das RAM-Programm P bestehe aus p Programmzeilen.
- Für jede Programmzeile schreiben wir ein TM-Unterprogramm. Sei M_i das Unterprogramm für Programmzeile i, $1 \leq i \leq p$.

Simulation RAM durch TM – Beweis

Beweis des Satzes

- Wir verwenden eine 2-Band-TM, die die RAM schrittweise simuliert.
- Das RAM-Programm P bestehe aus p Programmzeilen.
- Für jede Programmzeile schreiben wir ein TM-Unterprogramm. Sei M_i das Unterprogramm für Programmzeile i, $1 \leq i \leq p$.
- Außerdem spezifizieren wir ein Unterprogramm M_0 für die Initialisierung der TM und M_{p+1} für die Aufbereitung der Ausgabe des Ergebnisses.

Simulation RAM durch TM – Beweis

Abspeichern der *RAM-Konfiguration* auf der TM:

Simulation RAM durch TM – Beweis

Abspeichern der *RAM-Konfiguration* auf der TM:

- Den Befehlszähler kann die TM im Zustand abspeichern, da die Länge des RAM-Programms konstant ist.

Simulation RAM durch TM – Beweis

Abspeichern der *RAM-Konfiguration* auf der TM:

- Den Befehlszähler kann die TM im Zustand abspeichern, da die Länge des RAM-Programms konstant ist.
- Die Registerinhalte werden wie folgt auf Band 2 abgespeichert:

$$\#\#0\#\,\mathsf{bin}(c(0))\#\#\,\mathsf{bin}(i_1)\#\,\mathsf{bin}(c(i_1))\#\#\ldots$$

$$\ldots\#\#\,\mathsf{bin}(i_m)\#\,\mathsf{bin}(c(i_m))\#\#\#,$$

wobei $0, i_1, \ldots, i_m$ die Indizes der benutzten Register sind.

Simulation RAM durch TM – Beweis

Abspeichern der *RAM-Konfiguration* auf der TM:

- Den Befehlszähler kann die TM im Zustand abspeichern, da die Länge des RAM-Programms konstant ist.
- Die Registerinhalte werden wie folgt auf Band 2 abgespeichert:

$$\#\#0\#\,\mathsf{bin}(c(0))\#\#\,\mathsf{bin}(i_1)\#\,\mathsf{bin}(c(i_1))\#\#\ldots$$

$$\ldots\#\#\,\mathsf{bin}(i_m)\#\,\mathsf{bin}(c(i_m))\#\#\#,$$

wobei $0, i_1, \ldots, i_m$ die Indizes der benutzten Register sind.

Beobachtung

Der Platzbedarf auf Band 2 ist durch $O(n + t(n))$ beschränkt, weil die RAM für jedes neue Bit, das sie erzeugt, mindestens eine Zeiteinheit benötigt.

Simulation RAM durch TM – Beweis

Rechenschritt für Rechenschritt simuliert die TM nun die Konfigurationsveränderungen der RAM.

Simulation RAM durch TM – Beweis

Rechenschritt für Rechenschritt simuliert die TM nun die Konfigurationsveränderungen der RAM.

Dazu ruft die TM das im Programmzähler b angegebene Unterprogramm M_b auf.

Das Unterprogramm M_b

- kopiert den Inhalt der in Programmzeile b angesprochenen Register auf Band 1,
- führt die notwendigen Operationen auf diesen Registerinhalten durch,
- kopiert dann das Ergebnis in das in Zeile b angegebene Register auf Band 2 zurück, und
- aktualisiert zuletzt den Programmzähler b.

Simulation RAM durch TM – Beweis

Laufzeitanalyse:

Die Initialisierung erfordert Zeit $O(n)$.

Simulation RAM durch TM – Beweis

Laufzeitanalyse:

Die Initialisierung erfordert Zeit $O(n)$.

Alle Unterprogramme haben eine Laufzeit, die polynomiell in der Länge des aktuellen Wortes auf Band 2 beschränkt ist, also eine Laufzeit polynomiell in $n + t(n)$.

Simulation RAM durch TM – Beweis

Laufzeitanalyse:

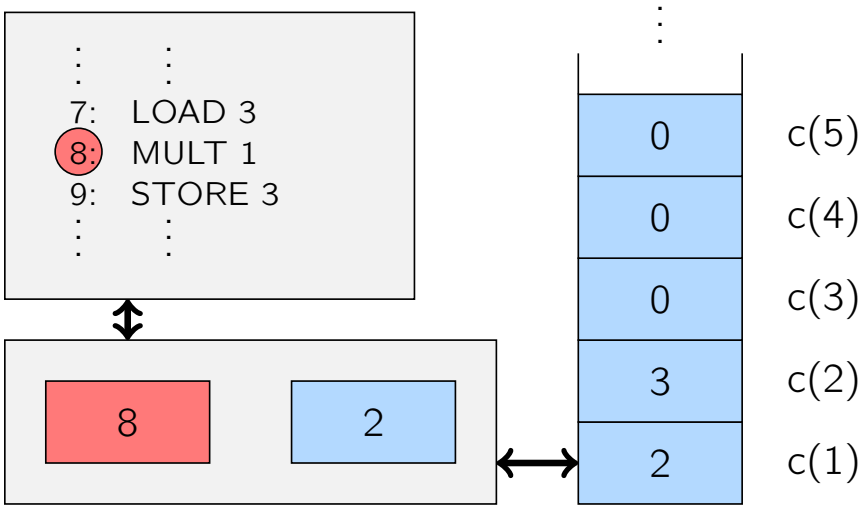
Die Initialisierung erfordert Zeit $O(n)$.

Alle Unterprogramme haben eine Laufzeit, die polynomiell in der Länge des aktuellen Wortes auf Band 2 beschränkt ist, also eine Laufzeit polynomiell in $n + t(n)$.

Somit ist auch die Gesamtlaufzeit der Simulation polynomiell in $n + t(n)$ beschränkt. □

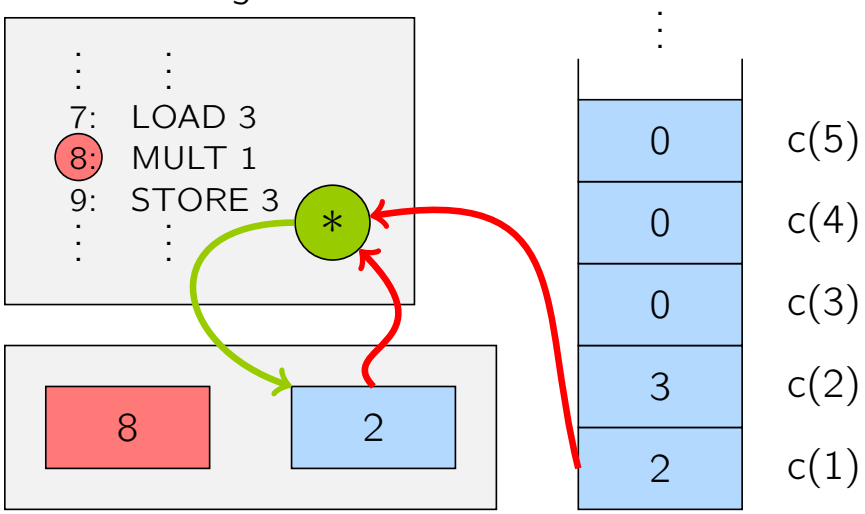
Simulation RAM durch TM – Illustration

simulierte Registermaschine M



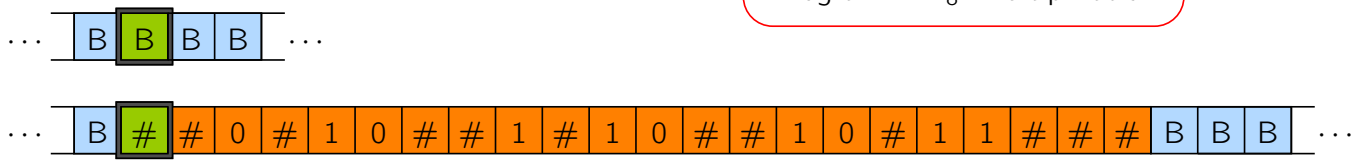
Simulation RAM durch TM – Illustration

simulierte Registermaschine M



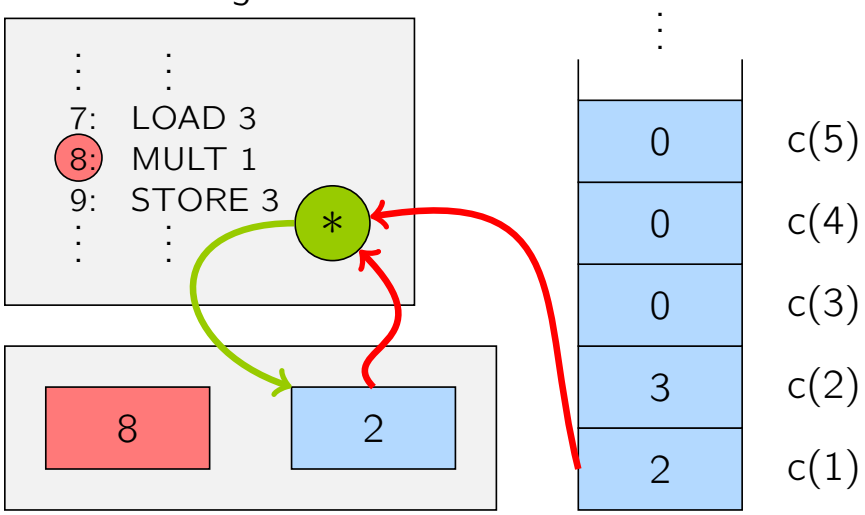
simulierende Turingmaschine

Programm M_8: Multiplikation



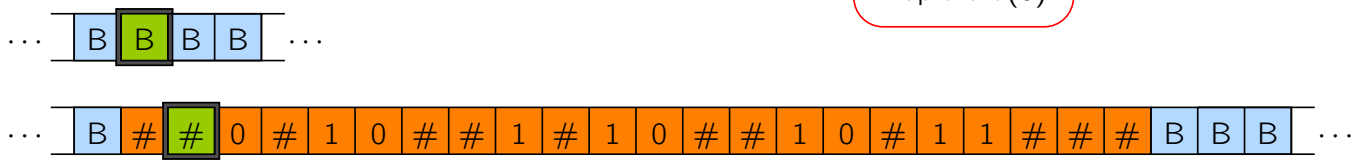
Simulation RAM durch TM – Illustration

simulierte Registermaschine M



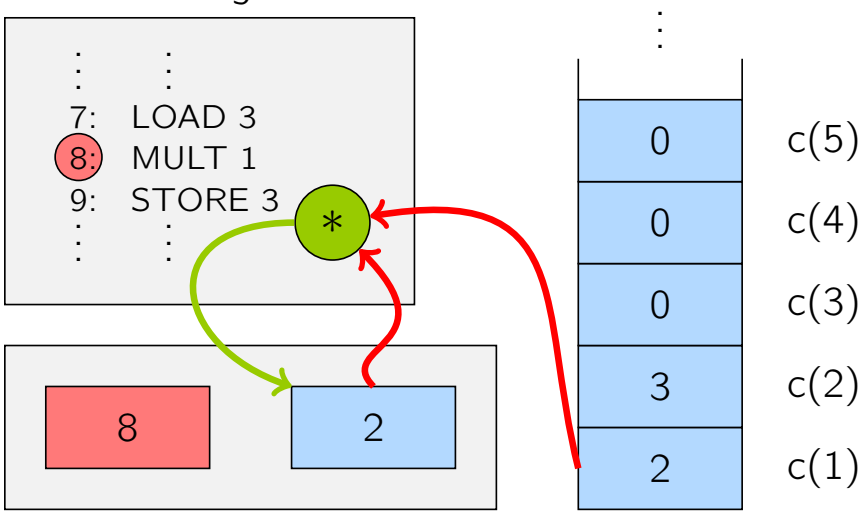
simulierende Turingmaschine

Kopiere $c(0)$

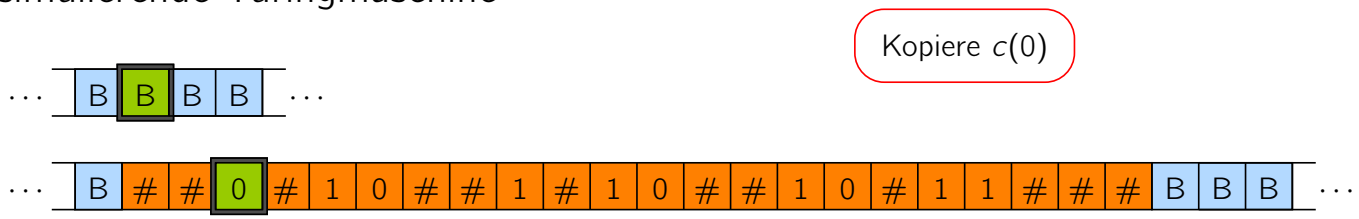


Simulation RAM durch TM – Illustration

simulierte Registermaschine M

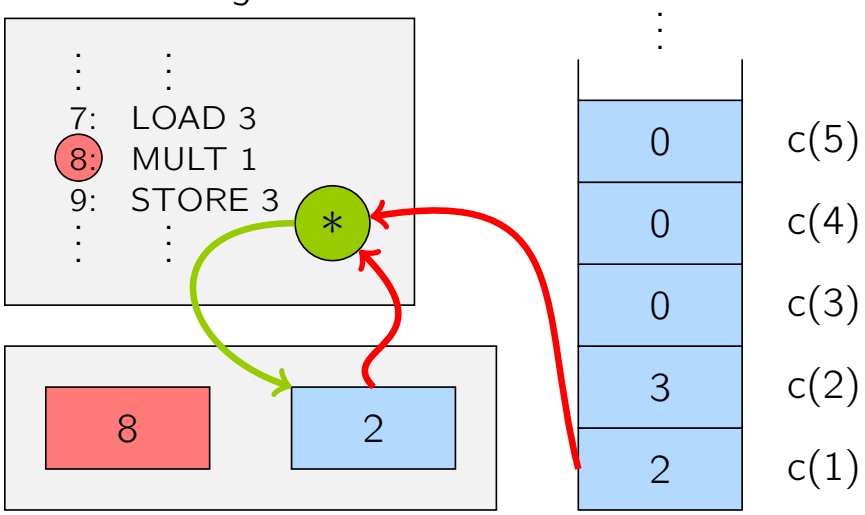


simulierende Turingmaschine

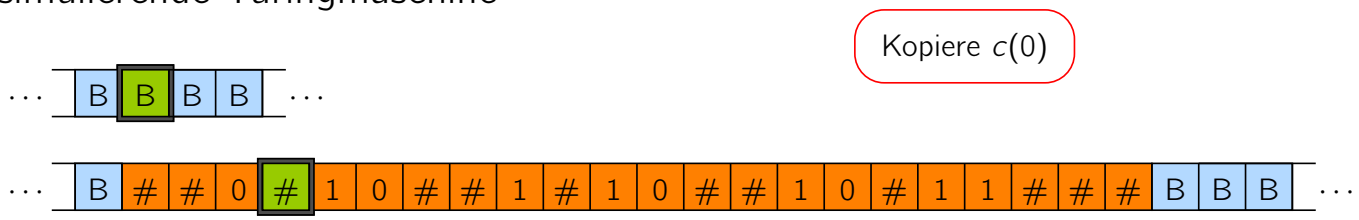


Simulation RAM durch TM – Illustration

simulierte Registermaschine M

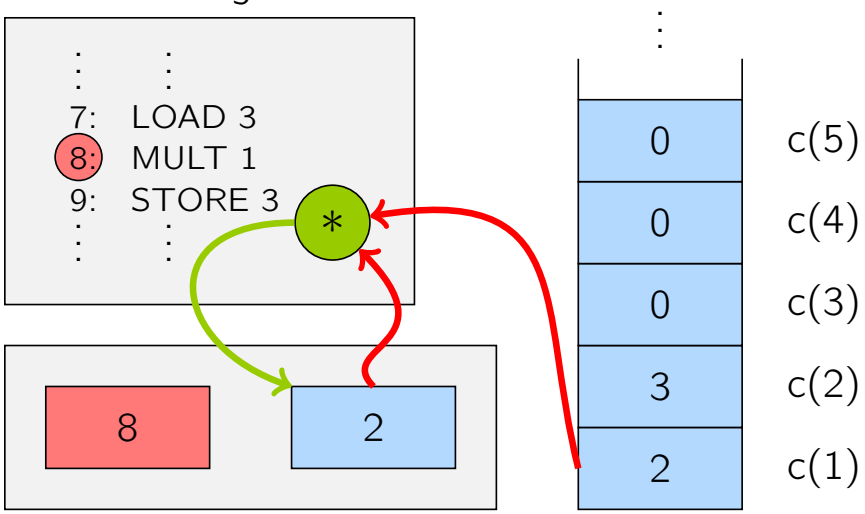


simulierende Turingmaschine



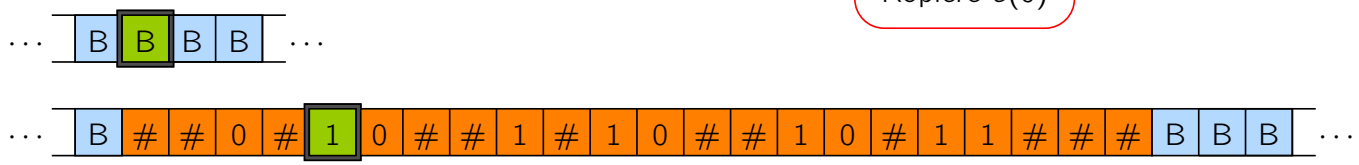
Simulation RAM durch TM – Illustration

simulierte Registermaschine M



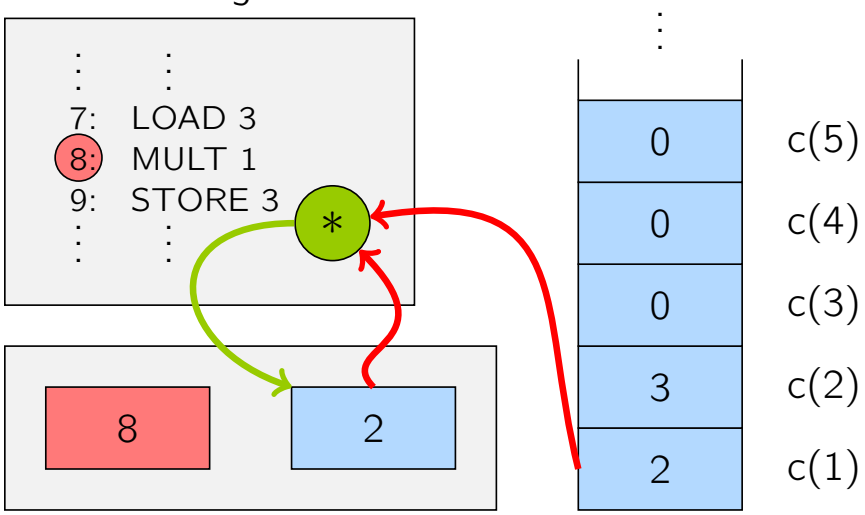
simulierende Turingmaschine

Kopiere $c(0)$



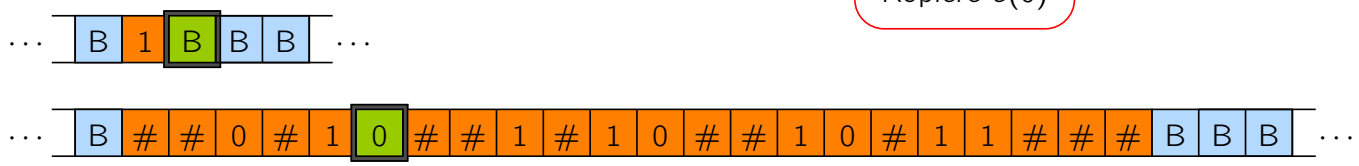
Simulation RAM durch TM – Illustration

simulierte Registermaschine M



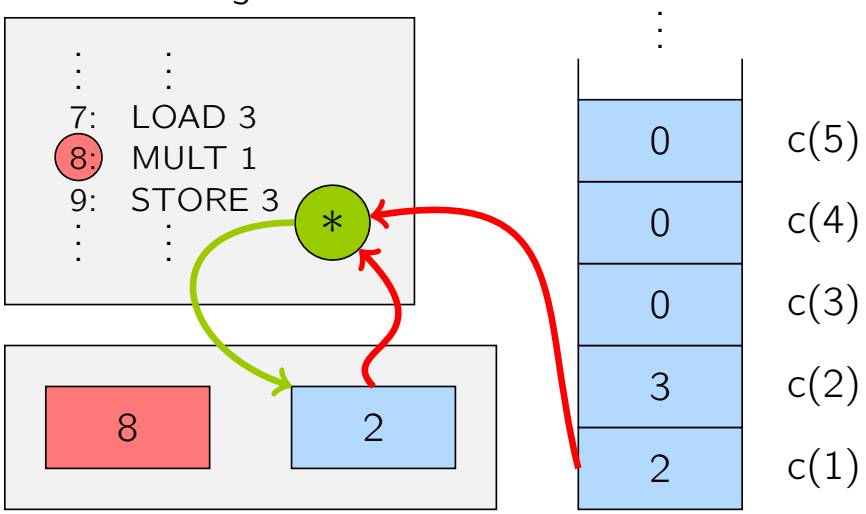
simulierende Turingmaschine

Kopiere $c(0)$

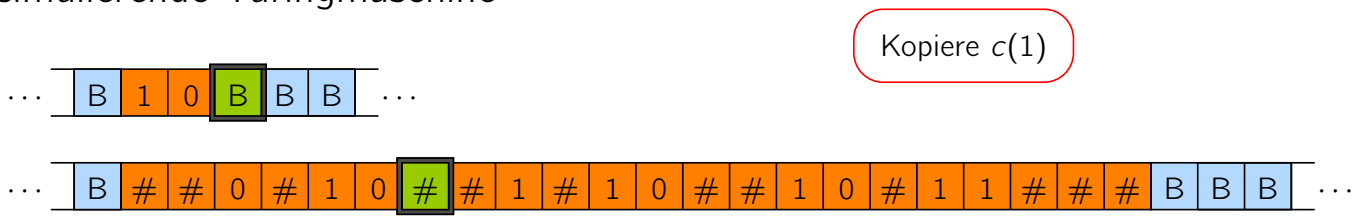


Simulation RAM durch TM – Illustration

simulierte Registermaschine M

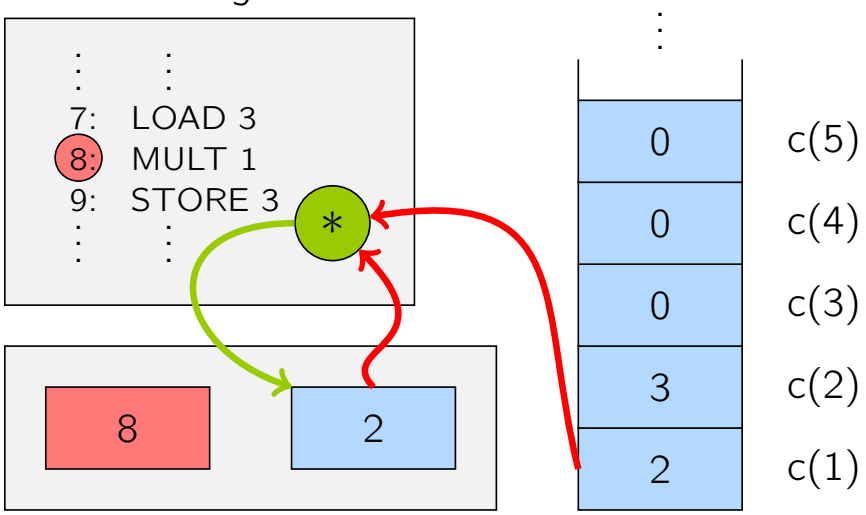


simulierende Turingmaschine

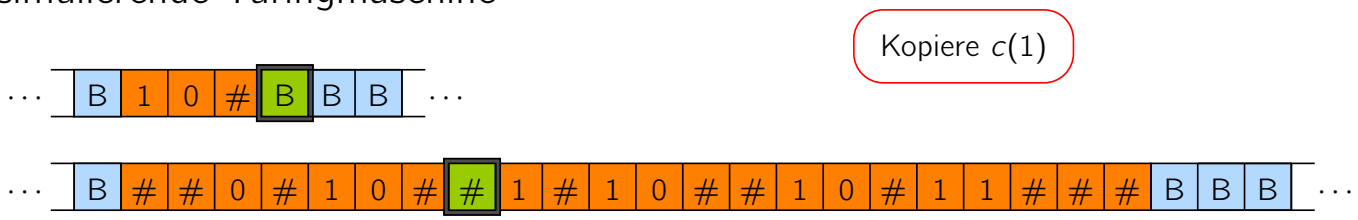


Simulation RAM durch TM – Illustration

simulierte Registermaschine M

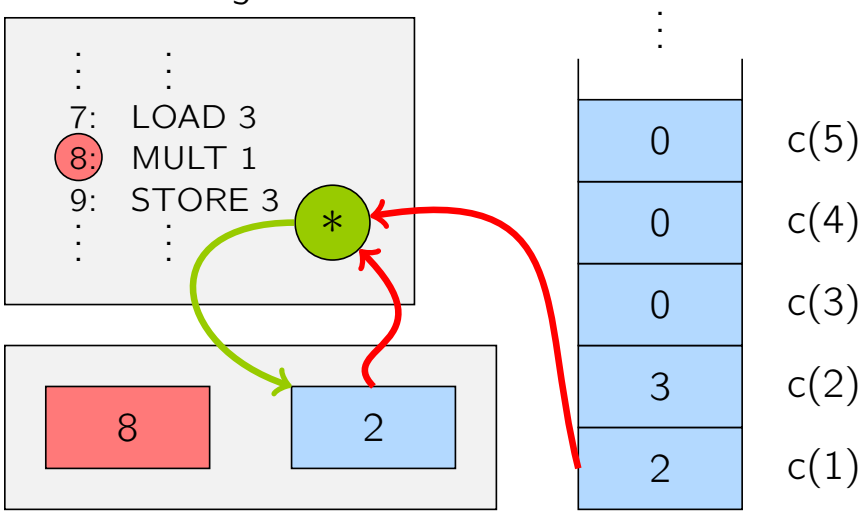


simulierende Turingmaschine



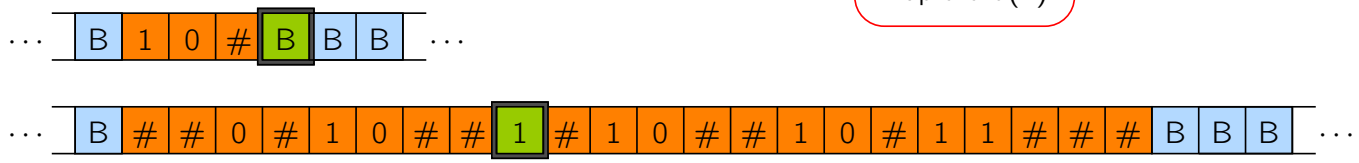
Simulation RAM durch TM – Illustration

simulierte Registermaschine M



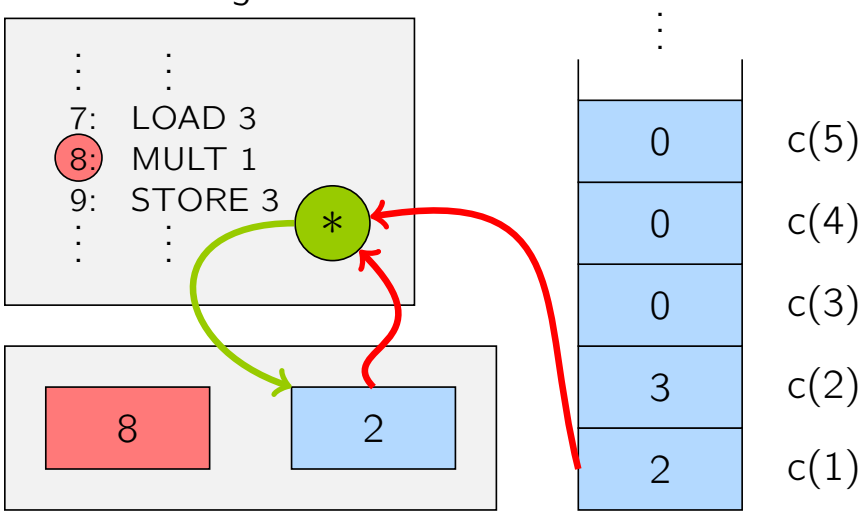
simulierende Turingmaschine

Kopiere $c(1)$



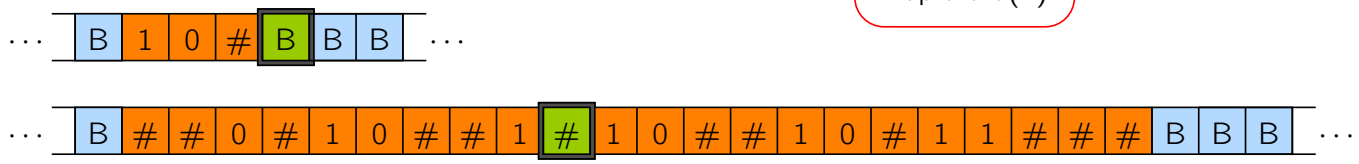
Simulation RAM durch TM – Illustration

simulierte Registermaschine M



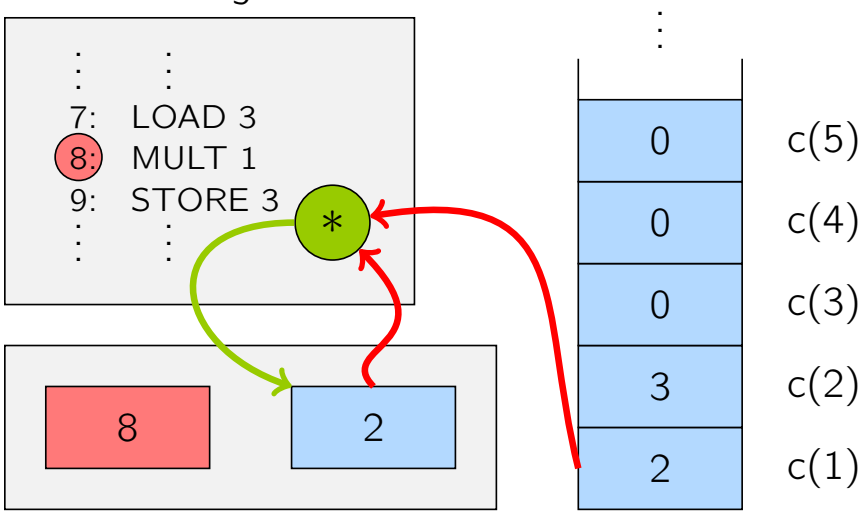
simulierende Turingmaschine

Kopiere $c(1)$



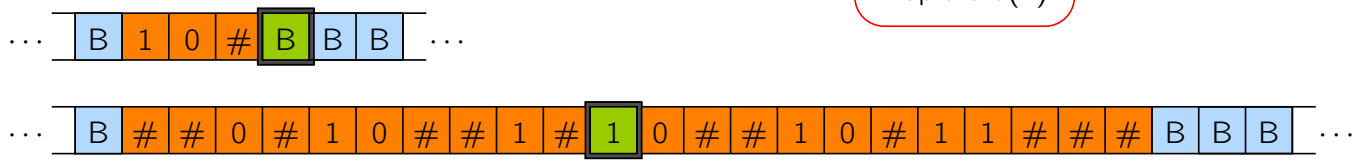
Simulation RAM durch TM – Illustration

simulierte Registermaschine M



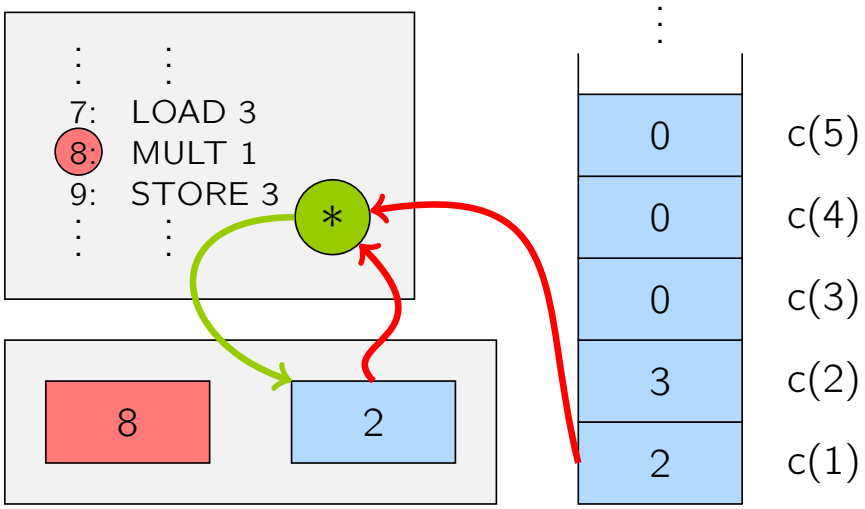
simulierende Turingmaschine

Kopiere $c(1)$

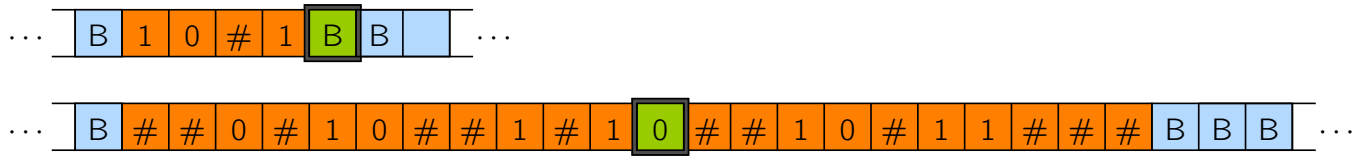


Simulation RAM durch TM – Illustration

simulierte Registermaschine M

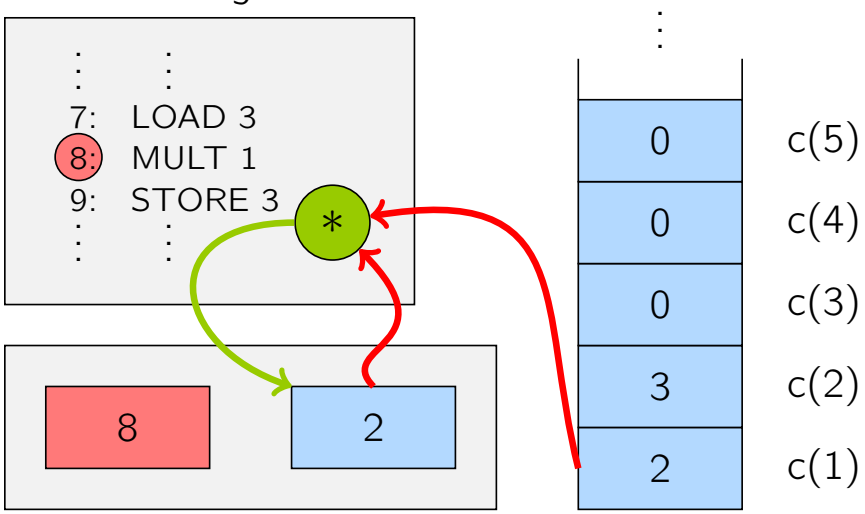


simulierende Turingmaschine

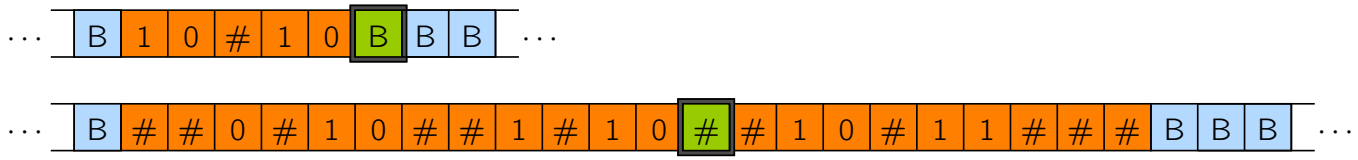


Simulation RAM durch TM – Illustration

simulierte Registermaschine M

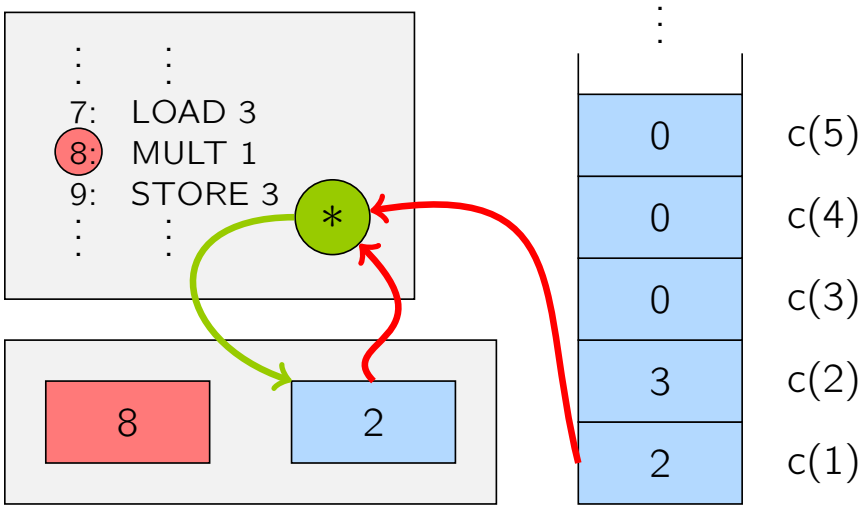


simulierende Turingmaschine



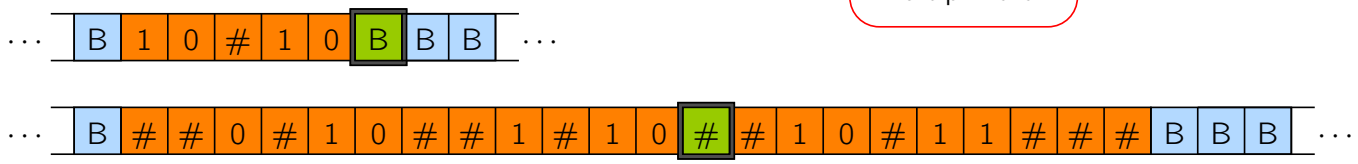
Simulation RAM durch TM – Illustration

simulierte Registermaschine M



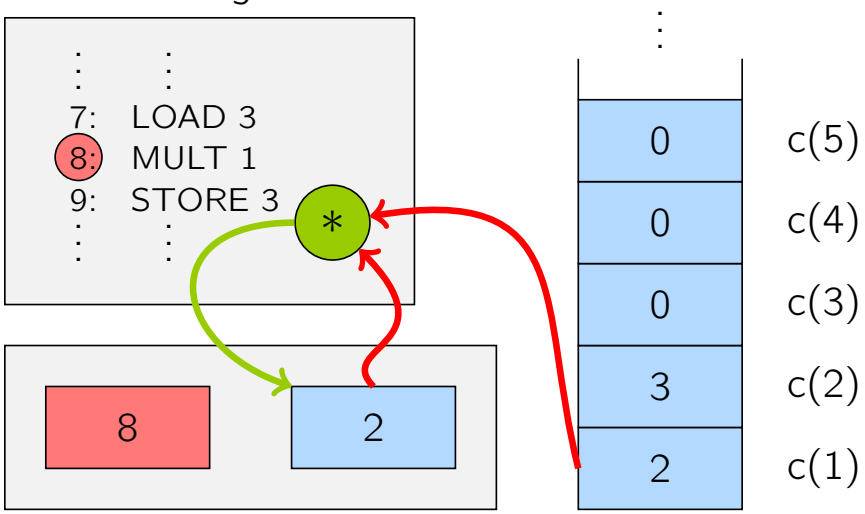
simulierende Turingmaschine

Multiplizieren

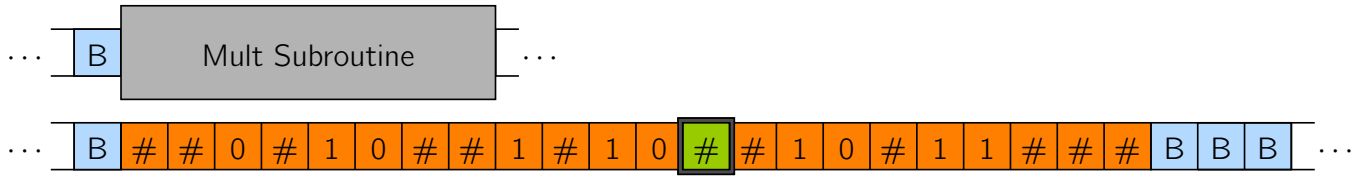


Simulation RAM durch TM – Illustration

simulierte Registermaschine M

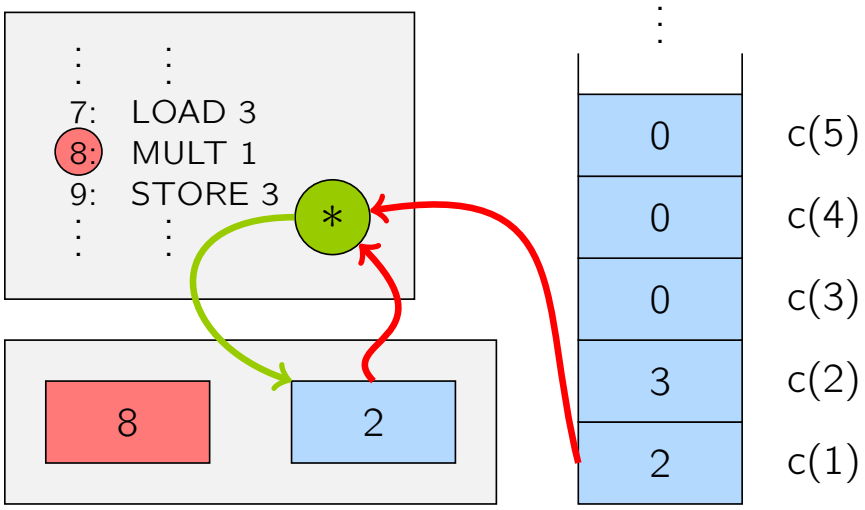


simulierende Turingmaschine

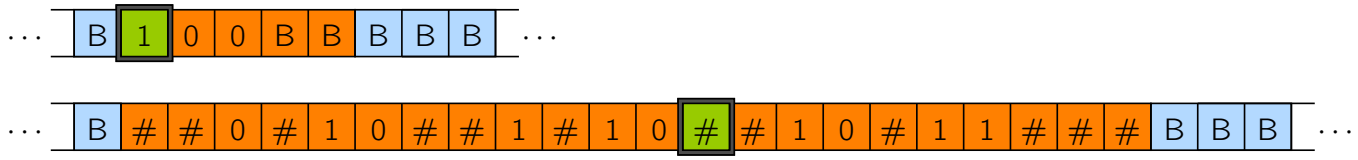


Simulation RAM durch TM – Illustration

simulierte Registermaschine M

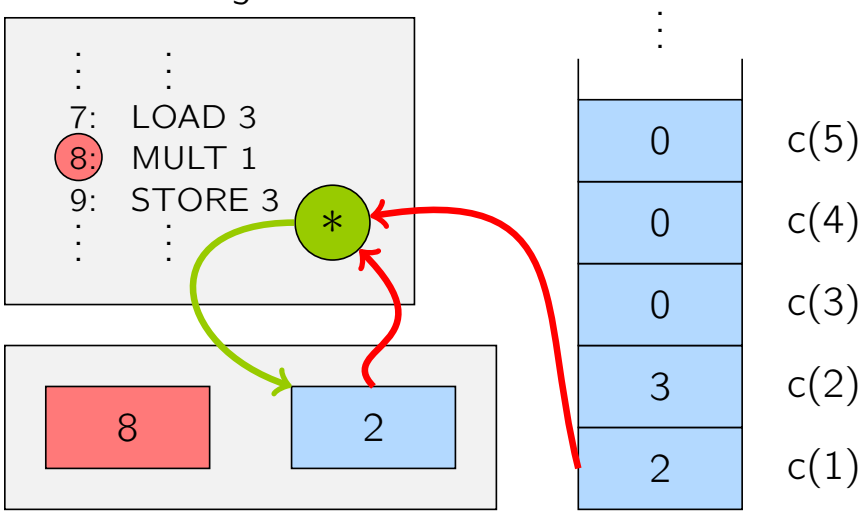


simulierende Turingmaschine

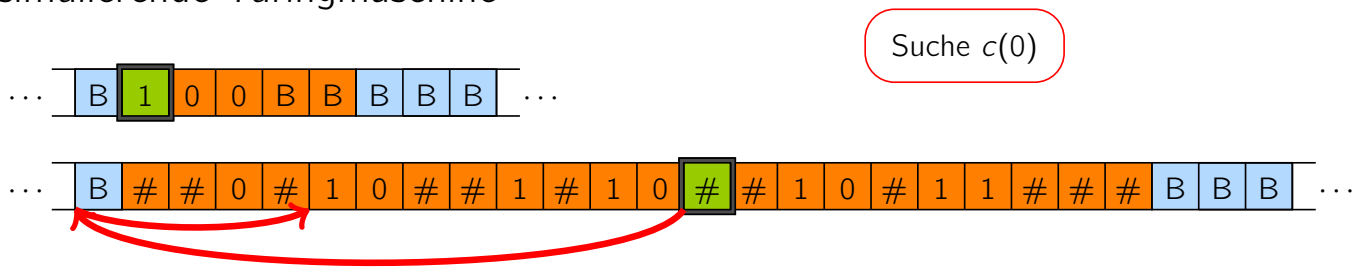


Simulation RAM durch TM – Illustration

simulierte Registermaschine M

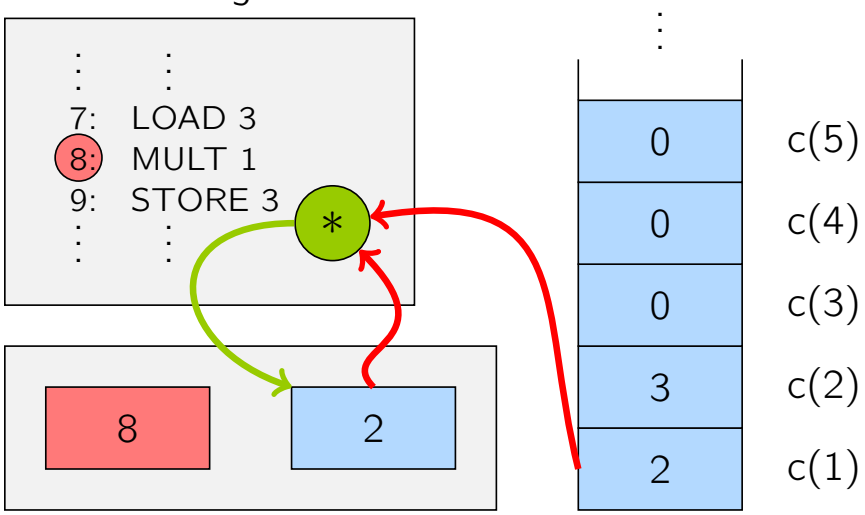


simulierende Turingmaschine

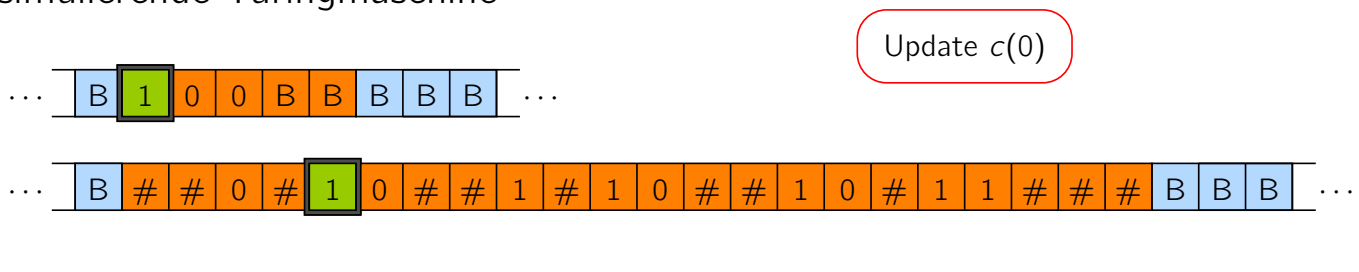


Simulation RAM durch TM – Illustration

simulierte Registermaschine M

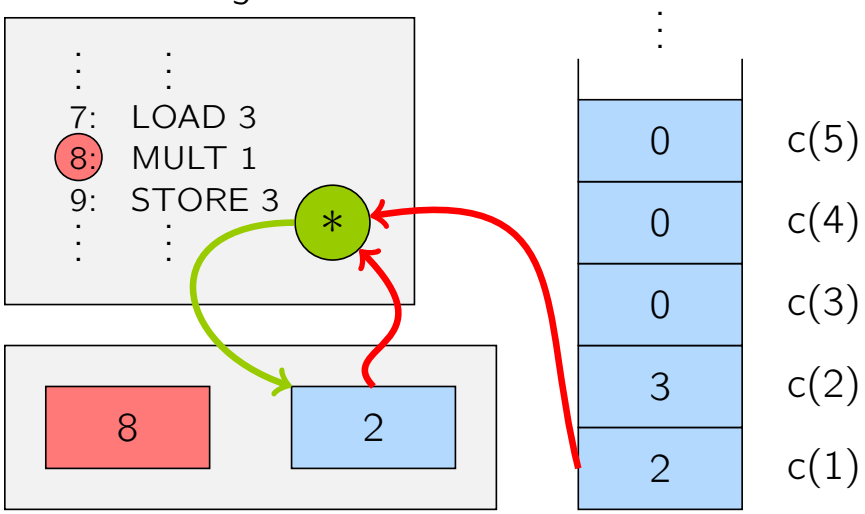


simulierende Turingmaschine



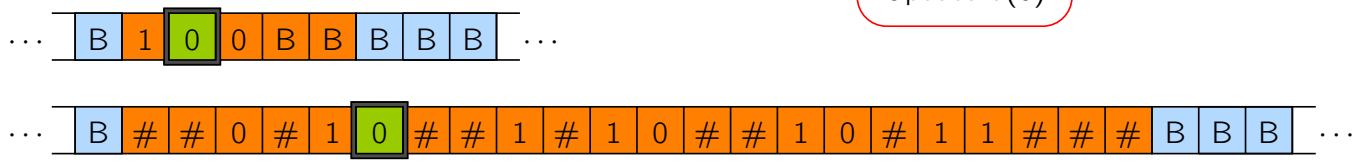
Simulation RAM durch TM – Illustration

simulierte Registermaschine M



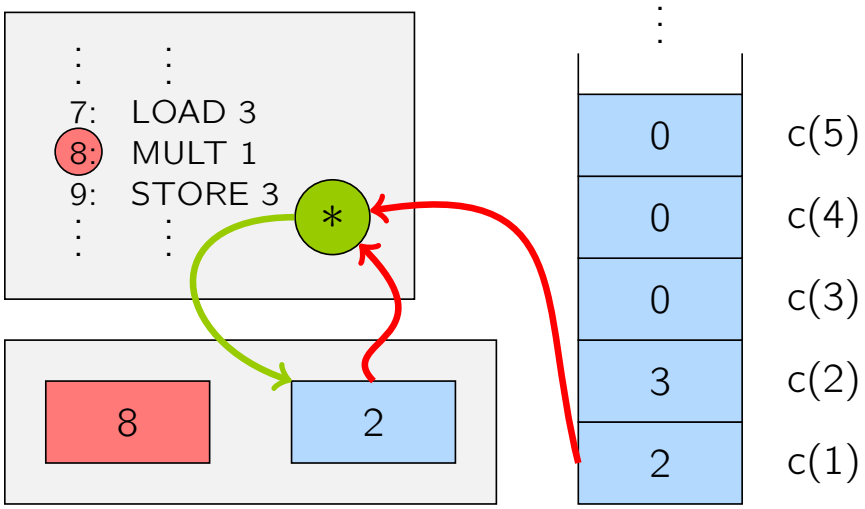
simulierende Turingmaschine

Update $c(0)$

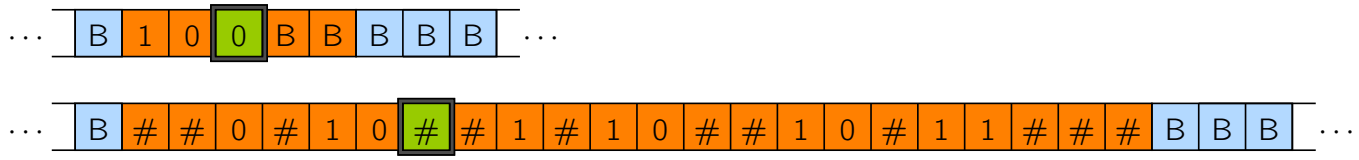


Simulation RAM durch TM – Illustration

simulierte Registermaschine M

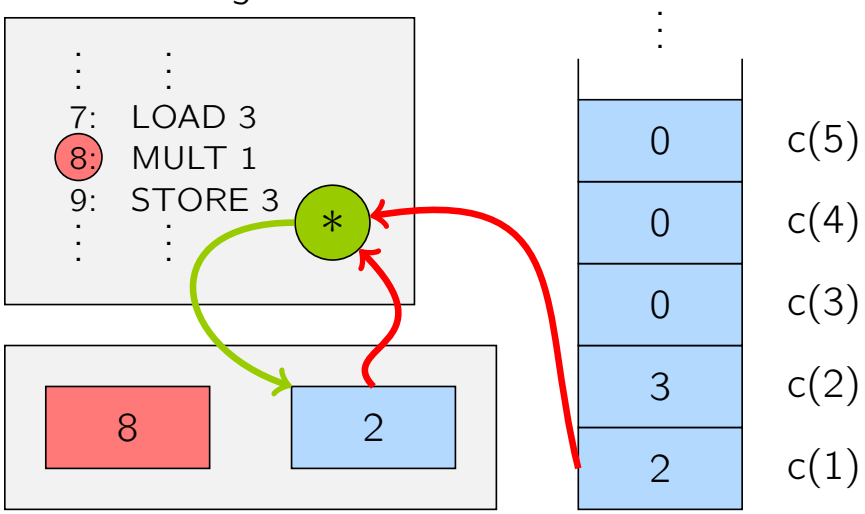


simulierende Turingmaschine



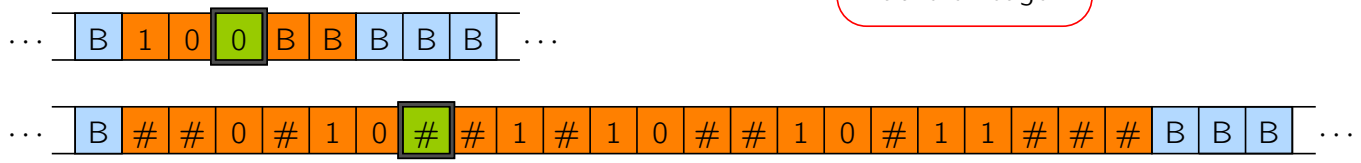
Simulation RAM durch TM – Illustration

simulierte Registermaschine M



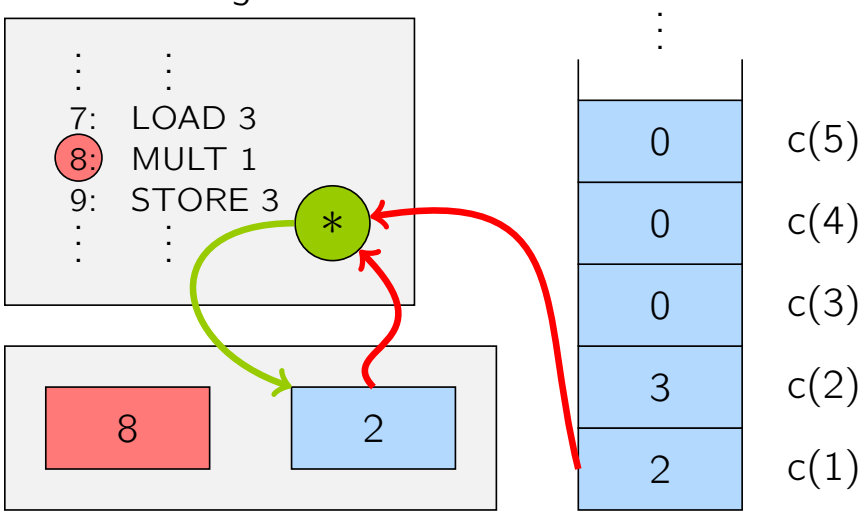
simulierende Turingmaschine

Lücke erzeugen



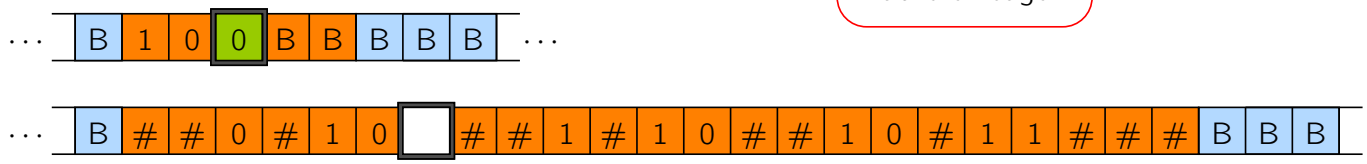
Simulation RAM durch TM – Illustration

simulierte Registermaschine M



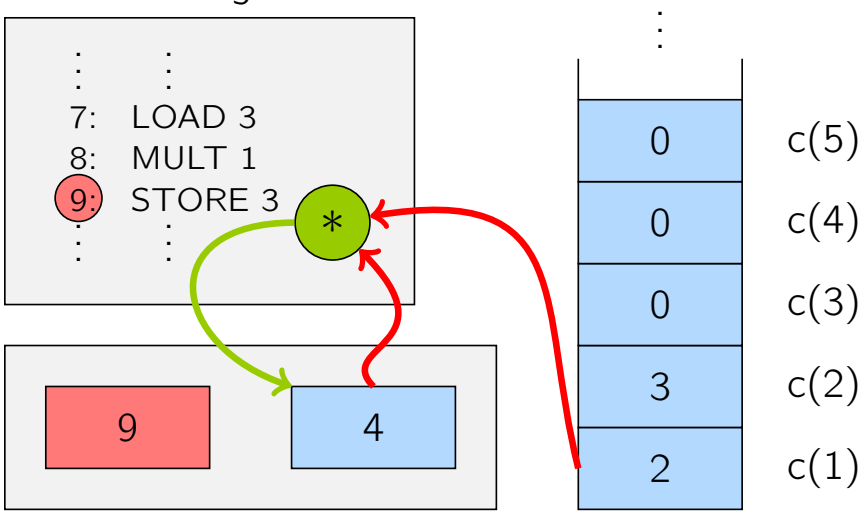
simulierende Turingmaschine

Lücke erzeugen

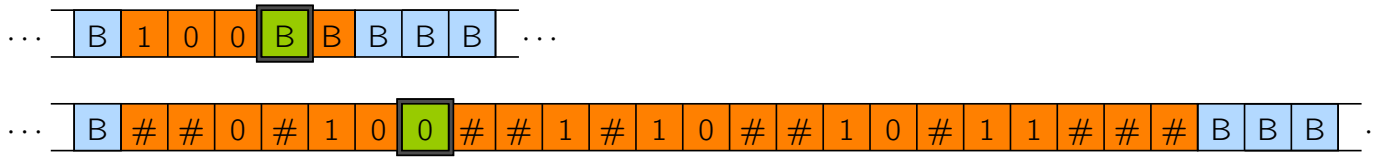


Simulation RAM durch TM – Illustration

simulierte Registermaschine M



simulierende Turingmaschine



Simulation TM durch RAM

Simulation TM durch RAM

Satz

Jede $t(n)$-zeitbeschränkte TM kann durch eine RAM simuliert werden, die zeitbeschränkt ist durch

- $O(t(n) + n)$ *im uniformen Kostenmaß und*
- $O((t(n) + n) \cdot \log(t(n) + n))$ *im logarithmischen Kostenmaß.*

Simulation TM durch RAM – Beweis

Beweis des Satzes

- O.B.d.A. nehmen wir an, es handelt sich um eine TM mit einseitig beschränktem Band, deren Zellen mit $0, 1, 2, 3, \ldots$ durchnummeriert sind. (Vgl. Übung)

Simulation TM durch RAM – Beweis

Beweis des Satzes

- O.B.d.A. nehmen wir an, es handelt sich um eine TM mit einseitig beschränktem Band, deren Zellen mit $0, 1, 2, 3, \ldots$ durchnummeriert sind. (Vgl. Übung)
- Die Zustände und Zeichen werden ebenfalls durchnummeriert und mit ihren Nummern identifiziert, so dass sie in den Registern abgespeichert werden können.

Simulation TM durch RAM – Beweis

Beweis des Satzes

- O.B.d.A. nehmen wir an, es handelt sich um eine TM mit einseitig beschränktem Band, deren Zellen mit $0, 1, 2, 3, \ldots$ durchnummeriert sind. (Vgl. Übung)
- Die Zustände und Zeichen werden ebenfalls durchnummeriert und mit ihren Nummern identifiziert, so dass sie in den Registern abgespeichert werden können.
- Register 1 speichert den Index der Kopfposition.

Simulation TM durch RAM – Beweis

Beweis des Satzes

- O.B.d.A. nehmen wir an, es handelt sich um eine TM mit einseitig beschränktem Band, deren Zellen mit $0, 1, 2, 3, \ldots$ durchnummeriert sind. (Vgl. Übung)
- Die Zustände und Zeichen werden ebenfalls durchnummeriert und mit ihren Nummern identifiziert, so dass sie in den Registern abgespeichert werden können.
- Register 1 speichert den Index der Kopfposition.
- Register 2 speichert den aktuellen Zustand.

Simulation TM durch RAM – Beweis

Beweis des Satzes

- O.B.d.A. nehmen wir an, es handelt sich um eine TM mit einseitig beschränktem Band, deren Zellen mit $0, 1, 2, 3, \ldots$ durchnummeriert sind. (Vgl. Übung)
- Die Zustände und Zeichen werden ebenfalls durchnummeriert und mit ihren Nummern identifiziert, so dass sie in den Registern abgespeichert werden können.
- Register 1 speichert den Index der Kopfposition.
- Register 2 speichert den aktuellen Zustand.
- Die Register $3, 4, 5, 6, \ldots$ speichern die Inhalte der besuchten Bandpositionen $0, 1, 2, 3, \ldots$

Simulation TM durch RAM – Beweis

Die TM wird nun Schritt für Schritt durch die RAM simuliert.

Auswahl des richtigen TM-Übergangs

Die RAM verwendet eine zweistufige *if*-Abfrage:

- Auf einer ersten Stufe von $|Q|$ vielen *if*-Abfragen wird der aktuelle Zustand selektiert.

Simulation TM durch RAM – Beweis

Die TM wird nun Schritt für Schritt durch die RAM simuliert.

Auswahl des richtigen TM-Übergangs

Die RAM verwendet eine zweistufige *if*-Abfrage:

- Auf einer ersten Stufe von $|Q|$ vielen *if*-Abfragen wird der aktuelle Zustand selektiert.
- Für jeden möglichen Zustand gibt es dann eine zweite Stufe von $|\Gamma|$ vielen *if*-Abfragen, die das gelesene Zeichen selektieren.

Simulation TM durch RAM – Beweis

Die TM wird nun Schritt für Schritt durch die RAM simuliert.

Auswahl des richtigen TM-Übergangs

Die RAM verwendet eine zweistufige *if*-Abfrage:

- Auf einer ersten Stufe von $|Q|$ vielen *if*-Abfragen wird der aktuelle Zustand selektiert.
- Für jeden möglichen Zustand gibt es dann eine zweite Stufe von $|\Gamma|$ vielen *if*-Abfragen, die das gelesene Zeichen selektieren.

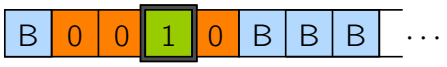
Durchführung des TM-Übergangs

Je nach Ausgang der *if*-Abfragen aktualisiert die RAM

- den TM-Zustand in Register 2,
- die TM-Bandinschrift in Register $c(1)$ und
- die TM-Bandposition in Register 1.

Simulation TM durch RAM – Illustration

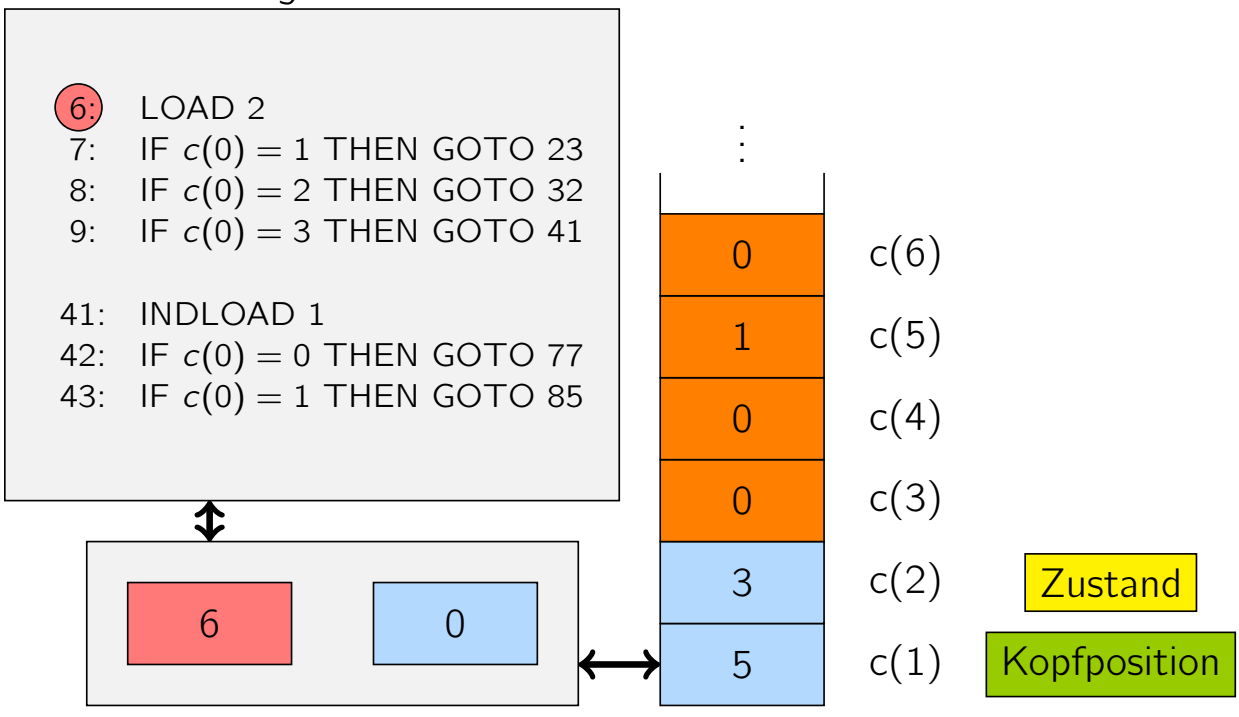
simulierte Turingmaschine M



δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

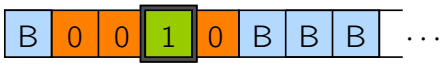
q_3

simulierende Registermaschine



Simulation TM durch RAM – Illustration

simulierte Turingmaschine M

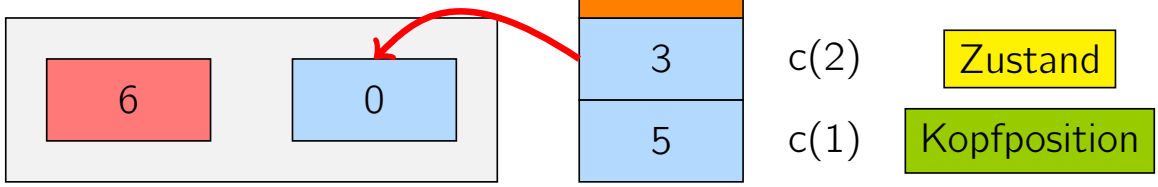


δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

q_3

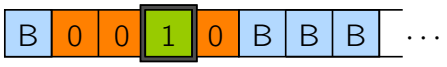
simulierende Registermaschine

```
6:   LOAD 2
7:   IF c(0) = 1 THEN GOTO 23
8:   IF c(0) = 2 THEN GOTO 32
9:   IF c(0) = 3 THEN GOTO 41

41:  INDLOAD 1
42:  IF c(0) = 0 THEN GOTO 77
43:  IF c(0) = 1 THEN GOTO 85
```



Simulation TM durch RAM – Illustration

simulierte Turingmaschine M



δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

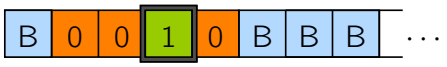
q_3

simulierende Registermaschine

```
6:   LOAD 2
7:   IF c(0) = 1 THEN GOTO 23
8:   IF c(0) = 2 THEN GOTO 32
9:   IF c(0) = 3 THEN GOTO 41

41:  INDLOAD 1
42:  IF c(0) = 0 THEN GOTO 77
43:  IF c(0) = 1 THEN GOTO 85
```

Simulation TM durch RAM – Illustration

simulierte Turingmaschine M

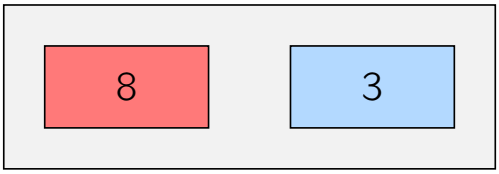


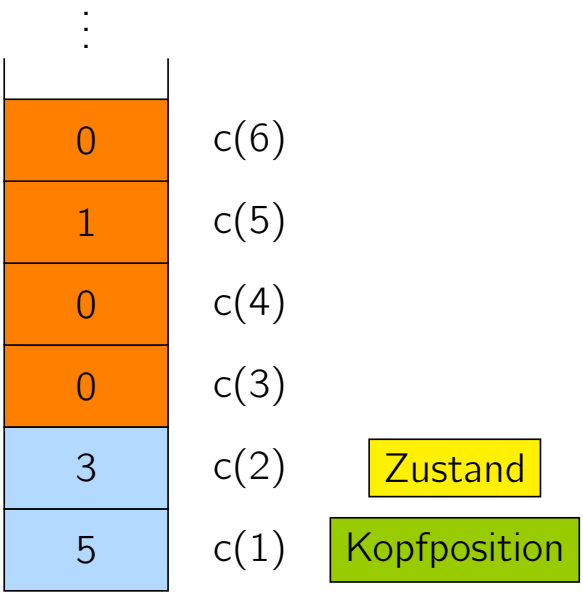
δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

q_3

simulierende Registermaschine

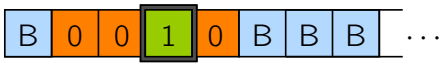
```
 6:   LOAD 2
 7:   IF c(0) = 1 THEN GOTO 23
 8:   IF c(0) = 2 THEN GOTO 32
 9:   IF c(0) = 3 THEN GOTO 41

41:   INDLOAD 1
42:   IF c(0) = 0 THEN GOTO 77
43:   IF c(0) = 1 THEN GOTO 85
```





Simulation TM durch RAM – Illustration

simulierte Turingmaschine M

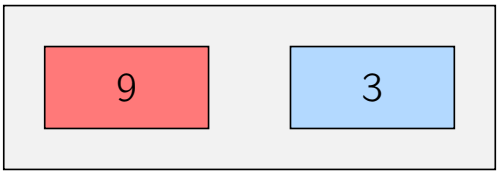


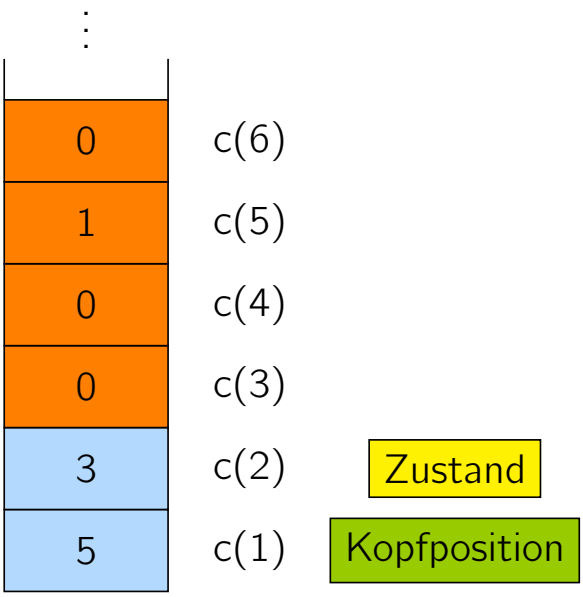
δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

q_3

simulierende Registermaschine

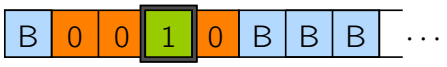
```
 6:   LOAD 2
 7:   IF c(0) = 1 THEN GOTO 23
 8:   IF c(0) = 2 THEN GOTO 32
 9:   IF c(0) = 3 THEN GOTO 41

41:   INDLOAD 1
42:   IF c(0) = 0 THEN GOTO 77
43:   IF c(0) = 1 THEN GOTO 85
```





Simulation TM durch RAM – Illustration

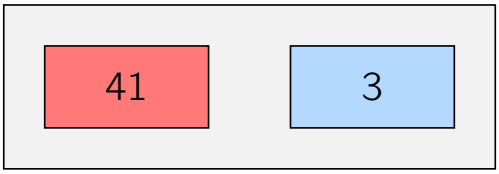
simulierte Turingmaschine M

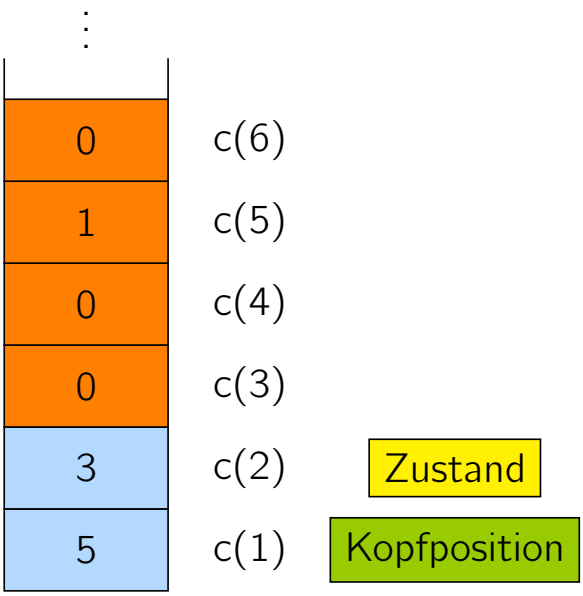


δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

simulierende Registermaschine

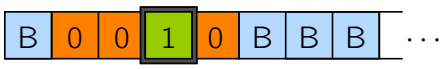
```
6:   LOAD 2
7:   IF c(0) = 1 THEN GOTO 23
8:   IF c(0) = 2 THEN GOTO 32
9:   IF c(0) = 3 THEN GOTO 41

41:  INDLOAD 1
42:  IF c(0) = 0 THEN GOTO 77
43:  IF c(0) = 1 THEN GOTO 85
```





Simulation TM durch RAM – Illustration

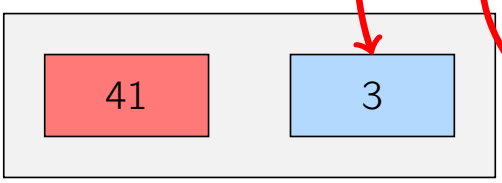
simulierte Turingmaschine M

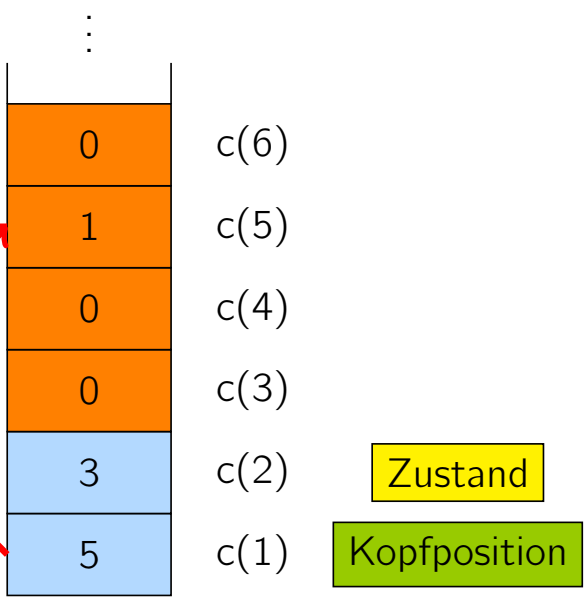


δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

simulierende Registermaschine

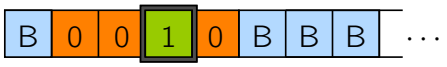
```
6:   LOAD 2
7:   IF c(0) = 1 THEN GOTO 23
8:   IF c(0) = 2 THEN GOTO 32
9:   IF c(0) = 3 THEN GOTO 41

41:  INDLOAD 1
42:  IF c(0) = 0 THEN GOTO 77
43:  IF c(0) = 1 THEN GOTO 85
```





Simulation TM durch RAM – Illustration

simulierte Turingmaschine M

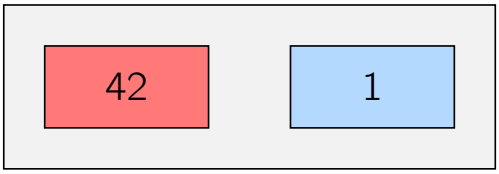


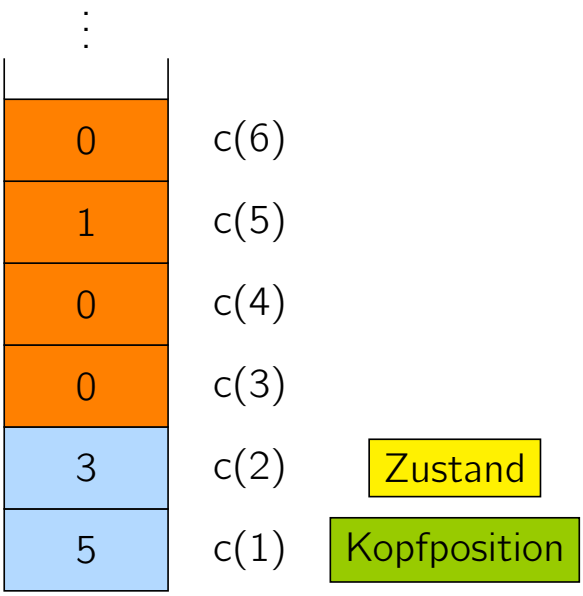
δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

q_3

simulierende Registermaschine

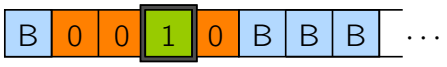
```
 6:  LOAD 2
 7:  IF c(0) = 1 THEN GOTO 23
 8:  IF c(0) = 2 THEN GOTO 32
 9:  IF c(0) = 3 THEN GOTO 41

41:  INDLOAD 1
42:  IF c(0) = 0 THEN GOTO 77
43:  IF c(0) = 1 THEN GOTO 85
```





Simulation TM durch RAM – Illustration

simulierte Turingmaschine M

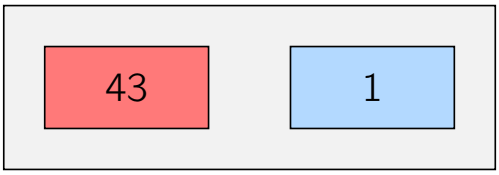


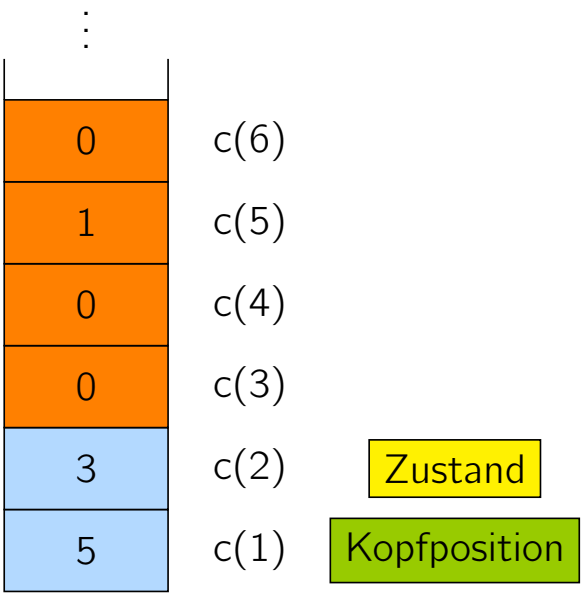
δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

q_3

simulierende Registermaschine

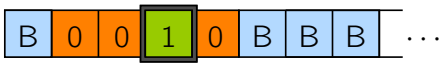
```
 6:  LOAD 2
 7:  IF c(0) = 1 THEN GOTO 23
 8:  IF c(0) = 2 THEN GOTO 32
 9:  IF c(0) = 3 THEN GOTO 41

41:  INDLOAD 1
42:  IF c(0) = 0 THEN GOTO 77
43:  IF c(0) = 1 THEN GOTO 85
```





Simulation TM durch RAM – Illustration

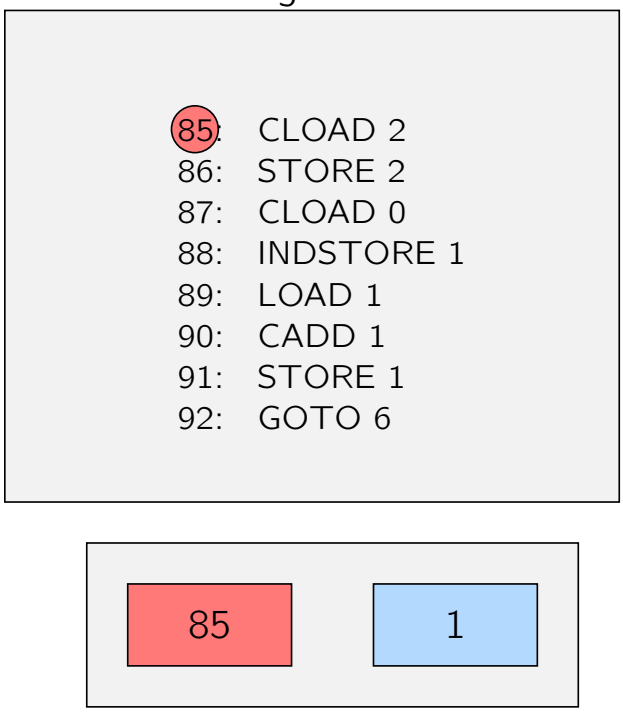
simulierte Turingmaschine M

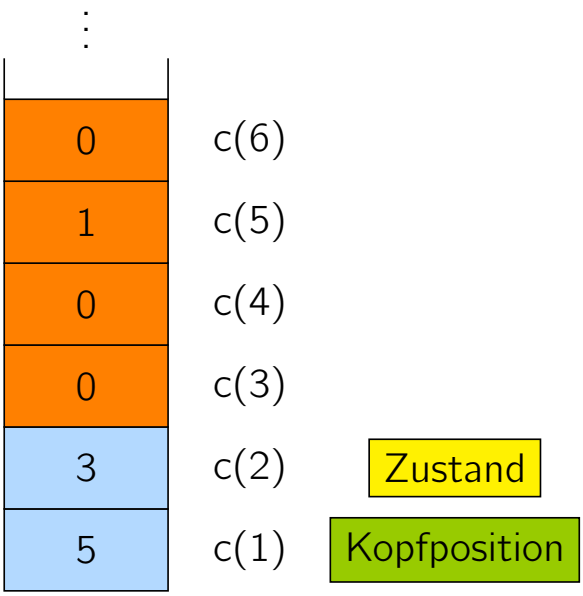


δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

q_3

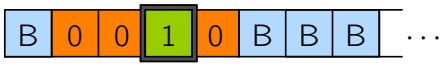
simulierende Registermaschine





Simulation TM durch RAM – Illustration

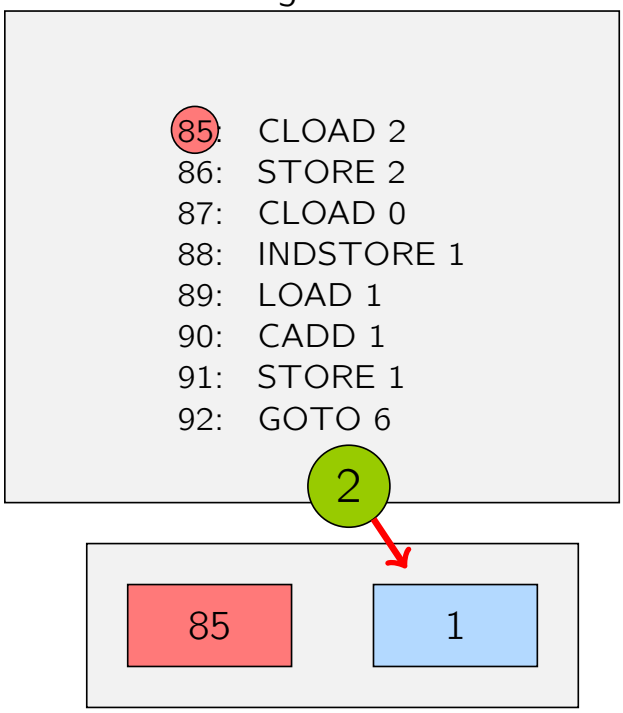
simulierte Turingmaschine M

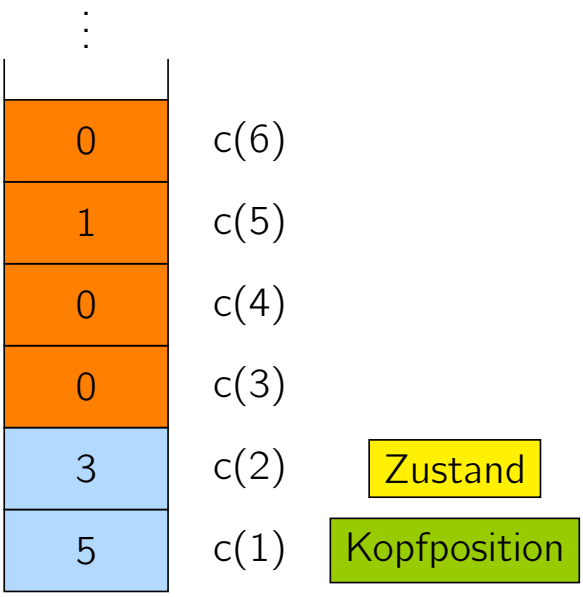


δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

q_3

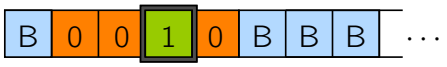
simulierende Registermaschine





Simulation TM durch RAM – Illustration

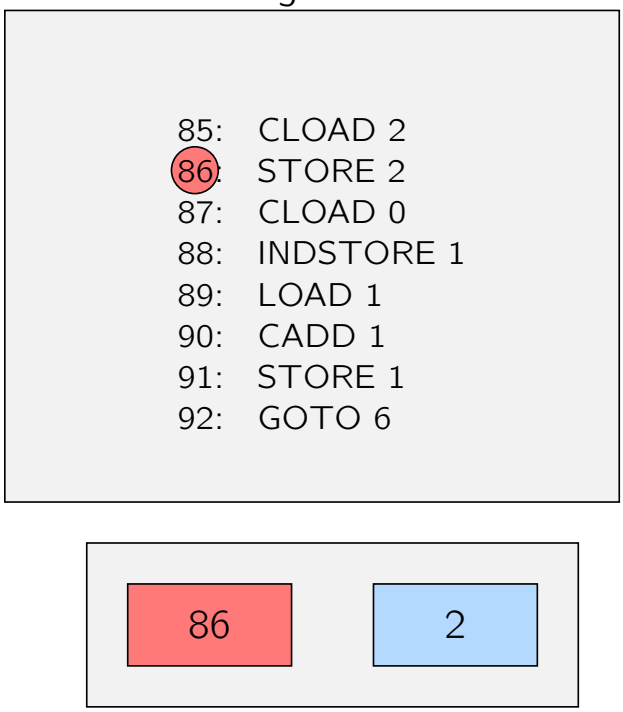
simulierte Turingmaschine M

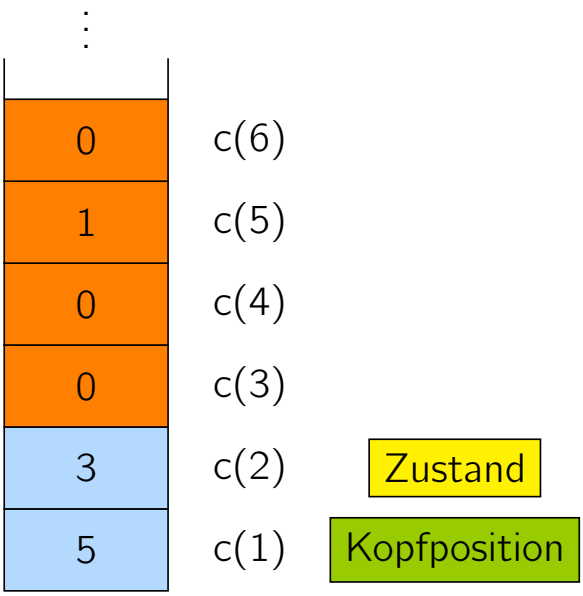


δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

q_3

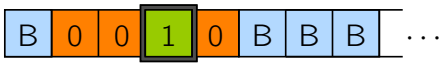
simulierende Registermaschine





Simulation TM durch RAM – Illustration

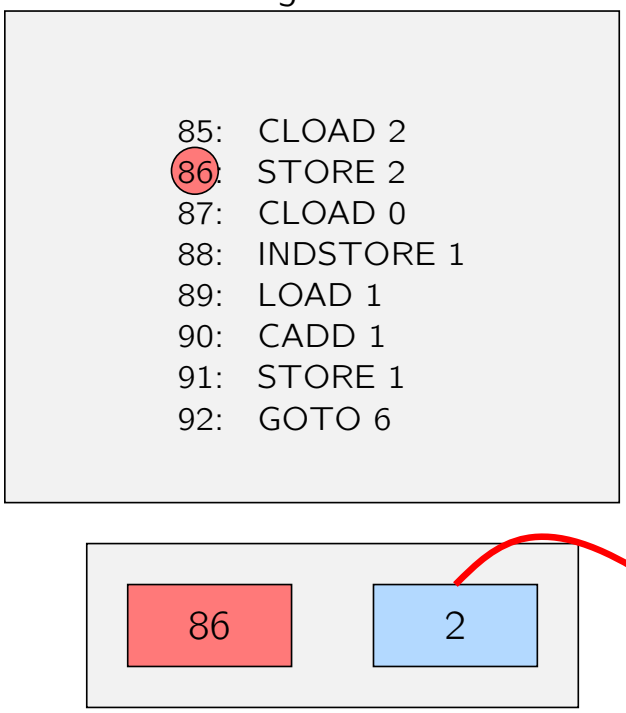
simulierte Turingmaschine M

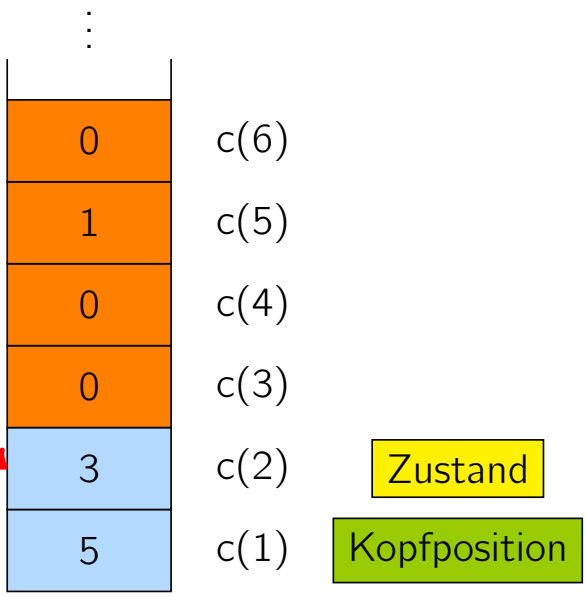


δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

q_3

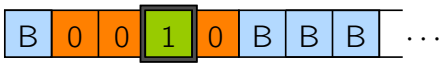
simulierende Registermaschine





Simulation TM durch RAM – Illustration

simulierte Turingmaschine M

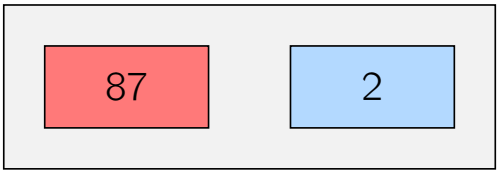


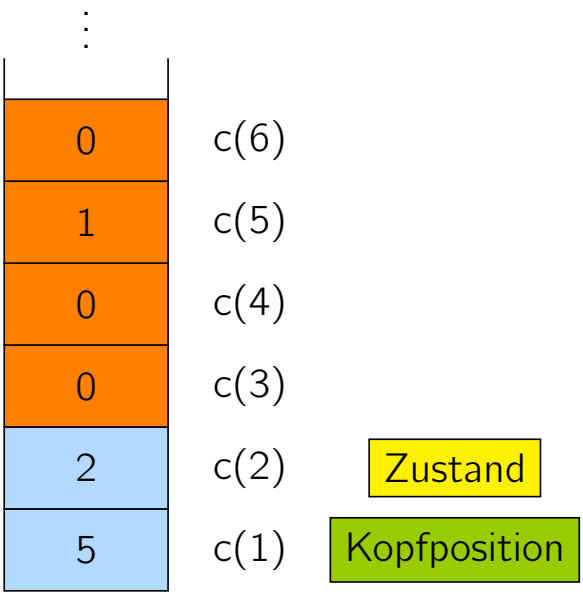
δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

q_3

simulierende Registermaschine

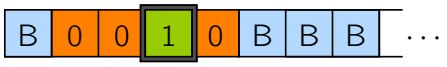
```
85:  CLOAD 2
86:  STORE 2
87:  CLOAD 0
88:  INDSTORE 1
89:  LOAD 1
90:  CADD 1
91:  STORE 1
92:  GOTO 6
```





Simulation TM durch RAM – Illustration

simulierte Turingmaschine M

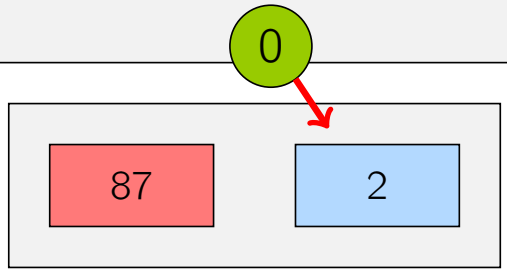


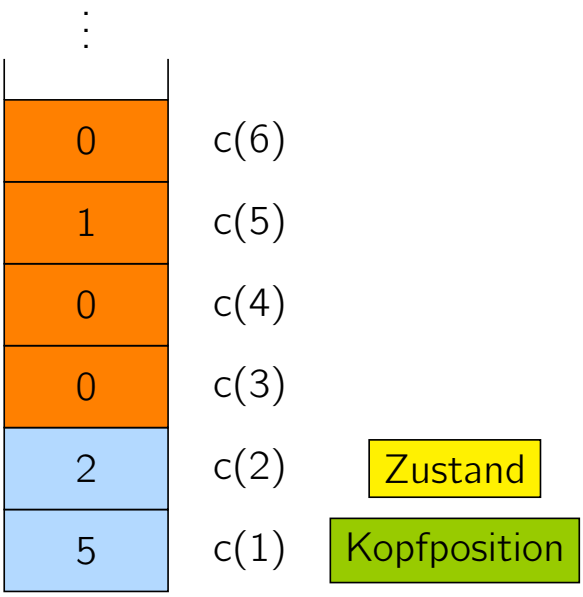
δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

q_3

simulierende Registermaschine

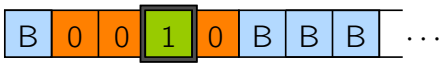
```
85:  CLOAD 2
86:  STORE 2
87:  CLOAD 0
88:  INDSTORE 1
89:  LOAD 1
90:  CADD 1
91:  STORE 1
92:  GOTO 6
```





Simulation TM durch RAM – Illustration

simulierte Turingmaschine M

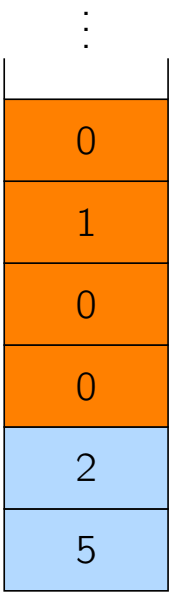


δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

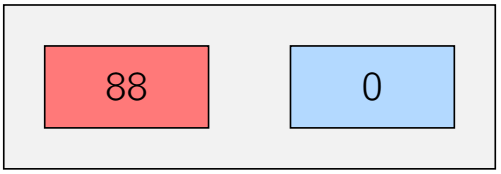
q_3

simulierende Registermaschine

```
85:  CLOAD 2
86:  STORE 2
87:  CLOAD 0
88:  INDSTORE 1
89:  LOAD 1
90:  CADD 1
91:  STORE 1
92:  GOTO 6
```



c(6) c(5) c(4) c(3) c(2) c(1)

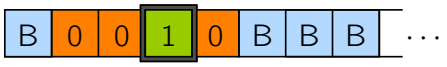


Zustand

Kopfposition

Simulation TM durch RAM – Illustration

simulierte Turingmaschine M

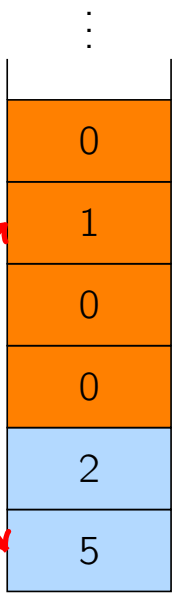


δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

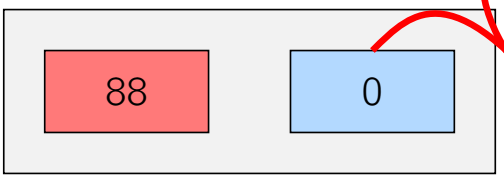
q_3

simulierende Registermaschine

```
85:  CLOAD 2
86:  STORE 2
87:  CLOAD 0
88:  INDSTORE 1
89:  LOAD 1
90:  CADD 1
91:  STORE 1
92:  GOTO 6
```



c(6) c(5) c(4) c(3) c(2) c(1)



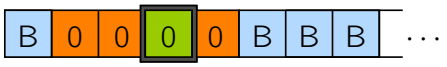
Zustand

Kopfposition

Vorlesung BuK im WS 22/23, M. Grohe Seite 109 Version 12. Oktober 2022

Simulation TM durch RAM – Illustration

simulierte Turingmaschine M

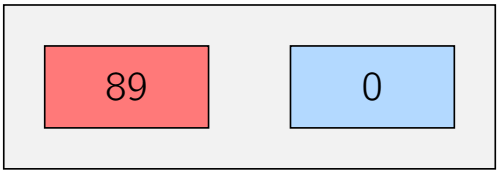


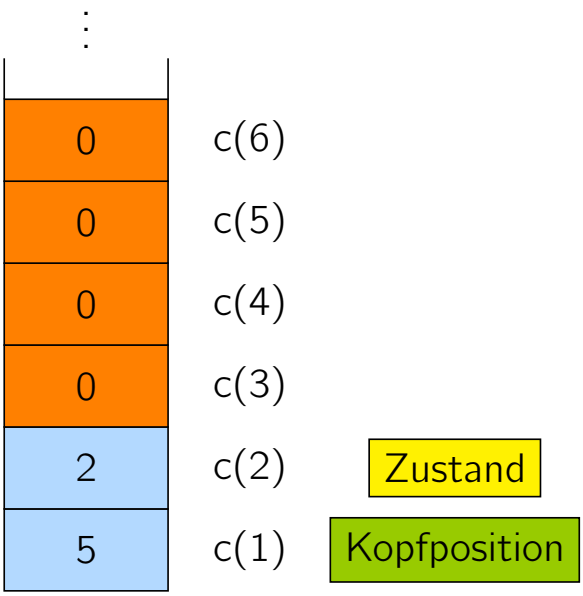
δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

q_3

simulierende Registermaschine

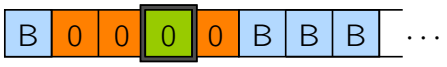
```
85:  CLOAD 2
86:  STORE 2
87:  CLOAD 0
88:  INDSTORE 1
89:  LOAD 1
90:  CADD 1
91:  STORE 1
92:  GOTO 6
```





Simulation TM durch RAM – Illustration

simulierte Turingmaschine M

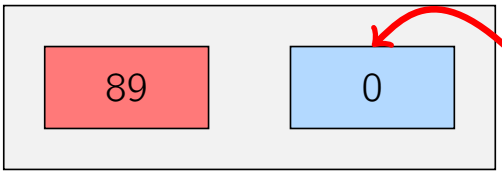


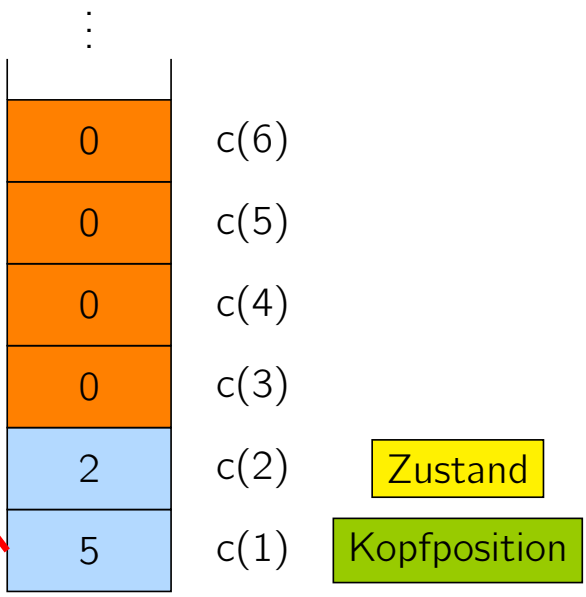
δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

q_3

simulierende Registermaschine

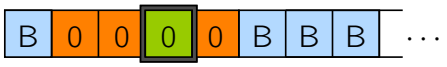
```
85:  CLOAD 2
86:  STORE 2
87:  CLOAD 0
88:  INDSTORE 1
89:  LOAD 1
90:  CADD 1
91:  STORE 1
92:  GOTO 6
```





Simulation TM durch RAM – Illustration

simulierte Turingmaschine M

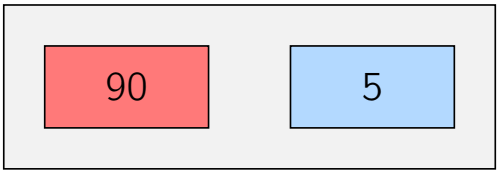


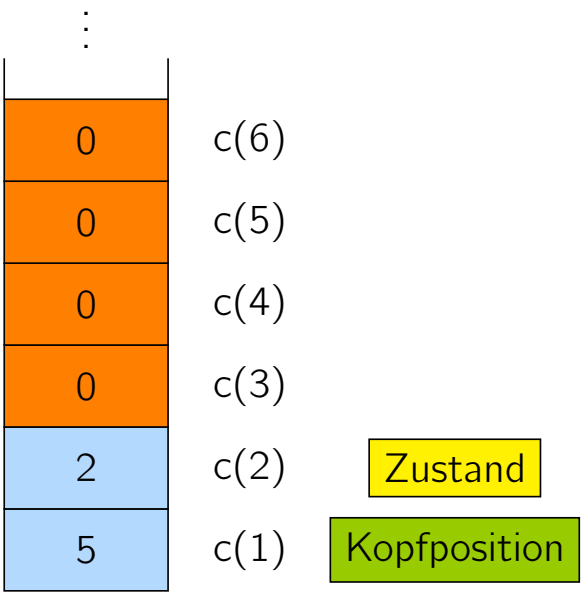
δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

q_3

simulierende Registermaschine

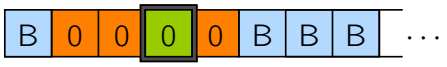
```
85:  CLOAD 2
86:  STORE 2
87:  CLOAD 0
88:  INDSTORE 1
89:  LOAD 1
90:  CADD 1
91:  STORE 1
92:  GOTO 6
```





Simulation TM durch RAM – Illustration

simulierte Turingmaschine M

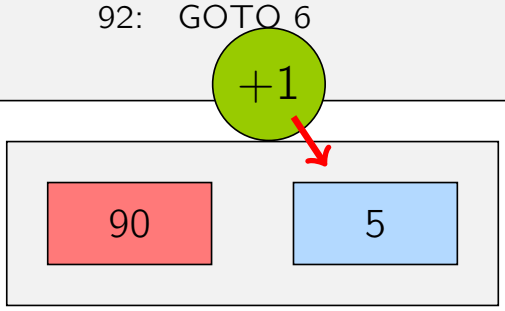


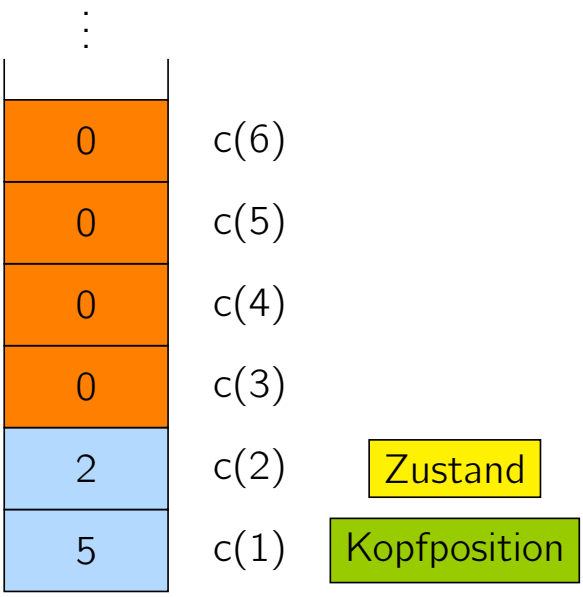
δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

q_3

simulierende Registermaschine

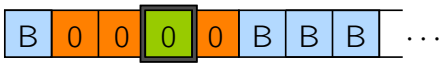
```
85:  CLOAD 2
86:  STORE 2
87:  CLOAD 0
88:  INDSTORE 1
89:  LOAD 1
90:  CADD 1
91:  STORE 1
92:  GOTO 6
```





Simulation TM durch RAM – Illustration

simulierte Turingmaschine M

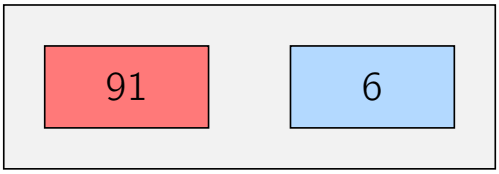


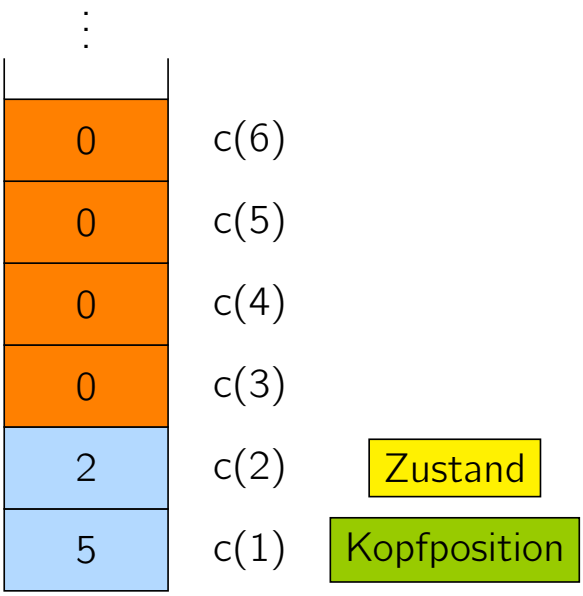
δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

q_3

simulierende Registermaschine

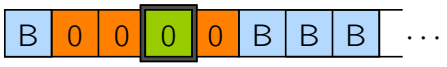
```
85:  CLOAD 2
86:  STORE 2
87:  CLOAD 0
88:  INDSTORE 1
89:  LOAD 1
90:  CADD 1
91:  STORE 1
92:  GOTO 6
```





Simulation TM durch RAM – Illustration

simulierte Turingmaschine M

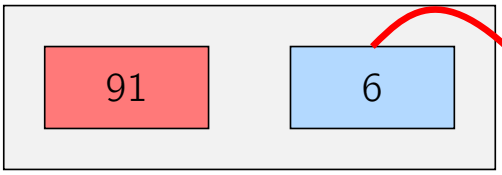


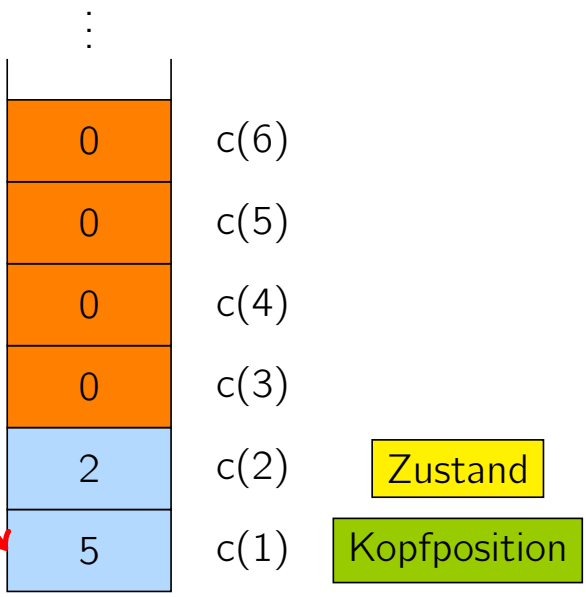
δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

q_3

simulierende Registermaschine

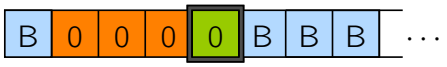
```
85:  CLOAD 2
86:  STORE 2
87:  CLOAD 0
88:  INDSTORE 1
89:  LOAD 1
90:  CADD 1
91:  STORE 1
92:  GOTO 6
```





Simulation TM durch RAM – Illustration

simulierte Turingmaschine M

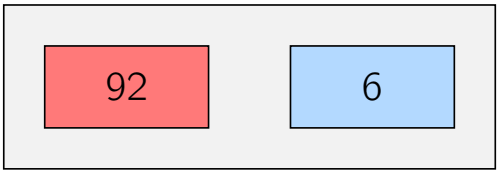


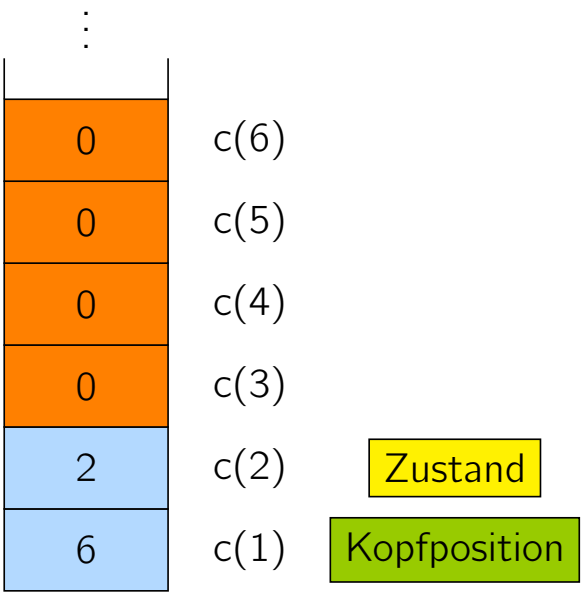
δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

q_3

simulierende Registermaschine

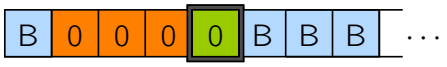
```
85:  CLOAD 2
86:  STORE 2
87:  CLOAD 0
88:  INDSTORE 1
89:  LOAD 1
90:  CADD 1
91:  STORE 1
92:  GOTO 6
```





Simulation TM durch RAM – Illustration

simulierte Turingmaschine M

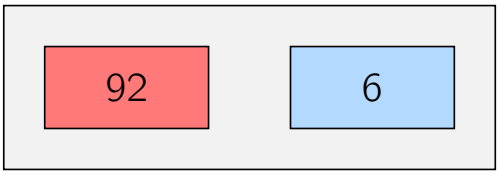


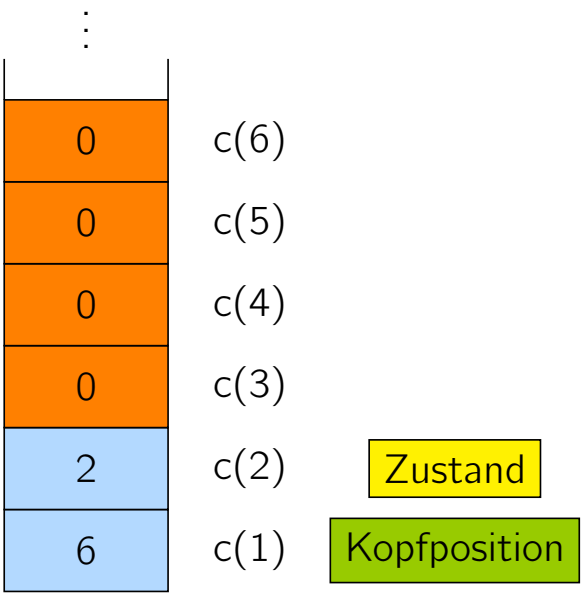
δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

q_2

simulierende Registermaschine

```
85:  CLOAD 2
86:  STORE 2
87:  CLOAD 0
88:  INDSTORE 1
89:  LOAD 1
90:  CADD 1
91:  STORE 1
92:  GOTO 6
```





Simulation TM durch RAM – Beweis

Laufzeitanalyse im uniformen Kostenmodell:

- Die Initialisierung kann in Zeit $O(n)$ durchgeführt werden.

Simulation TM durch RAM – Beweis

Laufzeitanalyse im uniformen Kostenmodell:

- Die Initialisierung kann in Zeit $O(n)$ durchgeführt werden.
- Die Simulation jedes einzelnen TM-Schrittes hat konstante Laufzeit.

Simulation TM durch RAM – Beweis

Laufzeitanalyse im uniformen Kostenmodell:

- Die Initialisierung kann in Zeit $O(n)$ durchgeführt werden.
- Die Simulation jedes einzelnen TM-Schrittes hat konstante Laufzeit.
- Insgesamt ist die Simulationszeit somit $O(n + t(n))$.

Simulation TM durch RAM – Beweis

Laufzeitanalyse im logarithmischen Kostenmodell:

- Die in den Registern gespeicherten Zahlen repräsentieren Zustände, Zeichen und Bandpositionen.

Simulation TM durch RAM – Beweis

Laufzeitanalyse im logarithmischen Kostenmodell:

- Die in den Registern gespeicherten Zahlen repräsentieren Zustände, Zeichen und Bandpositionen.
- Zustände und Zeichen haben eine konstante Kodierungslänge.

Simulation TM durch RAM – Beweis

Laufzeitanalyse im logarithmischen Kostenmodell:

- Die in den Registern gespeicherten Zahlen repräsentieren Zustände, Zeichen und Bandpositionen.
- Zustände und Zeichen haben eine konstante Kodierungslänge.
- Die Bandpositionen, die während der Simulation angesprochen werden, sind durch $\max\{n, t(n)\} \leq n + t(n)$ beschränkt. Die Kodierungslänge dieser Positionen ist also $O(\log(t(n) + n))$.

Simulation TM durch RAM – Beweis

Laufzeitanalyse im logarithmischen Kostenmodell:

- Die in den Registern gespeicherten Zahlen repräsentieren Zustände, Zeichen und Bandpositionen.
- Zustände und Zeichen haben eine konstante Kodierungslänge.
- Die Bandpositionen, die während der Simulation angesprochen werden, sind durch $\max\{n, t(n)\} \leq n + t(n)$ beschränkt. Die Kodierungslänge dieser Positionen ist also $O(\log(t(n) + n))$.
- Damit kann die Simulation jedes einzelnen TM-Schrittes in Zeit $O(\log(t(n) + n))$ durchgeführt werden.

Simulation TM durch RAM – Beweis

Laufzeitanalyse im logarithmischen Kostenmodell:

- Die in den Registern gespeicherten Zahlen repräsentieren Zustände, Zeichen und Bandpositionen.
- Zustände und Zeichen haben eine konstante Kodierungslänge.
- Die Bandpositionen, die während der Simulation angesprochen werden, sind durch $\max\{n, t(n)\} \leq n + t(n)$ beschränkt. Die Kodierungslänge dieser Positionen ist also $O(\log(t(n) + n))$.
- Damit kann die Simulation jedes einzelnen TM-Schrittes in Zeit $O(\log(t(n) + n))$ durchgeführt werden.
- Insgesamt ergibt sich somit eine Simulationszeit von $O((t(n) + n)\log(t(n) + n))$.

□

Zusammenfassung

- Die Mehrband-TM kann mit quadratischem Zeitverlust durch eine (1-Band-)TM simuliert werden.

Zusammenfassung

- Die Mehrband-TM kann mit quadratischem Zeitverlust durch eine (1-Band-)TM simuliert werden.
- TM und RAM (im logarithmischen Kostenmodell) können sich gegenseitig mit polynomiellem Zeitverlust simulieren.

Zusammenfassung

- Die Mehrband-TM kann mit quadratischem Zeitverlust durch eine (1-Band-)TM simuliert werden.
- TM und RAM (im logarithmischen Kostenmodell) können sich gegenseitig mit polynomiellem Zeitverlust simulieren.
- Wenn es uns also „nur“ um Fragen der Berechenbarkeit von Problemen (oder um ihre Lösbarkeit in polynomieller Zeit) geht, können wir wahlweise auf die TM, die Mehrband-TM oder die RAM zurückgreifen.

Die Church-Turing-These

Kein jemals bisher vorgeschlagenes „vernünftiges“ Rechnermodell ist mächtiger als die TM.

Die Church-Turing-These

Kein jemals bisher vorgeschlagenes „vernünftiges“ Rechnermodell ist mächtiger als die TM.

Diese Einsicht hat Church zur Formulierung der folgenden These veranlasst.

Church-Turing-These

Die Klassen der TM-berechenbaren Funktionen und TM-entscheidbaren Sprachen stimmen mit den Klassen der “intuitiv berechenbaren” Funktionen bzw. “intuitiv entscheidbaren” Sprachen überein.

Die Church-Turing-These

Kein jemals bisher vorgeschlagenes „vernünftiges“ Rechnermodell ist mächtiger als die TM.

Diese Einsicht hat Church zur Formulierung der folgenden These veranlasst.

Church-Turing-These

Die Klassen der TM-berechenbaren Funktionen und TM-entscheidbaren Sprachen stimmen mit den Klassen der “intuitiv berechenbaren” Funktionen bzw. “intuitiv entscheidbaren” Sprachen überein.

Wir werden deshalb nicht mehr von TM-berechenbaren Funktionen oder TM-entscheidbaren Sprachen sprechen, sondern allgemein von berechenbaren Funktionen bzw. entscheidbaren Sprachen.

Gleichbedeutend wird auch der Begriff rekursive Funktion bzw. Sprache verwendet.

Erweiterung: Partielle Funktionen

Wenn eine Turingmaschine bei einer Eingabe anhält, liefert sie eine wohldefinierte Ausgabe. Aber Turingmaschinen halten nicht immer an und liefern deswegen zu manchen Eingaben keine Ausgabe.

Erweiterung: Partielle Funktionen

Wenn eine Turingmaschine bei einer Eingabe anhält, liefert sie eine wohldefinierte Ausgabe. Aber Turingmaschinen halten nicht immer an und liefern deswegen zu manchen Eingaben keine Ausgabe.

Im allgemeinen berechnet eine Turingmaschine M mit Ein- und Ausgabealphabet Σ also nur eine partielle Funktion f_M von Σ^* nach Σ^*, die definiert ist durch

$$f_M(x) = \begin{cases} y & \text{falls } M \text{ bei Eingabe } x \text{ mit Ausgabe } y \text{ anhält,} \\ \perp \text{ (undefiniert)} & \text{falls } M \text{ bei Eingabe } x \text{ nicht anhält.} \end{cases}$$

Erweiterung: Partielle Funktionen

Wenn eine Turingmaschine bei einer Eingabe anhält, liefert sie eine wohldefinierte Ausgabe. Aber Turingmaschinen halten nicht immer an und liefern deswegen zu manchen Eingaben keine Ausgabe.

Im allgemeinen berechnet eine Turingmaschine M mit Ein- und Ausgabealphabet Σ also nur eine partielle Funktion f_M von Σ^* nach Σ^*, die definiert ist durch

$$f_M(x) = \begin{cases} y & \text{falls } M \text{ bei Eingabe } x \text{ mit Ausgabe } y \text{ anhält,} \\ \perp \text{ (undefiniert)} & \text{falls } M \text{ bei Eingabe } x \text{ nicht anhält.} \end{cases}$$

Wir nennen eine partielle Funktion f von Σ^* nach Σ^* berechenbar, wenn es eine Turingmaschine M gibt, so dass $f = f_M$.

Zum Inhalt der Vorlesung

Jetzt sind wir bereit, die zentrale Fragestellung des ersten Teils dieser Vorlesung formaler zu fassen.

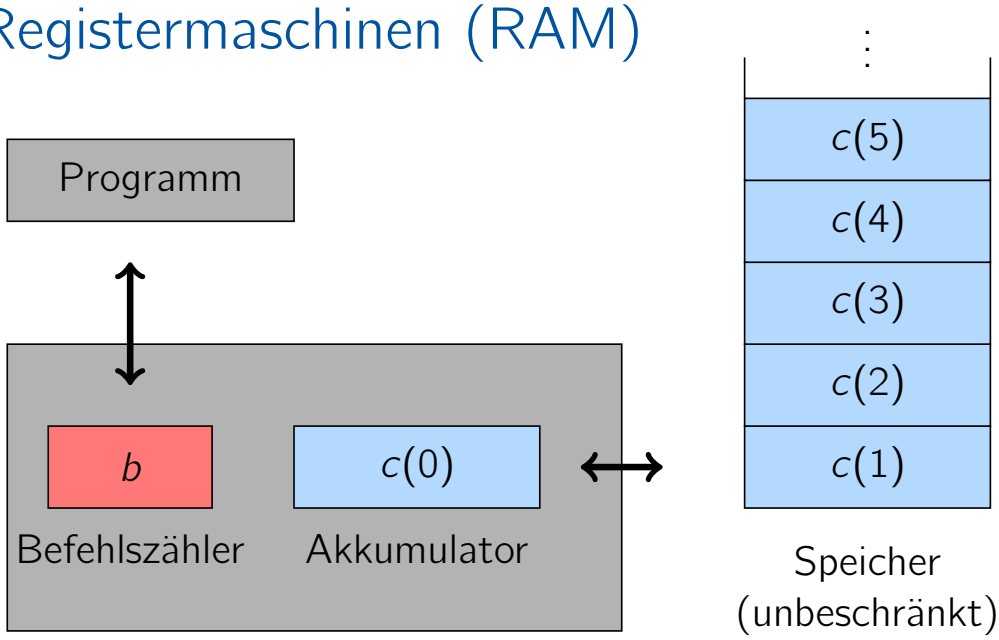
In der *Berechenbarkeitstheorie* ...

... wird untersucht, welche (partiellen) Funktionen berechenbar und welche Sprachen entscheidbar sind, d.h. welche Berechnungsprobleme durch einen Algorithmus – ohne jegliche Einschränkungen der Rechenzeit oder des Speicherplatzes – gelöst werden können.

Teil II
Berechenbarkeit

Vorlesung 5
Unentscheidbare Probleme: Diagonalisierung

Wdh.: Registermaschinen (RAM)



Befehlssatz:
LOAD, STORE, ADD, SUB, MULT, DIV
INDLOAD, INDSTORE, INDADD, INDSUB, INDMULT, INDDIV
CLOAD, CADD, CSUB, CMULT, CDIV
GOTO, IF c(0)?x THEN GOTO j (wobei ? aus {=,<,<=,>,>=} ist),
END

Wdh.: RAM vs. TM

Satz (RAM → TM)

Für jede im logarithmischen Kostenmaß $t(n)$-zeitbeschränkte RAM R gibt es ein Polynom q und zu diesem eine $O\big(q(n + t(n))\big)$-TM M, die R simuliert.

Satz (TM → RAM)

Jede $t(n)$-zeitbeschränkte TM kann durch eine RAM simuliert werden, die zeitbeschränkt ist durch

- $O(t(n) + n)$ *im uniformen Kostenmaß und*
- $O((t(n) + n) \cdot \log(t(n) + n))$ *im logarithmischen Kostenmaß.*

Wdh.: Hintereinanderausführung von Polynomen

Beobachtung

Die Klasse der Polynome ist unter Hintereinanderausführung abgeschlossen.

(Wenn $p(x)$ und $q(x)$ Polynome sind, dann ist $p(q(x))$ ein Polynom.

Des Weiteren gilt für Polynome p, q:
Wenn $t(n) \in O(p(n))$ und $t'(n) \in O(q(n))$ dann $t(t'(n)) \in O(p(q(n)))$.

Wdh.: Die Church-Turing-These

Kein jemals bisher vorgeschlagenes „vernünftiges“ Rechnermodell hat eine größere Mächtigkeit als die TM.

Wdh.: Die Church-Turing-These

Kein jemals bisher vorgeschlagenes „vernünftiges“ Rechnermodell hat eine größere Mächtigkeit als die TM.

Diese Einsicht hat Church zur Formulierung der folgenden These veranlasst.

Church-Turing-These

Die Klassen der TM-berechenbaren (partiellen) Funktionen und TM-entscheidbaren Sprachen stimmen mit den Klassen der “intuitiv berechenbaren” (partiellen) Funktionen bzw. “intuitiv entscheidbaren” Sprachen überein.

Wdh.: Die Church-Turing-These

Kein jemals bisher vorgeschlagenes „vernünftiges“ Rechnermodell hat eine größere Mächtigkeit als die TM.

Diese Einsicht hat Church zur Formulierung der folgenden These veranlasst.

Church-Turing-These

Die Klassen der TM-berechenbaren (partiellen) Funktionen und TM-entscheidbaren Sprachen stimmen mit den Klassen der “intuitiv berechenbaren” (partiellen) Funktionen bzw. “intuitiv entscheidbaren” Sprachen überein.

Wir werden deshalb nicht mehr von TM-berechenbaren (partiellen) Funktionen oder TM-entscheidbaren Sprachen sprechen, sondern allgemein von berechenbaren (partiellen) Funktionen bzw. entscheidbaren Sprachen.

Gibt es unentscheidbare Probleme?

Terminologie: unentscheidbar bedeutet einfach “nicht entscheidbar”

Gibt es unentscheidbare Probleme?

Terminologie: unentscheidbar bedeutet einfach “nicht entscheidbar”

Ja, es gibt unentscheidbare Probleme

Gibt es unentscheidbare Probleme?

Terminologie: unentscheidbar bedeutet einfach “nicht entscheidbar”

Ja, es gibt unentscheidbare Probleme,

denn die Mächtigkeit der Menge aller Sprachen ist größer

als die Mächtigkeit der Menge aller TMen.

Exkurs: abzählbare und überabzählbare Mengen

Definition (Abzählbare Menge)

Eine Menge M heißt abzählbar, wenn sie leer ist oder wenn es eine surjektive Funktion $c : \mathbb{N} \to M$ gibt.

Exkurs: abzählbare und überabzählbare Mengen

Definition (Abzählbare Menge)

Eine Menge M heißt abzählbar, wenn sie leer ist oder wenn es eine surjektive Funktion $c : \mathbb{N} \to M$ gibt.

- Jede endliche Menge M ist abzählbar.

Exkurs: abzählbare und überabzählbare Mengen

Definition (Abzählbare Menge)

Eine Menge M heißt abzählbar, wenn sie leer ist oder wenn es eine surjektive Funktion $c : \mathbb{N} \to M$ gibt.

- Jede endliche Menge M ist abzählbar.

- Im Fall einer abzählbar unendlichen Menge M gibt es immer auch eine bijektive Abbildung $c : \mathbb{N} \to M$, denn Wiederholungen können bei der Abzählung offensichtlich ausgelassen werden.

Exkurs: abzählbare und überabzählbare Mengen

Definition (Abzählbare Menge)

Eine Menge M heißt abzählbar, wenn sie leer ist oder wenn es eine surjektive Funktion $c : \mathbb{N} \to M$ gibt.

- Jede endliche Menge M ist abzählbar.
- Im Fall einer abzählbar unendlichen Menge M gibt es immer auch eine bijektive Abbildung $c : \mathbb{N} \to M$, denn Wiederholungen können bei der Abzählung offensichtlich ausgelassen werden.
- Die Elemente einer abzählbaren Menge können also *nummeriert* werden.

Exkurs: abzählbare und überabzählbare Mengen

Definition (Abzählbare Menge)

Eine Menge M heißt abzählbar, wenn sie leer ist oder wenn es eine surjektive Funktion $c : \mathbb{N} \to M$ gibt.

- Jede endliche Menge M ist abzählbar.
- Im Fall einer abzählbar unendlichen Menge M gibt es immer auch eine bijektive Abbildung $c : \mathbb{N} \to M$, denn Wiederholungen können bei der Abzählung offensichtlich ausgelassen werden.
- Die Elemente einer abzählbaren Menge können also *nummeriert* werden.
- Abzählbar unendliche Mengen haben somit dieselbe Mächtigkeit wie die Menge der natürlichen Zahlen $\mathbb{N}$.

Exkursion: abzählbare und überabzählbare Mengen

Beispiele für abzählbar unendliche Mengen

- die Menge der ganzen Zahlen $\mathbb{Z}$:

$$c(i) = \begin{cases} i/2 & \text{falls } i \text{ gerade} \\ -(i+1)/2 & \text{falls } i \text{ ungerade} \end{cases}$$

Exkursion: abzählbare und überabzählbare Mengen

Beispiele für abzählbar unendliche Mengen

- die Menge der ganzen Zahlen $\mathbb{Z}$:

$$c(i) = \begin{cases} i/2 & \text{falls } i \text{ gerade} \\ -(i+1)/2 & \text{falls } i \text{ ungerade} \end{cases}$$

$$0, -1, 1, -2, 2, -3, 3, -4, 4, \ldots,$$

- die Menge der rationalen Zahlen $\mathbb{Q}$

Exkursion: abzählbare und überabzählbare Mengen

Beispiele für abzählbar unendliche Mengen

- die Menge der ganzen Zahlen $\mathbb{Z}$:

$$c(i) = \begin{cases} i/2 & \text{falls } i \text{ gerade} \\ -(i+1)/2 & \text{falls } i \text{ ungerade} \end{cases}$$

$$0, -1, 1, -2, 2, -3, 3, -4, 4, \ldots,$$

- die Menge der rationalen Zahlen $\mathbb{Q}$

$$0, \frac{1}{1}, \frac{2}{1}, \frac{1}{2}, \frac{3}{1}, \ldots, \frac{i}{1}, \frac{i-1}{2}, \frac{i-2}{3}, \ldots \frac{1}{i}, \ldots$$

Abzählbarkeit der rationalen Zahlen

	1	2	3	4	5	6	$\cdots$
1	1/1	2/1	3/1	4/1	5/1	6/1	
2	1/2	2/2	3/2	4/2	5/2	6/2	
3	1/3	2/3	3/3	4/3	5/3	6/3	$\cdots$
4	1/4	2/4	3/4	4/4	5/4	6/4	
5	1/5	2/5	3/5	4/5	5/5	6/5	
6	1/6	2/6	3/6	4/6	5/6	6/6	
$\vdots$			$\vdots$				$\ddots$

Abzählbarkeit der rationalen Zahlen

	1	2	3	4	5	6	$\cdots$
1	1/1	2/1	3/1	4/1	5/1	6/1	
2	1/2	2/2	3/2	4/2	5/2	6/2	
3	1/3	2/3	3/3	4/3	5/3	6/3	$\cdots$
4	1/4	2/4	3/4	4/4	5/4	6/4	
5	1/5	2/5	3/5	4/5	5/5	6/5	
6	1/6	2/6	3/6	4/6	5/6	6/6	
$\vdots$				$\vdots$			$\ddots$

Abzählbarkeit der rationalen Zahlen

	1	2	3	4	5	6	$\cdots$
1	1/1	2/1	3/1	4/1	5/1	6/1	
2	1/2	2/2	3/2	4/2	5/2	6/2	
3	1/3	2/3	3/3	4/3	5/3	6/3	$\cdots$
4	1/4	2/4	3/4	4/4	5/4	6/4	
5	1/5	2/5	3/5	4/5	5/5	6/5	
6	1/6	2/6	3/6	4/6	5/6	6/6	
$\vdots$				$\vdots$			$\ddots$

Abzählbarkeit der rationalen Zahlen

	1	2	3	4	5	6	$\cdots$
1	1/1	2/1	3/1	4/1	5/1	6/1	
2	1/2	2/2	3/2	4/2	5/2	6/2	
3	1/3	2/3	3/3	4/3	5/3	6/3	$\cdots$
4	1/4	2/4	3/4	4/4	5/4	6/4	
5	1/5	2/5	3/5	4/5	5/5	6/5	
6	1/6	2/6	3/6	4/6	5/6	6/6	
$\vdots$				$\vdots$			$\ddots$

Abzählbarkeit der rationalen Zahlen

	1	2	3	4	5	6	$\cdots$
1	1/1	2/1	3/1	4/1	5/1	6/1	
2	1/2	2/2	3/2	4/2	5/2	6/2	
3	1/3	2/3	3/3	4/3	5/3	6/3	$\cdots$
4	1/4	2/4	3/4	4/4	5/4	6/4	
5	1/5	2/5	3/5	4/5	5/5	6/5	
6	1/6	2/6	3/6	4/6	5/6	6/6	
$\vdots$				$\vdots$			$\ddots$

Abzählbarkeit der rationalen Zahlen

	1	2	3	4	5	6	$\cdots$
1	1/1	2/1	3/1	4/1	5/1	6/1	
2	1/2	2/2	3/2	4/2	5/2	6/2	
3	1/3	2/3	3/3	4/3	5/3	6/3	$\cdots$
4	1/4	2/4	3/4	4/4	5/4	6/4	
5	1/5	2/5	3/5	4/5	5/5	6/5	
6	1/6	2/6	3/6	4/6	5/6	6/6	
$\vdots$				$\vdots$			$\ddots$

Abzählbarkeit der rationalen Zahlen

	1	2	3	4	5	6	$\cdots$
1	1/1	2/1	3/1	4/1	5/1	6/1	
2	1/2	2/2	3/2	4/2	5/2	6/2	
3	1/3	2/3	3/3	4/3	5/3	6/3	$\cdots$
4	1/4	2/4	3/4	4/4	5/4	6/4	
5	1/5	2/5	3/5	4/5	5/5	6/5	
6	1/6	2/6	3/6	4/6	5/6	6/6	
$\vdots$				$\vdots$			$\ddots$

Exkursion: abzählbare und überabzählbare Mengen

Beispiele für abzählbar unendliche Mengen

- die Menge der ganzen Zahlen $\mathbb{Z}$:

$$c(i) = \begin{cases} i/2 & \text{falls } i \text{ gerade} \\ -(i+1)/2 & \text{falls } i \text{ ungerade} \end{cases}$$

$$0, -1, 1, -2, 2, -3, 3, -4, 4, \ldots,$$

- die Menge der rationalen Zahlen $\mathbb{Q}$

$$0, \frac{1}{1}, \frac{2}{1}, \frac{1}{2}, \frac{3}{1}, \ldots, \frac{i}{1}, \frac{i-1}{2}, \frac{i-2}{3}, \ldots \frac{1}{i}, \ldots$$

- Σ^*, die Menge der Wörter über einem endlichen Alphabet Σ
  - Zum Beispiel: $\{0,1\}^*$ in kanonischer Reihenfolge:

$$\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, \ldots$$

Exkursion: abzählbare und überabzählbare Mengen

Beispiele für abzählbar unendliche Mengen

- die Menge der ganzen Zahlen $\mathbb{Z}$:

$$c(i) = \begin{cases} i/2 & \text{falls } i \text{ gerade} \\ -(i+1)/2 & \text{falls } i \text{ ungerade} \end{cases}$$

$$0, -1, 1, -2, 2, -3, 3, -4, 4, \ldots,$$

- die Menge der rationalen Zahlen $\mathbb{Q}$

$$0, \frac{1}{1}, \frac{2}{1}, \frac{1}{2}, \frac{3}{1}, \ldots, \frac{i}{1}, \frac{i-1}{2}, \frac{i-2}{3}, \ldots \frac{1}{i}, \ldots$$

- Σ^*, die Menge der Wörter über einem endlichen Alphabet Σ
  - Zum Beispiel: $\{0,1\}^*$ in kanonischer Reihenfolge:

$$\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, \ldots$$

  - Zum Beispiel: $\{$  ,  , $\}^*$ in kanonischer Reihenfolge:

$\{\epsilon,$  ,  , ,  , , , , , , $\ldots\}$

Exkursion: abzählbare und überabzählbare Mengen

Beispiele für abzählbar unendliche Mengen

- die Menge der Gödelnummern, da Gödelnummern Wörter über dem Alphabet $\{0, 1\}$ sind

Exkursion: abzählbare und überabzählbare Mengen

Beispiele für abzählbar unendliche Mengen

- die Menge der Gödelnummern, da Gödelnummern Wörter über dem Alphabet $\{0, 1\}$ sind, und somit auch
- die Menge der TMen, weil jede TM durch eine eindeutige Gödelnummer beschrieben wird.

Exkursion: abzählbare und überabzählbare Mengen

Beispiele für abzählbar unendliche Mengen

- die Menge der Gödelnummern, da Gödelnummern Wörter über dem Alphabet $\{0,1\}$ sind, und somit auch
- die Menge der TMen, weil jede TM durch eine eindeutige Gödelnummer beschrieben wird.

Die i-te Gödelnummer in der kanonischen Reihenfolge von $\{0,1\}^*$ bezeichnen wir mit w_i, die i-te Turingmaschine mit M_i. Es gilt also $w_i = \langle M_i \rangle$.

Exkursion: abzählbare und überabzählbare Mengen

Nun betrachte die Potenzmenge $\mathcal{P}(\mathbb{N})$, also die Menge aller Teilmengen von $\mathbb{N}$.

Exkursion: abzählbare und überabzählbare Mengen

Nun betrachte die Potenzmenge $\mathcal{P}(\mathbb{N})$, also die Menge aller Teilmengen von $\mathbb{N}$.

Satz

Die Menge $\mathcal{P}(\mathbb{N})$ ist überabzählbar.

Exkursion: abzählbare und überabzählbare Mengen

Nun betrachte die Potenzmenge $\mathcal{P}(\mathbb{N})$, also die Menge aller Teilmengen von $\mathbb{N}$.

Satz

Die Menge $\mathcal{P}(\mathbb{N})$ ist überabzählbar.

Beweis (Diagonalisierung)

- Zum Zweck des Widerspruchs nehmen wir an, dass $\mathcal{P}(\mathbb{N})$ abzählbar ist.

Exkursion: abzählbare und überabzählbare Mengen

Nun betrachte die Potenzmenge $\mathcal{P}(\mathbb{N})$, also die Menge aller Teilmengen von $\mathbb{N}$.

Satz

Die Menge $\mathcal{P}(\mathbb{N})$ ist überabzählbar.

Beweis (Diagonalisierung)

- Zum Zweck des Widerspruchs nehmen wir an, dass $\mathcal{P}(\mathbb{N})$ abzählbar ist.
- Sei $S_0, S_1, S_2, S_3, \ldots$ eine Aufzählung von $\mathcal{P}(\mathbb{N})$.

Exkursion: abzählbare und überabzählbare Mengen

Nun betrachte die Potenzmenge $\mathcal{P}(\mathbb{N})$, also die Menge aller Teilmengen von $\mathbb{N}$.

Satz

Die Menge $\mathcal{P}(\mathbb{N})$ ist überabzählbar.

Beweis (Diagonalisierung)

- Zum Zweck des Widerspruchs nehmen wir an, dass $\mathcal{P}(\mathbb{N})$ abzählbar ist.
- Sei $S_0, S_1, S_2, S_3, \ldots$ eine Aufzählung von $\mathcal{P}(\mathbb{N})$.
- Wir definieren eine zweidimensionale unendliche Matrix $(A_{i,j})_{i\in\mathbb{N},j\in\mathbb{N}}$ mit

$$A_{i,j} = \begin{cases} 1 & \text{falls } j \in S_i \\ 0 & \text{sonst} \end{cases}$$

Exkursion: abzählbare und überabzählbare Mengen

Die Matrix A könnte etwa folgendermaßen aussehen

	0	1	2	3	4	5	6	
S_0	**0**	1	1	0	1	0	1	$\cdots$
S_1	1	**1**	1	0	1	0	1	$\cdots$
S_2	0	0	**1**	0	1	0	1	$\cdots$
S_3	0	1	1	**0**	0	0	1	$\cdots$
S_4	0	1	0	0	**1**	0	1	$\cdots$
S_5	0	1	1	0	1	**0**	0	$\cdots$
S_6	1	1	1	0	1	0	**0**	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Exkursion: abzählbare und überabzählbare Mengen

Die Matrix A könnte etwa folgendermaßen aussehen

$\{1, 2, 4, 6, \ldots\} =$

	0	1	2	3	4	5	6	
S_0	**0**	1	1	0	1	0	1	$\cdots$
S_1	1	**1**	1	0	1	0	1	$\cdots$
S_2	0	0	**1**	0	1	0	1	$\cdots$
S_3	0	1	1	**0**	0	0	1	$\cdots$
S_4	0	1	0	0	**1**	0	1	$\cdots$
S_5	0	1	1	0	1	**0**	0	$\cdots$
S_6	1	1	1	0	1	0	**0**	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Exkursion: abzählbare und überabzählbare Mengen

Die Matrix A könnte etwa folgendermaßen aussehen

		0	1	2	3	4	5	6	
$\{1,2,4,6,\ldots\} =$	S_0	**0**	1	1	0	1	0	1	$\cdots$
$\{0,1,2,4,6,\ldots\} =$	S_1	1	**1**	1	0	1	0	1	$\cdots$
	S_2	0	0	**1**	0	1	0	1	$\cdots$
	S_3	0	1	1	**0**	0	0	1	$\cdots$
	S_4	0	1	0	0	**1**	0	1	$\cdots$
	S_5	0	1	1	0	1	**0**	0	$\cdots$
	S_6	1	1	1	0	1	0	**0**	$\cdots$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Exkursion: abzählbare und überabzählbare Mengen

Die Matrix A könnte etwa folgendermaßen aussehen

	0	1	2	3	4	5	6	
S_0	**0**	1	1	0	1	0	1	$\cdots$
S_1	1	**1**	1	0	1	0	1	$\cdots$
S_2	0	0	**1**	0	1	0	1	$\cdots$
S_3	0	1	1	**0**	0	0	1	$\cdots$
S_4	0	1	0	0	**1**	0	1	$\cdots$
S_5	0	1	1	0	1	**0**	0	$\cdots$
S_6	1	1	1	0	1	0	**0**	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Wir definieren die Menge

$$S_{\text{diag}} = \{i \in \mathbb{N} \mid A_{i,i} = 1\}.$$

Exkursion: abzählbare und überabzählbare Mengen

Die Matrix A könnte etwa folgendermaßen aussehen

	0	1	2	3	4	5	6	
S_0	**0**	1	1	0	1	0	1	$\cdots$
S_1	1	**1**	1	0	1	0	1	$\cdots$
S_2	0	0	**1**	0	1	0	1	$\cdots$
S_3	0	1	1	**0**	0	0	1	$\cdots$
S_4	0	1	0	0	**1**	0	1	$\cdots$
S_5	0	1	1	0	1	**0**	0	$\cdots$
S_6	1	1	1	0	1	0	**0**	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		$S_{\text{diag}} = \{1, 2, 4, \ldots\}$

Wir definieren die Menge

$$S_{\text{diag}} = \{i \in \mathbb{N} \mid A_{i,i} = 1\}.$$

Exkursion: abzählbare und überabzählbare Mengen

Die Matrix A könnte etwa folgendermaßen aussehen

	0	1	2	3	4	5	6	
S_0	**0**	1	1	0	1	0	1	$\cdots$
S_1	1	**1**	1	0	1	0	1	$\cdots$
S_2	0	0	**1**	0	1	0	1	$\cdots$
S_3	0	1	1	**0**	0	0	1	$\cdots$
S_4	0	1	0	0	**1**	0	1	$\cdots$
S_5	0	1	1	0	1	**0**	0	$\cdots$
S_6	1	1	1	0	1	0	**0**	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		$S_{\text{diag}} = \{1, 2, 4, \ldots\}$

Wir definieren die Menge

$$S_{\text{diag}} = \{i \in \mathbb{N} \mid A_{i,i} = 1\}.$$

Das Komplement dieser Menge ist

$$\overline{S}_{\text{diag}} = \mathbb{N} \setminus S_{\text{diag}} = \{i \in \mathbb{N} \mid A_{i,i} = 0\}.$$

Exkursion: abzählbare und überabzählbare Mengen

- Beachte: Auch $\overline{S}_{\text{diag}}$ ist eine Teilmenge von $\mathbb{N}$ und kommt somit in der Aufzählung $S_1, S_2, \ldots$ von $\mathcal{P}(\mathbb{N})$ vor.

Exkursion: abzählbare und überabzählbare Mengen

- Beachte: Auch $\overline{S}_{\text{diag}}$ ist eine Teilmenge von $\mathbb{N}$ und kommt somit in der Aufzählung $S_1, S_2, \ldots$ von $\mathcal{P}(\mathbb{N})$ vor.
- Es gibt also ein $k \in \mathbb{N}$, so dass $\overline{S}_{\text{diag}} = S_k$.

Exkursion: abzählbare und überabzählbare Mengen

- Beachte: Auch $\overline{S}_{\text{diag}}$ ist eine Teilmenge von $\mathbb{N}$ und kommt somit in der Aufzählung $S_1, S_2, \ldots$ von $\mathcal{P}(\mathbb{N})$ vor.
- Es gibt also ein $k \in \mathbb{N}$, so dass $\overline{S}_{\text{diag}} = S_k$.
- Jetzt gibt es zwei Fälle, die jeweils zum Widerspruch führen.

Exkursion: abzählbare und überabzählbare Mengen

- Beachte: Auch $\overline{S}_{\text{diag}}$ ist eine Teilmenge von $\mathbb{N}$ und kommt somit in der Aufzählung $S_1, S_2, \ldots$ von $\mathcal{P}(\mathbb{N})$ vor.
- Es gibt also ein $k \in \mathbb{N}$, so dass $\overline{S}_{\text{diag}} = S_k$.
- Jetzt gibt es zwei Fälle, die jeweils zum Widerspruch führen.
  - Fall 1:

$$A_{k,k} = 1 \overset{\text{Def. } \overline{S}_{\text{diag}}}{\Rightarrow} k \notin \overline{S}_{\text{diag}} \Rightarrow k \notin S_k \overset{\text{Def. } A}{\Rightarrow} A_{k,k} = 0$$

Widerspruch!

Exkursion: abzählbare und überabzählbare Mengen

- Beachte: Auch $\overline{S}_{\text{diag}}$ ist eine Teilmenge von $\mathbb{N}$ und kommt somit in der Aufzählung $S_1, S_2, \ldots$ von $\mathcal{P}(\mathbb{N})$ vor.
- Es gibt also ein $k \in \mathbb{N}$, so dass $\overline{S}_{\text{diag}} = S_k$.
- Jetzt gibt es zwei Fälle, die jeweils zum Widerspruch führen.
  - Fall 1:

$$A_{k,k} = 1 \overset{\text{Def. } \overline{S}_{\text{diag}}}{\Rightarrow} k \notin \overline{S}_{\text{diag}} \Rightarrow k \notin S_k \overset{\text{Def. } A}{\Rightarrow} A_{k,k} = 0$$

Widerspruch!

  - Fall 2:

$$A_{k,k} = 0 \overset{\text{Def. } \overline{S}_{\text{diag}}}{\Rightarrow} k \in \overline{S}_{\text{diag}} \Rightarrow k \in S_k \overset{\text{Def. } A}{\Rightarrow} A_{k,k} = 1$$

Widerspruch!

Exkursion: abzählbare und überabzählbare Mengen

- Beachte: Auch $\overline{S}_{\text{diag}}$ ist eine Teilmenge von $\mathbb{N}$ und kommt somit in der Aufzählung $S_1, S_2, \ldots$ von $\mathcal{P}(\mathbb{N})$ vor.
- Es gibt also ein $k \in \mathbb{N}$, so dass $\overline{S}_{\text{diag}} = S_k$.
- Jetzt gibt es zwei Fälle, die jeweils zum Widerspruch führen.
  - Fall 1:

$$A_{k,k} = 1 \overset{\text{Def. } \overline{S}_{\text{diag}}}{\Rightarrow} k \notin \overline{S}_{\text{diag}} \Rightarrow k \notin S_k \overset{\text{Def. } A}{\Rightarrow} A_{k,k} = 0$$

Widerspruch!

  - Fall 2:

$$A_{k,k} = 0 \overset{\text{Def. } \overline{S}_{\text{diag}}}{\Rightarrow} k \in \overline{S}_{\text{diag}} \Rightarrow k \in S_k \overset{\text{Def. } A}{\Rightarrow} A_{k,k} = 1$$

Widerspruch!

- Folglich gibt es keine Aufzählung von $\mathcal{P}(\mathbb{N})$. □

Vorlesung BuK im WS 22/23, M. Grohe Seite 130 Version 26. Oktober 2022

Kardinalitäts-Battle

Welches Ansammlung hat größere Kardinalität?

Kardinalitäts-Battle

1. $\{1, \ldots, n\}$ $\mathbb{N}$

Kardinalitäts-Battle

1. $\{1, \ldots, n\}$ $<$ $\mathbb{N}$

Kardinalitäts-Battle

1. $\{1, \ldots, n\}$ $<$ $\mathbb{N}$

endlich — abzählbar unendlich

Kardinalitäts-Battle

1.	$\{1, \ldots, n\}$	$<$	$\mathbb{N}$
	endlich		abzählbar unendlich
2.	$\{☺\}^*$		$\mathcal{P}(\mathbb{N})$

Kardinalitäts-Battle

1.	$\{1, \ldots, n\}$	$<$	$\mathbb{N}$
	endlich		abzählbar unendlich
2.	$\{☺\}^*$	$<$	$\mathcal{P}(\mathbb{N})$

Kardinalitäts-Battle

1.	$\{1, \ldots, n\}$	$<$	$\mathbb{N}$
	endlich		abzählbar unendlich
2.	$\{\text{☺}\}^*$	$<$	$\mathcal{P}(\mathbb{N})$
	abzählbar unendlich		überabzählbar

Kardinalitäts-Battle

1.	$\{1, \ldots, n\}$	$<$	$\mathbb{N}$
	endlich		abzählbar unendlich
2.	$\{\text{☺}\}^*$	$<$	$\mathcal{P}(\mathbb{N})$
	abzählbar unendlich		überabzählbar
3.	$\mathbb{R}$		$\mathcal{P}(\mathbb{N})$

Kardinalitäts-Battle

1.	$\{1, \ldots, n\}$	$<$	$\mathbb{N}$
	endlich		abzählbar unendlich
2.	$\{\text{☺}\}^*$	$<$	$\mathcal{P}(\mathbb{N})$
	abzählbar unendlich		überabzählbar
3.	$\mathbb{R}$	$=$	$\mathcal{P}(\mathbb{N})$

Kardinalitäts-Battle

1.	$\{1, \ldots, n\}$	$<$	$\mathbb{N}$
	endlich		abzählbar unendlich
2.	$\{\text{☺}\}^*$	$<$	$\mathcal{P}(\mathbb{N})$
	abzählbar unendlich		überabzählbar
3.	$\mathbb{R}$	$=$	$\mathcal{P}(\mathbb{N})$
	überabzählbar		überabzählbar

Kardinalitäts-Battle

1.	$\{1, \ldots, n\}$	$<$	$\mathbb{N}$
	endlich		abzählbar unendlich
2.	$\{\text{☺}\}^*$	$<$	$\mathcal{P}(\mathbb{N})$
	abzählbar unendlich		überabzählbar
3.	$\mathbb{R}$	$=$	$\mathcal{P}(\mathbb{N})$
	überabzählbar		überabzählbar
4.	Graphen mit $\exists n V(G) = \{1, \ldots, n\}$		$\{0,1\}^*$

Kardinalitäts-Battle

1.	$\{1, \ldots, n\}$	$<$	$\mathbb{N}$
	endlich		abzählbar unendlich
2.	$\{\text{☺}\}^*$	$<$	$\mathcal{P}(\mathbb{N})$
	abzählbar unendlich		überabzählbar
3.	$\mathbb{R}$	$=$	$\mathcal{P}(\mathbb{N})$
	überabzählbar		überabzählbar
4.	Graphen mit $\exists n V(G) = \{1, \ldots, n\}$	$=$	$\{0,1\}^*$

Vorlesung BuK im WS 22/23, M. Grohe Seite 131 Version 26. Oktober 2022

Kardinalitäts-Battle

1.	$\{1, \ldots, n\}$	$<$	$\mathbb{N}$
	endlich		abzählbar unendlich
2.	$\{\text{☺}\}^*$	$<$	$\mathcal{P}(\mathbb{N})$
	abzählbar unendlich		überabzählbar
3.	$\mathbb{R}$	$=$	$\mathcal{P}(\mathbb{N})$
	überabzählbar		überabzählbar
4.	Graphen mit $\exists n V(G) = \{1, \ldots, n\}$	$=$	$\{0,1\}^*$
	abzählbar unendlich		abzählbar unendlich

Kardinalitäts-Battle

1.	$\{1, \ldots, n\}$	$<$	$\mathbb{N}$
	endlich		abzählbar unendlich
2.	$\{\text{☺}\}^*$	$<$	$\mathcal{P}(\mathbb{N})$
	abzählbar unendlich		überabzählbar
3.	$\mathbb{R}$	$=$	$\mathcal{P}(\mathbb{N})$
	überabzählbar		überabzählbar
4.	Graphen mit $\exists n V(G) = \{1, \ldots, n\}$	$=$	$\{0,1\}^*$
	abzählbar unendlich		abzählbar unendlich
5.	$\mathbb{N}^*$		$\{\mathbb{R}\}$

Kardinalitäts-Battle

1.	$\{1, \ldots, n\}$	$<$	$\mathbb{N}$
	endlich		abzählbar unendlich
2.	$\{\text{☺}\}^*$	$<$	$\mathcal{P}(\mathbb{N})$
	abzählbar unendlich		überabzählbar
3.	$\mathbb{R}$	$=$	$\mathcal{P}(\mathbb{N})$
	überabzählbar		überabzählbar
4.	Graphen mit $\exists n V(G) = \{1, \ldots, n\}$	$=$	$\{0,1\}^*$
	abzählbar unendlich		abzählbar unendlich
5.	$\mathbb{N}^*$	$>$	$\{\mathbb{R}\}$

Kardinalitäts-Battle

1.	$\{1, \ldots, n\}$	$<$	$\mathbb{N}$
	endlich		abzählbar unendlich
2.	$\{\text{☺}\}^*$	$<$	$\mathcal{P}(\mathbb{N})$
	abzählbar unendlich		überabzählbar
3.	$\mathbb{R}$	$=$	$\mathcal{P}(\mathbb{N})$
	überabzählbar		überabzählbar
4.	Graphen mit $\exists n V(G) = \{1, \ldots, n\}$	$=$	$\{0,1\}^*$
	abzählbar unendlich		abzählbar unendlich
5.	$\mathbb{N}^*$	$>$	$\{\mathbb{R}\}$
	abzählbar unendlich		einelementig

Kardinalitäts-Battle

1.	$\{1, \ldots, n\}$	$<$	$\mathbb{N}$
	endlich		abzählbar unendlich
2.	$\{\text{☺}\}^*$	$<$	$\mathcal{P}(\mathbb{N})$
	abzählbar unendlich		überabzählbar
3.	$\mathbb{R}$	$=$	$\mathcal{P}(\mathbb{N})$
	überabzählbar		überabzählbar
4.	Graphen mit $\exists n V(G) = \{1, \ldots, n\}$	$=$	$\{0,1\}^*$
	abzählbar unendlich		abzählbar unendlich
5.	$\mathbb{N}^*$	$>$	$\{\mathbb{R}\}$
	abzählbar unendlich		einelementig
6.	Graphen mit $V(G) = \mathbb{N}$		$\mathbb{N}$

Kardinalitäts-Battle

1.	$\{1, \ldots, n\}$	$<$	$\mathbb{N}$
	endlich		abzählbar unendlich
2.	$\{\text{☺}\}^*$	$<$	$\mathcal{P}(\mathbb{N})$
	abzählbar unendlich		überabzählbar
3.	$\mathbb{R}$	$=$	$\mathcal{P}(\mathbb{N})$
	überabzählbar		überabzählbar
4.	Graphen mit $\exists n V(G) = \{1, \ldots, n\}$	$=$	$\{0,1\}^*$
	abzählbar unendlich		abzählbar unendlich
5.	$\mathbb{N}^*$	$>$	$\{\mathbb{R}\}$
	abzählbar unendlich		einelementig
6.	Graphen mit $V(G) = \mathbb{N}$	$>$	$\mathbb{N}$

Vorlesung BuK im WS 22/23, M. Grohe Seite 131 Version 26. Oktober 2022

Kardinalitäts-Battle

1.	$\{1, \ldots, n\}$	$<$	$\mathbb{N}$
	endlich		abzählbar unendlich
2.	$\{\text{☺}\}^*$	$<$	$\mathcal{P}(\mathbb{N})$
	abzählbar unendlich		überabzählbar
3.	$\mathbb{R}$	$=$	$\mathcal{P}(\mathbb{N})$
	überabzählbar		überabzählbar
4.	Graphen mit $\exists n V(G) = \{1, \ldots, n\}$	$=$	$\{0,1\}^*$
	abzählbar unendlich		abzählbar unendlich
5.	$\mathbb{N}^*$	$>$	$\{\mathbb{R}\}$
	abzählbar unendlich		einelementig
6.	Graphen mit $V(G) = \mathbb{N}$	$>$	$\mathbb{N}$
	überabzählbar		abzählbar unendlich

Kardinalitäts-Battle

1.	$\{1, \ldots, n\}$	$<$	$\mathbb{N}$
	endlich		abzählbar unendlich
2.	$\{\text{☺}\}^*$	$<$	$\mathcal{P}(\mathbb{N})$
	abzählbar unendlich		überabzählbar
3.	$\mathbb{R}$	$=$	$\mathcal{P}(\mathbb{N})$
	überabzählbar		überabzählbar
4.	Graphen mit $\exists n V(G) = \{1, \ldots, n\}$	$=$	$\{0,1\}^*$
	abzählbar unendlich		abzählbar unendlich
5.	$\mathbb{N}^*$	$>$	$\{\mathbb{R}\}$
	abzählbar unendlich		einelementig
6.	Graphen mit $V(G) = \mathbb{N}$	$>$	$\mathbb{N}$
	überabzählbar		abzählbar unendlich
7.	endliche Graphen		$\mathbb{N}$

Vorlesung BuK im WS 22/23, M. Grohe Seite 131 Version 26. Oktober 2022

Kardinalitäts-Battle

1.	$\{1, \ldots, n\}$	$<$	$\mathbb{N}$
	endlich		abzählbar unendlich
2.	$\{\text{☺}\}^*$	$<$	$\mathcal{P}(\mathbb{N})$
	abzählbar unendlich		überabzählbar
3.	$\mathbb{R}$	$=$	$\mathcal{P}(\mathbb{N})$
	überabzählbar		überabzählbar
4.	Graphen mit $\exists n V(G) = \{1, \ldots, n\}$	$=$	$\{0,1\}^*$
	abzählbar unendlich		abzählbar unendlich
5.	$\mathbb{N}^*$	$>$	$\{\mathbb{R}\}$
	abzählbar unendlich		einelementig
6.	Graphen mit $V(G) = \mathbb{N}$	$>$	$\mathbb{N}$
	überabzählbar		abzählbar unendlich
7.	endliche Graphen	???	$\mathbb{N}$

Kardinalitäts-Battle

1.	$\{1, \ldots, n\}$	$<$	$\mathbb{N}$
	endlich		abzählbar unendlich
2.	$\{\text{☺}\}^*$	$<$	$\mathcal{P}(\mathbb{N})$
	abzählbar unendlich		überabzählbar
3.	$\mathbb{R}$	$=$	$\mathcal{P}(\mathbb{N})$
	überabzählbar		überabzählbar
4.	Graphen mit $\exists n V(G) = \{1, \ldots, n\}$	$=$	$\{0,1\}^*$
	abzählbar unendlich		abzählbar unendlich
5.	$\mathbb{N}^*$	$>$	$\{\mathbb{R}\}$
	abzählbar unendlich		einelementig
6.	Graphen mit $V(G) = \mathbb{N}$	$>$	$\mathbb{N}$
	überabzählbar		abzählbar unendlich
7.	endliche Graphen	???	$\mathbb{N}$
	keine Menge		abzählbar unendlich

Kardinalitäts-Battle

1.	$\{1, \ldots, n\}$	$<$	$\mathbb{N}$
	endlich		abzählbar unendlich
2.	$\{\text{☺}\}^*$	$<$	$\mathcal{P}(\mathbb{N})$
	abzählbar unendlich		überabzählbar
3.	$\mathbb{R}$	$=$	$\mathcal{P}(\mathbb{N})$
	überabzählbar		überabzählbar
4.	Graphen mit $\exists n V(G) = \{1, \ldots, n\}$	$=$	$\{0,1\}^*$
	abzählbar unendlich		abzählbar unendlich
5.	$\mathbb{N}^*$	$>$	$\{\mathbb{R}\}$
	abzählbar unendlich		einelementig
6.	Graphen mit $V(G) = \mathbb{N}$	$>$	$\mathbb{N}$
	überabzählbar		abzählbar unendlich
7.	endliche Graphen	???	$\mathbb{N}$
	keine Menge		abzählbar unendlich
8.	Gödelnummern		$\mathcal{P}(\{0,1\}^*)$

Kardinalitäts-Battle

1.	$\{1, \ldots, n\}$	$<$	$\mathbb{N}$
	endlich		abzählbar unendlich
2.	$\{\text{☺}\}^*$	$<$	$\mathcal{P}(\mathbb{N})$
	abzählbar unendlich		überabzählbar
3.	$\mathbb{R}$	$=$	$\mathcal{P}(\mathbb{N})$
	überabzählbar		überabzählbar
4.	Graphen mit $\exists n V(G) = \{1, \ldots, n\}$	$=$	$\{0,1\}^*$
	abzählbar unendlich		abzählbar unendlich
5.	$\mathbb{N}^*$	$>$	$\{\mathbb{R}\}$
	abzählbar unendlich		einelementig
6.	Graphen mit $V(G) = \mathbb{N}$	$>$	$\mathbb{N}$
	überabzählbar		abzählbar unendlich
7.	endliche Graphen	???	$\mathbb{N}$
	keine Menge		abzählbar unendlich
8.	Gödelnummern	$<$	$\mathcal{P}(\{0,1\}^*)$

Vorlesung BuK im WS 22/23, M. Grohe Seite 131 Version 26. Oktober 2022

Kardinalitäts-Battle

1.	$\{1, \ldots, n\}$	$<$	$\mathbb{N}$
	endlich		abzählbar unendlich
2.	$\{\text{☺}\}^*$	$<$	$\mathcal{P}(\mathbb{N})$
	abzählbar unendlich		überabzählbar
3.	$\mathbb{R}$	$=$	$\mathcal{P}(\mathbb{N})$
	überabzählbar		überabzählbar
4.	Graphen mit $\exists n V(G) = \{1, \ldots, n\}$	$=$	$\{0,1\}^*$
	abzählbar unendlich		abzählbar unendlich
5.	$\mathbb{N}^*$	$>$	$\{\mathbb{R}\}$
	abzählbar unendlich		einelementig
6.	Graphen mit $V(G) = \mathbb{N}$	$>$	$\mathbb{N}$
	überabzählbar		abzählbar unendlich
7.	endliche Graphen	???	$\mathbb{N}$
	keine Menge		abzählbar unendlich
8.	Gödelnummern	$<$	$\mathcal{P}(\{0,1\}^*)$
	abzählbar unendlich		überabzählbar

Kardinalitäts-Battle

1.	$\{1, \ldots, n\}$	$<$	$\mathbb{N}$
	endlich		abzählbar unendlich
2.	$\{\text{☺}\}^*$	$<$	$\mathcal{P}(\mathbb{N})$
	abzählbar unendlich		überabzählbar
3.	$\mathbb{R}$	$=$	$\mathcal{P}(\mathbb{N})$
	überabzählbar		überabzählbar
4.	Graphen mit $\exists n V(G) = \{1, \ldots, n\}$	$=$	$\{0,1\}^*$
	abzählbar unendlich		abzählbar unendlich
5.	$\mathbb{N}^*$	$>$	$\{\mathbb{R}\}$
	abzählbar unendlich		einelementig
6.	Graphen mit $V(G) = \mathbb{N}$	$>$	$\mathbb{N}$
	überabzählbar		abzählbar unendlich
7.	endliche Graphen	???	$\mathbb{N}$
	keine Menge		abzählbar unendlich
8.	Gödelnummern	$<$	$\mathcal{P}(\{0,1\}^*)$
	abzählbar unendlich		überabzählbar

Zwischenfrage für übermotivierte Zuhörende: Gibt es eine Mächtigkeit zwischen $|\mathbb{N}|$ und $|\mathbb{R}|$?

Wie viele verschiedene Entscheidungsprobleme gibt es?

Wie viele verschiedene Entscheidungsprobleme gibt es?

Jedes Entscheidungsproblem mit binär kodierter Eingabe entspricht einer Sprache über dem Alphabet $\{0,1\}$ und umgekehrt.

Sei $\mathcal{L}$ die Menge aller Sprachen (bzw. Entscheidungsprobleme) über $\{0,1\}^*$.

Wie viele verschiedene Entscheidungsprobleme gibt es?

Jedes Entscheidungsproblem mit binär kodierter Eingabe entspricht einer Sprache über dem Alphabet $\{0,1\}$ und umgekehrt.

Sei $\mathcal{L}$ die Menge aller Sprachen (bzw. Entscheidungsprobleme) über $\{0,1\}^*$.

Eine Sprache L über dem Alphabet $\{0,1\}$ ist eine Teilmenge von $\{0,1\}^*$.

Wie viele verschiedene Entscheidungsprobleme gibt es?

Jedes Entscheidungsproblem mit binär kodierter Eingabe entspricht einer Sprache über dem Alphabet $\{0,1\}$ und umgekehrt.

Sei $\mathcal{L}$ die Menge aller Sprachen (bzw. Entscheidungsprobleme) über $\{0,1\}^*$.

Eine Sprache L über dem Alphabet $\{0,1\}$ ist eine Teilmenge von $\{0,1\}^*$.

$\mathcal{L}$ ist somit die Menge aller Teilmengen, also die Potenzmenge über $\{0,1\}^*$, d.h $\mathcal{L} = \mathcal{P}(\{0,1\}^*)$.

Wie viele verschiedene Entscheidungsprobleme gibt es?

Jedes Entscheidungsproblem mit binär kodierter Eingabe entspricht einer Sprache über dem Alphabet $\{0,1\}$ und umgekehrt.

Sei $\mathcal{L}$ die Menge aller Sprachen (bzw. Entscheidungsprobleme) über $\{0,1\}^*$.

Eine Sprache L über dem Alphabet $\{0,1\}$ ist eine Teilmenge von $\{0,1\}^*$.

$\mathcal{L}$ ist somit die Menge aller Teilmengen, also die Potenzmenge über $\{0,1\}^*$, d.h $\mathcal{L} = \mathcal{P}(\{0,1\}^*)$.

Wir beobachten:

- $\{0,1\}^*$ hat dieselbe Mächtigkeit wie $\mathbb{N}$.

Wie viele verschiedene Entscheidungsprobleme gibt es?

Jedes Entscheidungsproblem mit binär kodierter Eingabe entspricht einer Sprache über dem Alphabet $\{0,1\}$ und umgekehrt.

Sei $\mathcal{L}$ die Menge aller Sprachen (bzw. Entscheidungsprobleme) über $\{0,1\}^*$.

Eine Sprache L über dem Alphabet $\{0,1\}$ ist eine Teilmenge von $\{0,1\}^*$.

$\mathcal{L}$ ist somit die Menge aller Teilmengen, also die Potenzmenge über $\{0,1\}^*$, d.h $\mathcal{L} = \mathcal{P}(\{0,1\}^*)$.

Wir beobachten:

- $\{0,1\}^*$ hat dieselbe Mächtigkeit wie $\mathbb{N}$.
- $\mathcal{L} = \mathcal{P}(\{0,1\}^*)$ hat somit dieselbe Mächtigkeit wie $\mathcal{P}(\mathbb{N})$.

Wie viele verschiedene Entscheidungsprobleme gibt es?

Jedes Entscheidungsproblem mit binär kodierter Eingabe entspricht einer Sprache über dem Alphabet $\{0,1\}$ und umgekehrt.

Sei $\mathcal{L}$ die Menge aller Sprachen (bzw. Entscheidungsprobleme) über $\{0,1\}^*$.

Eine Sprache L über dem Alphabet $\{0,1\}$ ist eine Teilmenge von $\{0,1\}^*$.

$\mathcal{L}$ ist somit die Menge aller Teilmengen, also die Potenzmenge über $\{0,1\}^*$, d.h $\mathcal{L} = \mathcal{P}(\{0,1\}^*)$.

Wir beobachten:

- $\{0,1\}^*$ hat dieselbe Mächtigkeit wie $\mathbb{N}$.
- $\mathcal{L} = \mathcal{P}(\{0,1\}^*)$ hat somit dieselbe Mächtigkeit wie $\mathcal{P}(\mathbb{N})$.

Die Menge der Entscheidungsprobleme $\mathcal{L}$ ist also überabzählbar.

Existenz unentscheidbarer Probleme

Zusammengefasst:

- Es gibt überabzählbar viele Sprachen.

Existenz unentscheidbarer Probleme

Zusammengefasst:

- Es gibt überabzählbar viele Sprachen.
- Aber es gibt nur abzählbar viele TMen.

Existenz unentscheidbarer Probleme

Zusammengefasst:

- Es gibt überabzählbar viele Sprachen.
- Aber es gibt nur abzählbar viele TMen.

Schlussfolgerung

Es gibt unentscheidbare Sprachen.

Existenz unentscheidbarer Probleme

Zusammengefasst:

- Es gibt überabzählbar viele Sprachen.
- Aber es gibt nur abzählbar viele TMen.

Schlussfolgerung

Es gibt unentscheidbare Sprachen.

- Die reine Existenz unentscheidbarer Probleme ist noch nicht dramatisch, denn es könnte sich ja um uninteressante, nicht praxis-relevante Probleme handeln.

Existenz unentscheidbarer Probleme

Zusammengefasst:

- Es gibt überabzählbar viele Sprachen.
- Aber es gibt nur abzählbar viele TMen.

Schlussfolgerung

Es gibt unentscheidbare Sprachen.

- Die reine Existenz unentscheidbarer Probleme ist noch nicht dramatisch, denn es könnte sich ja um uninteressante, nicht praxis-relevante Probleme handeln.
- Leider werden wir sehen, dass diese Hoffnung sich nicht bestätigt.

Das Halteproblem

Beim Halteproblem geht es darum, zu entscheiden, ob ein gegebenes Programm mit einer gegebenen Eingabe terminiert.

In der Notation der TMen ergibt sich die folgende formale Problemdefinition.

$$H = \{\langle M \rangle w \mid M \text{ hält auf } w\}.$$

Das Halteproblem

Beim Halteproblem geht es darum, zu entscheiden, ob ein gegebenes Programm mit einer gegebenen Eingabe terminiert.

In der Notation der TMen ergibt sich die folgende formale Problemdefinition.

$$H = \{\langle M \rangle w \mid M \text{ hält auf } w\}.$$

Es wäre äußerst hilfreich, wenn Compiler das Halteproblem entscheiden könnten. Wir werden jedoch sehen, dass dieses elementare Problem nicht entscheidbar ist.

Unentscheidbarkeit der Diagonalsprache

Zum Beweis der Unentscheidbarkeit des Halteproblems machen wir einen Umweg über die sogenannte *Diagonalsprache*.

$$\begin{aligned} D &= \{ w \in \{0,1\}^* \mid w = w_i \text{ und } M_i \text{ akzeptiert } w \text{ nicht}\} \\ &= \{\langle M\rangle \mid M \text{ akteptiert } \langle M\rangle \text{ nicht}\}. \end{aligned}$$

Anders gesagt, die i-te Gödelnummer w_i ist genau dann in D, wenn die i-te TM, also M_i, dieses Wort nicht akzeptiert.

Unentscheidbarkeit der Diagonalsprache

Zum Beweis der Unentscheidbarkeit des Halteproblems machen wir einen Umweg über die sogenannte *Diagonalsprache*.

$$\begin{aligned} D &= \{ w \in \{0,1\}^* \mid w = w_i \text{ und } M_i \text{ akzeptiert } w \text{ nicht}\} \\ &= \{\langle M\rangle \mid M \text{ akteptiert } \langle M\rangle \text{ nicht}\}. \end{aligned}$$

Anders gesagt, die i-te Gödelnummer w_i ist genau dann in D, wenn die i-te TM, also M_i, dieses Wort nicht akzeptiert.

Satz

Die Diagonalsprache D ist unentscheidbar.

Unentscheidbarkeit der Diagonalsprache – Intuition

Warum trägt die Sprache den Namen *Diagonalsprache*? –
Betrachte eine unendliche binäre Matrix A mit

$$A_{i,j} = \begin{cases} 1 & \text{falls } M_i \text{ das Wort } w_j \text{ akzeptiert} \\ 0 & \text{sonst} \end{cases}$$

Unentscheidbarkeit der Diagonalsprache – Intuition

Warum trägt die Sprache den Namen *Diagonalsprache*? –
Betrachte eine unendliche binäre Matrix A mit

$$A_{i,j} = \begin{cases} 1 & \text{falls } M_i \text{ das Wort } w_j \text{ akzeptiert} \\ 0 & \text{sonst} \end{cases}$$

Beispiel:

	w_0	w_1	w_2	w_3	w_4	
M_0	**0**	1	1	0	1	$\cdots$
M_1	1	**0**	1	0	1	$\cdots$
M_2	0	0	**1**	0	1	$\cdots$
M_3	0	1	1	**1**	0	$\cdots$
M_4	0	1	0	0	**0**	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		

Unentscheidbarkeit der Diagonalsprache – Intuition

Warum trägt die Sprache den Namen *Diagonalsprache*? –
Betrachte eine unendliche binäre Matrix A mit

$$A_{i,j} = \begin{cases} 1 & \text{falls } M_i \text{ das Wort } w_j \text{ akzeptiert} \\ 0 & \text{sonst} \end{cases}$$

Beispiel:

	w_0	w_1	w_2	w_3	w_4	
M_0	**0**	1	1	0	1	$\cdots$
M_1	1	**0**	1	0	1	$\cdots$
M_2	0	0	**1**	0	1	$\cdots$
M_3	0	1	1	**1**	0	$\cdots$
M_4	0	1	0	0	**0**	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		

Die Diagonalsprache lässt sich auf der Diagonale der Matrix ablesen. Es ist

$$D = \{w_i \mid A_{i,i} = 0\} .$$

Unentscheidbarkeit der Diagonalsprache – Beweis

Satz

Die Diagonalsprache D ist unentscheidbar.

Beweis

- Wir führen einen Widerspruchsbeweis und nehmen an, D ist entscheidbar.

Unentscheidbarkeit der Diagonalsprache – Beweis

Satz

Die Diagonalsprache D ist unentscheidbar.

Beweis

- Wir führen einen Widerspruchsbeweis und nehmen an, D ist entscheidbar.
- Dann gibt es eine TM M_j, die D entscheidet.

Unentscheidbarkeit der Diagonalsprache – Beweis

Satz

Die Diagonalsprache D ist unentscheidbar.

Beweis

- Wir führen einen Widerspruchsbeweis und nehmen an, D ist entscheidbar.
- Dann gibt es eine TM M_j, die D entscheidet.
- Wir starten die TM M_j mit der Eingabe w_j. Es ergeben sich zwei Fälle, die jeweils direkt zum Widerspruch führen.

Unentscheidbarkeit der Diagonalsprache – Beweis

Satz

Die Diagonalsprache D ist unentscheidbar.

Beweis

- Wir führen einen Widerspruchsbeweis und nehmen an, D ist entscheidbar.
- Dann gibt es eine TM M_j, die D entscheidet.
- Wir starten die TM M_j mit der Eingabe w_j. Es ergeben sich zwei Fälle, die jeweils direkt zum Widerspruch führen.
  - Fall 1:

$$w_j \in D \overset{M_j \text{ entsch. } D}{\Rightarrow} M_j \text{ akzeptiert } w_j$$

Unentscheidbarkeit der Diagonalsprache – Beweis

Satz

Die Diagonalsprache D ist unentscheidbar.

Beweis

- Wir führen einen Widerspruchsbeweis und nehmen an, D ist entscheidbar.
- Dann gibt es eine TM M_j, die D entscheidet.
- Wir starten die TM M_j mit der Eingabe w_j. Es ergeben sich zwei Fälle, die jeweils direkt zum Widerspruch führen.
  - Fall 1:

$$w_j \in D \overset{M_j \text{ entsch. } D}{\Rightarrow} M_j \text{ akzeptiert } w_j \overset{\text{Def. von } D}{\Rightarrow} w_j \notin D$$

Unentscheidbarkeit der Diagonalsprache – Beweis

Satz

Die Diagonalsprache D ist unentscheidbar.

Beweis

- Wir führen einen Widerspruchsbeweis und nehmen an, D ist entscheidbar.
- Dann gibt es eine TM M_j, die D entscheidet.
- Wir starten die TM M_j mit der Eingabe w_j. Es ergeben sich zwei Fälle, die jeweils direkt zum Widerspruch führen.
  - Fall 1:

$$w_j \in D \overset{M_j \text{ entsch. } D}{\Rightarrow} M_j \text{ akzeptiert } w_j \overset{\text{Def. von } D}{\Rightarrow} w_j \notin D$$

Widerspruch!

Unentscheidbarkeit der Diagonalsprache – Beweis

Satz

Die Diagonalsprache D ist unentscheidbar.

Beweis

- Wir führen einen Widerspruchsbeweis und nehmen an, D ist entscheidbar.
- Dann gibt es eine TM M_j, die D entscheidet.
- Wir starten die TM M_j mit der Eingabe w_j. Es ergeben sich zwei Fälle, die jeweils direkt zum Widerspruch führen.
  - Fall 1:

$$w_j \in D \overset{M_j \text{ entsch. } D}{\Rightarrow} M_j \text{ akzeptiert } w_j \overset{\text{Def. von } D}{\Rightarrow} w_j \notin D$$

Widerspruch!

  - Fall 2:

$$w_j \notin D \overset{M_j \text{ entsch. } D}{\Rightarrow} M_j \text{ akzeptiert } w_j \text{ nicht}$$

Unentscheidbarkeit der Diagonalsprache – Beweis

Satz

Die Diagonalsprache D ist unentscheidbar.

Beweis

- Wir führen einen Widerspruchsbeweis und nehmen an, D ist entscheidbar.
- Dann gibt es eine TM M_j, die D entscheidet.
- Wir starten die TM M_j mit der Eingabe w_j. Es ergeben sich zwei Fälle, die jeweils direkt zum Widerspruch führen.
  - Fall 1:

    $$w_j \in D \overset{M_j \text{ entsch. } D}{\Rightarrow} M_j \text{ akzeptiert } w_j \overset{\text{Def. von } D}{\Rightarrow} w_j \notin D$$

    Widerspruch!
  - Fall 2:

    $$w_j \notin D \overset{M_j \text{ entsch. } D}{\Rightarrow} M_j \text{ akzeptiert } w_j \text{ nicht} \overset{\text{Def. von } D}{\Rightarrow} w_j \in D$$

Unentscheidbarkeit der Diagonalsprache – Beweis

Satz

Die Diagonalsprache D ist unentscheidbar.

Beweis

- Wir führen einen Widerspruchsbeweis und nehmen an, D ist entscheidbar.
- Dann gibt es eine TM M_j, die D entscheidet.
- Wir starten die TM M_j mit der Eingabe w_j. Es ergeben sich zwei Fälle, die jeweils direkt zum Widerspruch führen.
  - Fall 1:

    $$w_j \in D \overset{M_j \text{ entsch. } D}{\Rightarrow} M_j \text{ akzeptiert } w_j \overset{\text{Def. von } D}{\Rightarrow} w_j \notin D$$

    Widerspruch!
  - Fall 2:

    $$w_j \notin D \overset{M_j \text{ entsch. } D}{\Rightarrow} M_j \text{ akzeptiert } w_j \text{ nicht} \overset{\text{Def. von } D}{\Rightarrow} w_j \in D$$

    Widerspruch!

Unentscheidbarkeit der Diagonalsprache – Beweis

Satz

Die Diagonalsprache D ist unentscheidbar.

Beweis

- Wir führen einen Widerspruchsbeweis und nehmen an, D ist entscheidbar.
- Dann gibt es eine TM M_j, die D entscheidet.
- Wir starten die TM M_j mit der Eingabe w_j. Es ergeben sich zwei Fälle, die jeweils direkt zum Widerspruch führen.
  - Fall 1:

    $$w_j \in D \overset{M_j \text{ entsch. } D}{\Rightarrow} M_j \text{ akzeptiert } w_j \overset{\text{Def. von } D}{\Rightarrow} w_j \notin D$$

    Widerspruch!
  - Fall 2:

    $$w_j \notin D \overset{M_j \text{ entsch. } D}{\Rightarrow} M_j \text{ akzeptiert } w_j \text{ nicht} \overset{\text{Def. von } D}{\Rightarrow} w_j \in D$$

    Widerspruch!
- Somit is D unentscheidbar. □

Das Halteproblem für JAVA-Programme

Eingabe:	String `method` (Name der zu überprüfenden JAVA-Methode) Array `parameters` von Objekten (die Parameter, die an die Methode übergeben werden)
Ausgabe:	`true`, falls `method` wirklich eine Methode in einer verfügbaren Klasse bezeichnet, `parameters` die richtige Zahl von Parametern mit den richtigen Typen enthält und die Methode `method` bei Eingabe `parameters` (irgendwann) anhält, `false` sonst.

In JAVA:

```
1  class Halt {
2      ...
3
4      static boolean halt(String method,
5                          Object[] parameters) {
6          ...
7      }
8      ...
9  }
```

Wir können annehmen, dass die Methode `halt` Zugriff auf den Quellcode von `method` hat.

Die Unentscheidbarkeit des Halteproblems

Satz

Es gibt kein JAVA-Programm, welches das Halteproblem löst.

Beweis des Satzes

Angenommen, es gibt ein Programm, welches das Halteproblem löst. Wir können annehmen, dass dies mittels einer Methode

```
static boolean halt(String method,Object[] parameters)
```

in einer Klasse `Halt` geschieht.

Beweis des Satzes

Angenommen, es gibt ein Programm, welches das Halteproblem löst. Wir können annehmen, dass dies mittels einer Methode

```
static boolean halt(String method,Object[] parameters)
```

in einer Klasse `Halt` geschieht.

Wir definieren eine neue Methode `diag` in einer Klasse `Diag`:

```
1  class Diag{
2      static void diag(String method) {
3          Object[] parameters = { method };
4                  // Eingaben bestehen nur aus
5                  // dem einen String ‘‘method’’.
6          if ( Halt.halt(method,parameters) ) {
7              while ( true ) {};
8                  // Wenn Methode method bei
9                  // Eingabe method hält,
10                 // dann laufe in Endlosschleife.
11         }
12     }
13 }
```

Was passiert beim Aufruf

```
Diag.diag("foo")
```

wenn `foo` der Name einer Methode ist, die einen String als Parameter erwartet?

Was passiert beim Aufruf

```
Diag.diag("foo")
```

wenn `foo` der Name einer Methode ist, die einen String als Parameter erwartet?

`Diag.diag("foo")` hält.
$\iff$ `Halt.halt("foo",{"foo"})` gibt `false` zurück.
$\iff$ `foo("foo")` hält nicht.

Was passiert beim Aufruf

```
Diag.diag("foo")
```

wenn `foo` der Name einer Methode ist, die einen String als Parameter erwartet?

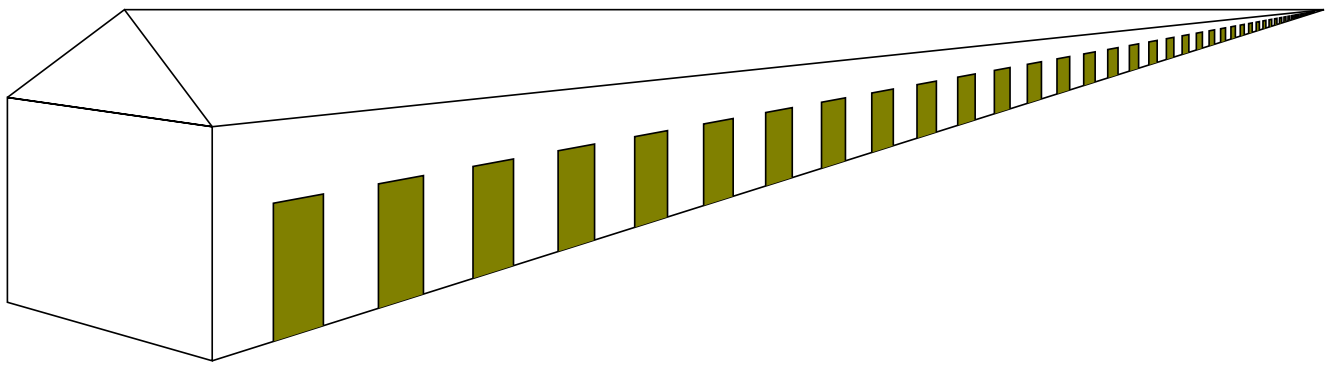
`Diag.diag("foo")` hält.
$\iff$ `Halt.halt("foo",{"foo"})` gibt `false` zurück.
$\iff$ `foo("foo")` hält nicht.

Also beim Aufruf `Diag.diag("Diag.diag")`:

`Diag.diag("Diag.diag")` hält.
$\iff$ `Diag.diag("Diag.diag")` hält nicht.

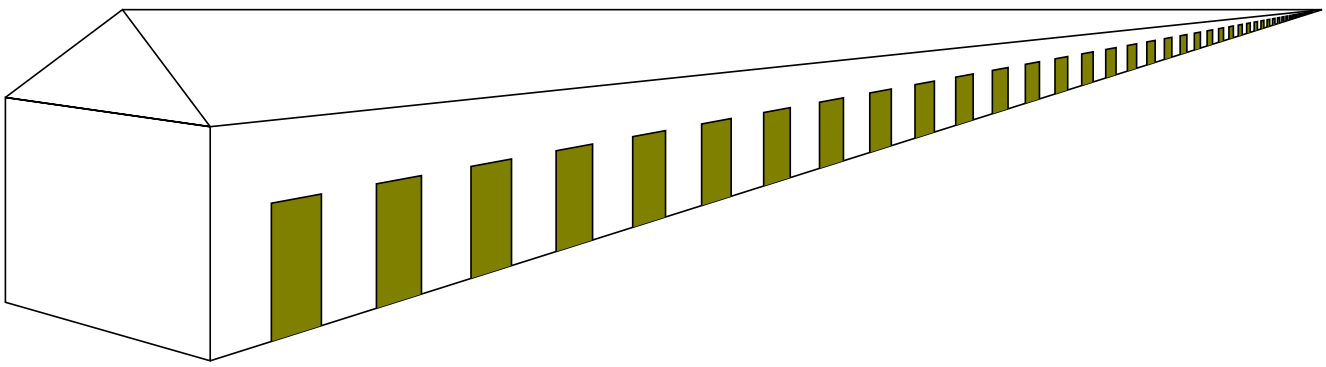
Widerspruch!

Hilbert's Hotel



Vorlesung 6
Unentscheidbarkeit des Halteproblems: Unterprogrammtechnik

Wdh.: Hilbert's Hotel



Wdh.: Abzählbarkeit

Definition (Abzählbare Menge)

Eine Menge M heißt abzählbar, wenn sie leer ist oder wenn es eine surjektive Funktion $c\colon \mathbb{N} \to M$ gibt.

Abzählbare Mengen: endlichen Mengen, $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$, $\{0,1\}^*$, Menge der Gödelnummern, die Menge der endlichen Teilmengen von $\mathbb{N}$.

Satz

Die Menge $\mathcal{P}(\mathbb{N})$ ist überabzählbar.

Überabzählbare Mengen: $\mathbb{R}$, $\mathcal{P}(\mathbb{N})$, $\mathcal{P}(\{0,1\}^*)$, Menge der Berechnungsprobleme.

Schlussfolgerung: Es gibt nicht-berechenbare Probleme.

Wdh.: Unentscheidbarkeit der Diagonalsprache

Die Diagonalsprache:

$$\begin{aligned} D &= \{\, w \in \{0,1\}^* \mid w = w_i \text{ und } M_i \text{ akzeptiert } w \text{ nicht}\} \\ &= \{\langle M\rangle \mid M \text{ akteptiert } \langle M\rangle \text{ nicht}\}. \end{aligned}$$

Wdh.: Unentscheidbarkeit der Diagonalsprache

Die Diagonalsprache:

$$\begin{aligned} D &= \{ w \in \{0,1\}^* \mid w = w_i \text{ und } M_i \text{ akzeptiert } w \text{ nicht}\} \\ &= \{\langle M \rangle \mid M \text{ akteptiert } \langle M \rangle \text{ nicht}\}. \end{aligned}$$

Satz

Die Diagonalsprache D ist unentscheidbar (= nicht entscheidbar).

Wdh.: Unentscheidbarkeit der Diagonalsprache

Die Diagonalsprache:

$$\begin{aligned} D &= \{ w \in \{0,1\}^* \mid w = w_i \text{ und } M_i \text{ akzeptiert } w \text{ nicht}\} \\ &= \{\langle M \rangle \mid M \text{ akteptiert } \langle M \rangle \text{ nicht}\}. \end{aligned}$$

Satz

Die Diagonalsprache D ist unentscheidbar (= nicht entscheidbar).

Beweisansatz: Diagonalisierung

Wdh.: Unentscheidbarkeit der Diagonalsprache

$$A_{i,j} = \begin{cases} 1 & \text{falls } M_i \text{ das Wort } w_j \text{ akzeptiert} \\ 0 & \text{sonst} \end{cases}$$

Wdh.: Unentscheidbarkeit der Diagonalsprache

$$A_{i,j} = \begin{cases} 1 & \text{falls } M_i \text{ das Wort } w_j \text{ akzeptiert} \\ 0 & \text{sonst} \end{cases}$$

Beispiel:

	w_0	w_1	w_2	w_3	w_4	
M_0	**0**	1	1	0	1	$\cdots$
M_1	1	**0**	1	0	1	$\cdots$
M_2	0	0	**1**	0	1	$\cdots$
M_3	0	1	1	**1**	0	$\cdots$
M_4	0	1	0	0	**0**	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		

Wdh.: Unentscheidbarkeit der Diagonalsprache

$$A_{i,j} = \begin{cases} 1 & \text{falls } M_i \text{ das Wort } w_j \text{ akzeptiert} \\ 0 & \text{sonst} \end{cases}$$

Beispiel:

	w_0	w_1	w_2	w_3	w_4	
M_0	**0**	1	1	0	1	$\cdots$
M_1	1	**0**	1	0	1	$\cdots$
M_2	0	0	**1**	0	1	$\cdots$
M_3	0	1	1	**1**	0	$\cdots$
M_4	0	1	0	0	**0**	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		

Die Diagonalsprache lässt sich auf der Diagonale der Matrix ablesen. Es ist

$$D = \{w_i \mid A_{i,i} = 0\} \ .$$

Das Komplement der Diagonalsprache

Das "Komplement" der Diagonalsprache ist

$$\begin{aligned} \overline{D} &= \{ w \in \{0,1\}^* \mid w = w_i \text{ und } M_i \text{ akzeptiert } w\} \\ &= \{\langle M \rangle \mid M \text{ akzeptiert } \langle M \rangle\}. \end{aligned}$$

Tatsächlich ist $\overline{D}$ nicht das Komplement on D in $\{0,1\}^*$, sondern lediglich in der Menge der Gödelnummern. Das heißt, sowohl D als auch $\overline{D}$ bestehen nur aus Gödelnummern, und eine Gödelnummer w_i ist genau dann in $\overline{D}$, wenn sie nicht in D ist.

Das Komplement der Diagonalsprache

Das "Komplement" der Diagonalsprache ist

$$\begin{aligned}\overline{D} &= \{ w \in \{0,1\}^* \mid w = w_i \text{ und } M_i \text{ akzeptiert } w\} \\ &= \{\langle M\rangle \mid M \text{ akzeptiert } \langle M\rangle\}.\end{aligned}$$

Tatsächlich ist $\overline{D}$ nicht das Komplement on D in $\{0,1\}^*$, sondern lediglich in der Menge der Gödelnummern. Das heißt, sowohl D als auch $\overline{D}$ bestehen nur aus Gödelnummern, und eine Gödelnummer w_i ist genau dann in $\overline{D}$, wenn sie nicht in D ist.

Satz

$\overline{D}$ ist unentscheidbar.

Unentscheidbarkeit des Diagonalsprachenkomplements

Satz

$\overline{D}$ ist unentscheidbar.

Unentscheidbarkeit des Diagonalsprachenkomplements

Satz

$\overline{D}$ ist unentscheidbar.

Beweis

- Zum Widerspruch nehmen wir an, es gibt eine TM $M_{\overline{D}}$, welche die Sprache $\overline{D}$ entscheidet.

Unentscheidbarkeit des Diagonalsprachenkomplements

Satz

$\overline{D}$ ist unentscheidbar.

Beweis

- Zum Widerspruch nehmen wir an, es gibt eine TM $M_{\overline{D}}$, welche die Sprache $\overline{D}$ entscheidet.
- Gemäß der Definition *entscheidbarer Sprachen* hält $M_{\overline{D}}$ auf jeder Eingabe w und akzeptiert genau dann, wenn $w \in \overline{D}$.

Unentscheidbarkeit des Diagonalsprachenkomplements

Satz

$\overline{D}$ ist unentscheidbar.

Beweis

- Zum Widerspruch nehmen wir an, es gibt eine TM $M_{\overline{D}}$, welche die Sprache $\overline{D}$ entscheidet.
- Gemäß der Definition *entscheidbarer Sprachen* hält $M_{\overline{D}}$ auf jeder Eingabe w und akzeptiert genau dann, wenn $w \in \overline{D}$.
- Wir konstruieren nun eine TM M, die $M_{\overline{D}}$ als Unterprogramm verwendet.

Unentscheidbarkeit des Diagonalsprachenkomplements

Satz

$\overline{D}$ ist unentscheidbar.

Beweis

- Zum Widerspruch nehmen wir an, es gibt eine TM $M_{\overline{D}}$, welche die Sprache $\overline{D}$ entscheidet.
- Gemäß der Definition *entscheidbarer Sprachen* hält $M_{\overline{D}}$ auf jeder Eingabe w und akzeptiert genau dann, wenn $w \in \overline{D}$.
- Wir konstruieren nun eine TM M, die $M_{\overline{D}}$ als Unterprogramm verwendet.
- M prüft zunächst, ob die Eingabe eine Gödelnummer ist und verwirft, wenn dass nicht der Fall ist.

Unentscheidbarkeit des Diagonalsprachenkomplements

Satz

$\overline{D}$ ist unentscheidbar.

Beweis

- Zum Widerspruch nehmen wir an, es gibt eine TM $M_{\overline{D}}$, welche die Sprache $\overline{D}$ entscheidet.
- Gemäß der Definition *entscheidbarer Sprachen* hält $M_{\overline{D}}$ auf jeder Eingabe w und akzeptiert genau dann, wenn $w \in \overline{D}$.
- Wir konstruieren nun eine TM M, die $M_{\overline{D}}$ als Unterprogramm verwendet.
- M prüft zunächst, ob die Eingabe eine Gödelnummer ist und verwirft, wenn dass nicht der Fall ist.
- Sonst startet M das Unterprogramm $M_{\overline{D}}$ auf der vorliegenden Eingabe und negiert anschließend die Ausgabe von $M_{\overline{D}}$.

Unentscheidbarkeit des Diagonalsprachenkomplements

Satz

$\overline{D}$ ist unentscheidbar.

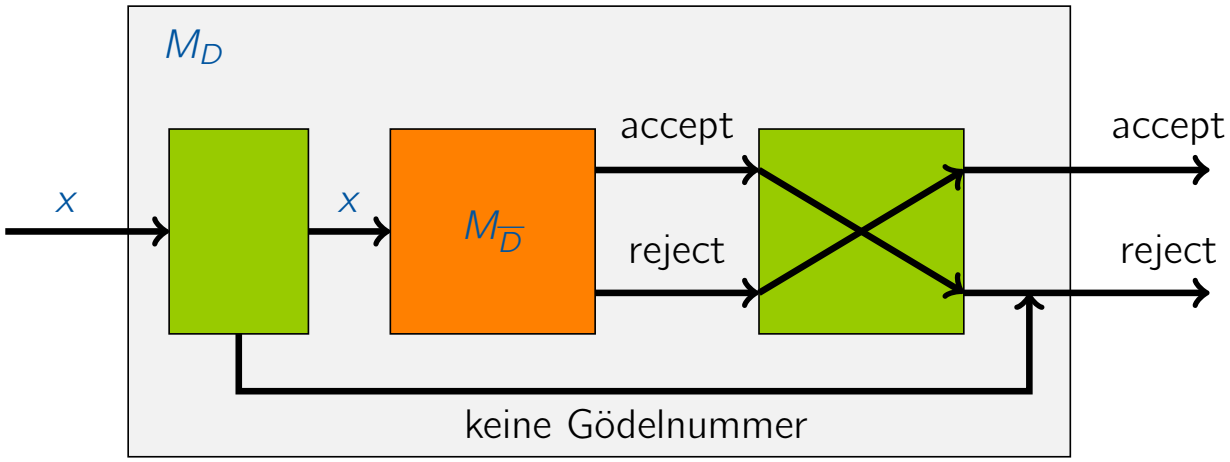
Beweis

- Zum Widerspruch nehmen wir an, es gibt eine TM $M_{\overline{D}}$, welche die Sprache $\overline{D}$ entscheidet.
- Gemäß der Definition *entscheidbarer Sprachen* hält $M_{\overline{D}}$ auf jeder Eingabe w und akzeptiert genau dann, wenn $w \in \overline{D}$.
- Wir konstruieren nun eine TM M, die $M_{\overline{D}}$ als Unterprogramm verwendet.
- M prüft zunächst, ob die Eingabe eine Gödelnummer ist und verwirft, wenn dass nicht der Fall ist.
- Sonst startet M das Unterprogramm $M_{\overline{D}}$ auf der vorliegenden Eingabe und negiert anschließend die Ausgabe von $M_{\overline{D}}$.
- Die TM M entscheidet nun offensichtlich D. Ein Widerspruch zur Unentscheidbarkeit von D. □

Vorlesung BuK im WS 22/23, M. Grohe Seite 150 Version 26. Oktober 2022

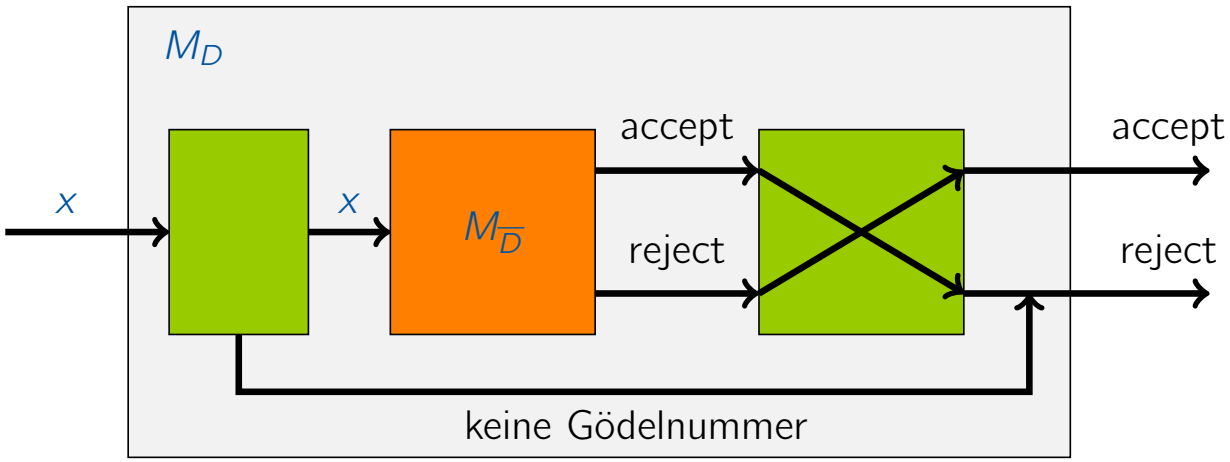
Unentscheidbarkeit des Komplements der Diagonalsprache

Illustration: Aus $M_{\overline{D}}$ konstruieren wir M_D.



Unentscheidbarkeit des Komplements der Diagonalsprache

Illustration: Aus $M_{\overline{D}}$ konstruieren wir M_D.



Aber die Existenz von M_D steht im Widerspruch zur Unentscheidbarkeit von D. Damit kann es $M_{\overline{D}}$ nicht geben, also ist $\overline{D}$ nicht entscheidbar.

Die Beweistechnik aus diesem Satz lässt sich allgemein wie folgt zusammenfassen:

Unterprogrammtechnik

Die Beweistechnik aus diesem Satz lässt sich allgemein wie folgt zusammenfassen:

Unterprogrammtechnik zum Beweis von Unentscheidbarkeit

Um nachzuweisen, dass eine Sprache L unentscheidbar ist, genügt es zu zeigen, dass man durch Unterprogrammaufruf einer TM M_L, die L entscheidet, ein andere Sprache L' entscheiden kann, von der bereits bewiesen wurde, dass sie unentscheidbar ist.

Unterprogrammtechnik

Die Beweistechnik aus diesem Satz lässt sich allgemein wie folgt zusammenfassen:

Unterprogrammtechnik zum Beweis von Unentscheidbarkeit

Um nachzuweisen, dass eine Sprache L unentscheidbar ist, genügt es zu zeigen, dass man durch Unterprogrammaufruf einer TM M_L, die L entscheidet, ein andere Sprache L' entscheiden kann, von der bereits bewiesen wurde, dass sie unentscheidbar ist.

Im Folgenden demonstrieren wir die Unterprogrammtechnik an einigen Beispielsprachen, inklusive dem Halteproblem.

Das Halteproblem

Das Halteproblem ist wie folgt definiert:

$$H \;=\; \{\langle M\rangle w \mid M \text{ hält auf } w\} \;.$$

Das Halteproblem

Das Halteproblem ist wie folgt definiert:

$$H \;=\; \{\langle M\rangle w \mid M \text{ hält auf } w\} \;.$$

Satz

Das Halteproblem H ist unentscheidbar.

Unentscheidbarkeit des Halteproblems

Satz

Das Halteproblem H ist unentscheidbar.

Beweis

Wir nutzen die Unterprogrammtechnik:

- Sei M_H eine TM, die H entscheidet, also eine TM, die auf jeder Eingabe hält und nur Eingaben der Form $\langle M\rangle w$ akzeptiert, bei denen M auf w hält.

Unentscheidbarkeit des Halteproblems

Satz

Das Halteproblem H ist unentscheidbar.

Beweis

Wir nutzen die Unterprogrammtechnik:

- Sei M_H eine TM, die H entscheidet, also eine TM, die auf jeder Eingabe hält und nur Eingaben der Form $\langle M \rangle w$ akzeptiert, bei denen M auf w hält.
- Wir konstruieren eine TM $M_{\overline{D}}$ mit M_H als Unterprogramm, die $\overline{D}$ entscheidet, was im Widerspruch zur Nicht-Berechenbarkeit von $\overline{D}$ steht.

Unentscheidbarkeit des Halteproblems

Satz

Das Halteproblem H ist unentscheidbar.

Beweis

Wir nutzen die Unterprogrammtechnik:

- Sei M_H eine TM, die H entscheidet, also eine TM, die auf jeder Eingabe hält und nur Eingaben der Form $\langle M \rangle w$ akzeptiert, bei denen M auf w hält.
- Wir konstruieren eine TM $M_{\overline{D}}$ mit M_H als Unterprogramm, die $\overline{D}$ entscheidet, was im Widerspruch zur Nicht-Berechenbarkeit von $\overline{D}$ steht.

Aus diesem Widerspruch ergibt sich die Nicht-Existenz der TM M_H.

Unentscheidbarkeit des Halteproblems – Beweis

Algorithmus der TM $M_{\overline{D}}$ mit Unterprogramm M_H:

1) Teste, ob w Gödelnummer. Wenn nicht, verwerfe.
   Sonst sei M die TM mit $w = \langle M \rangle$

Unentscheidbarkeit des Halteproblems – Beweis

Algorithmus der TM $M_{\overline{D}}$ mit Unterprogramm M_H:

1) Teste, ob w Gödelnummer. Wenn nicht, verwerfe.
   Sonst sei M die TM mit $w = \langle M \rangle$

2) Starte M_H als Unterprogramm mit Eingabe $ww = \langle M \rangle w$.

Unentscheidbarkeit des Halteproblems – Beweis

Algorithmus der TM $M_{\overline{D}}$ mit Unterprogramm M_H:

1) Teste, ob w Gödelnummer. Wenn nicht, verwerfe.
   Sonst sei M die TM mit $w = \langle M \rangle$
2) Starte M_H als Unterprogramm mit Eingabe $ww = \langle M \rangle w$.
   3.1) Falls M_H akzeptiert, so simuliere das Verhalten von M auf w mittels universeller TM U.

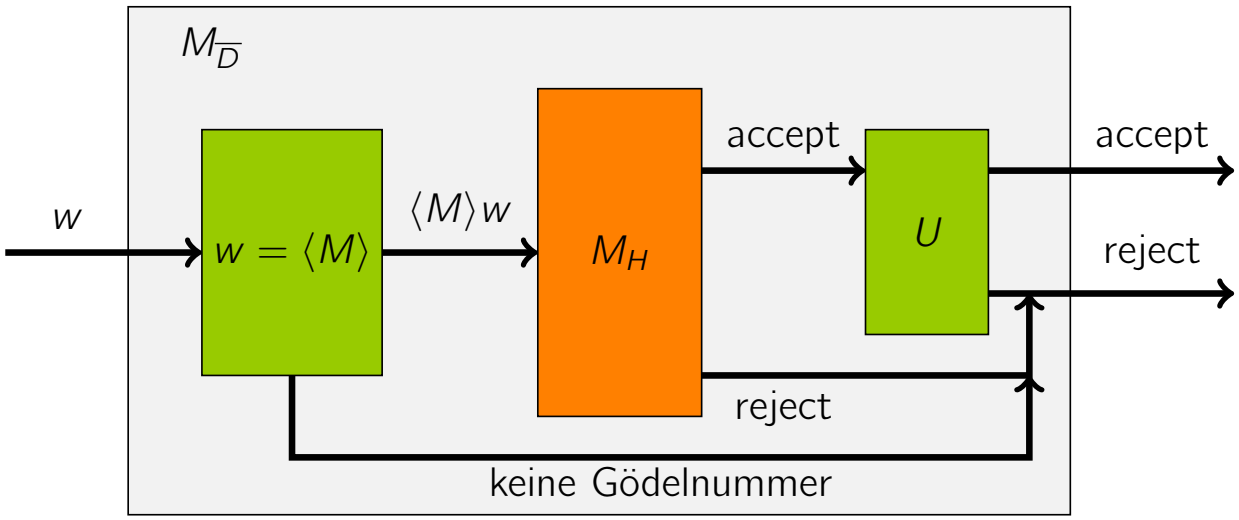
Unentscheidbarkeit des Halteproblems – Beweis

Algorithmus der TM $M_{\overline{D}}$ mit Unterprogramm M_H:

1) Teste, ob w Gödelnummer. Wenn nicht, verwerfe.
   Sonst sei M die TM mit $w = \langle M \rangle$
2) Starte M_H als Unterprogramm mit Eingabe $ww = \langle M \rangle w$.
   3.1) Falls M_H akzeptiert, so simuliere das Verhalten von M auf w mittels universeller TM U.
   3.2) Falls M_H verwirft, so verwirf die Eingabe.

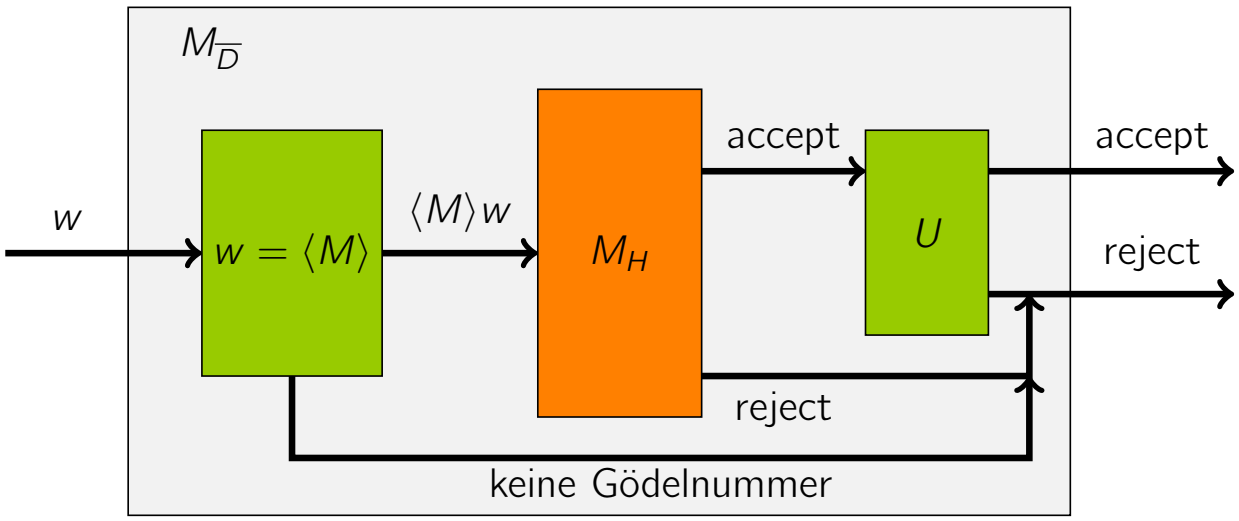
Unentscheidbarkeit des Halteproblems

Illustration: Aus M_H konstruieren wir $M_{\overline{D}}$.



Unentscheidbarkeit des Halteproblems

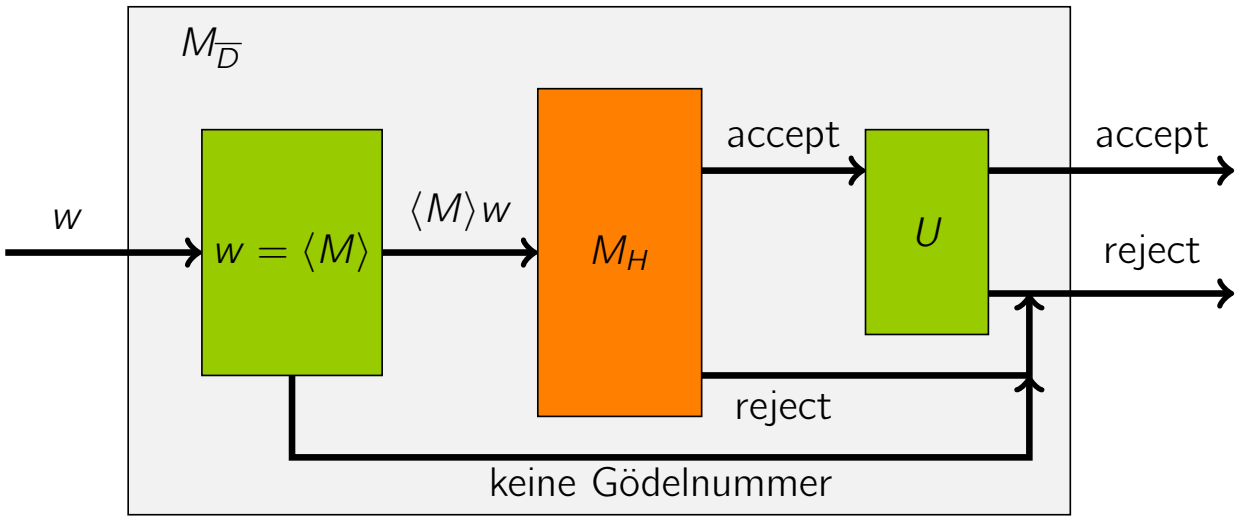
Illustration: Aus M_H konstruieren wir $M_{\overline{D}}$.



Aber die Existenz von $M_{\overline{D}}$ steht im Widerspruch zur Unentscheidbarkeit von $\overline{D}$. Damit kann es M_H nicht geben und das Halteproblem H ist nicht entscheidbar.

Unentscheidbarkeit des Halteproblems

Illustration: Aus M_H konstruieren wir $M_{\overline{D}}$.



Aber die Existenz von $M_{\overline{D}}$ steht im Widerspruch zur Unentscheidbarkeit von $\overline{D}$. Damit kann es M_H nicht geben und das Halteproblem H ist nicht entscheidbar.

Anmerkung: Der Test, ob w eine Gödelnummer ist ist nicht unbedingt erforderlich, weil M_H immer verwirft, wenn der erste Teil der Eingabe keine Gödelnummer ist.

Unentscheidbarkeit des Halteproblems – Forts. Beweis

Wir müssen zeigen, dass $M_{\overline{D}}$ korrekt arbeitet.

Für die Korrektheit ist zu zeigen:

1. $w \in \overline{D} \Rightarrow M_{\overline{D}}$ akzeptiert w
2. $w \notin \overline{D} \Rightarrow M_{\overline{D}}$ verwirft w

Unentscheidbarkeit des Halteproblems – Forts. Beweis

Nehmen wir an, $w = \langle M \rangle$ für eine TM M.

Sonst $w \notin \overline{D}$ und $M_{\overline{D}}$ verwirft D.

Unentscheidbarkeit des Halteproblems – Forts. Beweis

Nehmen wir an, $w = \langle M \rangle$ für eine TM M.

Sonst $w \notin \overline{D}$ und $M_{\overline{D}}$ verwirft D.

Es gilt

$$w \in \overline{D} \quad \Rightarrow$$

Unentscheidbarkeit des Halteproblems – Forts. Beweis

Nehmen wir an, $w = \langle M \rangle$ für eine TM M.

Sonst $w \notin \overline{D}$ und $M_{\overline{D}}$ verwirft D.

Es gilt

$$w \in \overline{D} \quad \Rightarrow \quad M \text{ akzeptiert } w.$$

Unentscheidbarkeit des Halteproblems – Forts. Beweis

Nehmen wir an, $w = \langle M \rangle$ für eine TM M.

Sonst $w \notin \overline{D}$ und $M_{\overline{D}}$ verwirft D.

Es gilt

$$\begin{aligned} w \in \overline{D} \quad &\Rightarrow \quad M \text{ akzeptiert } w. \\ &\Rightarrow \quad M_H \text{ und } U \text{ akzeptieren } \langle M \rangle w. \end{aligned}$$

Unentscheidbarkeit des Halteproblems – Forts. Beweis

Nehmen wir an, $w = \langle M \rangle$ für eine TM M.

Sonst $w \notin \overline{D}$ und $M_{\overline{D}}$ verwirft D.

Es gilt

$$\begin{aligned} w \in \overline{D} &\Rightarrow M \text{ akzeptiert } w. \\ &\Rightarrow M_H \text{ und } U \text{ akzeptieren } \langle M \rangle w. \\ &\Rightarrow M_{\overline{D}} \text{ akzeptiert } w. \end{aligned}$$

Unentscheidbarkeit des Halteproblems – Forts. Beweis

Nehmen wir an, $w = \langle M \rangle$ für eine TM M.

Sonst $w \notin \overline{D}$ und $M_{\overline{D}}$ verwirft D.

Es gilt

$$\begin{aligned} w \in \overline{D} &\Rightarrow M \text{ akzeptiert } w. \\ &\Rightarrow M_H \text{ und } U \text{ akzeptieren } \langle M \rangle w. \\ &\Rightarrow M_{\overline{D}} \text{ akzeptiert } w. \end{aligned}$$

$$w \notin \overline{D} \quad \Rightarrow$$

Unentscheidbarkeit des Halteproblems – Forts. Beweis

Nehmen wir an, $w = \langle M \rangle$ für eine TM M.

Sonst $w \notin \overline{D}$ und $M_{\overline{D}}$ verwirft D.

Es gilt

$$\begin{aligned} w \in \overline{D} &\Rightarrow M \text{ akzeptiert } w. \\ &\Rightarrow M_H \text{ und } U \text{ akzeptieren } \langle M \rangle w. \\ &\Rightarrow M_{\overline{D}} \text{ akzeptiert } w. \end{aligned}$$

$$w \notin \overline{D} \Rightarrow M \text{ akzeptiert } w \text{ nicht.}$$

Unentscheidbarkeit des Halteproblems – Forts. Beweis

Nehmen wir an, $w = \langle M \rangle$ für eine TM M.

Sonst $w \notin \overline{D}$ und $M_{\overline{D}}$ verwirft D.

Es gilt

$$\begin{aligned} w \in \overline{D} &\Rightarrow M \text{ akzeptiert } w. \\ &\Rightarrow M_H \text{ und } U \text{ akzeptieren } \langle M \rangle w. \\ &\Rightarrow M_{\overline{D}} \text{ akzeptiert } w. \end{aligned}$$

$$\begin{aligned} w \notin \overline{D} &\Rightarrow M \text{ akzeptiert } w \text{ nicht.} \\ &\Rightarrow (\, M \text{ hält nicht auf } w \,) \text{ oder } (\, M \text{ verwirft } w \,). \end{aligned}$$

Unentscheidbarkeit des Halteproblems – Forts. Beweis

Nehmen wir an, $w = \langle M \rangle$ für eine TM M.

Sonst $w \notin \overline{D}$ und $M_{\overline{D}}$ verwirft D.

Es gilt

$$\begin{aligned} w \in \overline{D} &\Rightarrow M \text{ akzeptiert } w. \\ &\Rightarrow M_H \text{ und } U \text{ akzeptieren } \langle M \rangle w. \\ &\Rightarrow M_{\overline{D}} \text{ akzeptiert } w. \end{aligned}$$

$$\begin{aligned} w \notin \overline{D} &\Rightarrow M \text{ akzeptiert } w \text{ nicht}. \\ &\Rightarrow (\, M \text{ hält nicht auf } w \,) \text{ oder } (\, M \text{ verwirft } w \,). \\ &\Rightarrow (\, M_H \text{ verwirft } \langle M \rangle w \,) \text{ oder} \\ &\qquad (\, M_H \text{ akzeptiert und } U \text{ verwirft } \langle M \rangle w \,). \end{aligned}$$

Unentscheidbarkeit des Halteproblems – Forts. Beweis

Nehmen wir an, $w = \langle M \rangle$ für eine TM M.

Sonst $w \notin \overline{D}$ und $M_{\overline{D}}$ verwirft D.

Es gilt

$$\begin{aligned} w \in \overline{D} &\Rightarrow M \text{ akzeptiert } w. \\ &\Rightarrow M_H \text{ und } U \text{ akzeptieren } \langle M \rangle w. \\ &\Rightarrow M_{\overline{D}} \text{ akzeptiert } w. \end{aligned}$$

$$\begin{aligned} w \notin \overline{D} &\Rightarrow M \text{ akzeptiert } w \text{ nicht}. \\ &\Rightarrow (\, M \text{ hält nicht auf } w \,) \text{ oder } (\, M \text{ verwirft } w \,). \\ &\Rightarrow (\, M_H \text{ verwirft } \langle M \rangle w \,) \text{ oder} \\ &\qquad (\, M_H \text{ akzeptiert und } U \text{ verwirft } \langle M \rangle w \,). \\ &\Rightarrow M_{\overline{D}} \text{ verwirft } w. \end{aligned}$$

□

Unentscheidbarkeit des speziellen Halteproblems

Das spezielle Halteproblem ist definiert als

$$H_\epsilon \;=\; \{\langle M\rangle \mid M \text{ hält auf Eingabe } \epsilon\} \;.$$

Unentscheidbarkeit des speziellen Halteproblems

Das spezielle Halteproblem ist definiert als

$$H_\epsilon \;=\; \{\langle M\rangle \mid M \text{ hält auf Eingabe } \epsilon\} \;.$$

Satz

Das spezielle Halteproblem H_ϵ ist unentscheidbar.

Unentscheidbarkeit des speziellen Halteproblems

Das spezielle Halteproblem ist definiert als

$$H_\epsilon \;=\; \{\langle M\rangle \mid M \text{ hält auf Eingabe } \epsilon\} \;.$$

Satz

Das spezielle Halteproblem H_ϵ ist unentscheidbar.

Beweis

Wir nutzen die Unterprogrammtechnik. Aus einer TM M_ϵ, die H_ϵ entscheidet, konstruieren wir eine TM M_H, die das unentscheidbare Halteproblem entscheiden würde.

Unentscheidbark. des speziellen Halteproblems – Beweis

Die TM M_H mit Unterprogramm M_ϵ arbeitet wie folgt:

1) Teste, ob Eingabewort x mit einer Gödelnummer beginnt. Wenn nicht, verwerfe.
   Sonst sei M die TM und $w \in \Sigma^*$ mit $x = \langle M\rangle w$

Unentscheidbark. des speziellen Halteproblems – Beweis

Die TM M_H mit Unterprogramm M_ϵ arbeitet wie folgt:

1) Teste, ob Eingabewort x mit einer Gödelnummer beginnt. Wenn nicht, verwerfe.
   Sonst sei M die TM und $w \in \Sigma^*$ mit $x = \langle M \rangle w$
2) Berechne die Gödelnummer einer TM M_w^* mit den folgenden Eigenschaften:

Unentscheidbark. des speziellen Halteproblems – Beweis

Die TM M_H mit Unterprogramm M_ϵ arbeitet wie folgt:

1) Teste, ob Eingabewort x mit einer Gödelnummer beginnt. Wenn nicht, verwerfe.
   Sonst sei M die TM und $w \in \Sigma^*$ mit $x = \langle M \rangle w$
2) Berechne die Gödelnummer einer TM M_w^* mit den folgenden Eigenschaften:

   - Falls M_w^* die Eingabe ϵ erhält, so schreibt sie das Wort w aufs Band und simuliert die TM M mit der Eingabe w.

Unentscheidbark. des speziellen Halteproblems – Beweis

Die TM M_H mit Unterprogramm M_ϵ arbeitet wie folgt:

1) Teste, ob Eingabewort x mit einer Gödelnummer beginnt. Wenn nicht, verwerfe.
   Sonst sei M die TM und $w \in \Sigma^*$ mit $x = \langle M \rangle w$
2) Berechne die Gödelnummer einer TM M_w^* mit den folgenden Eigenschaften:
   - Falls M_w^* die Eingabe ϵ erhält, so schreibt sie das Wort w aufs Band und simuliert die TM M mit der Eingabe w.
   - Bei anderen Eingaben kann sich M_w^* beliebig verhalten.

Unentscheidbark. des speziellen Halteproblems – Beweis

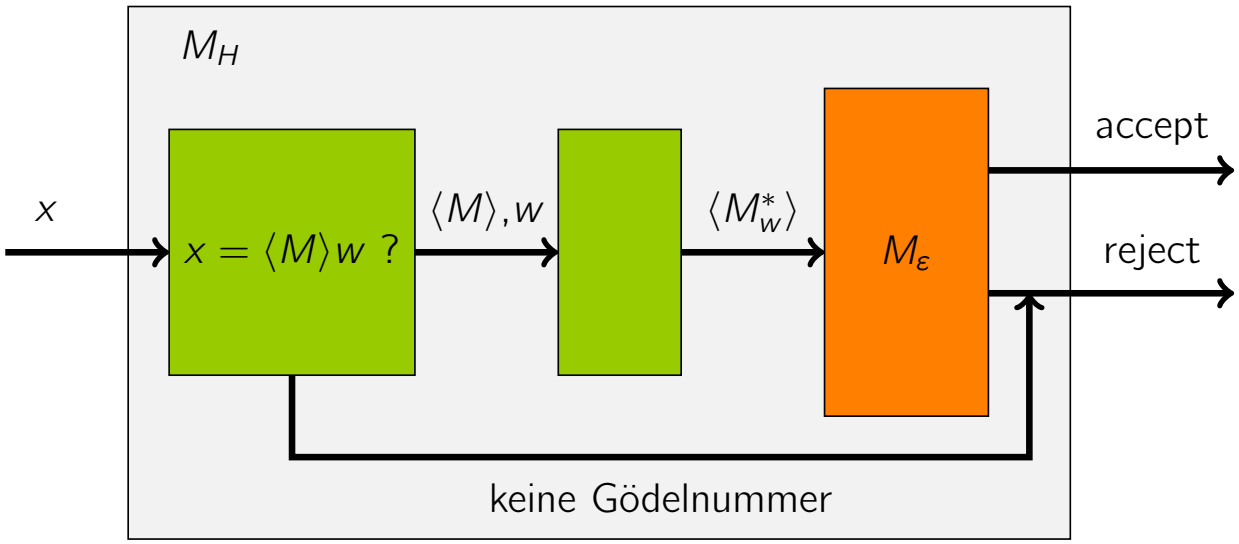
Die TM M_H mit Unterprogramm M_ϵ arbeitet wie folgt:

1) Teste, ob Eingabewort x mit einer Gödelnummer beginnt. Wenn nicht, verwerfe.
   Sonst sei M die TM und $w \in \Sigma^*$ mit $x = \langle M \rangle w$
2) Berechne die Gödelnummer einer TM M_w^* mit den folgenden Eigenschaften:
   - Falls M_w^* die Eingabe ϵ erhält, so schreibt sie das Wort w aufs Band und simuliert die TM M mit der Eingabe w.
   - Bei anderen Eingaben kann sich M_w^* beliebig verhalten.
3) Startet Unterprogramm M_ϵ mit der Eingabe $\langle M_w^* \rangle$ und akzeptiere (verwerfe) genau dann, wenn M_ϵ akzeptiert (verwirft).

Vorlesung BuK im WS 22/23, M. Grohe Seite 160 Version 26. Oktober 2022

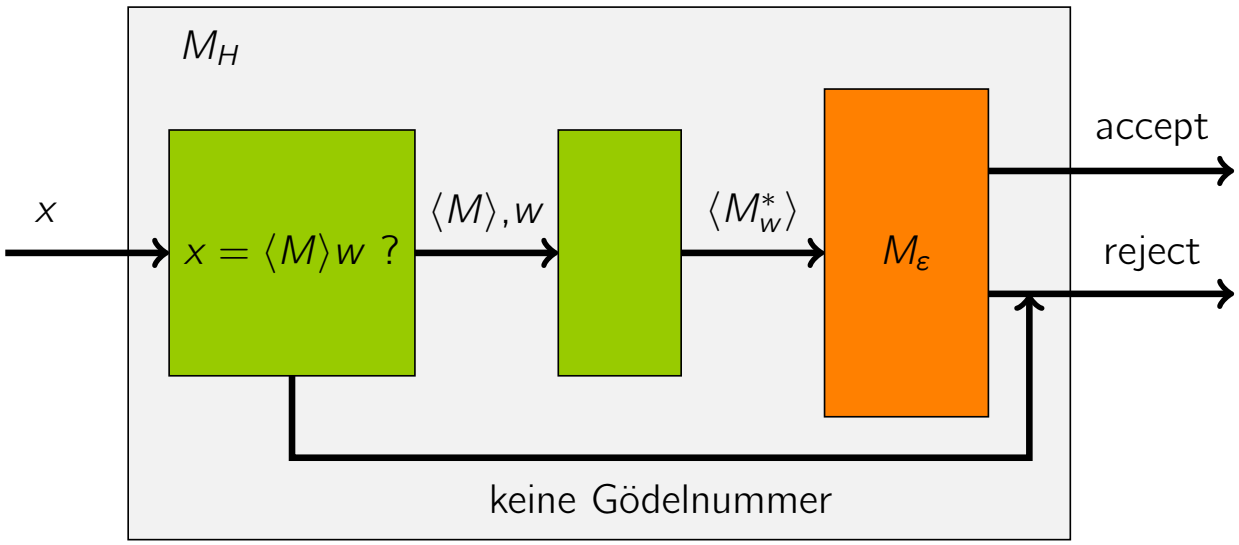
Unentscheidbarkeit des speziellen Halteproblems

Illustration: Aus M_ϵ konstruieren wir M_H.



Unentscheidbarkeit des speziellen Halteproblems

Illustration: Aus M_ϵ konstruieren wir M_H.



Aber die Existenz von M_H steht im Widerspruch zur Unentscheidbarkeit von H. Damit kann es M_ϵ nicht geben, und das spezielle Halteproblem H_ϵ ist nicht entscheidbar.

Unentscheidbarkeit des spez. Halteproblems – Beweis

Wir müssen noch zeigen, dass M_H korrekt arbeitet.

Unentscheidbarkeit des spez. Halteproblems – Beweis

Wir müssen noch zeigen, dass M_H korrekt arbeitet.

Falls die Eingabe x nicht von der Form $x = \langle M \rangle w$ ist, so verwirft M_H die Eingabe.

Unentscheidbarkeit des spez. Halteproblems – Beweis

Wir müssen noch zeigen, dass M_H korrekt arbeitet.

Falls die Eingabe x nicht von der Form $x = \langle M \rangle w$ ist, so verwirft M_H die Eingabe.

Wir gehen nun davon aus, dass eine Eingabe der Form $x = \langle M \rangle w$ vorliegt.

Für die Korrektheit ist somit noch zu zeigen:

1. $\langle M \rangle w \in H \Rightarrow M_H$ akzeptiert $\langle M \rangle w$
2. $\langle M \rangle w \notin H \Rightarrow M_H$ verwirft $\langle M \rangle w$

Unentscheidbarkeit des spez. Halteproblems – Beweis

$\langle M \rangle w \in H \quad \Rightarrow$

Unentscheidbarkeit des spez. Halteproblems – Beweis

$$\langle M \rangle w \in H \quad \Rightarrow \quad M \text{ hält auf Eingabe } w.$$

Unentscheidbarkeit des spez. Halteproblems – Beweis

$$\begin{aligned} \langle M \rangle w \in H \quad &\Rightarrow \quad M \text{ hält auf Eingabe } w. \\ &\Rightarrow \quad M^*_w \text{ hält auf der Eingabe } \epsilon. \end{aligned}$$

Unentscheidbarkeit des spez. Halteproblems – Beweis

$$\begin{aligned} \langle M \rangle w \in H &\Rightarrow M \text{ hält auf Eingabe } w. \\ &\Rightarrow M^*_w \text{ hält auf der Eingabe } \epsilon. \\ &\Rightarrow M_\epsilon \text{ akzeptiert } \langle M^*_w \rangle. \end{aligned}$$

Unentscheidbarkeit des spez. Halteproblems – Beweis

$$\begin{aligned} \langle M \rangle w \in H &\Rightarrow M \text{ hält auf Eingabe } w. \\ &\Rightarrow M^*_w \text{ hält auf der Eingabe } \epsilon. \\ &\Rightarrow M_\epsilon \text{ akzeptiert } \langle M^*_w \rangle. \\ &\Rightarrow M_H \text{ akzeptiert } \langle M \rangle w. \end{aligned}$$

Unentscheidbarkeit des spez. Halteproblems – Beweis

$$\begin{aligned} \langle M \rangle w \in H &\Rightarrow M \text{ hält auf Eingabe } w. \\ &\Rightarrow M_w^* \text{ hält auf der Eingabe } \epsilon. \\ &\Rightarrow M_\epsilon \text{ akzeptiert } \langle M_w^* \rangle. \\ &\Rightarrow M_H \text{ akzeptiert } \langle M \rangle w. \end{aligned}$$

$\langle M \rangle w \notin H$

Unentscheidbarkeit des spez. Halteproblems – Beweis

$$\begin{aligned} \langle M \rangle w \in H &\Rightarrow M \text{ hält auf Eingabe } w. \\ &\Rightarrow M_w^* \text{ hält auf der Eingabe } \epsilon. \\ &\Rightarrow M_\epsilon \text{ akzeptiert } \langle M_w^* \rangle. \\ &\Rightarrow M_H \text{ akzeptiert } \langle M \rangle w. \end{aligned}$$

$\langle M \rangle w \notin H \Rightarrow M$ hält nicht auf Eingabe w.

Unentscheidbarkeit des spez. Halteproblems – Beweis

$$
\begin{aligned}
\langle M \rangle w \in H &\Rightarrow M \text{ hält auf Eingabe } w. \\
&\Rightarrow M_w^* \text{ hält auf der Eingabe } \epsilon. \\
&\Rightarrow M_\epsilon \text{ akzeptiert } \langle M_w^* \rangle. \\
&\Rightarrow M_H \text{ akzeptiert } \langle M \rangle w.
\end{aligned}
$$

$$
\begin{aligned}
\langle M \rangle w \notin H &\Rightarrow M \text{ hält nicht auf Eingabe } w. \\
&\Rightarrow M_w^* \text{ hält nicht auf der Eingabe } \epsilon.
\end{aligned}
$$

Unentscheidbarkeit des spez. Halteproblems – Beweis

$$
\begin{aligned}
\langle M \rangle w \in H &\Rightarrow M \text{ hält auf Eingabe } w. \\
&\Rightarrow M_w^* \text{ hält auf der Eingabe } \epsilon. \\
&\Rightarrow M_\epsilon \text{ akzeptiert } \langle M_w^* \rangle. \\
&\Rightarrow M_H \text{ akzeptiert } \langle M \rangle w.
\end{aligned}
$$

$$
\begin{aligned}
\langle M \rangle w \notin H &\Rightarrow M \text{ hält nicht auf Eingabe } w. \\
&\Rightarrow M_w^* \text{ hält nicht auf der Eingabe } \epsilon. \\
&\Rightarrow M_\epsilon \text{ verwirft } \langle M_w^* \rangle.
\end{aligned}
$$

Unentscheidbarkeit des spez. Halteproblems – Beweis

$$
\begin{aligned}
\langle M\rangle w \in H &\Rightarrow M \text{ hält auf Eingabe } w.\\
&\Rightarrow M^*_w \text{ hält auf der Eingabe } \epsilon.\\
&\Rightarrow M_\epsilon \text{ akzeptiert } \langle M^*_w\rangle.\\
&\Rightarrow M_H \text{ akzeptiert } \langle M\rangle w.
\end{aligned}
$$

$$
\begin{aligned}
\langle M\rangle w \notin H &\Rightarrow M \text{ hält nicht auf Eingabe } w.\\
&\Rightarrow M^*_w \text{ hält nicht auf der Eingabe } \epsilon.\\
&\Rightarrow M_\epsilon \text{ verwirft } \langle M^*_w\rangle.\\
&\Rightarrow M_H \text{ verwirft } \langle M\rangle w.
\end{aligned}
$$

Unentscheidbarkeit des spez. Halteproblems – Beweis

$$
\begin{aligned}
\langle M\rangle w \in H &\Rightarrow M \text{ hält auf Eingabe } w.\\
&\Rightarrow M^*_w \text{ hält auf der Eingabe } \epsilon.\\
&\Rightarrow M_\epsilon \text{ akzeptiert } \langle M^*_w\rangle.\\
&\Rightarrow M_H \text{ akzeptiert } \langle M\rangle w.
\end{aligned}
$$

$$
\begin{aligned}
\langle M\rangle w \notin H &\Rightarrow M \text{ hält nicht auf Eingabe } w.\\
&\Rightarrow M^*_w \text{ hält nicht auf der Eingabe } \epsilon.\\
&\Rightarrow M_\epsilon \text{ verwirft } \langle M^*_w\rangle.\\
&\Rightarrow M_H \text{ verwirft } \langle M\rangle w.
\end{aligned}
$$

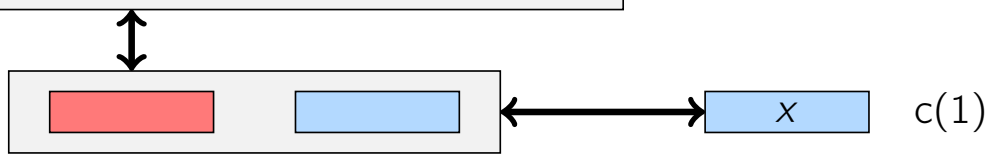
□

Vorlesung BuK im WS 22/23, M. Grohe Seite 163 Version 26. Oktober 2022

Das Collatz-Problem

Das Collatz-Problem

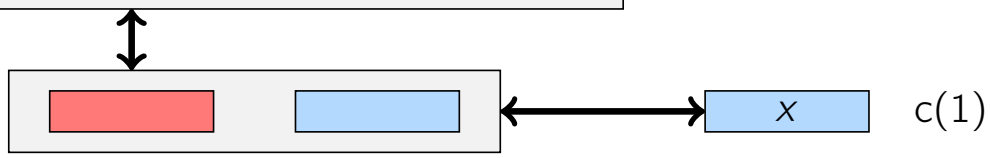
```
1:   LOAD 1
2:   IF c(0) > 1 THEN GOTO 4
3:   END
4:   CADD 1
5:   CDIV 2
6:   CMULT 2
7:   SUB 1
8:   IF c(0) > 0 THEN GOTO 13
9:   LOAD 1
10:  CDIV 2
11:  STORE 1
12:  GOTO 1
13:  LOAD 1
14:  CMULT 3
15:  ADD 1
16:  GOTO 1
```



Das Collatz-Problem

```
1:   LOAD 1
2:   IF c(0) > 1 THEN GOTO 4
3:   END
4:   CADD 1
5:   CDIV 2
6:   CMULT 2
7:   SUB 1
8:   IF c(0) > 0 THEN GOTO 13
9:   LOAD 1
10:  CDIV 2
11:  STORE 1
12:  GOTO 1
13:  LOAD 1
14:  CMULT 3
15:  ADD 1
16:  GOTO 1
```

$$x \leftarrow \begin{cases} x/2 & \text{wenn } x \text{ gerade} \\ 3x+1 & \text{sonst} \end{cases}$$



Das Collatz-Problem

$$x \leftarrow \begin{cases} x/2 & \text{wenn } x \text{ gerade} \\ 3x+1 & \text{sonst} \end{cases}$$

Das Collatz-Problem

$$x \leftarrow \begin{cases} x/2 & \text{wenn } x \text{ gerade} \\ 3x+1 & \text{sonst} \end{cases}$$

Mit dieser Iterationsgleichung erhält man z.B. die Zahlenfolgen

- 1, 4, 2, 1, ...

Das Collatz-Problem

$$x \leftarrow \begin{cases} x/2 & \text{wenn } x \text{ gerade} \\ 3x+1 & \text{sonst} \end{cases}$$

Mit dieser Iterationsgleichung erhält man z.B. die Zahlenfolgen

- 1, 4, 2, 1, ...
- 2, 1, 4, 2, 1,...

Das Collatz-Problem

$$x \leftarrow \begin{cases} x/2 & \text{wenn } x \text{ gerade} \\ 3x+1 & \text{sonst} \end{cases}$$

Mit dieser Iterationsgleichung erhält man z.B. die Zahlenfolgen

- 1, 4, 2, 1, ...
- 2, 1, 4, 2, 1,...
- 3, 10, 5, 16, 8, 4, 2, 1, ...

Das Collatz-Problem

$$x \leftarrow \begin{cases} x/2 & \text{wenn } x \text{ gerade} \\ 3x+1 & \text{sonst} \end{cases}$$

Mit dieser Iterationsgleichung erhält man z.B. die Zahlenfolgen

- 1, 4, 2, 1, ...
- 2, 1, 4, 2, 1,...
- 3, 10, 5, 16, 8, 4, 2, 1, ...
- 5, 16, 8, 4, 2, 1, 4, 2, 1

Das Collatz-Problem

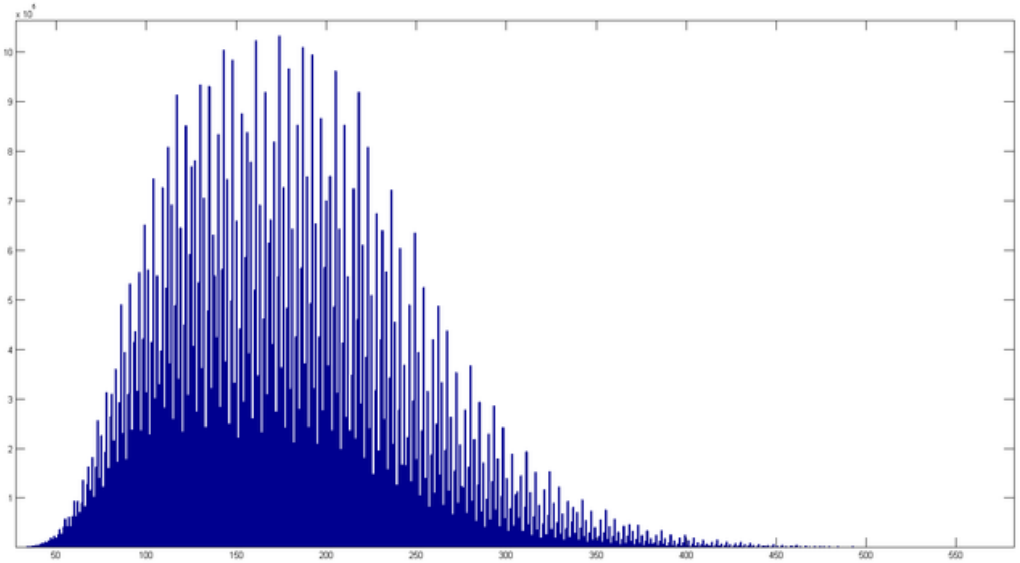
$$x \leftarrow \begin{cases} x/2 & \text{wenn } x \text{ gerade} \\ 3x+1 & \text{sonst} \end{cases}$$

Mit dieser Iterationsgleichung erhält man z.B. die Zahlenfolgen

- 1, 4, 2, 1, ...
- 2, 1, 4, 2, 1,...
- 3, 10, 5, 16, 8, 4, 2, 1, ...
- 5, 16, 8, 4, 2, 1, 4, 2, 1
- 19, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, 4, 2, 1

Offene Frage:
Es ist unbekannt ob obige Registermaschine auf allen Eingaben hält.

Statistik der Zahlenfolgenlängen



Zahlenfolgenlängen bei Eingaben bis 100 Millionen.

Vorlesung 7
Der Satz von Rice

Wdh.: Bisher betrachtete Probleme

Die Diagonalsprache:

$$D = \{\langle M \rangle \mid M \text{ akzeptiert } \langle M \rangle \text{ nicht}\}$$

Das Diagonalsprachenkomplement:

$$\overline{D} = \{\langle M \rangle \mid M \text{ akzeptiert } \langle M \rangle\}$$

Das Halteproblem:

$$H = \{\langle M \rangle w \mid M \text{ hält auf } w\}$$

Das spezielle Halteproblem:

$$H_\epsilon = \{\langle M \rangle \mid M \text{ hält auf Eingabe } \epsilon\}$$

Alle diese Probleme sind unentscheidbar.

Wdh.: Beweise durch Unterprogrammtechnik

D ist unentscheidbar auf Grund eines Diagonalisierungs-Argumentes.

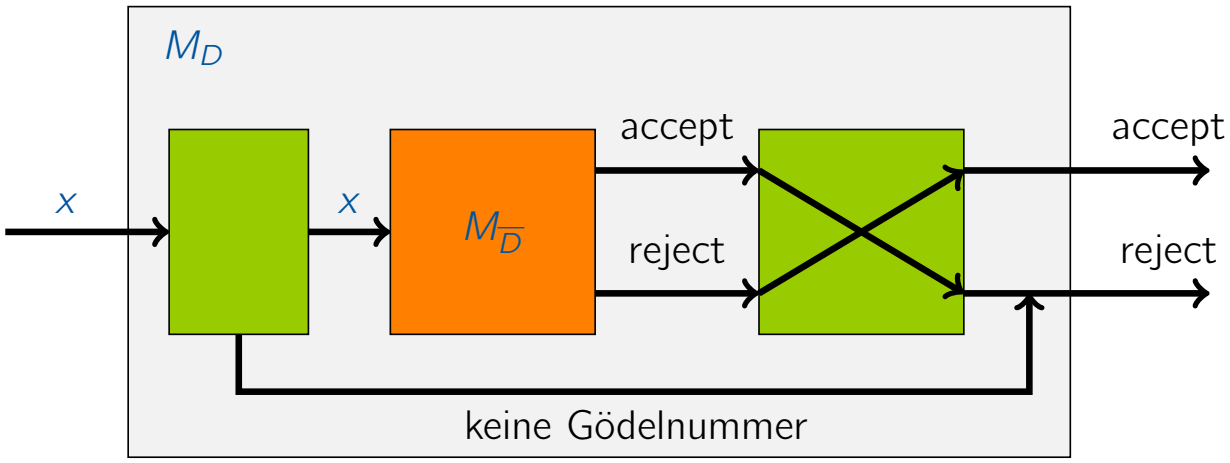
Wdh.: Beweise durch Unterprogrammtechnik

D ist unentscheidbar auf Grund eines Diagonalisierungs-Argumentes.

Die Argumentationskette war:

$$\begin{array}{ll} D \text{ ist unentscheidbar} & M_D \\ \Downarrow & \updownarrow \\ \overline{D} \text{ ist unentscheidbar} & M_{\overline{D}} \end{array}$$

Wdh.: Unentscheidbarkeit des Komplements der Diagonalsprache



Wdh.: Beweise durch Unterprogrammtechnik

D ist unentscheidbar auf Grund eines Diagonalisierungs-Argumentes.

Die Argumentationskette war:

$$\begin{array}{ll} D \text{ ist unentscheidbar} & M_D \\ \Downarrow & \updownarrow \\ \overline{D} \text{ ist unentscheidbar} & M_{\overline{D}} \end{array}$$

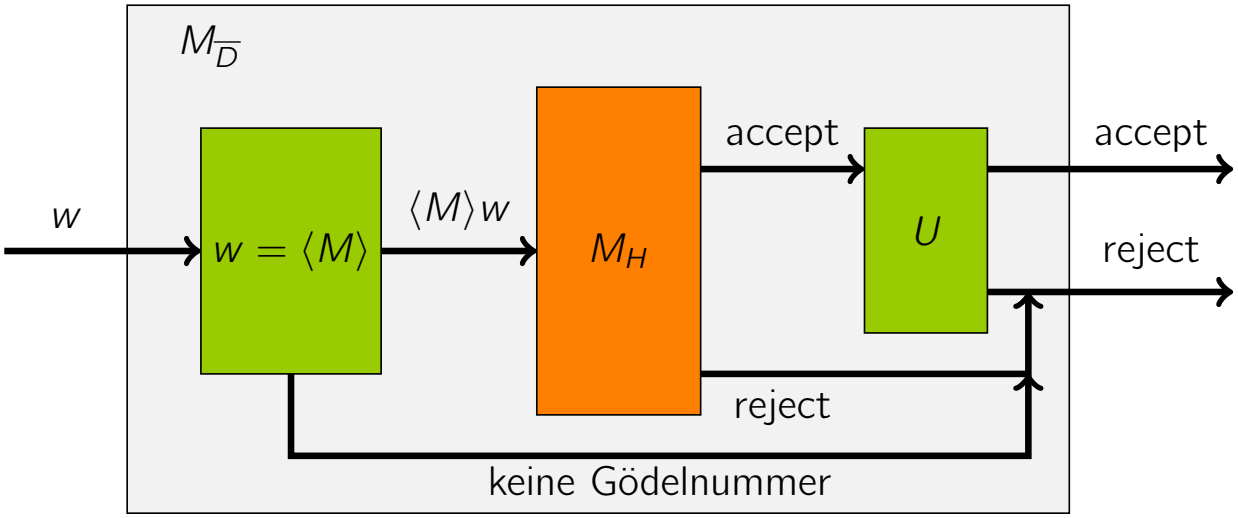
Wdh.: Beweise durch Unterprogrammtechnik

D ist unentscheidbar auf Grund eines Diagonalisierungs-Argumentes.

Die Argumentationskette war:

$$\begin{array}{ll} D \text{ ist unentscheidbar} & M_D \\ \Downarrow & \updownarrow \\ \overline{D} \text{ ist unentscheidbar} & M_{\overline{D}} \\ \Downarrow & \updownarrow \\ H \text{ ist unentscheidbar} & M_H \end{array}$$

Wdh.: Unentscheidbarkeit des Halteproblems



Wdh.: Beweise durch Unterprogrammtechnik

D ist unentscheidbar auf Grund eines Diagonalisierungs-Argumentes.

Die Argumentationskette war:

$$\begin{array}{ll} D \text{ ist unentscheidbar} & M_D \\ \Downarrow & \updownarrow \\ \overline{D} \text{ ist unentscheidbar} & M_{\overline{D}} \\ \Downarrow & \updownarrow \\ H \text{ ist unentscheidbar} & M_H \end{array}$$

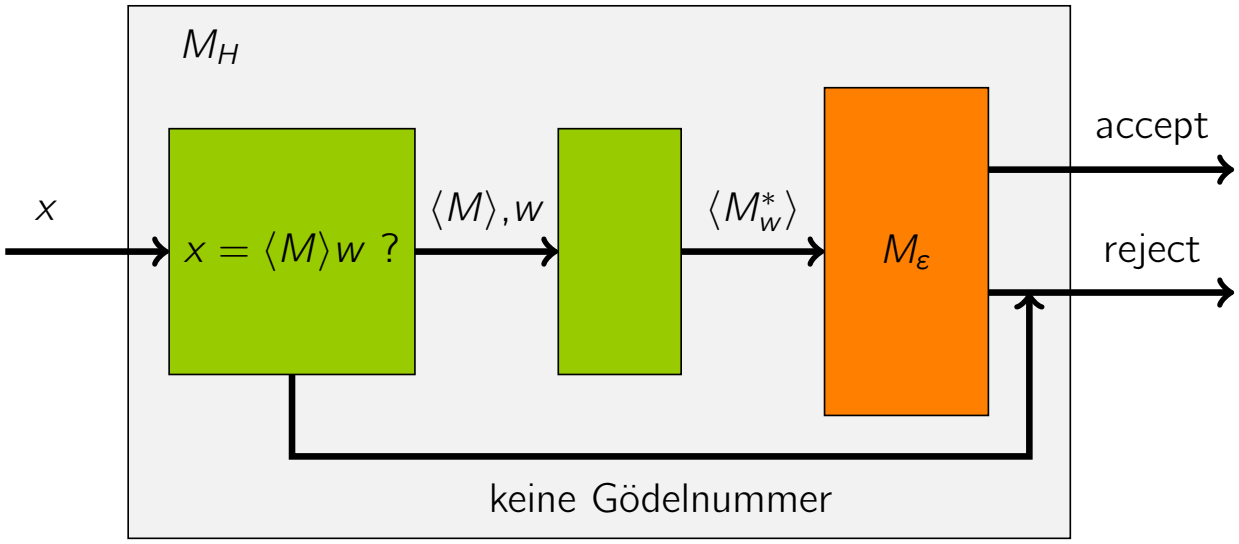
Wdh.: Beweise durch Unterprogrammtechnik

D ist unentscheidbar auf Grund eines Diagonalisierungs-Argumentes.

Die Argumentationskette war:

$$\begin{array}{ll} D \text{ ist unentscheidbar} & M_D \\ \Downarrow & \updownarrow \\ \overline{D} \text{ ist unentscheidbar} & M_{\overline{D}} \\ \Downarrow & \updownarrow \\ H \text{ ist unentscheidbar} & M_H \\ \Downarrow & \updownarrow \\ H_\epsilon \text{ ist unentscheidbar} & M_{H_\epsilon} \end{array}$$

Wdh.: Unentscheidbarkeit des speziellen Halteproblems



Wdh.: Beweise durch Unterprogrammtechnik

D ist unentscheidbar auf Grund eines Diagonalisierungs-Argumentes.

Die Argumentationskette war:

$$\begin{array}{ll} D \text{ ist unentscheidbar} & M_D \\ \Downarrow & \updownarrow \\ \overline{D} \text{ ist unentscheidbar} & M_{\overline{D}} \\ \Downarrow & \updownarrow \\ H \text{ ist unentscheidbar} & M_H \\ \Downarrow & \updownarrow \\ H_\epsilon \text{ ist unentscheidbar} & M_{H_\epsilon} \end{array}$$

Wdh.: Bisher betrachtete Probleme

Die Diagonalsprache:

$$D = \{\langle M \rangle \mid M \text{ akzeptiert } \langle M \rangle \text{ nicht}\}$$

Das Diagonalsprachenkomplement:

$$\overline{D} = \{\langle M \rangle \mid M \text{ akzeptiert } \langle M \rangle\}$$

Das Halteproblem:

$$H = \{\langle M \rangle w \mid M \text{ hält auf } w\}$$

Das spezielle Halteproblem:

$$H_\epsilon = \{\langle M \rangle \mid M \text{ hält auf Eingabe } \epsilon\}$$

Alle diese Probleme sind unentscheidbar.

Wdh.: Bisher betrachtete Probleme

Die Diagonalsprache:

$$D = \{\langle M \rangle \mid M \text{ akzeptiert } \langle M \rangle \text{ nicht}\}$$

Das Diagonalsprachenkomplement:

$$\overline{D} = \{\langle M \rangle \mid M \text{ akzeptiert } \langle M \rangle\}$$

Das Halteproblem:

$$H = \{\langle M \rangle w \mid M \text{ hält auf } w\}$$

Das spezielle Halteproblem:

$$H_\epsilon = \{\langle M \rangle \mid M \text{ hält auf Eingabe } \epsilon\}$$

Alle diese Probleme sind unentscheidbar.
Was haben sie strukturell gemeinsam?

Wdh.: Berechenbare partielle Funktionen

Im allgemeinen berechnet eine Turingmaschine M mit Ein- und Ausgabealphabet Σ eine partielle Funktion f_M von Σ^* nach Σ^*, die definiert ist durch

$$f_M(x) = \begin{cases} y & \text{falls } M \text{ bei Eingabe } x \text{ mit Ausgabe } y \text{ anhält,} \\ \perp \text{ (undefiniert)} & \text{falls } M \text{ bei Eingabe } x \text{ nicht anhält.} \end{cases}$$

Wdh.: Berechenbare partielle Funktionen

Im allgemeinen berechnet eine Turingmaschine M mit Ein- und Ausgabealphabet Σ eine partielle Funktion f_M von Σ^* nach Σ^*, die definiert ist durch

$$f_M(x) = \begin{cases} y & \text{falls } M \text{ bei Eingabe } x \text{ mit Ausgabe } y \text{ anhält,} \\ \perp \text{ (undefiniert)} & \text{falls } M \text{ bei Eingabe } x \text{ nicht anhält.} \end{cases}$$

Eine partielle Funktion f von Σ^* nach Σ^* ist berechenbar, wenn es eine Turingmaschine M gibt, so dass $f = f_M$.

Satz von Rice

Satz

Sei $\mathcal{R}$ die Menge der berechenbaren partiellen Funktionen und $\mathcal{S}$ eine Teilmenge von $\mathcal{R}$ mit $\emptyset \subsetneq \mathcal{S} \subsetneq \mathcal{R}$. Dann ist die Sprache

$$L(\mathcal{S}) = \{\langle M \rangle \mid M \text{ berechnet eine Funktion aus } \mathcal{S}\}$$

unentscheidbar.

Satz von Rice

Satz

Sei $\mathcal{R}$ die Menge der berechenbaren partiellen Funktionen und $\mathcal{S}$ eine Teilmenge von $\mathcal{R}$ mit $\emptyset \subsetneq \mathcal{S} \subsetneq \mathcal{R}$. Dann ist die Sprache

$$L(\mathcal{S}) = \{\langle M \rangle \mid M \text{ berechnet eine Funktion aus } \mathcal{S}\}$$

unentscheidbar.

Mit anderen Worten: Aussagen über die von einer TM berechnete Funktion sind nicht entscheidbar.

Satz von Rice – Anwendungsbeispiele

Beispiel 1

- Sei $\mathcal{S} = \{f_M \mid f_M(\epsilon) \neq \perp\}$.
- Dann ist

$$\begin{aligned} L(\mathcal{S}) &= \{\langle M \rangle \mid M \text{ berechnet eine Funktion aus } \mathcal{S}\} \\ &= \{\langle M \rangle \mid M \text{ hält auf Eingabe } \epsilon\} \\ &= H_\epsilon \end{aligned}$$

- Gemäß Satz von Rice ist H_ϵ nicht entscheidbar.
(Aber das wussten wir ja schon ☺.)

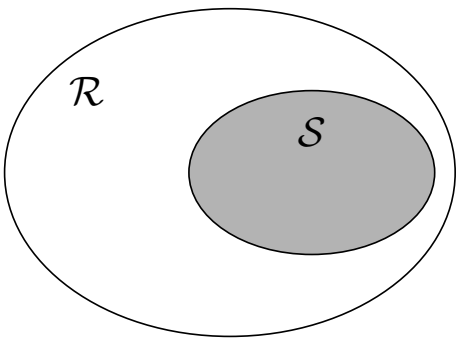
Satz von Rice – Anwendungsbeispiele

Beispiel 1

- Sei $\mathcal{S} = \{f_M \mid f_M(\epsilon) \neq \perp\}$.
- Dann ist

$$\begin{aligned} L(\mathcal{S}) &= \{\langle M \rangle \mid M \text{ berechnet eine Funktion aus } \mathcal{S}\} \\ &= \{\langle M \rangle \mid M \text{ hält auf Eingabe } \epsilon\} \\ &= H_\epsilon \end{aligned}$$

- Gemäß Satz von Rice ist H_ϵ nicht entscheidbar.
(Aber das wussten wir ja schon ☺.)



Satz von Rice – Anwendungsbeispiele

Beispiel 2

- Sei $\mathcal{S} = \{f_M \mid \forall w \in \{0,1\}^* \colon f_M(w) \neq \bot\}$.
- Dann ist

$$\begin{aligned} L(\mathcal{S}) &= \{\langle M \rangle \mid M \text{ berechnet eine Funktion aus } \mathcal{S}\} \\ &= \{\langle M \rangle \mid M \text{ hält auf jeder Eingabe}\} \end{aligned}$$

- Diese Sprache ist auch als das *allgemeine Halteproblem* H_{all} bekannt.
- Gemäß Satz von Rice ist H_{all} nicht entscheidbar.

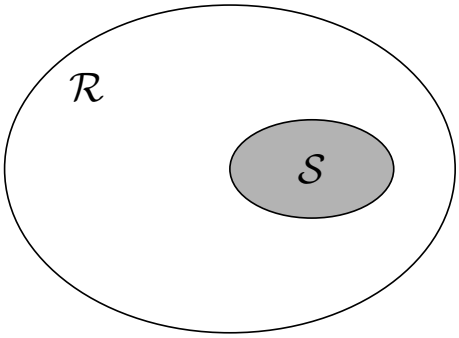
Satz von Rice – Anwendungsbeispiele

Beispiel 2

- Sei $\mathcal{S} = \{f_M \mid \forall w \in \{0,1\}^* \colon f_M(w) \neq \bot\}$.
- Dann ist

$$\begin{aligned} L(\mathcal{S}) &= \{\langle M \rangle \mid M \text{ berechnet eine Funktion aus } \mathcal{S}\} \\ &= \{\langle M \rangle \mid M \text{ hält auf jeder Eingabe}\} \end{aligned}$$

- Diese Sprache ist auch als das *allgemeine Halteproblem* H_{all} bekannt.
- Gemäß Satz von Rice ist H_{all} nicht entscheidbar.



Satz von Rice – Anwendungsbeispiele

Beispiel 3

- Sei $\mathcal{S} = \{f_M \mid \forall w \in \{0,1\}^* \colon f_M(w) = 1\}$.
- Dann ist

$$\begin{aligned} L(\mathcal{S}) &= \{\langle M \rangle \mid M \text{ berechnet eine Funktion aus } \mathcal{S}\} \\ &= \{\langle M \rangle \mid M \text{ hält auf jeder Eingabe mit Ausgabe 1}\} \end{aligned}$$

- Gemäß Satz von Rice ist $L(\mathcal{S})$ nicht entscheidbar.

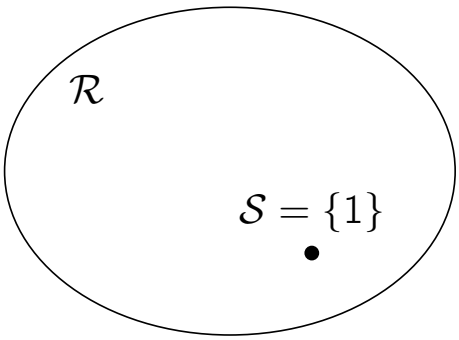
Satz von Rice – Anwendungsbeispiele

Beispiel 3

- Sei $\mathcal{S} = \{f_M \mid \forall w \in \{0,1\}^* \colon f_M(w) = 1\}$.
- Dann ist

$$\begin{aligned} L(\mathcal{S}) &= \{\langle M \rangle \mid M \text{ berechnet eine Funktion aus } \mathcal{S}\} \\ &= \{\langle M \rangle \mid M \text{ hält auf jeder Eingabe mit Ausgabe 1}\} \end{aligned}$$

- Gemäß Satz von Rice ist $L(\mathcal{S})$ nicht entscheidbar.



Satz von Rice – Beweis

Satz

Sei $\mathcal{R}$ die Menge der von TMen berechenbaren partiellen Funktionen und $\mathcal{S}$ eine Teilmenge von $\mathcal{R}$ mit $\emptyset \subsetneq \mathcal{S} \subsetneq \mathcal{R}$. Dann ist die Sprache

$$L(\mathcal{S}) = \{\langle M \rangle \mid M \text{ berechnet eine Funktion aus } \mathcal{S}\}$$

unentscheidbar.

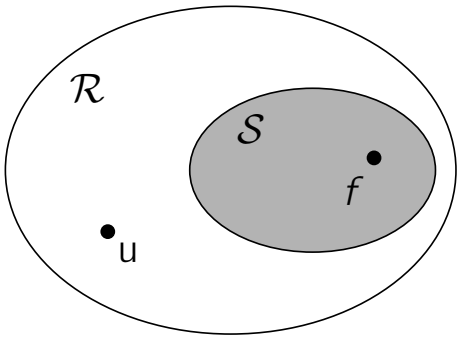
Satz von Rice – Beweis

Beweis

Wir nutzen die Unterprogrammtechnik. Aus einer TM $M_{L(\mathcal{S})}$, die $L(\mathcal{S})$ entscheidet, konstruieren wir eine TM M_{H_ϵ}, die das spezielle Halteproblem H_ϵ entscheidet.

Einige Vereinbarungen:

- Sei u die überall undefinierte Funktion.
- O.B.d.A. $u \notin \mathcal{S}$.



Vorlesung BuK im WS 22/23, M. Grohe Seite 182 Version 26. Oktober 2022

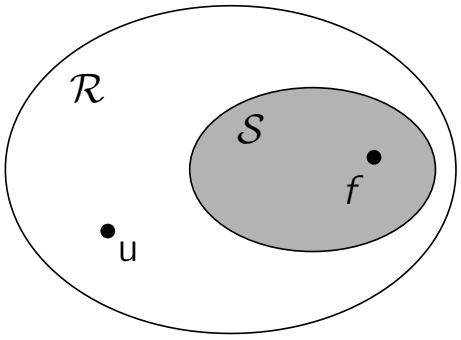
Satz von Rice – Beweis

Beweis

Wir nutzen die Unterprogrammtechnik. Aus einer TM $M_{L(\mathcal{S})}$, die $L(\mathcal{S})$ entscheidet, konstruieren wir eine TM M_{H_ϵ}, die das spezielle Halteproblem H_ϵ entscheidet.

Einige Vereinbarungen:

- Sei u die überall undefinierte Funktion.
- O.B.d.A. $u \notin \mathcal{S}$.



Bemerkung: Im Falle $u \in \mathcal{S}$ betrachten wir $\mathcal{R} \setminus \mathcal{S}$ statt $\mathcal{S}$ und zeigen die Unentscheidbarkeit von $L(\mathcal{R} \setminus \mathcal{S})$. Hieraus ergibt sich dann unmittelbar die Unentscheidbarkeit von $L(\mathcal{S})$.

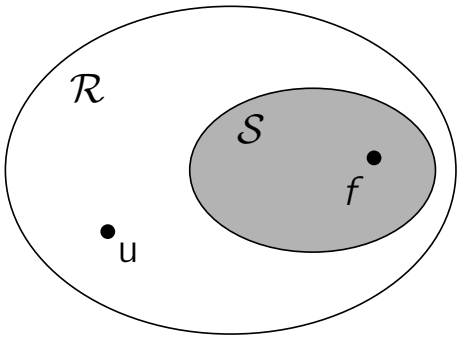
Satz von Rice – Beweis

Beweis

Wir nutzen die Unterprogrammtechnik. Aus einer TM $M_{L(\mathcal{S})}$, die $L(\mathcal{S})$ entscheidet, konstruieren wir eine TM M_{H_ϵ}, die das spezielle Halteproblem H_ϵ entscheidet.

Einige Vereinbarungen:

- Sei u die überall undefinierte Funktion.
- O.B.d.A. $u \notin \mathcal{S}$.
- Sei f eine Funktion aus $\mathcal{S}$.
- Sei N eine TM, die f berechnet.



Bemerkung: Im Falle $u \in \mathcal{S}$ betrachten wir $\mathcal{R} \setminus \mathcal{S}$ statt $\mathcal{S}$ und zeigen die Unentscheidbarkeit von $L(\mathcal{R} \setminus \mathcal{S})$. Hieraus ergibt sich dann unmittelbar die Unentscheidbarkeit von $L(\mathcal{S})$.

Satz von Rice – Fortsetzung Beweis

Die TM M_{H_ϵ} mit Unterprogramm $M_{L(\mathcal{S})}$ arbeitet wie folgt

1)

1) Teste, ob w Gödelnummer. Wenn nicht, verwerfe.
   Sonst sei M die TM mit $w = \langle M \rangle$

Satz von Rice – Fortsetzung Beweis

Die TM M_{H_ϵ} mit Unterprogramm $M_{L(\mathcal{S})}$ arbeitet wie folgt

1)

1) Teste, ob w Gödelnummer. Wenn nicht, verwerfe.
   Sonst sei M die TM mit $w = \langle M \rangle$
2) Sonst berechnet M_{H_ϵ} aus der Eingabe $\langle M \rangle$ die Gödelnummer der TM M^* (nächste Folie).

Satz von Rice – Fortsetzung Beweis

Die TM M_{H_ϵ} mit Unterprogramm $M_{L(\mathcal{S})}$ arbeitet wie folgt

1)

1) Teste, ob w Gödelnummer. Wenn nicht, verwerfe.
   Sonst sei M die TM mit $w = \langle M \rangle$
2) Sonst berechnet M_{H_ϵ} aus der Eingabe $\langle M \rangle$ die Gödelnummer der TM M^* (nächste Folie).
3) Starte $M_{L(\mathcal{S})}$ mit der Eingabe $\langle M^* \rangle$ und akzeptiere (verwirf) genau dann, wenn $M_{L(\mathcal{S})}$ akzeptiert (verwirft).

Satz von Rice – Fortsetzung Beweis

Verhalten von M^* auf Eingabe x

Phase A: Simuliere das Verhalten von M bei Eingabe ϵ auf einer für diesen Zweck reservierten Spur.

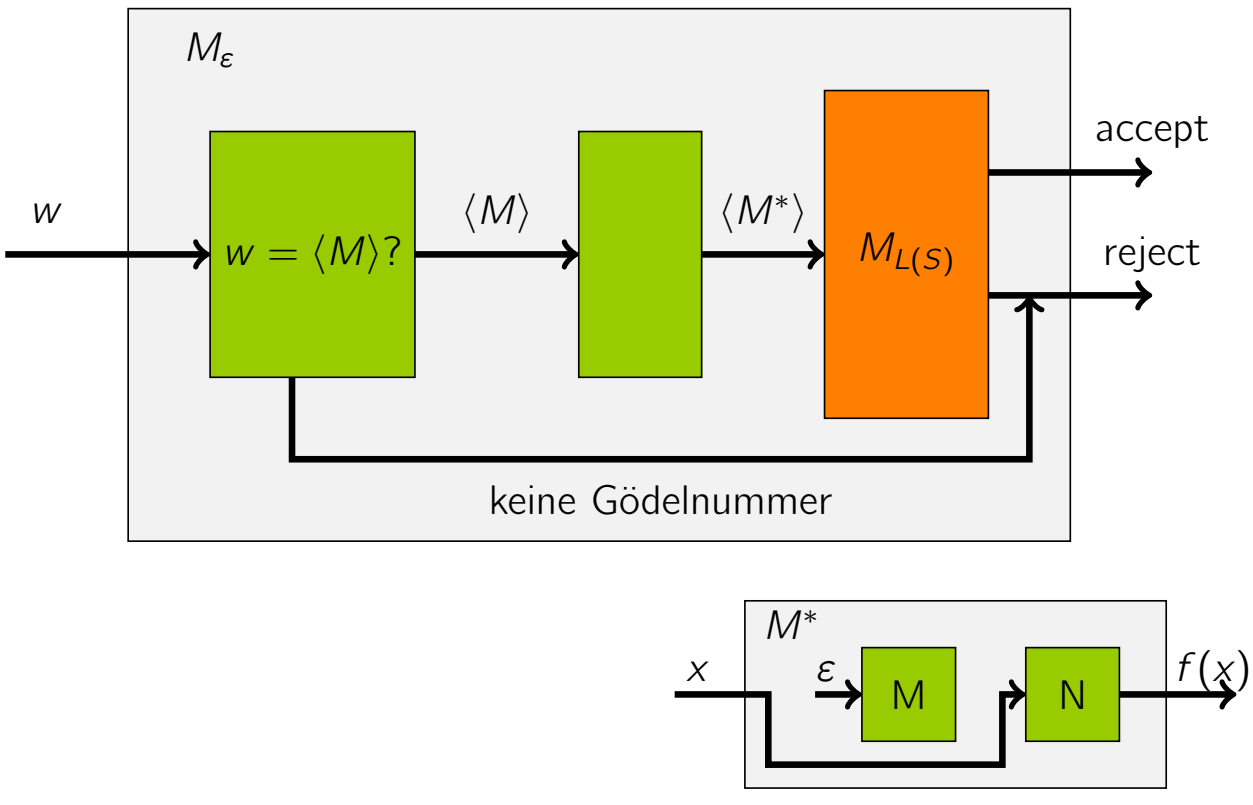
Satz von Rice – Fortsetzung Beweis

Verhalten von M^* auf Eingabe x

Phase A: Simuliere das Verhalten von M bei Eingabe ϵ auf einer für diesen Zweck reservierten Spur.

Phase B: Simuliere das Verhalten von N auf x, halte, sobald N hält, und übernimm die Ausgabe.

Satz von Rice – Illustration



Satz von Rice – Fortsetzung Beweis

Korrektheit:

Bei Eingabe von w, wobei w keine Gödelnummer ist, verwirft M_{H_ϵ}.

Bei Eingabe von $w = \langle M \rangle$ gilt:

$$w \in H_\epsilon$$

Satz von Rice – Fortsetzung Beweis

Korrektheit:

Bei Eingabe von w, wobei w keine Gödelnummer ist, verwirft M_{H_ϵ}.

Bei Eingabe von $w = \langle M \rangle$ gilt:

$$w \in H_\epsilon \quad \Rightarrow \quad M \text{ hält auf } \epsilon$$

Satz von Rice – Fortsetzung Beweis

Korrektheit:

Bei Eingabe von w, wobei w keine Gödelnummer ist, verwirft M_{H_ϵ}.

Bei Eingabe von $w = \langle M \rangle$ gilt:

$$\begin{aligned} w \in H_\epsilon &\Rightarrow M \text{ hält auf } \epsilon \\ &\Rightarrow M^* \text{ berechnet } f \end{aligned}$$

Satz von Rice – Fortsetzung Beweis

Korrektheit:

Bei Eingabe von w, wobei w keine Gödelnummer ist, verwirft M_{H_ϵ}.

Bei Eingabe von $w = \langle M \rangle$ gilt:

$$\begin{aligned} w \in H_\epsilon &\Rightarrow M \text{ hält auf } \epsilon \\ &\Rightarrow M^* \text{ berechnet } f \\ &\overset{f \in \mathcal{S}}{\Rightarrow} \langle M^* \rangle \in L(\mathcal{S}) \end{aligned}$$

Satz von Rice – Fortsetzung Beweis

Korrektheit:

Bei Eingabe von w, wobei w keine Gödelnummer ist, verwirft M_{H_ϵ}.

Bei Eingabe von $w = \langle M \rangle$ gilt:

$$\begin{aligned} w \in H_\epsilon &\Rightarrow M \text{ hält auf } \epsilon \\ &\Rightarrow M^* \text{ berechnet } f \\ &\overset{f \in \mathcal{S}}{\Rightarrow} \langle M^* \rangle \in L(\mathcal{S}) \\ &\Rightarrow M_{L(\mathcal{S})} \text{ akzeptiert } \langle M^* \rangle \end{aligned}$$

Satz von Rice – Fortsetzung Beweis

Korrektheit:

Bei Eingabe von w, wobei w keine Gödelnummer ist, verwirft M_{H_ϵ}.

Bei Eingabe von $w = \langle M \rangle$ gilt:

$$\begin{aligned} w \in H_\epsilon &\Rightarrow M \text{ hält auf } \epsilon \\ &\Rightarrow M^* \text{ berechnet } f \\ &\overset{f \in \mathcal{S}}{\Rightarrow} \langle M^* \rangle \in L(\mathcal{S}) \\ &\Rightarrow M_{L(\mathcal{S})} \text{ akzeptiert } \langle M^* \rangle \\ &\Rightarrow M_{H_\epsilon} \text{ akzeptiert } w \end{aligned}$$

Satz von Rice – Fortsetzung Beweis

Korrektheit:

Bei Eingabe von w, wobei w keine Gödelnummer ist, verwirft M_{H_ϵ}.

Bei Eingabe von $w = \langle M \rangle$ gilt:

$$\begin{aligned} w \in H_\epsilon &\Rightarrow M \text{ hält auf } \epsilon \\ &\Rightarrow M^* \text{ berechnet } f \\ &\overset{f \in \mathcal{S}}{\Rightarrow} \langle M^* \rangle \in L(\mathcal{S}) \\ &\Rightarrow M_{L(\mathcal{S})} \text{ akzeptiert } \langle M^* \rangle \\ &\Rightarrow M_{H_\epsilon} \text{ akzeptiert } w \end{aligned}$$

$$w \notin H_\epsilon$$

Satz von Rice – Fortsetzung Beweis

Korrektheit:

Bei Eingabe von w, wobei w keine Gödelnummer ist, verwirft M_{H_ϵ}.

Bei Eingabe von $w = \langle M \rangle$ gilt:

$$\begin{aligned} w \in H_\epsilon &\Rightarrow M \text{ hält auf } \epsilon \\ &\Rightarrow M^* \text{ berechnet } f \\ &\overset{f \in \mathcal{S}}{\Rightarrow} \langle M^* \rangle \in L(\mathcal{S}) \\ &\Rightarrow M_{L(\mathcal{S})} \text{ akzeptiert } \langle M^* \rangle \\ &\Rightarrow M_{H_\epsilon} \text{ akzeptiert } w \end{aligned}$$

$$w \notin H_\epsilon \quad \Rightarrow \quad M \text{ hält nicht auf } \epsilon$$

Satz von Rice – Fortsetzung Beweis

Korrektheit:

Bei Eingabe von w, wobei w keine Gödelnummer ist, verwirft M_{H_ϵ}.

Bei Eingabe von $w = \langle M \rangle$ gilt:

$$\begin{aligned} w \in H_\epsilon &\Rightarrow M \text{ hält auf } \epsilon \\ &\Rightarrow M^* \text{ berechnet } f \\ &\overset{f \in \mathcal{S}}{\Rightarrow} \langle M^* \rangle \in L(\mathcal{S}) \\ &\Rightarrow M_{L(\mathcal{S})} \text{ akzeptiert } \langle M^* \rangle \\ &\Rightarrow M_{H_\epsilon} \text{ akzeptiert } w \end{aligned}$$

$$\begin{aligned} w \notin H_\epsilon &\Rightarrow M \text{ hält nicht auf } \epsilon \\ &\Rightarrow M^* \text{ berechnet } u \end{aligned}$$

Satz von Rice – Fortsetzung Beweis

Korrektheit:

Bei Eingabe von w, wobei w keine Gödelnummer ist, verwirft M_{H_ϵ}.

Bei Eingabe von $w = \langle M \rangle$ gilt:

$$\begin{aligned} w \in H_\epsilon &\Rightarrow M \text{ hält auf } \epsilon \\ &\Rightarrow M^* \text{ berechnet } f \\ &\overset{f \in \mathcal{S}}{\Rightarrow} \langle M^* \rangle \in L(\mathcal{S}) \\ &\Rightarrow M_{L(\mathcal{S})} \text{ akzeptiert } \langle M^* \rangle \\ &\Rightarrow M_{H_\epsilon} \text{ akzeptiert } w \end{aligned}$$

$$\begin{aligned} w \notin H_\epsilon &\Rightarrow M \text{ hält nicht auf } \epsilon \\ &\Rightarrow M^* \text{ berechnet } u \\ &\overset{u \notin \mathcal{S}}{\Rightarrow} \langle M^* \rangle \notin L(\mathcal{S}) \end{aligned}$$

Satz von Rice – Fortsetzung Beweis

Korrektheit:

Bei Eingabe von w, wobei w keine Gödelnummer ist, verwirft M_{H_ϵ}.

Bei Eingabe von $w = \langle M \rangle$ gilt:

$$
\begin{aligned}
w \in H_\epsilon \quad &\Rightarrow \quad M \text{ hält auf } \epsilon \\
&\Rightarrow \quad M^* \text{ berechnet } f \\
&\overset{f \in \mathcal{S}}{\Rightarrow} \quad \langle M^* \rangle \in L(\mathcal{S}) \\
&\Rightarrow \quad M_{L(\mathcal{S})} \text{ akzeptiert } \langle M^* \rangle \\
&\Rightarrow \quad M_{H_\epsilon} \text{ akzeptiert } w
\end{aligned}
$$

$$
\begin{aligned}
w \notin H_\epsilon \quad &\Rightarrow \quad M \text{ hält nicht auf } \epsilon \\
&\Rightarrow \quad M^* \text{ berechnet } u \\
&\overset{u \notin \mathcal{S}}{\Rightarrow} \quad \langle M^* \rangle \notin L(\mathcal{S}) \\
&\Rightarrow \quad M_{L(\mathcal{S})} \text{ verwirft } \langle M^* \rangle
\end{aligned}
$$

Satz von Rice – Fortsetzung Beweis

Korrektheit:

Bei Eingabe von w, wobei w keine Gödelnummer ist, verwirft M_{H_ϵ}.

Bei Eingabe von $w = \langle M \rangle$ gilt:

$$
\begin{aligned}
w \in H_\epsilon \quad &\Rightarrow \quad M \text{ hält auf } \epsilon \\
&\Rightarrow \quad M^* \text{ berechnet } f \\
&\overset{f \in \mathcal{S}}{\Rightarrow} \quad \langle M^* \rangle \in L(\mathcal{S}) \\
&\Rightarrow \quad M_{L(\mathcal{S})} \text{ akzeptiert } \langle M^* \rangle \\
&\Rightarrow \quad M_{H_\epsilon} \text{ akzeptiert } w
\end{aligned}
$$

$$
\begin{aligned}
w \notin H_\epsilon \quad &\Rightarrow \quad M \text{ hält nicht auf } \epsilon \\
&\Rightarrow \quad M^* \text{ berechnet } u \\
&\overset{u \notin \mathcal{S}}{\Rightarrow} \quad \langle M^* \rangle \notin L(\mathcal{S}) \\
&\Rightarrow \quad M_{L(\mathcal{S})} \text{ verwirft } \langle M^* \rangle \\
&\Rightarrow \quad M_{H_\epsilon} \text{ verwirft } w
\end{aligned}
$$

□

Satz von Rice – Weitere Anwendungsbeispiele

Beispiel 4

- Sei
  $L_{17} = \{\langle M \rangle \mid M \text{ berechnet bei Eingabe der Zahl 17 die Zahl 42}\}$.

Satz von Rice – Weitere Anwendungsbeispiele

Beispiel 4

- Sei
  $L_{17} = \{\langle M \rangle \mid M \text{ berechnet bei Eingabe der Zahl 17 die Zahl 42}\}$.
- Es ist $L_{17} = L(\mathcal{S})$ für $\mathcal{S} = \{f_M \mid f_M(\mathsf{bin}(17)) = \mathsf{bin}(42)\}$.

Satz von Rice – Weitere Anwendungsbeispiele

Beispiel 4

- Sei
  $L_{17} = \{\langle M \rangle \mid M \text{ berechnet bei Eingabe der Zahl 17 die Zahl 42}\}$.
- Es ist $L_{17} = L(\mathcal{S})$ für $\mathcal{S} = \{f_M \mid f_M(\mathsf{bin}(17)) = \mathsf{bin}(42)\}$.
- Somit, (da $\emptyset \subsetneq \mathcal{S} \subsetneq \mathcal{R}$), ist diese Sprache gemäß des Satzes von Rice nicht entscheidbar.

Satz von Rice – Weitere Anwendungsbeispiele

Beispiel 4

- Sei
  $L_{17} = \{\langle M \rangle \mid M \text{ berechnet bei Eingabe der Zahl 17 die Zahl 42}\}$.
- Es ist $L_{17} = L(\mathcal{S})$ für $\mathcal{S} = \{f_M \mid f_M(\mathsf{bin}(17)) = \mathsf{bin}(42)\}$.
- Somit, (da $\emptyset \subsetneq \mathcal{S} \subsetneq \mathcal{R}$), ist diese Sprache gemäß des Satzes von Rice nicht entscheidbar.

Beispiel 5

- Sei $H_{42} = \{\langle M \rangle \mid$
  Auf jeder Eingabe hält M nach höchstens 42 Schritten$\}$.

Satz von Rice – Weitere Anwendungsbeispiele

Beispiel 4

- Sei
  $L_{17} = \{\langle M \rangle \mid M \text{ berechnet bei Eingabe der Zahl 17 die Zahl 42}\}$.
- Es ist $L_{17} = L(\mathcal{S})$ für $\mathcal{S} = \{f_M \mid f_M(\text{bin}(17)) = \text{bin}(42)\}$.
- Somit, (da $\emptyset \subsetneq \mathcal{S} \subsetneq \mathcal{R}$), ist diese Sprache gemäß des Satzes von Rice nicht entscheidbar.

Beispiel 5

- Sei $H_{42} = \{\langle M \rangle \mid$
  Auf jeder Eingabe hält M nach höchstens 42 Schritten$\}$.
- Über diese Sprache sagt der Satz von Rice nichts aus!

Satz von Rice – Weitere Anwendungsbeispiele

Beispiel 4

- Sei
  $L_{17} = \{\langle M \rangle \mid M \text{ berechnet bei Eingabe der Zahl 17 die Zahl 42}\}$.
- Es ist $L_{17} = L(\mathcal{S})$ für $\mathcal{S} = \{f_M \mid f_M(\text{bin}(17)) = \text{bin}(42)\}$.
- Somit, (da $\emptyset \subsetneq \mathcal{S} \subsetneq \mathcal{R}$), ist diese Sprache gemäß des Satzes von Rice nicht entscheidbar.

Beispiel 5

- Sei $H_{42} = \{\langle M \rangle \mid$
  Auf jeder Eingabe hält M nach höchstens 42 Schritten$\}$.
- Über diese Sprache sagt der Satz von Rice nichts aus!
- Ist H_{42} entscheidbar?

Satz von Rice für C++ Programme

Konsequenz für C++
Es gibt keine algorithmische Methode (von Hand oder automatisiert) festzustellen, ob ein C++ Programm einer (nicht-trivialen) Spezifikation entspricht.

Die analoge Konsequenz gilt auch für ähnliche Programmiersprachen wie Java.

Satz von Rice – Weitere Anwendungsbeispiele

Beispiel 6

- Sei $L_D = \{\langle M \rangle \mid M \text{ entscheidet die Diagonalsprache}\}$.

Satz von Rice – Weitere Anwendungsbeispiele

Beispiel 6

- Sei $L_D = \{\langle M \rangle \mid M \text{ entscheidet die Diagonalsprache}\}$.
- Dann ist $L_D = L(\mathcal{S})$ für $\mathcal{S} = \{f_D\}$ wobei

$$f_D(w) = \begin{cases} 1 & \text{wenn } w \in D \\ 0 & \text{sonst.} \end{cases}$$

Satz von Rice – Weitere Anwendungsbeispiele

Beispiel 6

- Sei $L_D = \{\langle M \rangle \mid M \text{ entscheidet die Diagonalsprache}\}$.
- Dann ist $L_D = L(\mathcal{S})$ für $\mathcal{S} = \{f_D\}$ wobei

$$f_D(w) = \begin{cases} 1 & \text{wenn } w \in D \\ 0 & \text{sonst.} \end{cases}$$

- Über diese Sprache sagt der Satz von Rice nichts aus!

Satz von Rice – Weitere Anwendungsbeispiele

Beispiel 6

- Sei $L_D = \{\langle M \rangle \mid M \text{ entscheidet die Diagonalsprache}\}$.
- Dann ist $L_D = L(\mathcal{S})$ für $\mathcal{S} = \{f_D\}$ wobei

$$f_D(w) = \begin{cases} 1 & \text{wenn } w \in D \\ 0 & \text{sonst.} \end{cases}$$

- Über diese Sprache sagt der Satz von Rice nichts aus!
- Aber: Diese Sprache ist entscheidbar, denn $L_D = \{\}$.

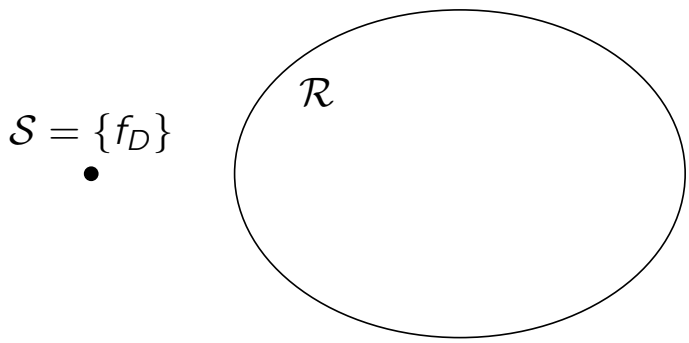
Satz von Rice – Weitere Anwendungsbeispiele

Beispiel 6

- Sei $L_D = \{\langle M \rangle \mid M \text{ entscheidet die Diagonalsprache}\}$.
- Dann ist $L_D = L(\mathcal{S})$ für $\mathcal{S} = \{f_D\}$ wobei

$$f_D(w) = \begin{cases} 1 & \text{wenn } w \in D \\ 0 & \text{sonst.} \end{cases}$$

- Über diese Sprache sagt der Satz von Rice nichts aus!
- Aber: Diese Sprache ist entscheidbar, denn $L_D = \{\}$.



Collatz Problem

Erinnerung an die Iterationsgleichung

$$x \leftarrow \begin{cases} x/2 & \text{wenn } x \text{ gerade} \\ 3x+1 & \text{sonst.} \end{cases}$$

Das Collatz-Problem ist eine Instanz des allgemeinen Halteproblems.

Wir wissen nicht, ob diese Instanz eine Ja- oder eine Nein-Instanz ist.

Vorlesung 8
Rekursive Aufzählbarkeit

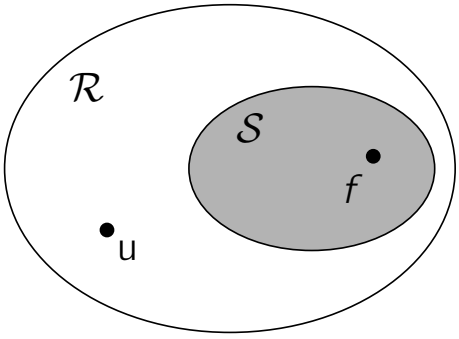
Wdh.: Der Satz von Rice

Satz

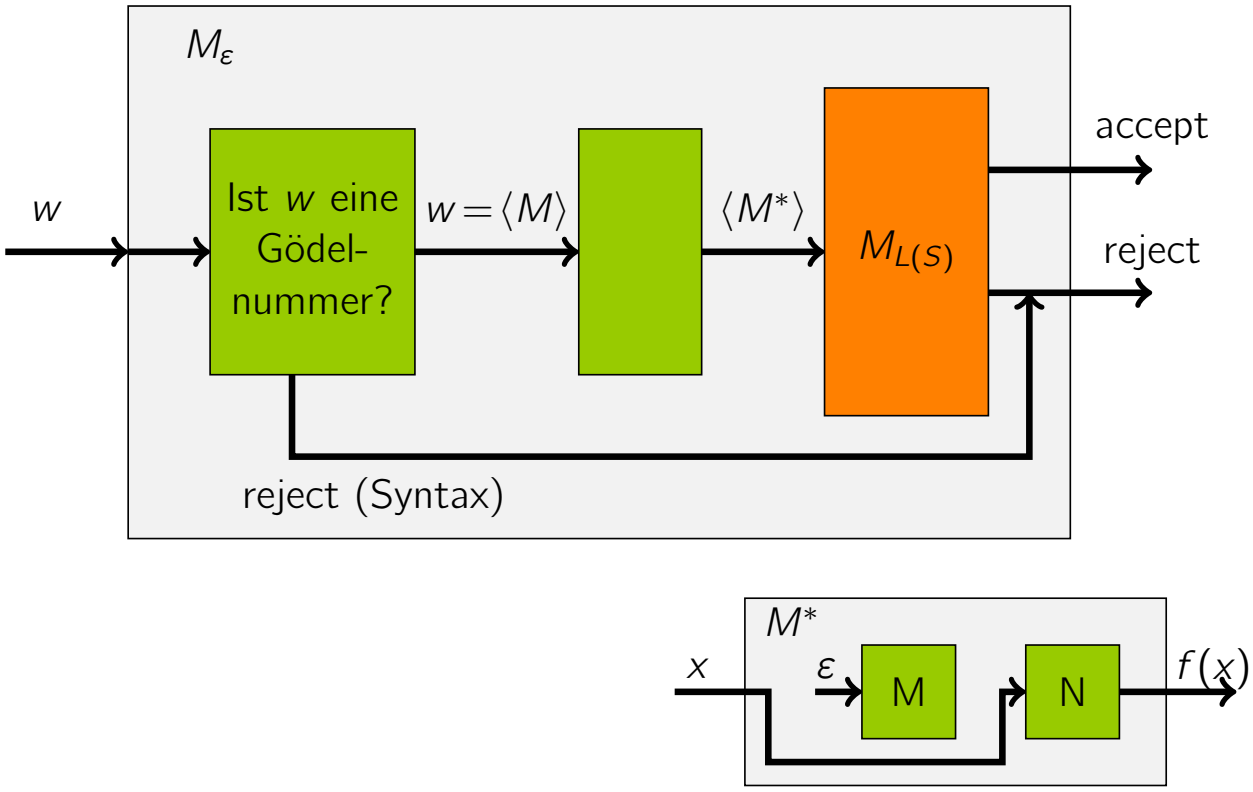
Sei $\mathcal{R}$ die Menge der von TMen berechenbaren partiellen Funktionen und $\mathcal{S}$ eine Teilmenge von $\mathcal{R}$ mit $\emptyset \subsetneq \mathcal{S} \subsetneq \mathcal{R}$. Dann ist die Sprache

$$L(\mathcal{S}) = \{\langle M \rangle \mid M \text{ berechnet eine Funktion aus } \mathcal{S}\}$$

unentscheidbar.



Wdh.: Satz von Rice – Illustration



Wdh.: Satz von Rice – Anwendungsbeispiele

Beispiel 2

- Sei $\mathcal{S} = \{f_M \mid \forall w \in \{0,1\}^* \colon f_M(w) \neq \bot\}$.
- Dann ist

$$\begin{aligned} L(\mathcal{S}) &= \{\langle M \rangle \mid M \text{ berechnet eine Funktion aus } \mathcal{S}\} \\ &= \{\langle M \rangle \mid M \text{ hält auf jeder Eingabe}\} \end{aligned}$$

- Diese Sprache ist auch als das *allgemeine Halteproblem* H_{all} bekannt.
- Gemäß Satz von Rice ist H_{all} unentscheidbar.

Beispiel 5

- Sei $H_{42} = \{\langle M \rangle \mid$
  Auf jeder Eingabe hält M nach höchstens 42 Schritten$\}$.
- Über diese Sprache sagt der Satz von Rice nichts aus!
- Ist H_{42} entscheidbar?

Erkennen vs Entscheiden

Definition

Sei M eine Turingmaschine. Dann ist $L(M)$ die Menge der von M akzeptieren Wörter, also

$$L(M) = \{w \in \{0,1\}^* \mid M \text{ akzeptiert } w\}.$$

Wir nennen $L(M)$ die von M erkannte Sprache.

Achtung: Bei Eingabe eines Wortes $w \notin L(M)$ hält M entweder nicht an oder hält an und verwirft.

Beobachtung

$$L(M) = \{w \mid \text{das Wort } f_M(w) \text{ beginnt mit einer } 1\}.$$

Beachte

Wenn M auf allen Eingaben hält, dann ist $L(M)$ gerade die von M entschiedene Sprache.

Semi-Entscheidbarkeit

Eine Sprache L wird von einer TM M entschieden, wenn

- M auf jeder Eingabe hält, und
- M genau die Wörter aus L akzeptiert.

Eine Sprache L, für die eine TM existiert, die L entscheidet, wird als entscheidbar bezeichnet.

Eine Sprache L wird von einer TM M erkannt, wenn

- M jedes Wort aus L akzeptiert, und
- M kein Wort akzeptiert, das nicht in L enthalten ist.

Es ist also $L(M)$ gerade die von M erkannte Sprache.

Definition

Eine Sprache L, für die eine TM existiert, die L erkennt, wird als semi-entscheidbar bezeichnet.

Beispiel

Das Halteproblem

$$H = \{\langle M \rangle w \mid M \text{ hält auf } w\}.$$

ist unentscheidbar.

Behauptung

Das Halteproblem ist semi-entscheidbar.

Beweis

Die folgende TM M' erkennt H:
Erhält M' eine Eingabe der Form $\langle M \rangle w$, so

- simuliert M' die TM M mit Eingabe w, und
- akzeptiert, falls M auf w hält.

Syntaktisch inkorrekte Eingaben werden von M' verworfen. □

Aufzähler, rekursive Aufzählbarkeit

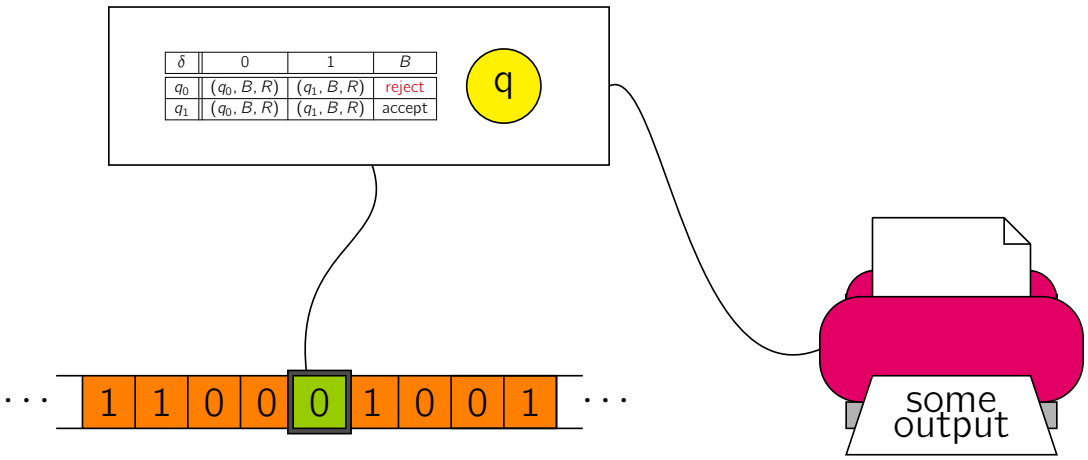
Ein Aufzähler für eine Sprache $L \subseteq \Sigma^*$ ist eine Variante einer TM mit einem angeschlossenen Drucker im Sinne eines zusätzlichen Ausgabebandes, auf dem sich der Kopf nur nach rechts bewegt.

- Gestartet mit leerem Arbeitsband, gibt der Aufzähler alle Wörter aus L (möglicherweise mit Wiederholungen) auf dem Drucker aus.
- Die ausgegebenen Wörter sind dabei durch ein Zeichen getrennt, das nicht in Σ enthalten ist.

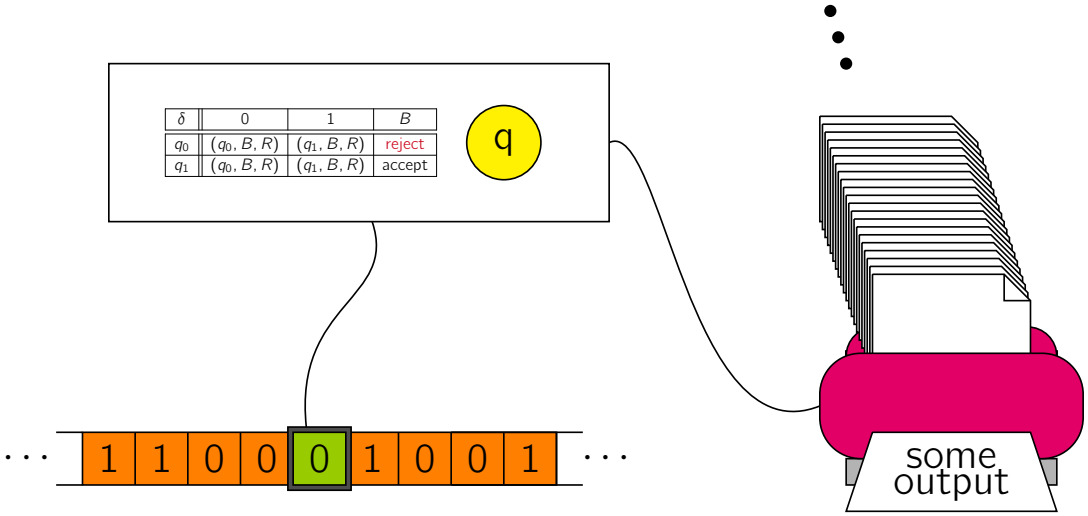
Definition

Eine Sprache, für die es einen Aufzähler gibt, heißt rekursiv aufzählbar.

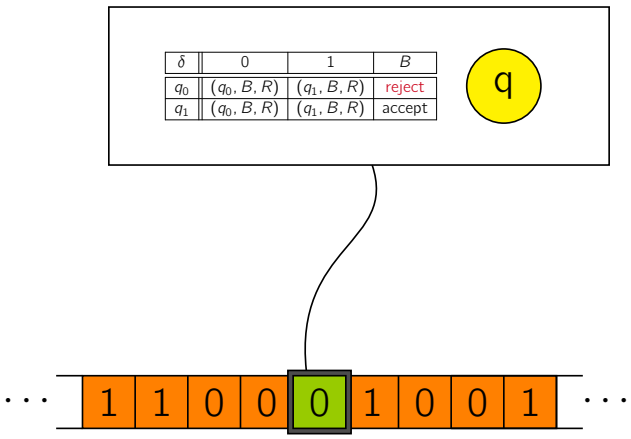
Aufzähler – Illustration



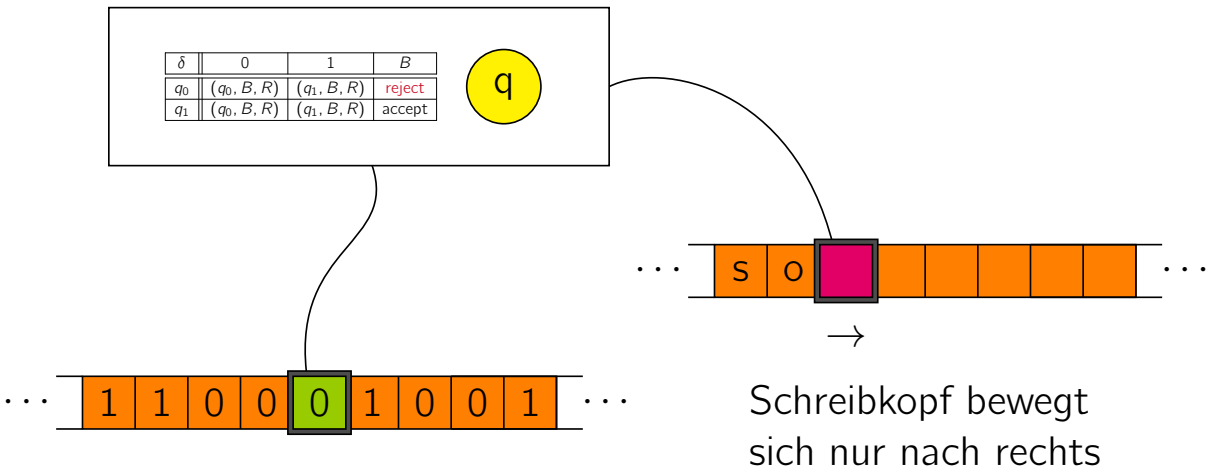
Aufzähler – Illustration



Aufzähler – Illustration

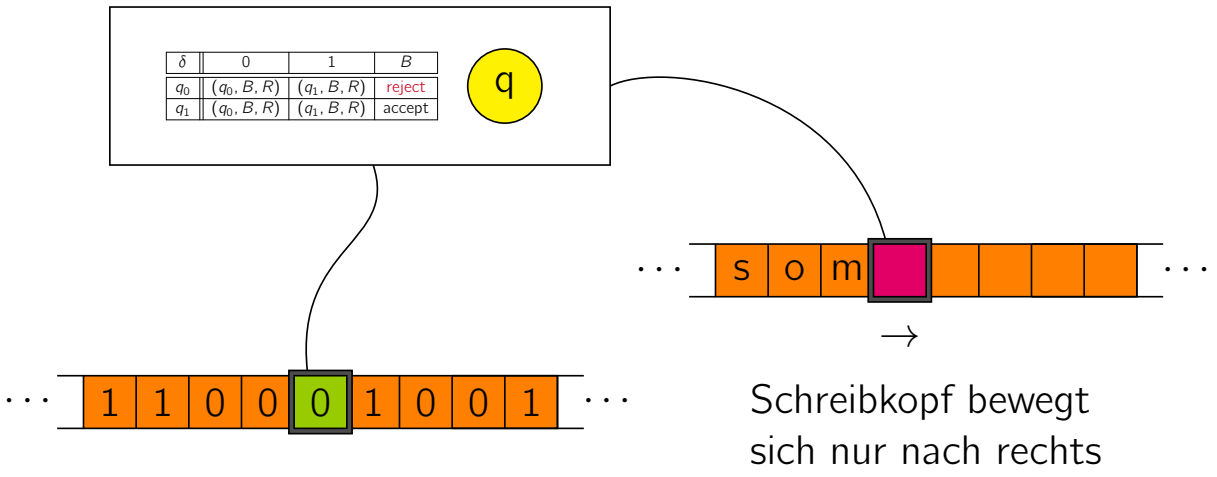


Aufzähler – Illustration

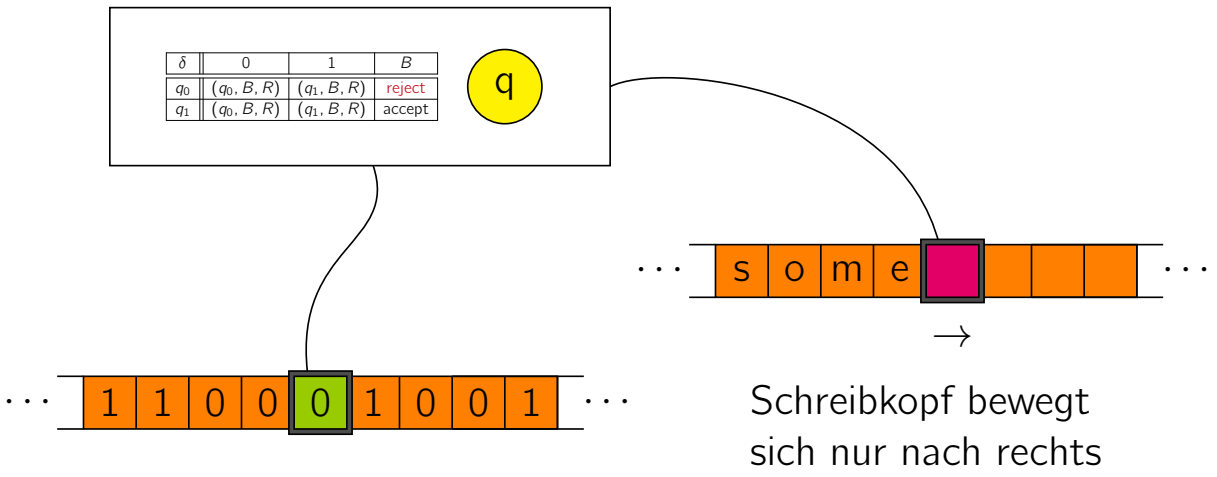


Schreibkopf bewegt sich nur nach rechts

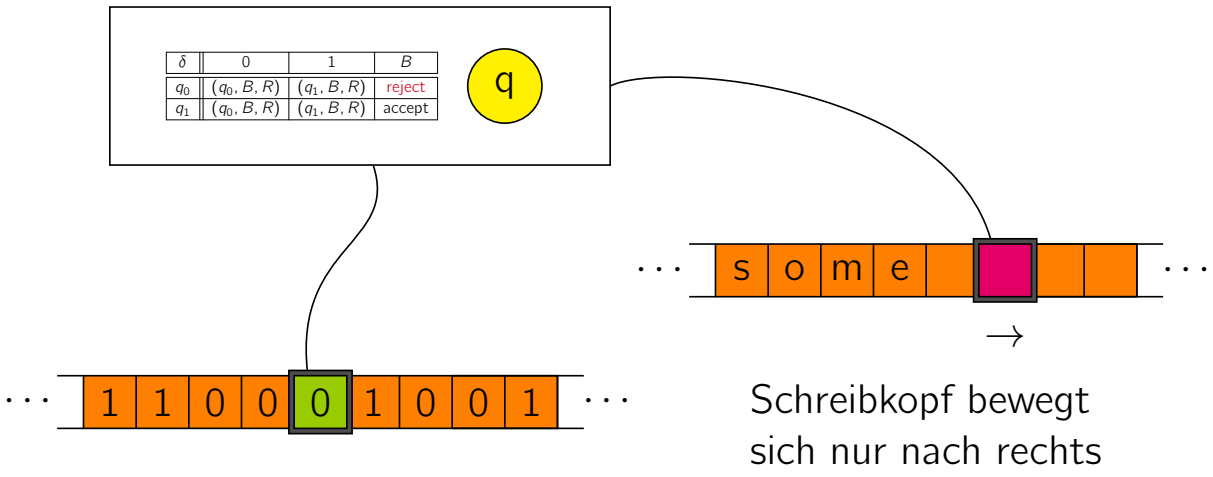
Aufzähler – Illustration



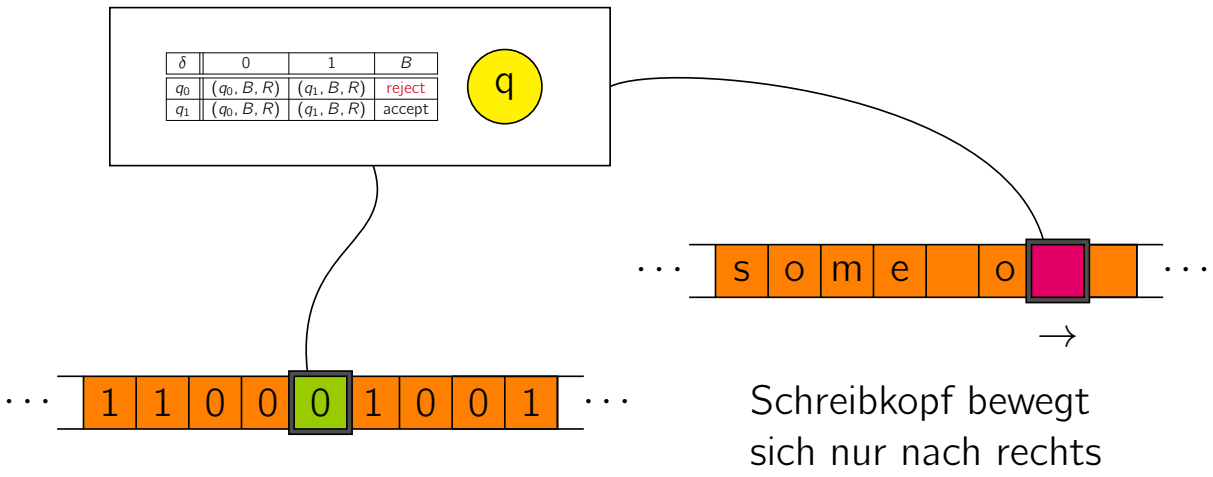
Aufzähler – Illustration



Aufzähler – Illustration



Aufzähler – Illustration



rekursiv aufzählbar = semi-entscheidbar

Satz

Eine Sprache L ist genau dann semi-entscheidbar, wenn sie rekursiv aufzählbar ist.

Beweis: L rekursiv aufzählbar $\Rightarrow$ L semi-entscheidbar

Sei A ein Aufzähler für L. Wir konstruieren eine TM M, die L erkennt.

Bei Eingabe w arbeitet M wie folgt:

- M simuliert A mit Hilfe einer Spur, welche die Rolle des Druckers übernimmt.
- Immer wenn ein neues Wort gedruckt worden ist, vergleicht M dieses Wort mit w und akzeptiert bei Übereinstimmung.

Korrektheit:

- Falls $w \in L$, so wird w irgendwann gedruckt und somit von M akzeptiert.
- Falls $w \notin L$, so wird w nicht gedruckt und somit nicht akzeptiert.

Beweis: L semi-entscheidbar $\Rightarrow$ L rekursiv aufzählbar

Sei M eine TM, die L erkennt. Wir konstruieren einen Aufzähler A für L.

Sei $x_0, x_1, x_2, \ldots$ die Aufzählung von Σ^* in kanonischer Reihenfolge.

Programm von A:

Für $i = 1, 2, 3, \ldots$

- Simuliere je i Schritte von M auf jedem Wort aus $\{x_0, \ldots, x_{i-1}\}$.
- Wann immer dabei eines der Worte akzeptiert wird, so drucke es aus.

Korrektheit:

Aufzähler A druckt offensichtlich nur Wörter aus L aus. Aber druckt er auch alle Wörter aus L aus?

- Sei x_k ein Wort aus L.
- Dann wird x_k von M nach einer endlichen Anzahl von Schritten t_k akzeptiert.
- Das heißt, x_k wird in jeder Iteration i mit $i \geq \max\{k, t_k\}$ von A ausgedruckt.

□

Schnitte von Sprachen

Satz

a) *Wenn die Sprachen L_1 und L_2 entscheidbar sind, so ist auch die Sprache $L_1 \cap L_2$ entscheidbar.*

b) *Wenn die Sprachen L_1 und L_2 semi-entscheidbar sind, so ist auch die Sprache $L_1 \cap L_2$ semi-entscheidbar.*

Schnitte von Sprachen – Beweis a)

Seien M_1 und M_2 zwei TMen, die L_1 bzw. L_2 entscheiden.

TM M, die $L_1 \cap L_2$ entscheidet:

1. Bei Eingabe w simuliert M zunächst das Verhalten von M_1 auf w und dann das Verhalten von M_2 auf w.
2. Falls M_1 und M_2 das Wort w akzeptieren, so akzeptiert auch M; sonst verwirft M.

Korrektheit:

- Falls $w \in L_1 \cap L_2$, so wird w akzeptiert.
- Sonst wird w verworfen.

□

Schnitte von Sprachen – Beweis b)

Seien nun M_1 und M_2 zwei TMen, die L_1 bzw. L_2 erkennen.

TM M, die $L_1 \cap L_2$ erkennt:

1. Bei Eingabe w simuliert M zunächst das Verhalten von M_1 auf w und dann das Verhalten von M_2 auf w.
2. Falls M_1 und M_2 akzeptieren, so akzeptiert auch M.

Wir verwenden die gleiche Konstruktion für M wie in a).

Korrektheit:

- Falls $w \in L_1 \cap L_2$, so wird w von M akzeptiert.
- Sonst wird w nicht akzeptiert.

□

Vereinigungen von Sprachen

Satz

a) *Wenn die Sprachen L_1 und L_2 entscheidbar sind, so ist auch die Sprache $L_1 \cup L_2$ entscheidbar.*

b) *Wenn die Sprachen L_1 und L_2 semi-entscheidbar sind, so ist auch die Sprache $L_1 \cup L_2$ semi-entscheidbar.*

Vereinigungen von Sprachen – Beweis a)

Seien M_1 und M_2 zwei TMen, die L_1 bzw. L_2 entscheiden.

TM M, die $L_1 \cup L_2$ entscheidet

- Bei Eingabe w simuliert M zunächst das Verhalten von M_1 auf w und dann das Verhalten von M_2 auf w.
- Falls M_1 oder M_2 akzeptiert, so akzeptiert auch M. Sonst verwirft M.

Korrektheit:

- Falls $w \in L_1 \cup L_2$, so wird w von M_1 oder von M_2 und somit auch von M akzeptiert.
- Sonst verwerfen M_1 und M_2, so dass auch M verwirft.

□

Vereinigungen von Sprachen – Beweis b)

Seien nun M_1 und M_2 zwei TMen, die L_1 bzw. L_2 erkennen.

Welches Problem tritt auf, wenn wir die Simulation aus a) einfach übernehmen?

Idee: Simuliere M_1 und M_2 parallel statt sequentiell.

Vereinigungen von Sprachen – Beweis b)

TM M, die $L_1 \cup L_2$ erkennt

- Wir nehmen o.B.d.A. an, dass M über zwei Bänder verfügt.
- Auf Band 1 wird M_1 auf w simuliert.
- Auf Band 2 wird M_2 auf w simuliert.
- Sobald ein Schritt erreicht wird, in dem M_1 oder M_2 akzeptiert, akzeptiert auch M.

Korrektheit:

- Falls $w \in L_1 \cup L_2$, so wird w von M_1 oder von M_2 und somit auch von M akzeptiert.
- Sonst wird w nicht akzeptiert.

□

„2×semi-entscheidbar ⇔ entscheidbar“

Lemma ∗
Seien $L \subseteq \Sigma^*$ und $\overline{L} := \Sigma^* \setminus L$ semi-entscheidbar. Dann ist L entscheidbar.

Beweis
Seien M und $\overline{M}$ Maschinen, die L bzw. $\overline{L}$ erkennen.

Die TM M' entscheidet L durch eine parallele Simulation von M und $\overline{M}$ auf der Eingabe w:

- M' akzeptiert w, sobald M akzeptiert.
- M' verwirft w, sobald $\overline{M}$ akzeptiert.

Da entweder $w \in L$ oder $w \notin L$, tritt eines der obigen Ereignisse nach endlicher Zeit ein, so dass die Terminierung von M' sichergestellt ist. □

Komplemente von Sprachen

Beobachtung 1
Wenn die Sprache L entscheidbar ist, so ist auch $\overline{L}$ entscheidbar, da wir das Akzeptanzverhalten einer TM M, die M entscheidet, invertieren können.

Beobachtung 2
Die Menge der semi-entscheidbaren Sprachen ist hingegen nicht unter Komplementbildung abgeschlossen.

Beispiel
- H ist semi-entscheidbar.
- Wäre $\overline{H}$ ebenfalls semi-entscheidbar, so wäre H nach Lemma ∗ entscheidbar.
- Also ist $\overline{H}$ nicht semi-entscheidbar.

Schlussfolgerung

Korollar

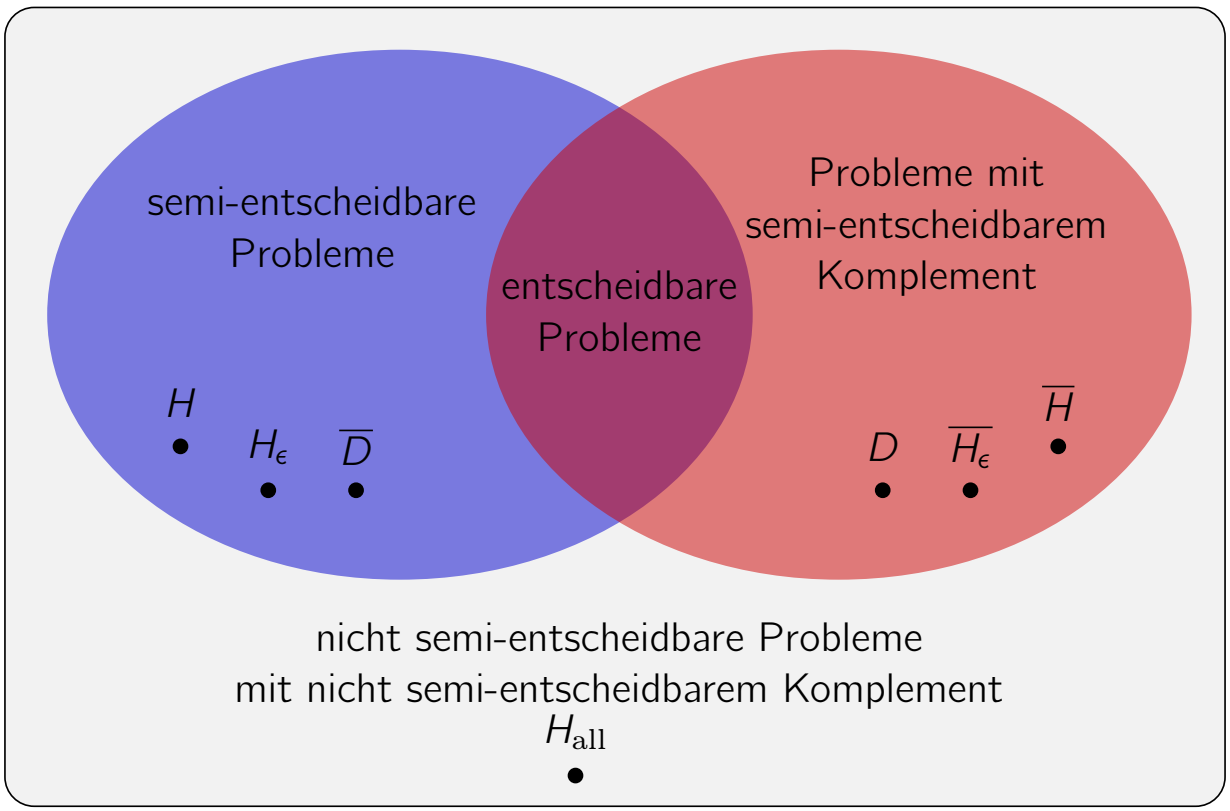
Für jede Sprache L gilt genau eine der vier folgenden Eigenschaften.

1. *L ist entscheidbar und sowohl L als auch $\overline{L}$ sind semi-entscheidbar;*
2. *L ist semi-entscheidbar, aber $\overline{L}$ ist nicht semi-entscheidbar;*
3. *$\overline{L}$ ist semi-entscheidbar, aber L ist nicht semi-entscheidbar;*
4. *sowohl L als auch $\overline{L}$ sind nicht semi-entscheidbar*

Beispiele

- Kategorie 1: Graphzusammenhang, Hamiltonkreis
- Kategorie 2: H, H_ϵ, $\overline{D}$
- Kategorie 3: $\overline{H}$, $\overline{H_\epsilon}$, D,
- Kategorie 4: $H_{\text{all}} = \{\langle M \rangle \mid M \text{ hält auf jeder Eingabe}\}$

Berechenbarkeitslandschaft



Allgemeines Halteproblem

Das *allgemeine Halteproblem* ist definiert als

$$H_{\text{all}} \;=\; \{\langle M\rangle \mid M \text{ hält auf jeder Eingabe}\}$$

Wie kann man nachweisen, dass sowohl H_{all} als auch $\overline{H}_{\text{all}}$ nicht semi-entscheidbar sind?

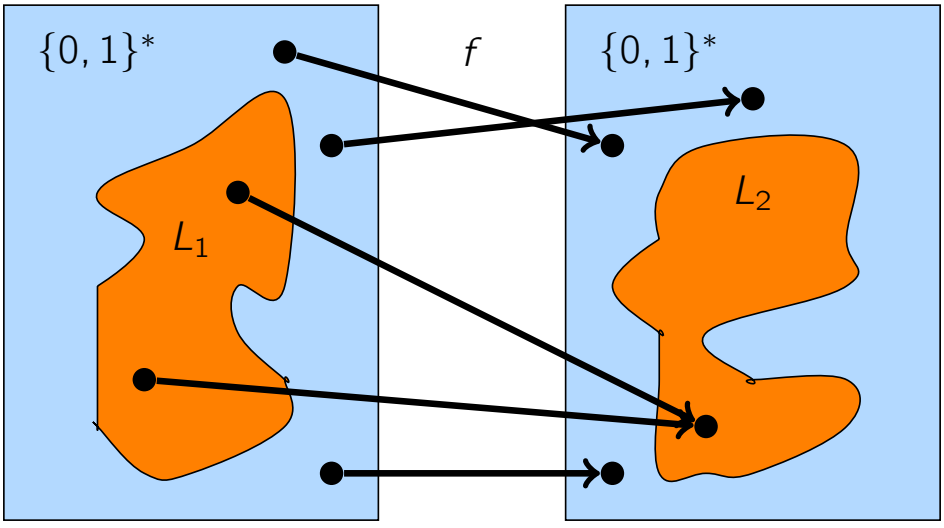
Wir verwenden eine spezielle Variante der Unterprogrammtechnik, die sogenannte Reduktion.

Reduktionen

Definition

Es seien L_1 und L_2 Sprachen über einem Alphabet Σ. Dann heißt L_1 auf L_2 reduzierbar, Notation $L_1 \leq L_2$, wenn es eine berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ gibt, so dass für alle $x \in \Sigma^*$ gilt

$$x \in L_1 \;\Leftrightarrow\; f(x) \in L_2.$$



Reduktionen

Lemma

Falls $L_1 \leq L_2$ und L_2 semi-entscheidbar ist, so ist L_1 semi-entscheidbar.

Beweis

Wir konstruieren eine TM M_1, die L_1 erkennt, indem als Unterprogramm eine TM M_2 verwenden, die L_2 erkennt:

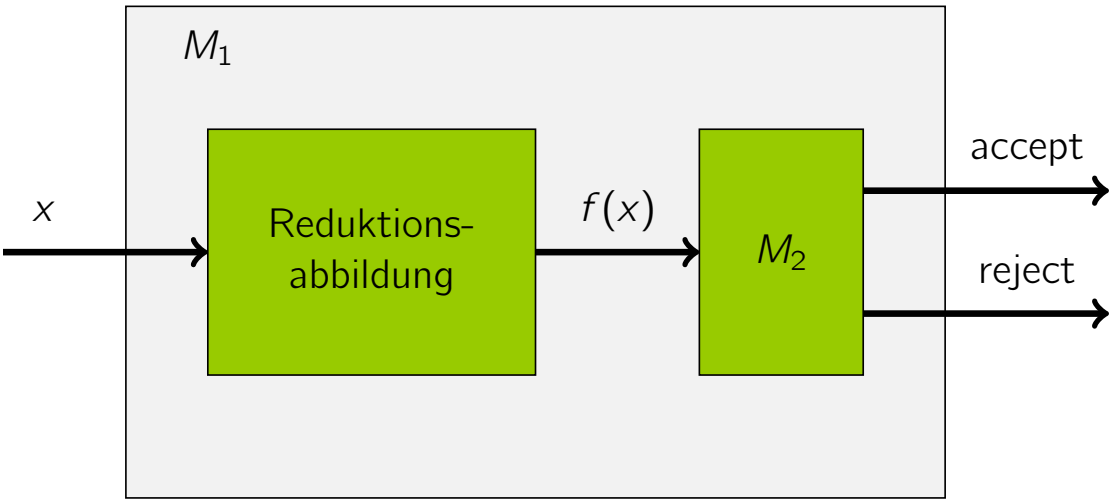
- Die TM M_1 berechnet $f(x)$ auf ihrer Eingabe x. (Dies ist möglich da f berechenbar ist.)
- Dann simuliert M_1 die TM M_2 mit der Eingabe $f(x)$ und übernimmt das Akzeptanzverhalten.

Korrektheit:

$$M_1 \text{ akzeptiert } x \;\Leftrightarrow\; M_2 \text{ akzeptiert } f(x) \;\Leftrightarrow\; f(x) \in L_2 \;\Leftrightarrow\; x \in L_1.$$

□

Reduktionen – Illustration



Verwendung von Reduktionen in umgekehrter Richtung

Im Umkehrschluss gilt:

Korollar

Falls $L_1 \leq L_2$ und L_1 nicht semi-entscheidbar ist, so ist L_2 nicht semi-entscheidbar.

Anwendung

Vorbeobachtung: H_ϵ ist unentscheidbar, aber semi-entscheidbar. Folglich ist $\overline{H}_\epsilon$ nicht semi-entscheidbar.

Wir werden zeigen

Behauptung A

$$\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$$

Behauptung B

$$\overline{H}_\epsilon \leq H_{\text{all}}$$

Aus diesen Reduktionen folgt:

Satz

Weder $\overline{H}_{all}$ noch H_{all} ist semi-entscheidbar.

Vorlesung 9
Allgemeines Halteproblem und Hilberts 10. Problem

Wdh.: Semi-Entscheidbarkeit

Eine Sprache L wird von einer TM M erkannt, wenn

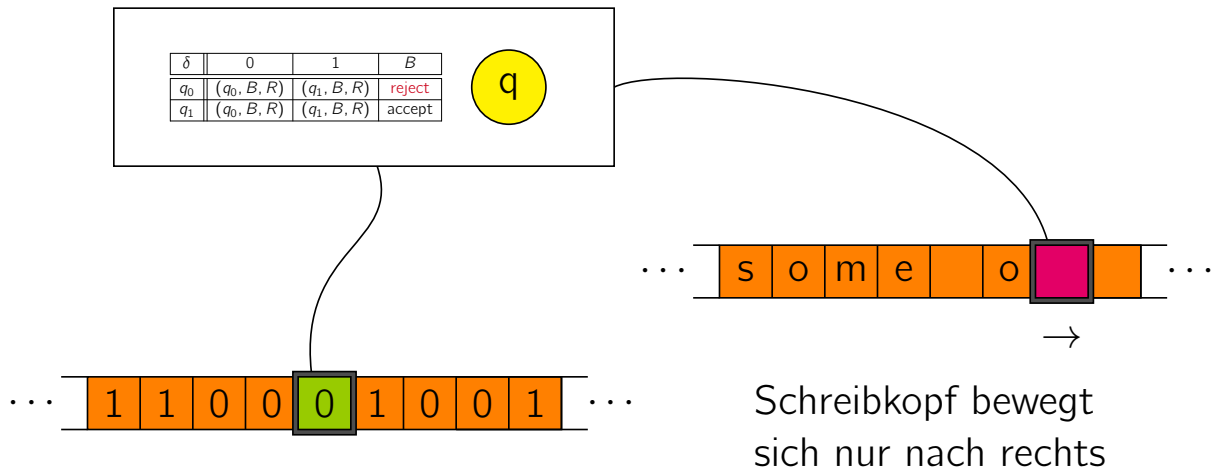
- M jedes Wort aus L akzeptiert, und
- M kein Wort akzeptiert, das nicht in L enthalten ist.

Es ist $L(M)$ die von M erkannte Sprache.

Definition
Eine Sprache L, für die eine TM existiert, die L erkennt, wird als semi-entscheidbar bezeichnet.

Beobachtung
Das Halteproblem ist semi-entscheidbar.

Wdh.: Aufzähler



Wdh.: rekursiv aufzählbar = semi-entscheidbar

Satz

Eine Sprache L ist genau dann semi-entscheidbar, wenn sie rekursiv aufzählbar ist.

Wdh.: rekursiv aufzählbar = semi-entscheidbar

Satz

Eine Sprache L ist genau dann semi-entscheidbar, wenn sie rekursiv aufzählbar ist.

Beweisrichtung „⇒“: Vorhanden ist eine TM M, die L erkennt.

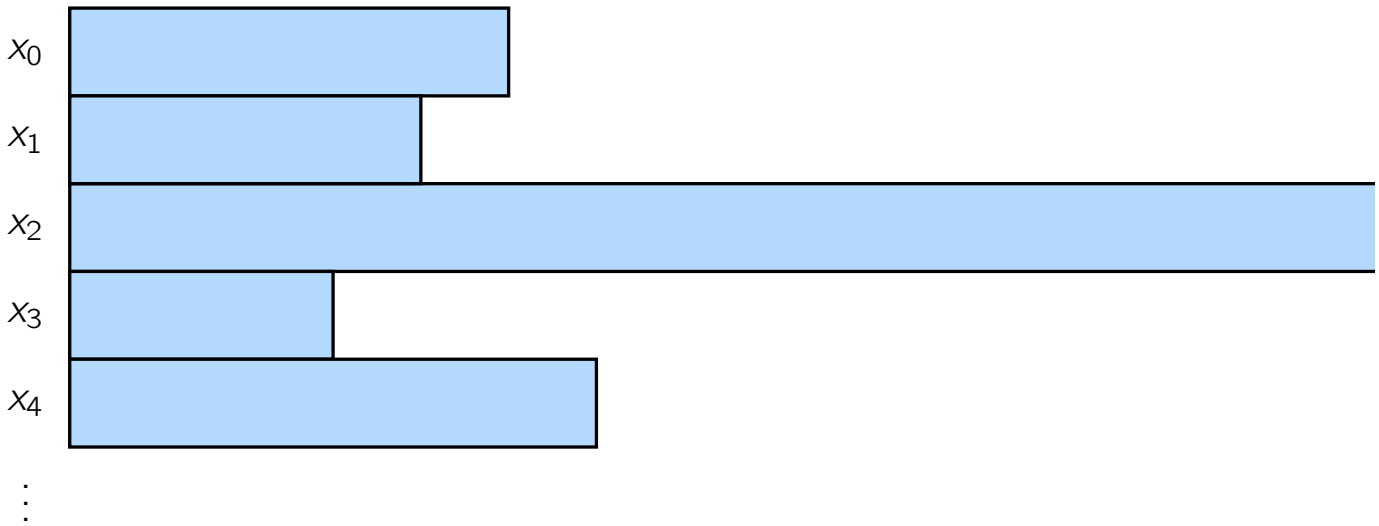
Wdh.: rekursiv aufzählbar = semi-entscheidbar

Satz

Eine Sprache L ist genau dann semi-entscheidbar, wenn sie rekursiv aufzählbar ist.

Beweisrichtung „⇒“: Vorhanden ist eine TM M, die L erkennt.

Anzahl der Schritte, die M auf x_i benötigt.



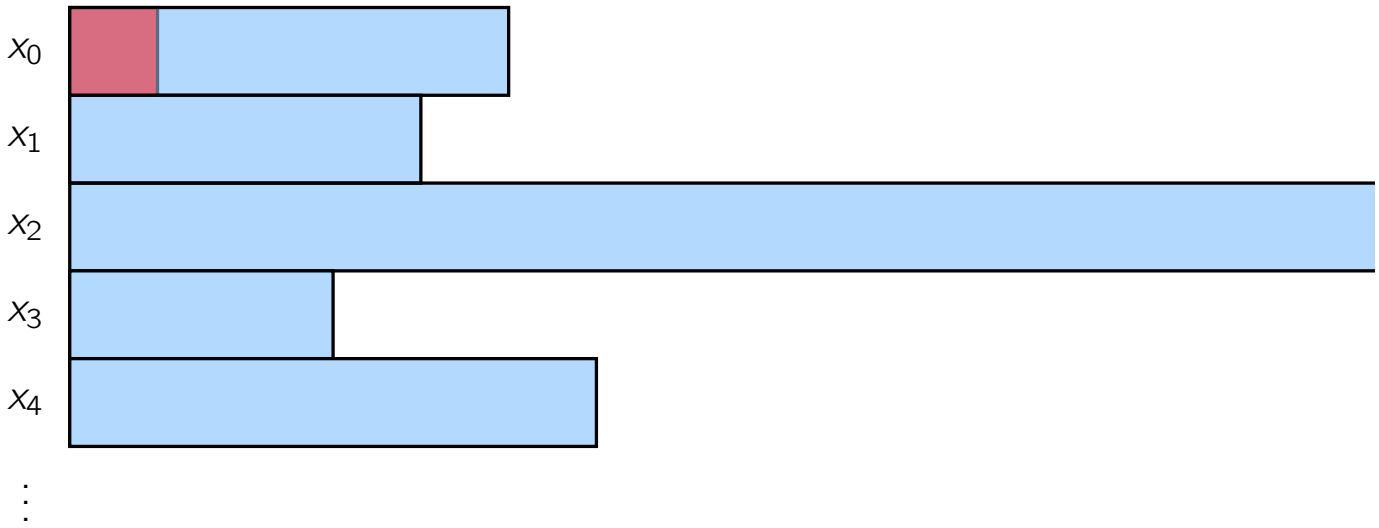
Wdh.: rekursiv aufzählbar = semi-entscheidbar

Satz

Eine Sprache L ist genau dann semi-entscheidbar, wenn sie rekursiv aufzählbar ist.

Beweisrichtung „⇒“: Vorhanden ist eine TM M, die L erkennt.

Anzahl der Schritte, die M auf x_i benötigt.



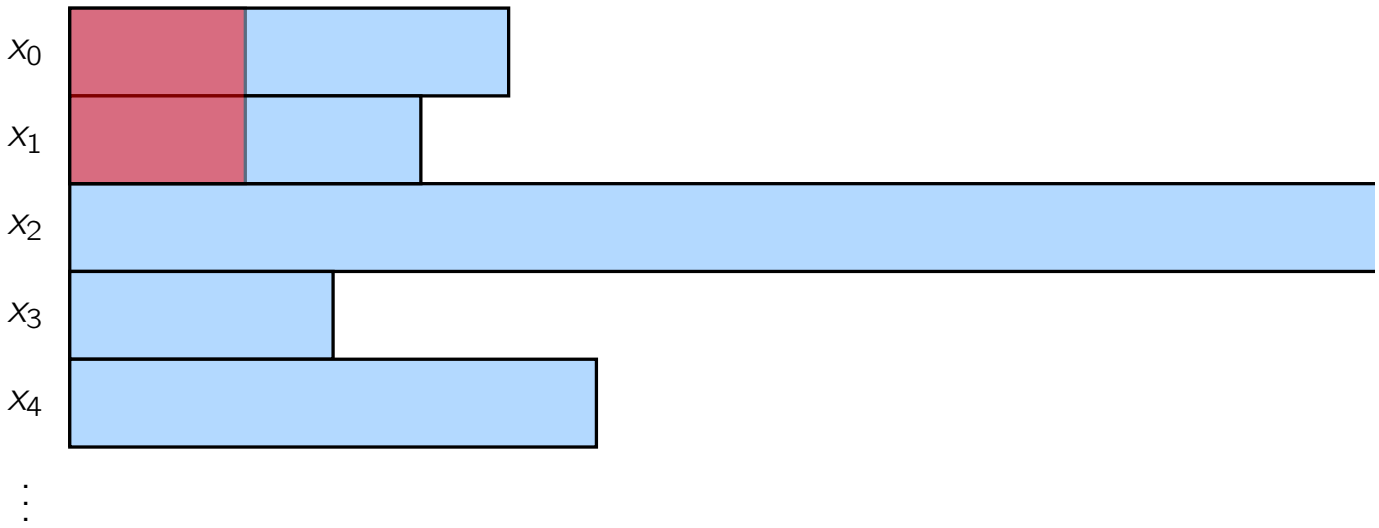
Wdh.: rekursiv aufzählbar = semi-entscheidbar

Satz

Eine Sprache L ist genau dann semi-entscheidbar, wenn sie rekursiv aufzählbar ist.

Beweisrichtung „⇒“: Vorhanden ist eine TM M, die L erkennt.

Anzahl der Schritte, die M auf x_i benötigt.



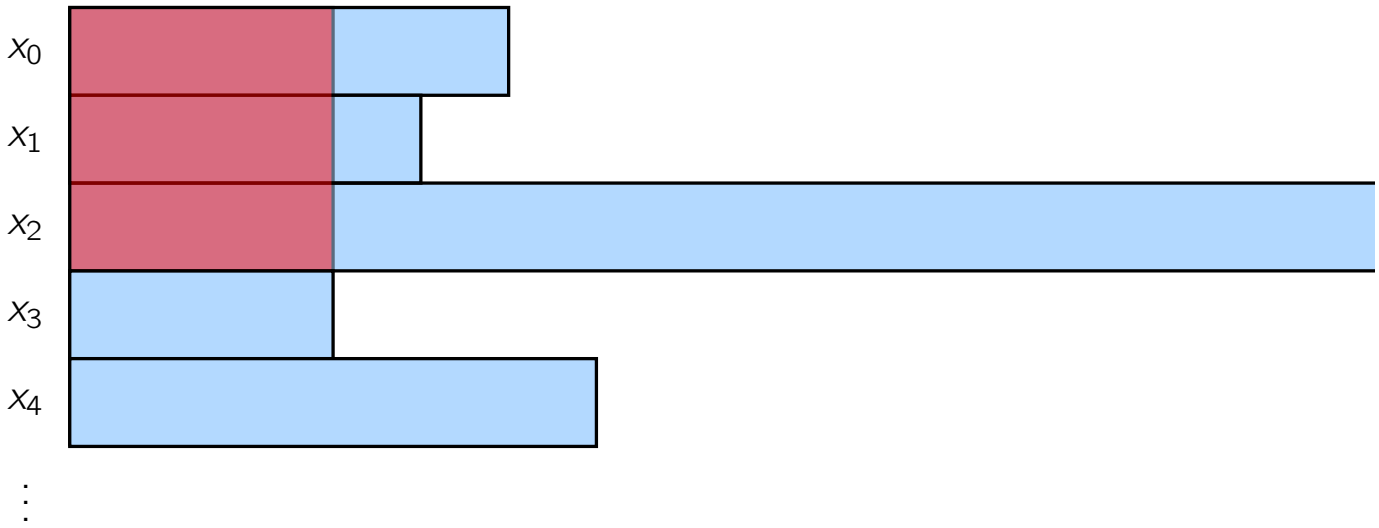
Wdh.: rekursiv aufzählbar = semi-entscheidbar

Satz

Eine Sprache L ist genau dann semi-entscheidbar, wenn sie rekursiv aufzählbar ist.

Beweisrichtung „⇒“: Vorhanden ist eine TM M, die L erkennt.

Anzahl der Schritte, die M auf x_i benötigt.



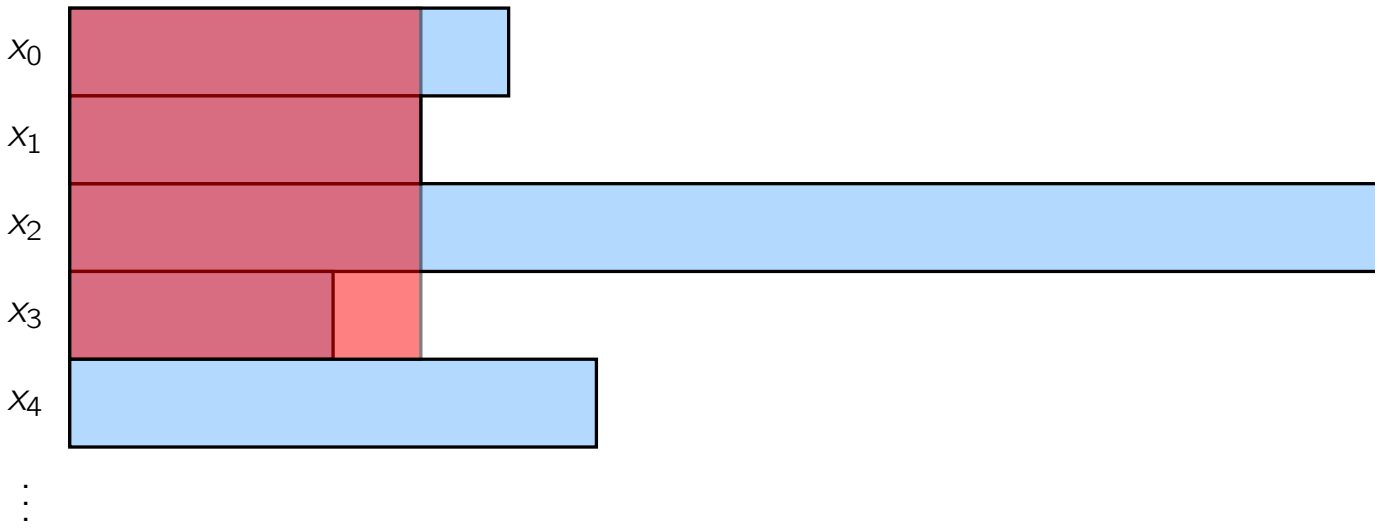
Wdh.: rekursiv aufzählbar = semi-entscheidbar

Satz

Eine Sprache L ist genau dann semi-entscheidbar, wenn sie rekursiv aufzählbar ist.

Beweisrichtung „⇒“: Vorhanden ist eine TM M, die L erkennt.

Anzahl der Schritte, die M auf x_i benötigt.



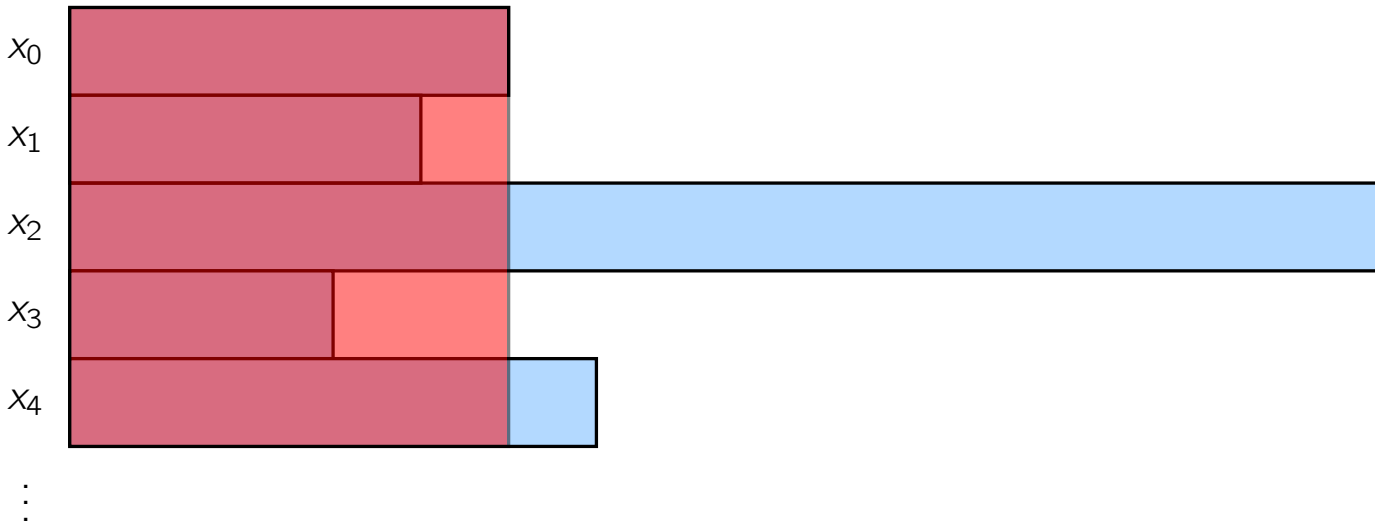
Wdh.: rekursiv aufzählbar = semi-entscheidbar

Satz

Eine Sprache L ist genau dann semi-entscheidbar, wenn sie rekursiv aufzählbar ist.

Beweisrichtung „⇒“: Vorhanden ist eine TM M, die L erkennt.

Anzahl der Schritte, die M auf x_i benötigt.



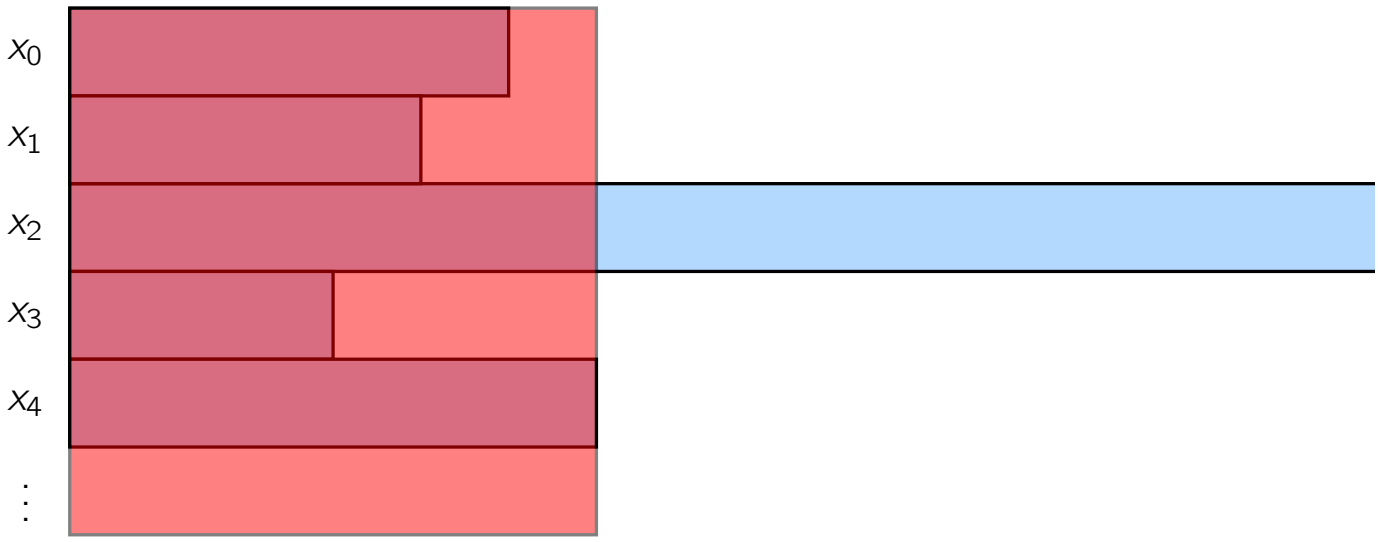
Wdh.: rekursiv aufzählbar = semi-entscheidbar

Satz

Eine Sprache L ist genau dann semi-entscheidbar, wenn sie rekursiv aufzählbar ist.

Beweisrichtung „⇒“: Vorhanden ist eine TM M, die L erkennt.

Anzahl der Schritte, die M auf x_i benötigt.



Wdh.: Abschlusseigenschaften von Sprachen

Wdh.: Abschlusseigenschaften von Sprachen

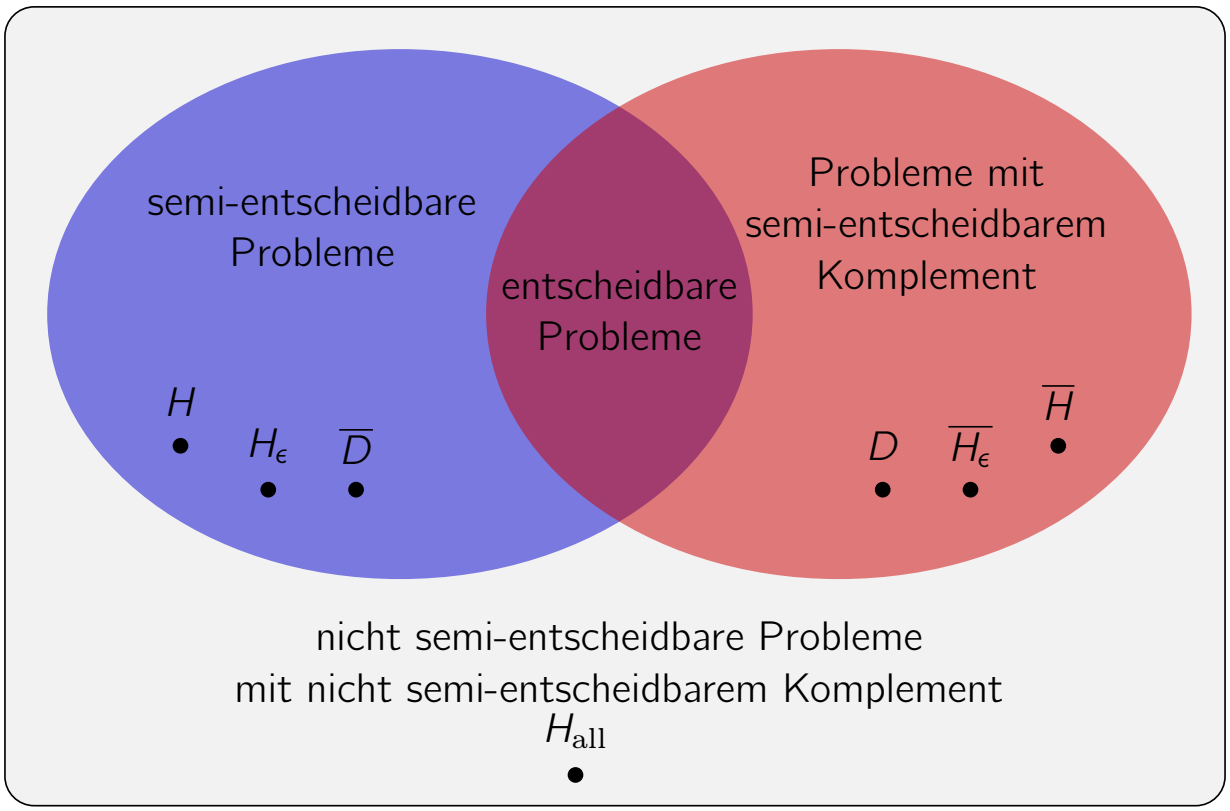
Satz

Die Menge der entscheidbaren Sprachen ist abgeschlossen unter Komplementbildung, Vereinigungen und Schnitten.

Satz

Die Menge der semi-entscheidbaren Sprachen ist abgeschlossen unter Vereinigungen und Schnitten.
Sie ist nicht abgeschlossen unter Komplementbildung.

Wdh.: Berechenbarkeitslandschaft



Wdh.: Allgemeines Halteproblem

Das allgemeine Halteproblem ist definiert als

$$H_{\text{all}} = \{\langle M \rangle \mid M \text{ hält auf jeder Eingabe}\}$$

Wie kann man nachweisen, dass sowohl H_{all} als auch $\overline{H}_{\text{all}}$ nicht semi-entscheidbar sind?

Wdh.: Allgemeines Halteproblem

Das allgemeine Halteproblem ist definiert als

$$H_{\text{all}} \;=\; \{\langle M\rangle \mid M \text{ hält auf jeder Eingabe}\}$$

Wie kann man nachweisen, dass sowohl H_{all} als auch $\overline{H}_{\text{all}}$ nicht semi-entscheidbar sind?

Wir verwenden Reduktionen, eine spezielle Variante der Unterprgrammtechnik.

Wdh.: Reduktionen

Definition

Es seien L_1 und L_2 Sprachen über einem Alphabet Σ. Dann heißt L_1 auf L_2 reduzierbar, Notation $L_1 \leq L_2$, wenn es eine berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ gibt, so dass für alle $x \in \Sigma^*$ gilt

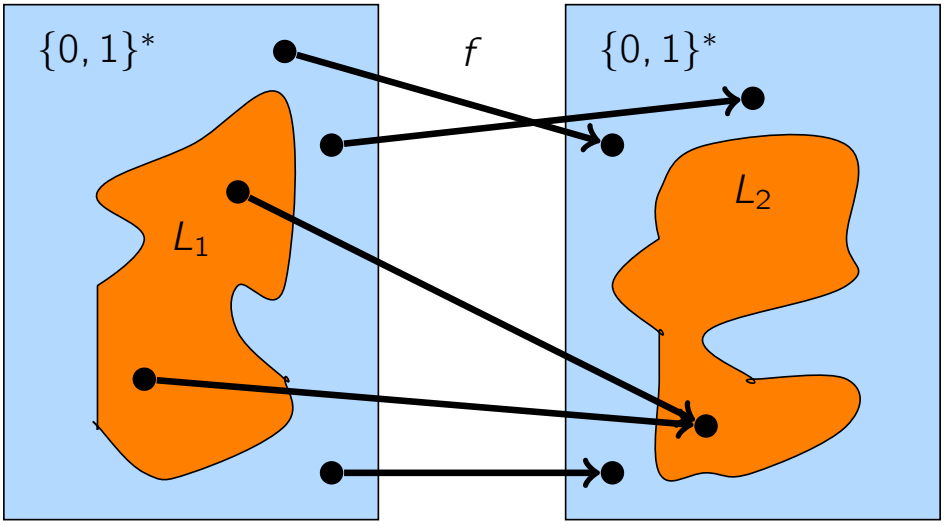
$$x \in L_1 \;\Leftrightarrow\; f(x) \in L_2 \quad .$$

Wdh.: Reduktionen

Definition

Es seien L_1 und L_2 Sprachen über einem Alphabet Σ. Dann heißt L_1 auf L_2 reduzierbar, Notation $L_1 \leq L_2$, wenn es eine berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ gibt, so dass für alle $x \in \Sigma^*$ gilt

$$x \in L_1 \;\Leftrightarrow\; f(x) \in L_2 \;\; .$$



Wdh.: Reduktionen

Wir haben gezeigt:

Lemma

Falls $L_1 \leq L_2$ und L_2 semi-entscheidbar ist, so ist L_1 semi-entscheidbar.

Wdh.: Reduktionen

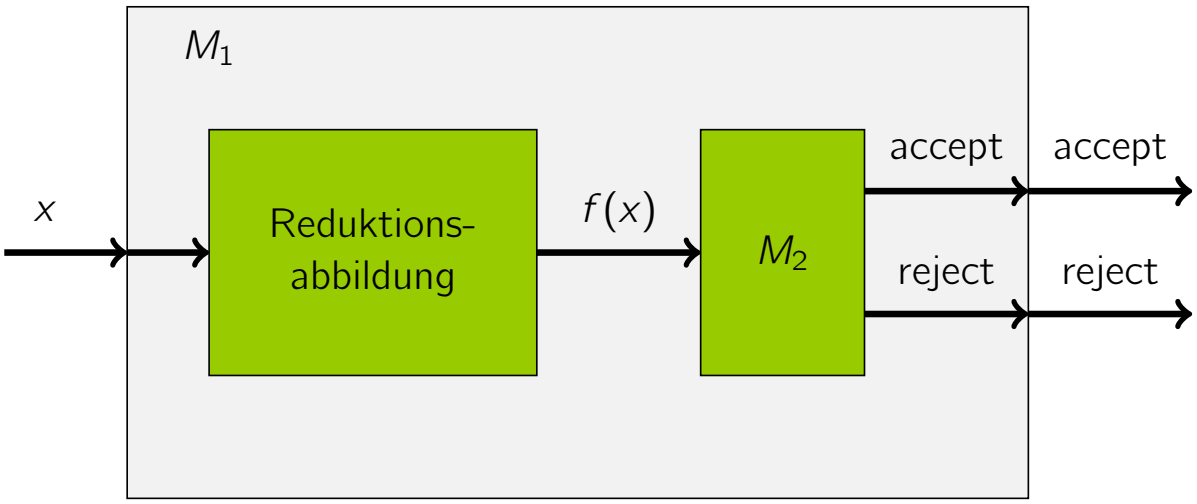
Wir haben gezeigt:

Lemma
Falls $L_1 \leq L_2$ und L_2 semi-entscheidbar ist, so ist L_1 semi-entscheidbar.

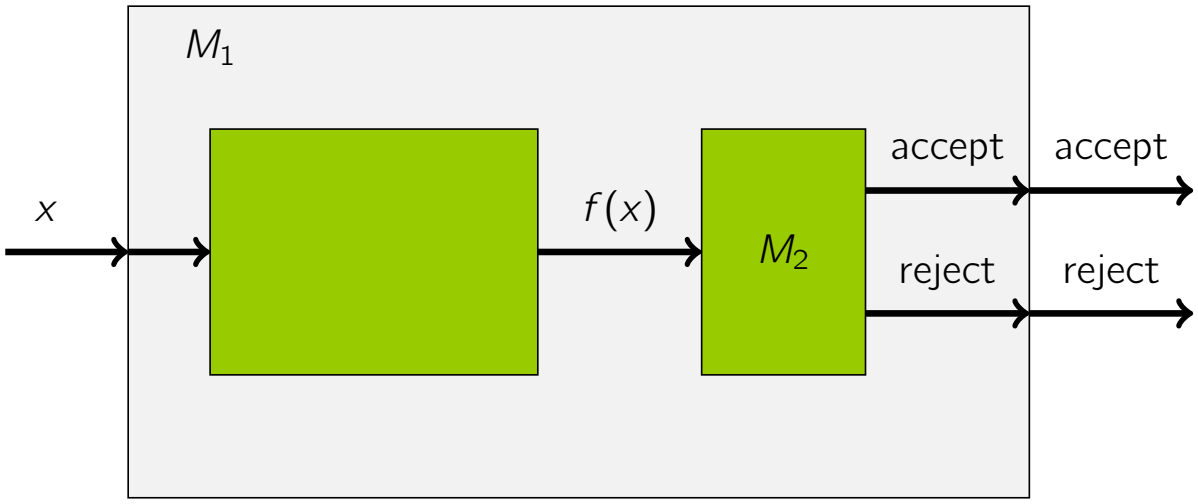
Im Umkehrschluss gilt:

Lemma
Falls $L_1 \leq L_2$ und L_1 nicht semi-entscheidbar ist, so ist L_2 nicht semi-entscheidbar.

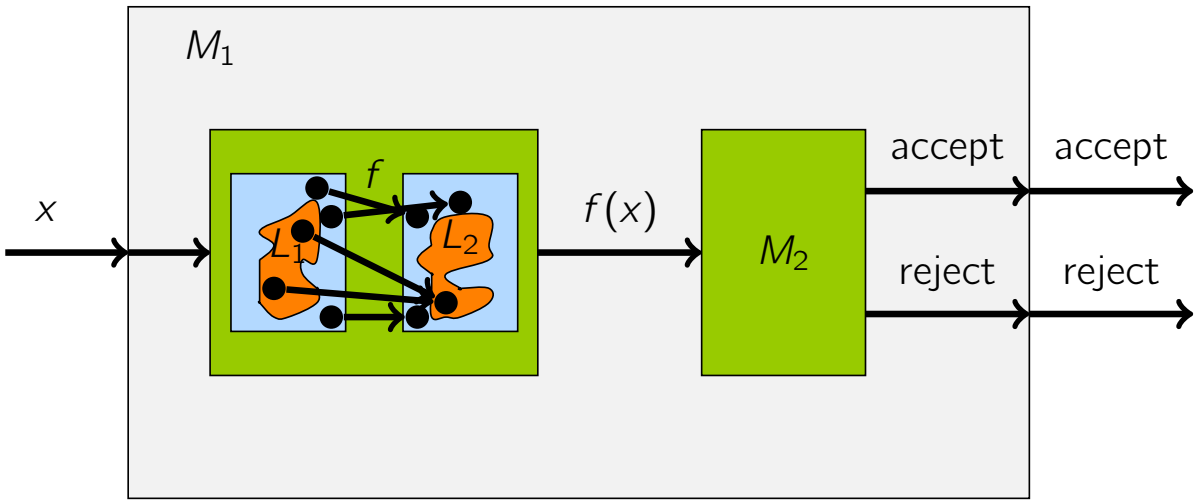
Wdh.: Reduktionen



Wdh.: Reduktionen



Wdh.: Reduktionen



Anwendung

Vorbeobachtung: H_ϵ ist unentscheidbar, aber semi-entscheidbar. Folglich ist $\overline{H}_\epsilon$ nicht semi-entscheidbar.

Anwendung

Vorbeobachtung: H_ϵ ist unentscheidbar, aber semi-entscheidbar. Folglich ist $\overline{H}_\epsilon$ nicht semi-entscheidbar.

Wir zeigen nun

Behauptung A

$$\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$$

Behauptung B

$$\overline{H}_\epsilon \leq H_{\text{all}}$$

Anwendung

Vorbeobachtung: H_ϵ ist unentscheidbar, aber semi-entscheidbar. Folglich ist $\overline{H}_\epsilon$ nicht semi-entscheidbar.

Wir zeigen nun

Behauptung A

$$\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$$

Behauptung B

$$\overline{H}_\epsilon \leq H_{\text{all}}$$

Aus diesen Reduktionen folgt:

Satz

Weder $\overline{H}_{all}$ noch H_{all} sind semi-entscheidbar.

Beweis von Behauptung A: $\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$

Zur Durchführung der Reduktion gehen wir in zwei Schritten vor:

Beweis von Behauptung A: $\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$

Zur Durchführung der Reduktion gehen wir in zwei Schritten vor:

1) Wir beschreiben eine berechenbare Funktion f, die Ja-Instanzen von $\overline{H}_\epsilon$ auf Ja-Instanzen von $\overline{H}_{\text{all}}$ und Nein-Instanzen von $\overline{H}_\epsilon$ auf Nein-Instanzen von $\overline{H}_{\text{all}}$ abbildet.

Beweis von Behauptung A: $\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$

Zur Durchführung der Reduktion gehen wir in zwei Schritten vor:

1) Wir beschreiben eine berechenbare Funktion f, die Ja-Instanzen von $\overline{H}_\epsilon$ auf Ja-Instanzen von $\overline{H}_{\text{all}}$ und Nein-Instanzen von $\overline{H}_\epsilon$ auf Nein-Instanzen von $\overline{H}_{\text{all}}$ abbildet.

2) Für die Korrektheit zeigen wir:
   a) $w \notin \overline{H}_\epsilon \;\Rightarrow\; f(w) \notin \overline{H}_{\text{all}}$
   b) $w \in \overline{H}_\epsilon \;\Rightarrow\; f(w) \in \overline{H}_{\text{all}}$

Beweis von Behauptung A: $\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$

Beschreibung der Funktion f:

Sei w die Eingabe für $\overline{H}_\epsilon$.

- Wenn w keine gültige Gödelnummer ist, so sei $f(w) = w$.

Beweis von Behauptung A: $\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$

Beschreibung der Funktion f:

Sei w die Eingabe für $\overline{H}_\epsilon$.

- Wenn w keine gültige Gödelnummer ist, so sei $f(w) = w$.
- Falls $w = \langle M \rangle$ für eine TM M, so sei $f(w)$ die Gödelnummer einer TM M^*_ϵ mit der folgenden Eigenschaft:

Beweis von Behauptung A: $\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$

Beschreibung der Funktion f:

Sei w die Eingabe für $\overline{H}_\epsilon$.

- Wenn w keine gültige Gödelnummer ist, so sei $f(w) = w$.
- Falls $w = \langle M \rangle$ für eine TM M, so sei $f(w)$ die Gödelnummer einer TM M^*_ϵ mit der folgenden Eigenschaft:

M^*_ϵ ignoriert die Eingabe und simuliert M mit der Eingabe ϵ.

Beweis von Behauptung A: $\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$

Beschreibung der Funktion f:

Sei w die Eingabe für $\overline{H}_\epsilon$.

- Wenn w keine gültige Gödelnummer ist, so sei $f(w) = w$.
- Falls $w = \langle M \rangle$ für eine TM M, so sei $f(w)$ die Gödelnummer einer TM M^*_ϵ mit der folgenden Eigenschaft:

M^*_ϵ ignoriert die Eingabe und simuliert M mit der Eingabe ϵ.

Die Funktion f ist berechenbar. (Warum?)

Beweis von Behauptung A: $\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$ — Korrektheit

Korrektheit:

Falls w keine Gödelnummer ist, so ist die Korrektheit klar, denn in diesem Fall gilt $w \in \overline{H}_\epsilon$ und $f(w) \in \overline{H}_{\text{all}}$.

Sei nun $w = \langle M \rangle$ für eine TM M und sei $f(w) = \langle M^*_\epsilon \rangle$.

Beweis von Behauptung A: $\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$ — Korrektheit

Korrektheit:

Falls w keine Gödelnummer ist, so ist die Korrektheit klar, denn in diesem Fall gilt $w \in \overline{H}_\epsilon$ und $f(w) \in \overline{H}_{\text{all}}$.

Sei nun $w = \langle M \rangle$ für eine TM M und sei $f(w) = \langle M^*_\epsilon \rangle$.

Es gilt

$$w \notin \overline{H}_\epsilon \quad \Rightarrow \quad w \in H_\epsilon$$

Beweis von Behauptung A: $\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$ — Korrektheit

Korrektheit:

Falls w keine Gödelnummer ist, so ist die Korrektheit klar, denn in diesem Fall gilt $w \in \overline{H}_\epsilon$ und $f(w) \in \overline{H}_{\text{all}}$.

Sei nun $w = \langle M \rangle$ für eine TM M und sei $f(w) = \langle M^*_\epsilon \rangle$.

Es gilt

$$\begin{aligned} w \notin \overline{H}_\epsilon &\Rightarrow w \in H_\epsilon \\ &\Rightarrow M \text{ hält auf der Eingabe } \epsilon. \end{aligned}$$

Beweis von Behauptung A: $\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$ — Korrektheit

Korrektheit:

Falls w keine Gödelnummer ist, so ist die Korrektheit klar, denn in diesem Fall gilt $w \in \overline{H}_\epsilon$ und $f(w) \in \overline{H}_{\text{all}}$.

Sei nun $w = \langle M \rangle$ für eine TM M und sei $f(w) = \langle M^*_\epsilon \rangle$.

Es gilt

$$\begin{aligned} w \notin \overline{H}_\epsilon &\Rightarrow w \in H_\epsilon \\ &\Rightarrow M \text{ hält auf der Eingabe } \epsilon. \\ &\Rightarrow M^*_\epsilon \text{ hält auf jeder Eingabe.} \end{aligned}$$

Beweis von Behauptung A: $\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$ — Korrektheit

Korrektheit:

Falls w keine Gödelnummer ist, so ist die Korrektheit klar, denn in diesem Fall gilt $w \in \overline{H}_\epsilon$ und $f(w) \in \overline{H}_{\text{all}}$.

Sei nun $w = \langle M \rangle$ für eine TM M und sei $f(w) = \langle M^*_\epsilon \rangle$.

Es gilt

$$\begin{aligned} w \notin \overline{H}_\epsilon &\Rightarrow w \in H_\epsilon \\ &\Rightarrow M \text{ hält auf der Eingabe } \epsilon. \\ &\Rightarrow M^*_\epsilon \text{ hält auf jeder Eingabe.} \\ &\Rightarrow \langle M^*_\epsilon \rangle \in H_{\text{all}} \end{aligned}$$

Beweis von Behauptung A: $\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$ — Korrektheit

Korrektheit:

Falls w keine Gödelnummer ist, so ist die Korrektheit klar, denn in diesem Fall gilt $w \in \overline{H}_\epsilon$ und $f(w) \in \overline{H}_{\text{all}}$.

Sei nun $w = \langle M \rangle$ für eine TM M und sei $f(w) = \langle M^*_\epsilon \rangle$.

Es gilt

$$\begin{aligned} w \notin \overline{H}_\epsilon &\Rightarrow w \in H_\epsilon \\ &\Rightarrow M \text{ hält auf der Eingabe } \epsilon. \\ &\Rightarrow M^*_\epsilon \text{ hält auf jeder Eingabe.} \\ &\Rightarrow \langle M^*_\epsilon \rangle \in H_{\text{all}} \\ &\Rightarrow f(w) \notin \overline{H}_{\text{all}}. \end{aligned}$$

Beweis von Behauptung A: $\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$ — Korrektheit

Es sei weiter $w = \langle M \rangle$ für eine TM M und sei $f(w) = \langle M^*_\epsilon \rangle$.

Beweis von Behauptung A: $\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$ — Korrektheit

Es sei weiter $w = \langle M \rangle$ für eine TM M und sei $f(w) = \langle M^*_\epsilon \rangle$.

$$w \in \overline{H}_\epsilon \quad \Rightarrow \quad M \text{ hält nicht auf der Eingabe } \epsilon.$$

Beweis von Behauptung A: $\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$ — Korrektheit

Es sei weiter $w = \langle M \rangle$ für eine TM M und sei $f(w) = \langle M^*_\epsilon \rangle$.

$$\begin{aligned} w \in \overline{H}_\epsilon &\Rightarrow M \text{ hält nicht auf der Eingabe } \epsilon. \\ &\Rightarrow M^*_\epsilon \text{ hält auf keiner Eingabe.} \end{aligned}$$

Beweis von Behauptung A: $\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$ — Korrektheit

Es sei weiter $w = \langle M \rangle$ für eine TM M und sei $f(w) = \langle M^*_\epsilon \rangle$.

$$\begin{aligned} w \in \overline{H}_\epsilon &\Rightarrow M \text{ hält nicht auf der Eingabe } \epsilon. \\ &\Rightarrow M^*_\epsilon \text{ hält auf keiner Eingabe.} \\ &\Rightarrow \langle M^*_\epsilon \rangle \notin H_{\text{all}} \end{aligned}$$

Beweis von Behauptung A: $\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$ — Korrektheit

Es sei weiter $w = \langle M \rangle$ für eine TM M und sei $f(w) = \langle M^*_\epsilon \rangle$.

$$\begin{aligned} w \in \overline{H}_\epsilon &\Rightarrow M \text{ hält nicht auf der Eingabe } \epsilon. \\ &\Rightarrow M^*_\epsilon \text{ hält auf keiner Eingabe.} \\ &\Rightarrow \langle M^*_\epsilon \rangle \notin H_{\text{all}} \\ &\Rightarrow f(w) \in \overline{H}_{\text{all}} \end{aligned}$$

Beweis von Behauptung A: $\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$ — Korrektheit

Es sei weiter $w = \langle M \rangle$ für eine TM M und sei $f(w) = \langle M^*_\epsilon \rangle$.

$$\begin{aligned} w \in \overline{H}_\epsilon &\Rightarrow M \text{ hält nicht auf der Eingabe } \epsilon. \\ &\Rightarrow M^*_\epsilon \text{ hält auf keiner Eingabe.} \\ &\Rightarrow \langle M^*_\epsilon \rangle \notin H_{\text{all}} \\ &\Rightarrow f(w) \in \overline{H}_{\text{all}} \end{aligned}$$

Also gilt $w \in \overline{H}_\epsilon \Leftrightarrow f(w) \in \overline{H}_{\text{all}}$ und somit ist die Funktion f korrekt konstruiert.

□

Daher ist $\overline{H}_{\text{all}}$ nicht semi-entscheidbar.

Beweis von Behauptung B: $\overline{H}_\epsilon \leq H_{\text{all}}$

Beweis von Behauptung B: $\overline{H}_\epsilon \leq H_{\text{all}}$

Wir gehen wiederum in zwei Schritten vor:

Beweis von Behauptung B: $\overline{H}_\epsilon \leq H_{\text{all}}$

Wir gehen wiederum in zwei Schritten vor:

1) Wir beschreiben eine berechenbare Funktion f, die Ja-Instanzen von $\overline{H}_\epsilon$ auf Ja-Instanzen von H_{all} und Nein-Instanzen von $\overline{H}_\epsilon$ auf Nein-Instanzen von H_{all} abbildet.

Beweis von Behauptung B: $\overline{H}_\epsilon \leq H_{\text{all}}$

Wir gehen wiederum in zwei Schritten vor:

1) Wir beschreiben eine berechenbare Funktion f, die Ja-Instanzen von $\overline{H}_\epsilon$ auf Ja-Instanzen von H_{all} und Nein-Instanzen von $\overline{H}_\epsilon$ auf Nein-Instanzen von H_{all} abbildet.

2) Für die Korrektheit zeigen wir:
   a) $w \notin \overline{H}_\epsilon \;\Rightarrow\; f(w) \notin H_{\text{all}}$
   b) $w \in \overline{H}_\epsilon \;\Rightarrow\; f(w) \in H_{\text{all}}$

Beweis von Behauptung B: $\overline{H}_\epsilon \leq H_{\text{all}}$

Beschreibung der Funktion f:

Sei w die Eingabe für $\overline{H}_\epsilon$. Sei w' irgendein Wort aus H_{all}.

- Wenn w keine gültige Gödelnummer ist, so sei $f(w) = w'$.

Beweis von Behauptung B: $\overline{H}_\epsilon \leq H_{\text{all}}$

Beschreibung der Funktion f:

Sei w die Eingabe für $\overline{H}_\epsilon$. Sei w' irgendein Wort aus H_{all}.

- Wenn w keine gültige Gödelnummer ist, so sei $f(w) = w'$.
- Falls $w = \langle M \rangle$ für eine TM M, so sei $f(w)$ die Gödelnummer einer TM M', die sich auf Eingaben der Länge i wie folgt verhält:

Beweis von Behauptung B: $\overline{H}_\epsilon \leq H_{\text{all}}$

Beschreibung der Funktion f:

Sei w die Eingabe für $\overline{H}_\epsilon$. Sei w' irgendein Wort aus H_{all}.

- Wenn w keine gültige Gödelnummer ist, so sei $f(w) = w'$.
- Falls $w = \langle M \rangle$ für eine TM M, so sei $f(w)$ die Gödelnummer einer TM M', die sich auf Eingaben der Länge i wie folgt verhält:

M' simuliert die ersten i Schritte von M auf der Eingabe ϵ. Wenn M innerhalb dieser i Schritte hält, dann geht M' in eine Endlosschleife, ansonsten hält M'.

Beweis von Behauptung B: $\overline{H}_\epsilon \leq H_{\text{all}}$

Beschreibung der Funktion f:

Sei w die Eingabe für $\overline{H}_\epsilon$. Sei w' irgendein Wort aus H_{all}.

- Wenn w keine gültige Gödelnummer ist, so sei $f(w) = w'$.
- Falls $w = \langle M \rangle$ für eine TM M, so sei $f(w)$ die Gödelnummer einer TM M', die sich auf Eingaben der Länge i wie folgt verhält:

M' simuliert die ersten i Schritte von M auf der Eingabe ϵ. Wenn M innerhalb dieser i Schritte hält, dann geht M' in eine Endlosschleife, ansonsten hält M'.

Die Funktion f ist berechenbar. (Warum?)

Beweis von Behauptung B: $\overline{H}_\epsilon \leq H_{\text{all}}$

Korrektheit

Falls w keine Gödelnummer ist, so ist die Korrektheit klar, denn in diesem Fall gilt $w \in \overline{H}_\epsilon$ und $f(w) = w' \in H_{\text{all}}$.

Sei nun $w = \langle M \rangle$ für eine TM M und sei $f(w) = \langle M' \rangle$.

Beweis von Behauptung B: $\overline{H}_\epsilon \leq H_{\text{all}}$

Korrektheit

Falls w keine Gödelnummer ist, so ist die Korrektheit klar, denn in diesem Fall gilt $w \in \overline{H}_\epsilon$ und $f(w) = w' \in H_{\text{all}}$.

Sei nun $w = \langle M \rangle$ für eine TM M und sei $f(w) = \langle M' \rangle$.

Es gilt

$$w \notin \overline{H}_\epsilon \quad \Rightarrow \quad M \text{ hält auf der Eingabe } \epsilon.$$

Beweis von Behauptung B: $\overline{H}_\epsilon \leq H_{\text{all}}$

Korrektheit

Falls w keine Gödelnummer ist, so ist die Korrektheit klar, denn in diesem Fall gilt $w \in \overline{H}_\epsilon$ und $f(w) = w' \in H_{\text{all}}$.

Sei nun $w = \langle M \rangle$ für eine TM M und sei $f(w) = \langle M' \rangle$.

Es gilt

$$\begin{aligned} w \notin \overline{H}_\epsilon &\Rightarrow M \text{ hält auf der Eingabe } \epsilon. \\ &\Rightarrow \exists i\colon M \text{ hält innerhalb von } i \text{ Schritten auf } \epsilon. \end{aligned}$$

Beweis von Behauptung B: $\overline{H}_\epsilon \leq H_{\text{all}}$

Korrektheit

Falls w keine Gödelnummer ist, so ist die Korrektheit klar, denn in diesem Fall gilt $w \in \overline{H}_\epsilon$ und $f(w) = w' \in H_{\text{all}}$.

Sei nun $w = \langle M \rangle$ für eine TM M und sei $f(w) = \langle M' \rangle$.

Es gilt

$$\begin{aligned} w \notin \overline{H}_\epsilon &\Rightarrow M \text{ hält auf der Eingabe } \epsilon. \\ &\Rightarrow \exists i\colon M \text{ hält innerhalb von } i \text{ Schritten auf } \epsilon. \\ &\Rightarrow \exists i\colon M' \text{ hält auf keiner Eingabe der Länge mindestens } i. \end{aligned}$$

Beweis von Behauptung B: $\overline{H}_\epsilon \leq H_{\text{all}}$

Korrektheit

Falls w keine Gödelnummer ist, so ist die Korrektheit klar, denn in diesem Fall gilt $w \in \overline{H}_\epsilon$ und $f(w) = w' \in H_{\text{all}}$.

Sei nun $w = \langle M \rangle$ für eine TM M und sei $f(w) = \langle M' \rangle$.

Es gilt

$$\begin{aligned} w \notin \overline{H}_\epsilon &\Rightarrow M \text{ hält auf der Eingabe } \epsilon. \\ &\Rightarrow \exists i\colon M \text{ hält innerhalb von } i \text{ Schritten auf } \epsilon. \\ &\Rightarrow \exists i\colon M' \text{ hält auf keiner Eingabe der Länge mindestens } i. \\ &\Rightarrow M' \text{ hält nicht auf jeder Eingabe.} \end{aligned}$$

Beweis von Behauptung B: $\overline{H}_\epsilon \leq H_{\text{all}}$

Korrektheit

Falls w keine Gödelnummer ist, so ist die Korrektheit klar, denn in diesem Fall gilt $w \in \overline{H}_\epsilon$ und $f(w) = w' \in H_{\text{all}}$.

Sei nun $w = \langle M \rangle$ für eine TM M und sei $f(w) = \langle M' \rangle$.

Es gilt

$$\begin{aligned} w \notin \overline{H}_\epsilon &\Rightarrow M \text{ hält auf der Eingabe } \epsilon. \\ &\Rightarrow \exists i\colon M \text{ hält innerhalb von } i \text{ Schritten auf } \epsilon. \\ &\Rightarrow \exists i\colon M' \text{ hält auf keiner Eingabe der Länge mindestens } i. \\ &\Rightarrow M' \text{ hält nicht auf jeder Eingabe.} \\ &\Rightarrow f(w) = \langle M' \rangle \notin H_{\text{all}} \end{aligned}$$

Beweis von Behauptung B: $\overline{H}_\epsilon \leq H_{\text{all}}$

$$w \in \overline{H}_\epsilon \quad \Rightarrow \quad M \text{ hält nicht auf der Eingabe } \epsilon$$

Beweis von Behauptung B: $\overline{H}_\epsilon \leq H_{\text{all}}$

$$\begin{aligned} w \in \overline{H}_\epsilon \quad &\Rightarrow \quad M \text{ hält nicht auf der Eingabe } \epsilon \\ &\Rightarrow \quad \neg\exists i\text{: } M \text{ hält innerhalb von } i \text{ Schritten auf } \epsilon \\ &\Rightarrow \quad \forall i\text{: } M \text{ hält nicht innerhalb von } i \text{ Schritten auf } \epsilon \end{aligned}$$

Beweis von Behauptung B: $\overline{H}_\epsilon \leq H_{\text{all}}$

$$
\begin{aligned}
w \in \overline{H}_\epsilon &\Rightarrow M \text{ hält nicht auf der Eingabe } \epsilon \\
&\Rightarrow \neg\exists i\text{: } M \text{ hält innerhalb von } i \text{ Schritten auf } \epsilon \\
&\Rightarrow \forall i\text{: } M \text{ hält nicht innerhalb von } i \text{ Schritten auf } \epsilon \\
&\Rightarrow \forall i\text{: } M' \text{ hält auf allen Eingaben der Länge } i
\end{aligned}
$$

Beweis von Behauptung B: $\overline{H}_\epsilon \leq H_{\text{all}}$

$$
\begin{aligned}
w \in \overline{H}_\epsilon &\Rightarrow M \text{ hält nicht auf der Eingabe } \epsilon \\
&\Rightarrow \neg\exists i\text{: } M \text{ hält innerhalb von } i \text{ Schritten auf } \epsilon \\
&\Rightarrow \forall i\text{: } M \text{ hält nicht innerhalb von } i \text{ Schritten auf } \epsilon \\
&\Rightarrow \forall i\text{: } M' \text{ hält auf allen Eingaben der Länge } i \\
&\Rightarrow M' \text{ hält auf jeder Eingabe}
\end{aligned}
$$

Beweis von Behauptung B: $\overline{H}_\epsilon \leq H_{\text{all}}$

$$
\begin{aligned}
w \in \overline{H}_\epsilon \quad &\Rightarrow \quad M \text{ hält nicht auf der Eingabe } \epsilon \\
&\Rightarrow \quad \neg\exists i\text{: } M \text{ hält innerhalb von } i \text{ Schritten auf } \epsilon \\
&\Rightarrow \quad \forall i\text{: } M \text{ hält nicht innerhalb von } i \text{ Schritten auf } \epsilon \\
&\Rightarrow \quad \forall i\text{: } M' \text{ hält auf allen Eingaben der Länge } i \\
&\Rightarrow \quad M' \text{ hält auf jeder Eingabe} \\
&\Rightarrow \quad f(w) = \langle M' \rangle \in H_{\text{all}} \quad .
\end{aligned}
$$

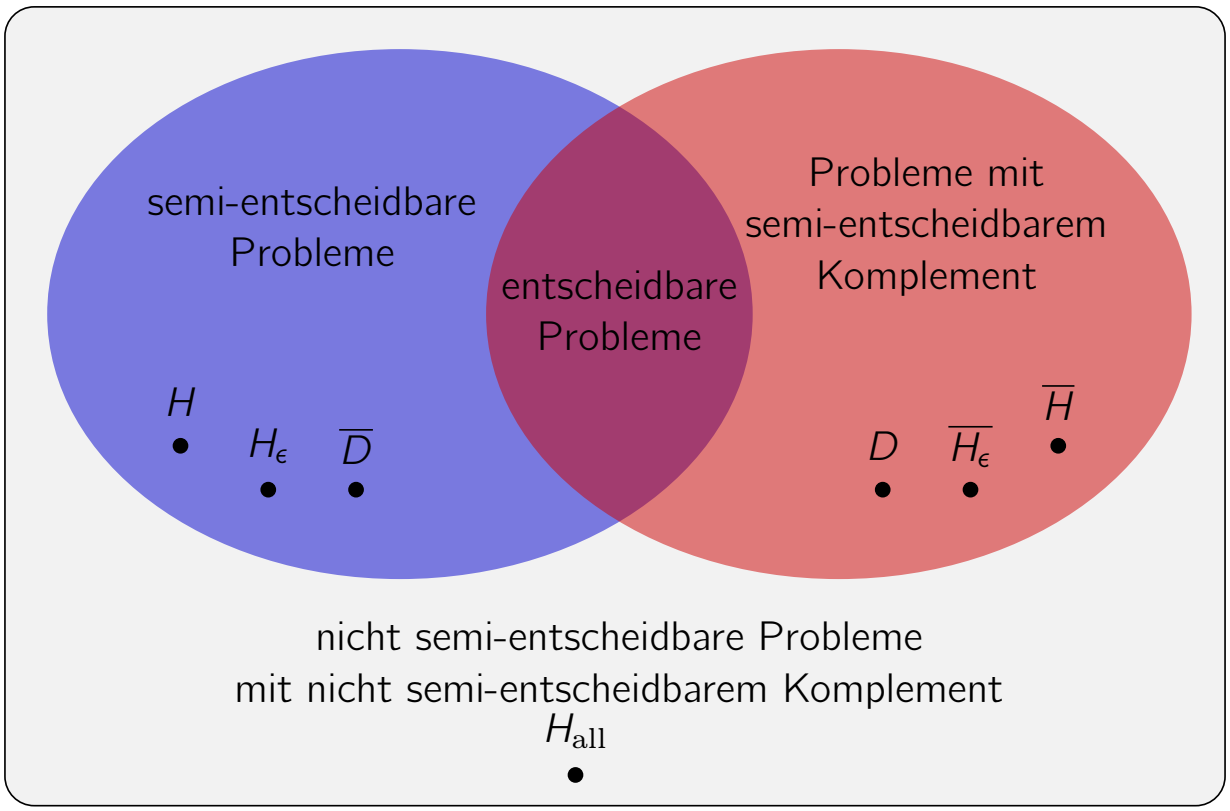
Beweis von Behauptung B: $\overline{H}_\epsilon \leq H_{\text{all}}$

$$
\begin{aligned}
w \in \overline{H}_\epsilon \quad &\Rightarrow \quad M \text{ hält nicht auf der Eingabe } \epsilon \\
&\Rightarrow \quad \neg\exists i\text{: } M \text{ hält innerhalb von } i \text{ Schritten auf } \epsilon \\
&\Rightarrow \quad \forall i\text{: } M \text{ hält nicht innerhalb von } i \text{ Schritten auf } \epsilon \\
&\Rightarrow \quad \forall i\text{: } M' \text{ hält auf allen Eingaben der Länge } i \\
&\Rightarrow \quad M' \text{ hält auf jeder Eingabe} \\
&\Rightarrow \quad f(w) = \langle M' \rangle \in H_{\text{all}} \quad .
\end{aligned}
$$

Also gilt $w \in \overline{H}_\epsilon \;\Leftrightarrow\; f(w) \in H_{\text{all}}$ und somit ist die Funktion f korrekt konstruiert. □

Es folgt, dass H_{all} nicht semi-entscheidbar ist.

Berechenbarkeitslandschaft



Anmerkung zu Reduktionen

Es gilt übrigens auch

Lemma

Falls $L_1 \leq L_2$ und L_2 entscheidbar ist, so ist L_1 entscheidbar.

Oder umgekehrt:

Lemma

Falls $L_1 \leq L_2$ und L_1 unentscheidbar ist, so ist L_2 unentscheidbar.

Beweis: analog zur Semientscheidbarkeit. □

Hilberts zehntes Problem

Im Jahr 1900 präsentierte der Mathematiker David Hilbert 23 mathematische Probleme auf einem Kongress in Paris.

Hilberts zehntes Problem

Im Jahr 1900 präsentierte der Mathematiker David Hilbert 23 mathematische Probleme auf einem Kongress in Paris.

Hilberts zehntes Problem (im Originalwortlaut)

Eine diophantische Gleichung mit irgendwelchen Unbekannten und mit ganzen rationalen Zahlenkoeffizienten sei vorgelegt: *Man soll ein Verfahren angeben, nach welchem sich mittels einer endlichen Anzahl von Operationen entscheiden läßt, ob die Gleichung in den ganzen rationalen Zahlen lösbar ist.*

Hilberts zehntes Problem

Im Jahr 1900 präsentierte der Mathematiker David Hilbert 23 mathematische Probleme auf einem Kongress in Paris.

Hilberts zehntes Problem (im Originalwortlaut)

Eine diophantische Gleichung mit irgendwelchen Unbekannten und mit ganzen rationalen Zahlenkoeffizienten sei vorgelegt: *Man soll ein Verfahren angeben, nach welchem sich mittels einer endlichen Anzahl von Operationen entscheiden läßt, ob die Gleichung in den ganzen rationalen Zahlen lösbar ist.*

Die „ganzen rationalen Zahlen“, von denen in diesem Problem die Rede ist, sind die Zahlen aus $\mathbb{Z}$, wie wir sie kennen.

„Diophantische Gleichungen“ bezeichnen Gleichungen über Polynomen in mehreren Variablen.

Diophantische Gleichungen

- Ein Term ist ein Produkt aus Variablen mit einem konstanten Koeffizienten, z.B. ist

$$6 \cdot x \cdot x \cdot x \cdot y \cdot z \cdot z \text{ bzw. } 6x^3yz^2$$

ein Term über den Variablen x, y, z mit dem Koeffizienten 6.

Diophantische Gleichungen

- Ein Term ist ein Produkt aus Variablen mit einem konstanten Koeffizienten, z.B. ist

$$6 \cdot x \cdot x \cdot x \cdot y \cdot z \cdot z \text{ bzw. } 6x^3yz^2$$

  ein Term über den Variablen x, y, z mit dem Koeffizienten 6.
- Ein Polynom ist eine Summe von Termen, z.B.

$$6x^3yz^2 + 3xy^2 - x^3 - 10.$$

Diophantische Gleichungen

- Ein Term ist ein Produkt aus Variablen mit einem konstanten Koeffizienten, z.B. ist

$$6 \cdot x \cdot x \cdot x \cdot y \cdot z \cdot z \text{ bzw. } 6x^3yz^2$$

  ein Term über den Variablen x, y, z mit dem Koeffizienten 6.
- Ein Polynom ist eine Summe von Termen, z.B.

$$6x^3yz^2 + 3xy^2 - x^3 - 10.$$

- Eine diophantische Gleichung setzt ein Polynom gleich Null. Die Lösungen der Gleichung entsprechen also den Nullstellen des Polynoms. Obiges Polynom hat beispielsweise die Nullstelle

$$(x, y, z) = (5, 3, 0) \ .$$

Diophantische Gleichungen – Beispiele

Beispiele

- Quadratische Gleichung $5x^2 - 3x + 6 = 0$. Lösbar mit diversen Methoden. (Quadratische Ergänzung, Lösungsformel, ...) Also kann man auch feststellen, ob die Gleichung ganzzahlige Lösungen hat.
- Fermatsche Gleichung $x^n + y^n - z^n = 0$ für $n \in \mathbb{N}$. Hat eine ganzzahlige positive Lösung nur, wenn $n \leq 2$. (Satz von Wiles, Fermats letzter Satz.)
- Hat die Gleichung $x^4 + y^6x^2 - z^7 + xy - x = 0$ eine ganzzahlige Lösung?

Formulierung als Entscheidungsproblem

Hilberts zehntes Problem (in unseren Worten)

Beschreibe einen Algorithmus, der entscheidet, ob ein gegebenes Polynom mit ganzzahligen Koeffizienten eine ganzzahlige Nullstelle hat.

Formulierung als Entscheidungsproblem

Hilberts zehntes Problem (in unseren Worten)

Beschreibe einen Algorithmus, der entscheidet, ob ein gegebenes Polynom mit ganzzahligen Koeffizienten eine ganzzahlige Nullstelle hat.

Die diesem Entscheidungsproblem zugrundeliegende Sprache ist

$$N = \{ p \mid p \text{ ist ein Polynom mit einer ganzzahligen Nullstelle} \} .$$

Semientscheidbarkeit von N

Gegeben sei ein Polynom p mit ℓ Variablen.

Der Wertebereich von p entspricht der abzählbar unendlichen Menge $\mathbb{Z}^\ell$.

Semientscheidbarkeit von N

Gegeben sei ein Polynom p mit ℓ Variablen.

Der Wertebereich von p entspricht der abzählbar unendlichen Menge $\mathbb{Z}^\ell$.

Der folgende Algorithmus erkennt N:

- Zähle die ℓ-Tupel aus $\mathbb{Z}^\ell$ in kanonischer Reihenfolge auf und werte p für jedes dieser Tupel aus.
- Akzeptiere, sobald eine der Auswertungen den Wert 0 ergibt.

Semientscheidbarkeit von N

Gegeben sei ein Polynom p mit ℓ Variablen.

Der Wertebereich von p entspricht der abzählbar unendlichen Menge $\mathbb{Z}^\ell$.

Der folgende Algorithmus erkennt N:

- Zähle die ℓ-Tupel aus $\mathbb{Z}^\ell$ in kanonischer Reihenfolge auf und werte p für jedes dieser Tupel aus.
- Akzeptiere, sobald eine der Auswertungen den Wert 0 ergibt.

Fazit: N ist semi-entscheidbar.

Ist N entscheidbar? – Diskussion

- Falls wir eine obere Schranke für die Absolutwerte der Nullstellen hätten, so bräuchten wir nur eine endliche Menge von ℓ-Tupeln aufzählen und N wäre somit entscheidbar.

Ist N entscheidbar? – Diskussion

- Falls wir eine obere Schranke für die Absolutwerte der Nullstellen hätten, so bräuchten wir nur eine endliche Menge von ℓ-Tupeln aufzählen und N wäre somit entscheidbar.
- Für Polynome über nur einer Variablen gibt es tatsächlich eine obere Schranke: Für ein Polynom der Form

$$p(x) = a_k x^k + a_{k-1} x^{k-1} + \cdots + a_1 x + a_0$$

mit ganzzahligen Koeffizienten gilt

$$p(x) = 0, x \in \mathbb{Z} \quad \Rightarrow \quad x \text{ teilt } a_0. \text{ (Warum?)}$$

Also gibt es keine Nullstelle mit Absolutwert größer als $|a_0|$.

Ist N entscheidbar? – Diskussion

- Falls wir eine obere Schranke für die Absolutwerte der Nullstellen hätten, so bräuchten wir nur eine endliche Menge von ℓ-Tupeln aufzählen und N wäre somit entscheidbar.
- Für Polynome über nur einer Variablen gibt es tatsächlich eine obere Schranke: Für ein Polynom der Form

$$p(x) = a_k x^k + a_{k-1} x^{k-1} + \cdots + a_1 x + a_0$$

mit ganzzahligen Koeffizienten gilt

$$p(x) = 0, x \in \mathbb{Z} \quad \Rightarrow \quad x \text{ teilt } a_0. \text{ (Warum?)}$$

Also gibt es keine Nullstelle mit Absolutwert größer als $|a_0|$.

- Eingeschränkt auf Polynome mit nur einer Variablen ist das Nullstellenproblem damit entscheidbar.

Ist N entscheidbar? – Diskussion

- Für Polynome mit mehreren Variablen gibt es leider keine obere Schranke für die Absolutwerte der Nullstellen. Um das einzusehen, betrachte beispielsweise das Polynom $x + y$.

Ist N entscheidbar? – Diskussion

- Für Polynome mit mehreren Variablen gibt es leider keine obere Schranke für die Absolutwerte der Nullstellen. Um das einzusehen, betrachte beispielsweise das Polynom $x + y$.
- Aber vielleicht gibt es ja immer eine Nullstelle mit kleinen Absolutwerten und somit eine obere Schranke für die Nullstelle mit den kleinsten Absolutwerten?

Ist N entscheidbar? – Diskussion

- Für Polynome mit mehreren Variablen gibt es leider keine obere Schranke für die Absolutwerte der Nullstellen. Um das einzusehen, betrachte beispielsweise das Polynom $x + y$.
- Aber vielleicht gibt es ja immer eine Nullstelle mit kleinen Absolutwerten und somit eine obere Schranke für die Nullstelle mit den kleinsten Absolutwerten?
- Oder vielleicht gibt es ganz andere Möglichkeiten einem Polynom anzusehen, ob es eine ganzzahlige Nullstelle hat?

Ist N entscheidbar? – Diskussion

- Für Polynome mit mehreren Variablen gibt es leider keine obere Schranke für die Absolutwerte der Nullstellen. Um das einzusehen, betrachte beispielsweise das Polynom $x + y$.
- Aber vielleicht gibt es ja immer eine Nullstelle mit kleinen Absolutwerten und somit eine obere Schranke für die Nullstelle mit den kleinsten Absolutwerten?
- Oder vielleicht gibt es ganz andere Möglichkeiten einem Polynom anzusehen, ob es eine ganzzahlige Nullstelle hat?
- Erst knapp siebzig Jahre, nachdem Hilbert sein Problem präsentiert hatte, konnte Yuri Matijasevič alle diese Fragen negativ beantworten.

Unentscheidbarkeit des Nullstellenproblems

Hilbert hat die folgende Antwort nicht erwartet.

Unentscheidbarkeit des Nullstellenproblems

Hilbert hat die folgende Antwort nicht erwartet.

Satz von Matijasevič (1970)

Das Problem, ob ein ganzzahliges Polynom eine ganzzahlige Nullstelle hat, ist unentscheidbar.

Unentscheidbarkeit des Nullstellenproblems

Hilbert hat die folgende Antwort nicht erwartet.

Satz von Matijasevič (1970)

Das Problem, ob ein ganzzahliges Polynom eine ganzzahlige Nullstelle hat, ist unentscheidbar.

Damit ist Hilberts Aufgabenstellung unlösbar.

Unentscheidbarkeit des Nullstellenproblems

- Der Beweis des Satzes von Matijasevič beruht auf einer Kette von Reduktionen, durch die letztendlich das Halteproblem H auf das Nullstellenproblem N reduziert wird.
- Yuri Matijasevič hat das letzte Glied dieser Kette geschlossen. Andere wichtige Beiträge zu diesem Ergebnis wurden zuvor von Martin Davis, Julia Robinson und Hilary Putnam erbracht.

Unentscheidbarkeit des Nullstellenproblems

- Der Beweis des Satzes von Matijasevič beruht auf einer Kette von Reduktionen, durch die letztendlich das Halteproblem H auf das Nullstellenproblem N reduziert wird.
- Yuri Matijasevič hat das letzte Glied dieser Kette geschlossen. Andere wichtige Beiträge zu diesem Ergebnis wurden zuvor von Martin Davis, Julia Robinson und Hilary Putnam erbracht.
- Leider ist der Beweis zu lang, um ihn im Rahmen dieser Vorlesung vollständig präsentieren zu können.

Unentscheidbarkeit des Nullstellenproblems

- Der Beweis des Satzes von Matijasevič beruht auf einer Kette von Reduktionen, durch die letztendlich das Halteproblem H auf das Nullstellenproblem N reduziert wird.
- Yuri Matijasevič hat das letzte Glied dieser Kette geschlossen. Andere wichtige Beiträge zu diesem Ergebnis wurden zuvor von Martin Davis, Julia Robinson und Hilary Putnam erbracht.
- Leider ist der Beweis zu lang, um ihn im Rahmen dieser Vorlesung vollständig präsentieren zu können.
- Wir werden nur einen Überblick sehen und alternativ das Postsche Korrespondenzproblem betrachten.

Unentscheidbarkeit des Nullstellenproblems – Zutaten

Unentscheidbarkeit des Nullstellenproblems – Zutaten

Sei $\mathrm{Dioph}(M)$ das Problem zu entscheiden, ob eine gegebene diophantische Gleichung über M lösbar ist.

Sei $\mathrm{ExpDioph}(M)$ das Problem zu entscheiden, ob eine gegebene Gleichung, die zusätzlich noch Terme der Form x^y haben darf, über M lösbar ist.

Unentscheidbarkeit des Nullstellenproblems – Zutaten

Sei $\mathrm{Dioph}(M)$ das Problem zu entscheiden, ob eine gegebene diophantische Gleichung über M lösbar ist.

Sei $\mathrm{ExpDioph}(M)$ das Problem zu entscheiden, ob eine gegebene Gleichung, die zusätzlich noch Terme der Form x^y haben darf, über M lösbar ist.

Die Reduktionskette im Beweis ist:

$$H \leq \mathrm{ExpDioph}(\mathbb{N}) \leq \mathrm{Dioph}(\mathbb{N}) \leq \mathrm{Dioph}(\mathbb{Z}) = N$$

$\mathrm{Dioph}(\mathbb{N}) \leq \mathrm{Dioph}(\mathbb{Z})$

Wir benötigen eine Reduktion, die zu einem Polynom p ein neues Polynom $f(p)$ berechnet, so dass $p = 0$ genau dann eine Lösung über $\mathbb{N}$ hat, wenn $f(p) = 0$ eine Lösung über $\mathbb{Z}$ hat.

$\mathrm{Dioph}(\mathbb{N}) \leq \mathrm{Dioph}(\mathbb{Z})$

Wir benötigen eine Reduktion, die zu einem Polynom p ein neues Polynom $f(p)$ berechnet, so dass $p = 0$ genau dann eine Lösung über $\mathbb{N}$ hat, wenn $f(p) = 0$ eine Lösung über $\mathbb{Z}$ hat.

Theorem (Vier-Quadrate-Satz von Lagrange)

Jede natürliche Zahl x lässt sich als Summe von vier Quadratzahlen darstellen.

$$\forall x \in \mathbb{N} \quad \exists a, b, c, d \in \mathbb{Z} \colon x = a^2 + b^2 + c^2 + d^2.$$

$\mathrm{Dioph}(\mathbb{N}) \leq \mathrm{Dioph}(\mathbb{Z})$

Wir benötigen eine Reduktion, die zu einem Polynom p ein neues Polynom $f(p)$ berechnet, so dass $p = 0$ genau dann eine Lösung über $\mathbb{N}$ hat, wenn $f(p) = 0$ eine Lösung über $\mathbb{Z}$ hat.

Theorem (Vier-Quadrate-Satz von Lagrange)

Jede natürliche Zahl x lässt sich als Summe von vier Quadratzahlen darstellen.

$$\forall x \in \mathbb{N} \quad \exists a, b, c, d \in \mathbb{Z} \colon x = a^2 + b^2 + c^2 + d^2.$$

Hieraus folgt, dass die Gleichung $p(x_1, x_2 \ldots, x_t) = 0$ genau dann eine Lösung über $\mathbb{N}$ hat, wenn

$$p(a_1^2 + b_1^2 + c_1^2 + d_1^2, \ldots, a_t^2 + b_t^2 + c_t^2 + d_t^2) = 0$$

eine Lösung über $\mathbb{Z}$ hat.

Es folgt $\mathrm{Dioph}(\mathbb{N}) \leq \mathrm{Dioph}(\mathbb{Z})$.

$H \leq \mathrm{ExpDioph}(\mathbb{N})$

Für den Beweisschritt $H \leq \mathrm{ExpDioph}(\mathbb{N})$ benutzt man das Halteproblem von Registermaschinen.

$H \leq \mathrm{ExpDioph}(\mathbb{N})$

Für den Beweisschritt $H \leq \mathrm{ExpDioph}(\mathbb{N})$ benutzt man das Halteproblem von Registermaschinen.
Es genügt hier Registermaschinen mit sehr eingeschränkten Befehlssatz und endlich vielen Registern zu betrachten. (Siehe spätere Vorlesungen.)

$H \leq \mathrm{ExpDioph}(\mathbb{N})$

Für den Beweisschritt $H \leq \mathrm{ExpDioph}(\mathbb{N})$ benutzt man das Halteproblem von Registermaschinen.
Es genügt hier Registermaschinen mit sehr eingeschränkten Befehlssatz und endlich vielen Registern zu betrachten. (Siehe spätere Vorlesungen.)

Idee:

- Man kodiert die Befehlszählerfolge $b_1, \ldots, b_t$ als natürliche Zahl $b_1 \cdots b_t$ über einer ausreichend großen Basis.

$H \leq \mathrm{ExpDioph}(\mathbb{N})$

Für den Beweisschritt $H \leq \mathrm{ExpDioph}(\mathbb{N})$ benutzt man das Halteproblem von Registermaschinen.
Es genügt hier Registermaschinen mit sehr eingeschränkten Befehlssatz und endlich vielen Registern zu betrachten. (Siehe spätere Vorlesungen.)

Idee:

- Man kodiert die Befehlszählerfolge $b_1, \ldots, b_t$ als natürliche Zahl $b_1 \cdots b_t$ über einer ausreichend großen Basis.
- Die Folge der Inhalte eines Registers $c(i)$ werden ebenso als eine natürliche Zahl kodiert.

$H \leq \mathrm{ExpDioph}(\mathbb{N})$

Für den Beweisschritt $H \leq \mathrm{ExpDioph}(\mathbb{N})$ benutzt man das Halteproblem von Registermaschinen.
Es genügt hier Registermaschinen mit sehr eingeschränkten Befehlssatz und endlich vielen Registern zu betrachten. (Siehe spätere Vorlesungen.)

Idee:

- Man kodiert die Befehlszählerfolge $b_1, \ldots, b_t$ als natürliche Zahl $b_1 \cdots b_t$ über einer ausreichend großen Basis.
- Die Folge der Inhalte eines Registers $c(i)$ werden ebenso als eine natürliche Zahl kodiert.
- Man verwendet nun (exponentielle) Gleichungen um zu garantieren, dass die Zahlen der Konfigurationsabfolge der Registermaschine entsprechen.

$H \leq \mathrm{ExpDioph}(\mathbb{N})$

Für den Beweisschritt $H \leq \mathrm{ExpDioph}(\mathbb{N})$ benutzt man das Halteproblem von Registermaschinen.
Es genügt hier Registermaschinen mit sehr eingeschränkten Befehlssatz und endlich vielen Registern zu betrachten. (Siehe spätere Vorlesungen.)

Idee:

- Man kodiert die Befehlszählerfolge $b_1, \ldots, b_t$ als natürliche Zahl $b_1 \cdots b_t$ über einer ausreichend großen Basis.
- Die Folge der Inhalte eines Registers $c(i)$ werden ebenso als eine natürliche Zahl kodiert.
- Man verwendet nun (exponentielle) Gleichungen um zu garantieren, dass die Zahlen der Konfigurationsabfolge der Registermaschine entsprechen.

  (Diverse Details sind in dieser groben Übersicht unterschlagen.)

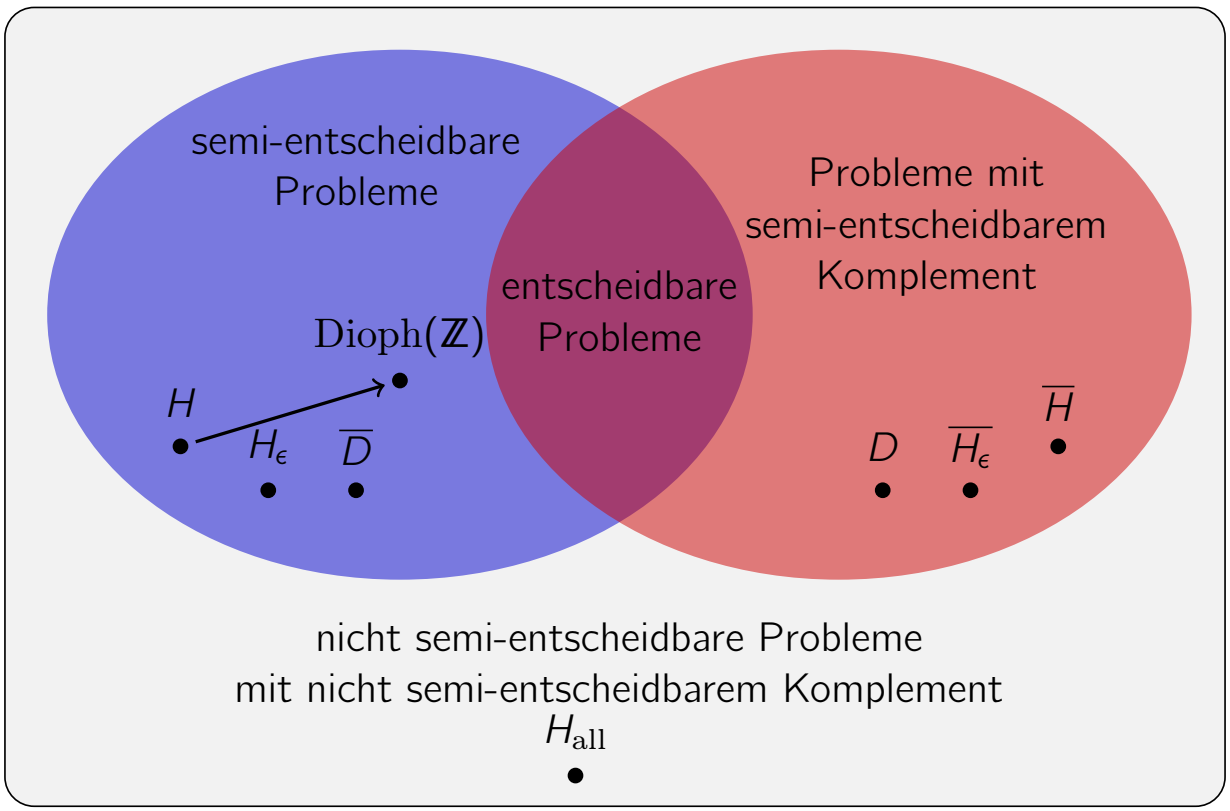
$\mathrm{ExpDioph}(\mathbb{N}) \leq \mathrm{Dioph}(\mathbb{N})$

$\mathrm{ExpDioph}(\mathbb{N}) \leq \mathrm{Dioph}(\mathbb{N})$

Dieser Teil ist aufwendig...

Vorlesung 10
Das Postsche Korrespondenzproblem

Wdh.: Berechenbarkeitslandschaft



Wdh.: Komplexität des allgemeines Halteproblem

Vorbeobachtung: H_ϵ ist unentscheidbar, aber semi-entscheidbar. Folglich ist $\overline{H}_\epsilon$ nicht semi-entscheidbar.

Wir haben gezeigt

Behauptung A

$$\overline{H}_\epsilon \leq \overline{H}_{\text{all}}$$

Behauptung B

$$\overline{H}_\epsilon \leq H_{\text{all}}$$

Aus diesen Reduktionen folgt:

Satz

Weder $\overline{H}_{all}$ noch H_{all} sind semi-entscheidbar.

Wdh.: Hilberts zehntes Problem

Hilberts zehntes Problem

Beschreibe einen Algorithmus, der entscheidet, ob ein gegebenes Polynom mit ganzzahligen Koeffizienten eine ganzzahlige Nullstelle hat.

Die diesem Entscheidungsproblem zugrundeliegende Sprache ist

$$N = \{ p \mid p \text{ ist ein Polynom mit einer ganzzahligen Nullstelle} \} .$$

Satz von Matijasevič (1970)

Das Problem, ob ein ganzzahliges Polynom eine ganzzahlige Nullstelle hat, ist unentscheidbar.

Wdh.: Unentscheidbarkeit des Nullstellenproblems

Sei $\mathrm{Dioph}(M)$ das Problem zu entscheiden, ob eine gegebene diophantische Gleichung über M lösbar ist.

Sei $\mathrm{ExpDioph}(M)$ das Problem zu entscheiden, ob eine gegebene Gleichung, die zusätzlich noch Terme der Form x^y haben darf, über M lösbar ist.

Die Reduktionskette im Beweis ist:

$$H \leq \mathrm{ExpDioph}(\mathbb{N}) \leq \mathrm{Dioph}(\mathbb{N}) \leq \mathrm{Dioph}(\mathbb{Z}) = N$$

Das Postsche Korrespondenzproblem – Einführung

Das Postsche Korrespondenzproblem (PKP) ist eine Art Puzzle aus Dominos.

- Jeder Domino ist mit zwei Wörtern über einem Alphabet Σ beschrieben, ein Wort in der oberen Hälfte und eines in der unteren. Gegeben sei eine Menge K von Dominos z.B.

$$K = \left\{ \left[\frac{b}{ca}\right], \left[\frac{a}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{abc}{c}\right] \right\} .$$

- Die Aufgabe besteht darin, eine Folge von Dominos aus K zu ermitteln, so dass sich oben und unten das gleiche Wort ergibt.
- Die Folge soll aus mindestens einem Domino bestehen.
- Wiederholungen von Dominos sind erlaubt. Ein Beispiel für eine derartige korrespondierende Folge über K ist

$$\left[\frac{a}{ab}\right] \left[\frac{b}{ca}\right] \left[\frac{ca}{a}\right] \left[\frac{a}{ab}\right] \left[\frac{abc}{c}\right] .$$

Das Postsche Korrespondenzproblem – Einführung

Nicht für jede Menge K ist dies möglich. Betrachte beispielsweise

$$K = \left\{ \left[\frac{abc}{ca}\right], \left[\frac{b}{aa}\right], \left[\frac{abcb}{abc}\right], \left[\frac{abc}{bc}\right] \right\} .$$

Warum gibt es keine Lösung des PKP-Puzzles für diese Dominos?

Definition des Postschen Korrespondenzproblem

Definition (Postsches Korrespondenzproblem, kurz: PKP)

Eine Instanz des PKP besteht aus einer Menge

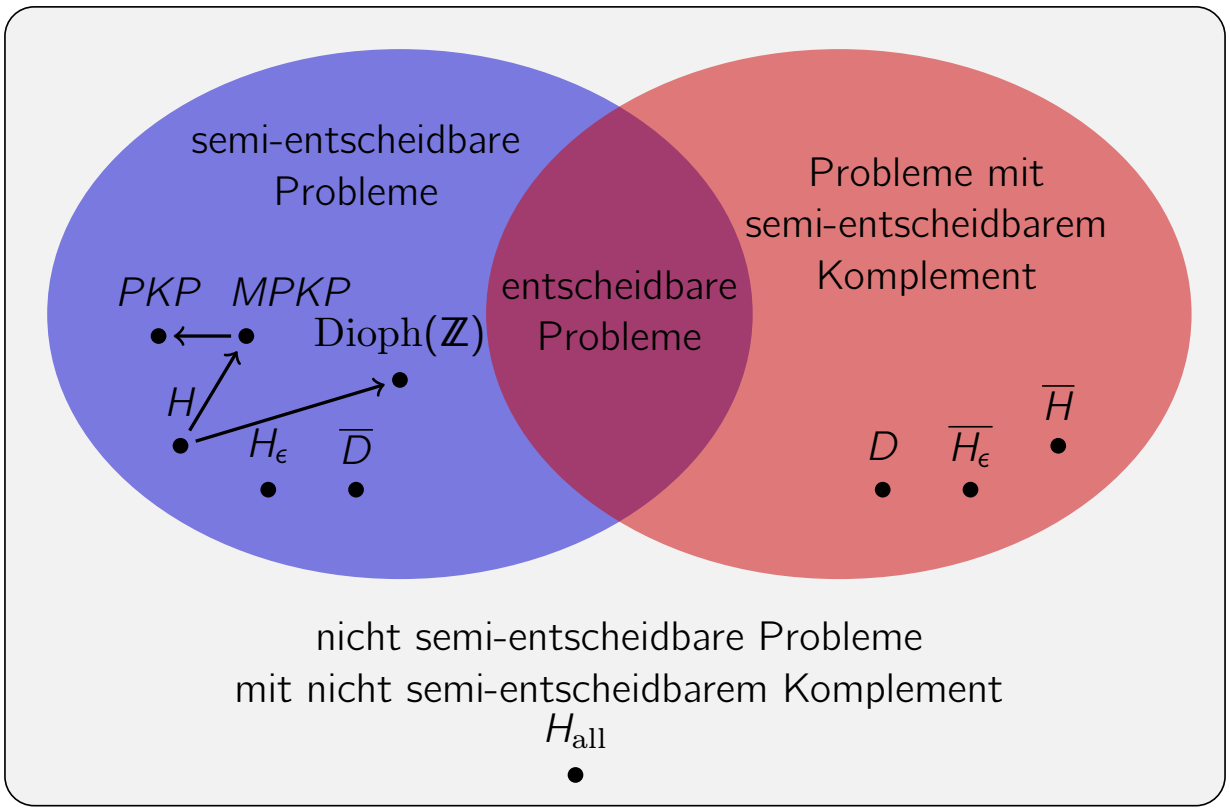
$$K = \left\{ \left[\frac{x_1}{y_1}\right], \ldots, \left[\frac{x_k}{y_k}\right] \right\} ,$$

wobei x_i und y_i nichtleere Wörter über einem endlichen Alphabet Σ sind. Es soll entschieden werden, ob es eine korrespondierende Folge von Indizes $i_1, \ldots, i_n \in \{1, \ldots, k\}$ für ein $n \geq 1$ gibt, so dass gilt $x_{i_1} x_{i_2} \ldots x_{i_n} = y_{i_1} y_{i_2} \ldots y_{i_n}$.

Die Elemente der Menge K bezeichnen wir als Dominos.

Wir werden die Unentscheidbarkeit des PKP durch eine kurze Reduktionskette nachweisen, die einen Umweg über eine Variante des PKP nimmt.

Berechenbarkeitslandschaft



Definition des Modifizierten PKP

Definition (Modifiziertes PKP, kurz: MPKP)

Eine Instanz des MPKP besteht aus einer geordneten Menge

$$K = \left(\left[\frac{x_1}{y_1}\right], \ldots, \left[\frac{x_k}{y_k}\right]\right) ,$$

wobei x_i und y_i nichtleere Wörter über einem endlichen Alphabet Σ sind. Es soll entschieden werden, ob es eine *korrespondierende Folge* von Indizes $i_2, \ldots, i_n \in \{1, \ldots, k\}$, für ein $n \geq 1$ gibt, so dass gilt $x_1 x_{i_2} \ldots x_{i_n} = y_1 y_{i_2} \ldots y_{i_n}$.

Die Modifizierung liegt also darin, dass wir einen Startdomino $\left[\frac{x_1}{y_1}\right]$ bestimmt haben, mit dem die korrespondierende Folge beginnen muss.

Reduktionskette

Wir werden die folgenden zwei Aussagen beweisen.

Lemma A
$MPKP \leq PKP$.

Lemma B
$H \leq MPKP$.

Aus der Transitivität der Reduzierbarkeit (Übungsaufgabe) folgt $H \leq PKP$ und somit folgt:

Satz
Das PKP ist nicht entscheidbar.

Beweis von Lemma A ($MPKP \leq PKP$)

Beschreibung der Funktion f:

Seien $\#$ und $\$$ zwei Symbole, die nicht im Alphabet Σ des MPKP enthalten sind.

Wir bilden $K = \left(\left[\frac{x_1}{y_1}\right], \ldots, \left[\frac{x_k}{y_k}\right]\right)$ auf

$$f(K) = \left\{\left[\frac{x'_0}{y'_0}\right], \left[\frac{x'_1}{y'_1}\right], \ldots, \left[\frac{x'_k}{y'_k}\right], \left[\frac{x'_{k+1}}{y'_{k+1}}\right]\right\}$$

ab, wobei

- x'_i aus x_i (für $1 \leq i \leq k$) entsteht, indem wir hinter jedem Zeichen ein $\#$ einfügen, und
- y'_i aus y_i (für $1 \leq i \leq k$) entsteht, indem wir vor jedem Zeichen ein $\#$ einfügen.

Ferner setzen wir $x'_0 = \#x'_1$; $x'_{k+1} = \$$; $y'_0 = y'_1$ und $y'_{k+1} = \#\$$.

Die Funktion f ist berechenbar.

Beweis von Lemma A ($MPKP \leq PKP$)

Beispiel:

$$K = \left(\left[\frac{ab}{a} \right], \left[\frac{c}{abc} \right], \left[\frac{a}{b} \right] \right)$$

wird abgebildet auf

$$f(K) = \left\{ \left[\frac{\#a\#b\#}{\#a} \right], \left[\frac{a\#b\#}{\#a} \right], \left[\frac{c\#}{\#a\#b\#c} \right], \left[\frac{a\#}{\#b} \right], \left[\frac{\$}{\#\$} \right] \right\}.$$

Lösung des MPKP:

$$\left[\frac{ab}{a} \right] \left[\frac{a}{b} \right] \left[\frac{ab}{a} \right] \left[\frac{c}{abc} \right]$$

Lösung des PKP:

$$\left[\frac{\#a\#b\#}{\#a} \right] \left[\frac{a\#}{\#b} \right] \left[\frac{a\#b\#}{\#a} \right] \left[\frac{c\#}{\#a\#b\#c} \right] \left[\frac{\$}{\#\$} \right]$$

Beweis von Lemma A ($MPKP \leq PKP$)

Zu zeigen: $K \in MPKP \Rightarrow f(K) \in PKP$

Sei $(i_2, \ldots, i_n)$ eine Lösung für K, d.h.

$$x_1 x_{i_2} \ldots x_{i_n} = y_1 y_{i_2} \ldots y_{i_n} = a_1 a_2 \ldots a_s$$

für geeignet gewählte Symbole $a_1, \ldots, a_s$ aus Σ.

Dann ist $(0, i_2, \ldots, i_n, k+1)$ eine Lösung für $f(K)$, denn

$$x'_0 x'_{i_2} \ldots x'_{i_n} \$ = \#a_1\#a_2\# \ldots \#a_s\#\$ = y'_0 y'_{i_2} \ldots y'_{i_n} \#\$$$

Aus einer Lösung für K bzgl. $MPKP$ lässt sich also eine Lösung für $f(K)$ bzgl. PKP konstruieren und die obige Behauptung ist gezeigt.

Beweis von Lemma A ($MPKP \leq PKP$)

Zu zeigen: $f(K) \in PKP \;\Rightarrow\; K \in MPKP$

Sei nun $(i_1, i_2, \ldots, i_n)$ eine Lösung minimaler Länge für $f(K)$.

- Beobachtung 1: Es gilt $i_1 = 0$ und $i_n = k+1$, weil nur x'_0 und y'_0 mit dem gleichen Zeichen beginnen und nur x'_{k+1} und y'_{k+1} mit dem gleichen Zeichen enden.
- Beobachtung 2: Es gilt $i_j \neq 0$ für $2 \leq j \leq n$, weil sonst zwei #-Zeichen im oberen Wort direkt aufeinanderfolgen würden, was im unteren Wort unmöglich ist.
- Beobachtung 3: Es gilt $i_j \neq k+1$ für $1 \leq j < n$, denn würde das \$-Zeichen vorher auftreten, könnten wir die vorliegende minimale korrespondierende Folge nach dem ersten Vorkommen des \$-Zeichens abschneiden und hätten eine noch kürzere Lösung gefunden.

Beweis von Lemma A ($MPKP \leq PKP$)

Aus den Beobachtungen folgt: Unsere PKP-Lösung für $f(K)$ hat die Struktur

$$\underbrace{x'_{i_1}}_{=x'_0=\#x'_1}\; x'_{i_2} \ldots x'_{i_{n-1}}\; \underbrace{x'_{i_n}}_{=x'_{k+1}=\$} \;=\; \#a_1\#a_2\#\ldots\#a_s\#\$$$

$$= \underbrace{y'_{i_1}}_{=y'_0=y'_1}\; y'_{i_2} \ldots y'_{i_{n-1}}\; \underbrace{y'_{i_n}}_{=y'_{k+1}=\#\$}$$

für geeignet gewählte Symbole $a_1, \ldots, a_s$ aus Σ.

Daraus können wir die folgende MPKP-Lösung für K konstruieren:

$$x_1 x_{i_2} \ldots x_{i_{n-1}} \;=\; a_1 a_2 \ldots a_s \;=\; y_1 y_{i_2} \ldots y_{i_{n-1}} \;.$$

Somit gilt $f(K) \in PKP \;\Rightarrow\; K \in MPKP$. □

Simulation einer TM durch Dominos – ein Beispiel

- Scheinbar haben Dominos wenig mit Turingmaschinen zu tun.
- In Lemma B ($H \leq MPKP$) wird dennoch behauptet, dass man mit Hilfe eines Puzzles aus Dominos das Halteproblem für Turingmaschinen entscheiden kann.
- Bevor wir in den Beweis des Lemmas einsteigen, möchten wir auf der Basis eines Beispiels illustrieren, wie die Rechnung einer Turingmaschine durch ein Puzzle aus Dominos „simuliert“ werden kann.

Simulation einer TM durch Dominos – ein Beispiel

Betrachte die TM $M = (Q, \Sigma, \Gamma, B, q_0, \overline{q}, \delta)$ wobei:

$\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, B\}$, $Q = \{q_0, q_1, q_2, \overline{q}\}$.

Die Übergangsfunktion δ sei gegeben durch

δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	$(\overline{q}, 1, N)$
q_1	$(q_2, 0, R)$	$(q_1, 1, R)$	$(\overline{q}, 1, N)$
q_2	$(q_2, 0, R)$	$(q_2, 1, R)$	(q_2, B, R)

- Die TM M erkennt, ob das Eingabewort von der Form $0^i 1^j$, mit $i, j \geq 0$, ist.
- Bei Eingabe eines Wortes dieser Form terminiert die Rechnung im Zustand $\overline{q}$ und die Maschine akzeptiert. Ansonsten läuft der Kopf im Zustand q_2 weiter und weiter nach rechts.

Simulation einer TM durch Dominos – ein Beispiel

Die Rechnung der TM auf einer gegebenen Eingabe kann durch eine Konfigurationsfolge beschrieben werden.

Konfigurationsfolge von M auf Eingabe $w = 0011$

$$q_0 0011 \vdash 0q_0 011 \vdash 00q_0 11 \vdash 001q_1 1 \vdash 0011q_1 B \vdash 0011\overline{q}1$$

- Wir möchten die Rechnung einer TM auf einer Eingabe durch ein Puzzle aus Dominos „simulieren“. Dieses Puzzle entspricht dem MPKP.
- Als Startdomino für das MPKP wählen wir einen Domino, bei dem das untere Wort aus der Anfangskonfiguration mit drei zusätzlichen Trennsymbolen besteht.

$$\left[\frac{\#}{\#\#q_0 0011\#}\right].$$

Simulation einer TM durch Dominos – ein Beispiel

Das Puzzle für unsere Beispielrechnung (M, w) enthält unter anderem jeweils einen Domino für jedes Zeichen aus $\Gamma \cup \{\#\}$.

$$\left[\frac{0}{0}\right], \left[\frac{1}{1}\right], \left[\frac{B}{B}\right], \left[\frac{\#}{\#}\right]$$

Liste erlaubter Dominos: Wir erweitern diese Menge um je einen Domino für jeden Eintrag in der Tabelle der Überführungsfunktion δ, der den jeweiligen Übergang inklusive der Kopfbewegung beschreibt.

$$\left[\frac{q_0 0}{0 q_0}\right], \left[\frac{q_0 1}{1 q_1}\right], \left[\frac{q_0 B}{\overline{q}1}\right], \left[\frac{q_1 0}{0 q_2}\right], \left[\frac{q_1 1}{1 q_1}\right], \left[\frac{q_1 B}{\overline{q}1}\right], \left[\frac{q_2 0}{0 q_2}\right], \left[\frac{q_2 1}{1 q_2}\right], \left[\frac{q_2 B}{B q_2}\right]$$

Wir werden später noch weitere Steine zur Liste erlaubter Dominos hinzufügen.

Simulation einer TM durch Dominos – ein Beispiel

Beobachtung:

Wenn wir den Startdomino $\left[\frac{\#}{\#\#q_0 0011\#}\right]$ mit einer Folge von Dominos aus der Liste der erlaubten Dominos derart ergänzen, dass der obere String ein Präfix des unteren Strings ist, so

- rekonstruieren wir im unteren String die Konfigurationsfolge von M auf w, und
- der obere String folgt dem unteren mit einer Konfiguration Rückstand.

Simulation einer TM durch Dominos – ein Beispiel

Rekonstruktion der Konfigurationsfolge

Die ersten Dominos in der Lösung des Puzzles sind

$$\left[\frac{\#}{\#\#q_0 0011\#}\right] \quad \left[\frac{\#}{\#}\right]\left[\frac{q_0 0}{0 q_0}\right]\left[\frac{0}{0}\right]\left[\frac{1}{1}\right]\left[\frac{1}{1}\right]\left[\frac{\#}{\#}\right]$$
$$\left[\frac{\#}{\#}\right]\left[\frac{0}{0}\right]\left[\frac{q_0 0}{0 q_0}\right]\left[\frac{1}{1}\right]\left[\frac{1}{1}\right]\left[\frac{\#}{\#}\right]$$
$$\left[\frac{\#}{\#}\right]\left[\frac{0}{0}\right]\left[\frac{0}{0}\right]\left[\frac{q_0 1}{1 q_1}\right]\left[\frac{1}{1}\right]\left[\frac{\#}{\#}\right]$$
$$\left[\frac{\#}{\#}\right]\left[\frac{0}{0}\right]\left[\frac{0}{0}\right]\left[\frac{1}{1}\right]\left[\frac{q_1 1}{1 q_1}\right]\left[\frac{\#}{\#}\right]$$
$$\left[\frac{\#}{\#}\right]\left[\frac{0}{0}\right]\left[\frac{0}{0}\right]\left[\frac{1}{1}\right]\left[\frac{1}{1}\right]\left[\frac{q_1 \#}{\overline{q} 1\#}\right] \ldots$$

Simulation einer TM durch Dominos – ein Beispiel

- Im letzten Schritt haben wir allerdings einen Domino verwendet, der nicht in der zuvor spezifizierten Liste erlaubter Dominos enthalten ist.
- Tatsächlich ergänzen wir die Liste erlaubter Dominos um die folgenden Elemente.

$$\left[\frac{q_0\#}{\overline{q}1\#}\right], \left[\frac{q_1\#}{\overline{q}1\#}\right]$$

Die Aufgabe dieser Dominos ist es, Überführungen zu realisieren, die ein neues Blank-Symbol benötigen, da der Kopf am Ende des Wortes steht.

Simulation einer TM durch Dominos – ein Beispiel

Wie können wir es schaffen, dass der obere String seinen Rückstand am Ende der Rechnung aufholt? – Zu diesem Zweck ergänzen wir die Liste der erlaubten Dominos um die folgenden Elemente.

$$\left[\frac{\overline{q}0}{\overline{q}}\right], \left[\frac{\overline{q}1}{\overline{q}}\right], \left[\frac{\overline{q}B}{\overline{q}}\right], \left[\frac{0\overline{q}}{\overline{q}}\right], \left[\frac{1\overline{q}}{\overline{q}}\right], \left[\frac{B\overline{q}}{\overline{q}}\right]$$

Des Weiteren fügen wir noch einen Abschlussdomino hinzu.

$$\left[\frac{\#\overline{q}\#\#}{\#}\right]$$

Beachte: Diese neuen Dominos können nur dann zum Einsatz kommen, wenn der Endzustand $\overline{q}$ erreicht ist, also nur wenn die Rechnung der TM terminiert.

Simulation einer TM durch Dominos – ein Beispiel

Rekonstruktion der Konfigurationsfolge – Fortsetzung

$$\ldots \quad \left[\frac{\#}{\#}\right]\left[\frac{0}{0}\right]\left[\frac{0}{0}\right]\left[\frac{1}{1}\right]\left[\frac{1}{1}\right]\left[\frac{q_1\#}{\overline{q}1\#}\right]$$
$$\left[\frac{\#}{\#}\right]\left[\frac{0}{0}\right]\left[\frac{0}{0}\right]\left[\frac{1}{1}\right]\left[\frac{1}{1}\right]\left[\frac{\overline{q}1}{\overline{q}}\right]\left[\frac{\#}{\#}\right]$$
$$\left[\frac{\#}{\#}\right]\left[\frac{0}{0}\right]\left[\frac{0}{0}\right]\left[\frac{1}{1}\right]\left[\frac{1\overline{q}}{\overline{q}}\right]\left[\frac{\#}{\#}\right]$$
$$\left[\frac{\#}{\#}\right]\left[\frac{0}{0}\right]\left[\frac{0}{0}\right]\left[\frac{1\overline{q}}{\overline{q}}\right]\left[\frac{\#}{\#}\right]$$
$$\left[\frac{\#}{\#}\right]\left[\frac{0}{0}\right]\left[\frac{0\overline{q}}{\overline{q}}\right]\left[\frac{\#}{\#}\right]$$
$$\left[\frac{\#}{\#}\right]\left[\frac{0\overline{q}}{\overline{q}}\right]\left[\frac{\#}{\#}\right] \qquad \left[\frac{\#\overline{q}\#\#}{\#}\right] \; .$$

Simulation einer TM durch Dominos – ein Beispiel

Jetzt stimmt der obere mit dem unteren String überein.

- Die Idee hinter der obigen Konstruktion ist es, eine Eingabe für das Halteproblem in ein MPKP-Puzzle zu transformieren, so dass das Puzzle genau dann eine Lösung hat, wenn die im Halteproblem betrachtete TM auf ihrer Eingabe hält.
- Unser Beispiel hat erläutert, wie eine derartige Transformation für eine bestimmte Eingabe des Halteproblems aussehen könnte. Der folgende Beweis für Lemma B verallgemeinert und formalisiert das Vorgehen aus unserem Beispiel.

Beweis von Lemma B ($H \leq MPKP$)

Wir beschreiben eine berechenbare Funktion f, die eine syntaktisch korrekte Eingabe für H der Form $(\langle M \rangle, w)$ auf eine syntaktisch korrekte Instanz $K = f\big((\langle M \rangle, w)\big)$ für das MPKP abbildet, so dass gilt

$$M \text{ hält auf } w \;\Leftrightarrow\; K \text{ hat eine Lösung } .$$

Syntaktisch nicht korrekte Eingaben für H werden auf syntaktisch nicht korrekte Eingaben für MPKP abgebildet.

Das Alphabet, das wir für die MPKP-Instanz verwenden, ist $\Gamma \cup Q \cup \{\#\}$, wobei gelte $\# \notin \Gamma \cup Q$.

Beweis von Lemma B ($H \leq MPKP$)

Konstruktion der Funktion f

Gegeben sei das Tupel $(\langle M \rangle, w)$. Wir beschreiben, welche Dominos die Menge $K = f\big((\langle M \rangle, w)\big)$ enthält.

Der Startdomino ist von der Form

$$\left[\frac{\#}{\#\#q_0 w \#}\right].$$

Des Weiteren enthalte K die folgenden Arten von Dominos.

Kopierdominos:

$$\left[\frac{a}{a}\right] \text{ für alle } a \in \Gamma \cup \{\#\}$$

Beweis von Lemma B ($H \leq MPKP$)

Überführungsdominos:

$$\left[\frac{qa}{q'c}\right] \quad \text{falls } \delta(q,a) = (q',c,N), \text{ für } q \in Q \setminus \{\overline{q}\},\ a \in \Gamma$$

$$\left[\frac{qa}{cq'}\right] \quad \text{falls } \delta(q,a) = (q',c,R), \text{ für } q \in Q \setminus \{\overline{q}\},\ a \in \Gamma$$

$$\left[\frac{bqa}{q'bc}\right] \quad \text{falls } \delta(q,a) = (q',c,L), \text{ für } q \in Q \setminus \{\overline{q}\},\ a,b \in \Gamma$$

Beweis von Lemma B ($H \leq MPKP$)

Spezielle Überführungsdominos, die implizite Blanks berücksichtigen:

$$\left[\frac{\#qa}{\#q'Bc}\right] \quad \text{falls } \delta(q,a) = (q',c,L), \text{ für } q \in Q \setminus \{\overline{q}\},\ a \in \Gamma$$

$$\left[\frac{q\#}{q'c\#}\right] \quad \text{falls } \delta(q,B) = (q',c,N), \text{ für } q \in Q \setminus \{\overline{q}\}$$

$$\left[\frac{q\#}{cq'\#}\right] \quad \text{falls } \delta(q,B) = (q',c,R), \text{ für } q \in Q \setminus \{\overline{q}\}$$

$$\left[\frac{bq\#}{q'bc\#}\right] \quad \text{falls } \delta(q,B) = (q',c,L), \text{ für } q \in Q \setminus \{\overline{q}\},\ b \in \Gamma$$

$$\left[\frac{\#q\#}{\#q'Bc\#}\right] \quad \text{falls } \delta(q,B) = (q',c,L), \text{ für } q \in Q \setminus \{\overline{q}\}$$

Beweis von Lemma B ($H \leq MPKP$)

Löschdominos:

$$\left[\frac{a\overline{q}}{\overline{q}}\right] \text{ und } \left[\frac{\overline{q}a}{\overline{q}}\right] \text{ für } a \in \Gamma$$

Abschlussdomino:

$$\left[\frac{\#\overline{q}\#\#}{\#}\right]$$

Dies sind alle Dominos in der MPKP-Instanz. Die Beschreibung der Funktion f ist somit abgeschlossen.

Beweis von Lemma B ($H \leq MPKP$)

Wir beweisen nun die Korrektheit der Konstruktion:

Zu zeigen 1.) f ist berechenbar. ✓

Zu zeigen 2.) M hält auf$w \Rightarrow K \in MPKP$

Wenn M auf w hält, so entspricht die Rechnung von M auf w einer endlichen Konfigurationsfolge der Form

$$k_0 \vdash k_1 \vdash \cdots \vdash k_{t-1} \vdash k_t \ ,$$

wobei k_0 die Startkonfiguration und k_t die Endkonfiguration im Zustand $\overline{q}$ ist.

Beweis von Lemma B ($H \leq MPKP$)

In diesem Fall können wir, beginnend mit dem Startdomino, nach und nach Kopier- und Überführungsdominos hinzulegen, so dass

- der untere String die vollständige Konfigurationsfolge von M auf w in der folgenden Form darstellt

$$\#\# \, k_0 \, \#\# \, k_1 \, \#\# \cdots \#\# \, k_{t-1} \, \#\# \, k_t \, \# \quad ,$$

  und
- der obere String ein Präfix des unteren Strings ist, nämlich

$$\#\# \, k_0 \, \#\# \, k_1 \, \#\# \cdots \#\# \, k_{t-1} \, \# \quad .$$

Beweis von Lemma B ($H \leq MPKP$)

Durch Hinzufügen von Löschdominos kann jetzt der Rückstand des oberen Strings fast ausgeglichen werden. Danach sind beide Strings identisch bis auf ein Suffix der Form

$$\#\overline{q}\# \quad .$$

Dieses Suffix fehlt im oberen String.

Nach Hinzufügen des Abschlussdominos

$$\left[\frac{\#\overline{q}\#\#}{\#}\right]$$

sind beide Strings somit identisch.

Wenn M auf w hält, gilt somit $K \in MPKP$.

Beweis von Lemma B ($H \leq MPKP$)

Zu zeigen: M hält nicht auf $w \Rightarrow K \notin MPKP$

Unsere Ausgangsitutation ist eine TM M und eine Eingabe w, so dass M nicht auf w hält.

Zum Zweck des Widerspruchs nehmen wir nun an, dass das MPKP-Puzzle K aus den Dominos für M und w eine Lösung hat, d.h., es gibt eine korrespondierende Folge aus diesen Dominos.

Allgemeine Beobachtungen zu korrespondierenden Folgen:

- Beim Startdomino ist der obere String kürzer als der untere.
- Auch bei den Kopier- und Überführungsdominos ist der obere String niemals länger als der untere.
- Nur Abschluss- und Löschdominos zeigen unten kürzere Strings als oben.

Beweis von Lemma B ($H \leq MPKP$)

Aus den Beobachtungen folgt:

- Die (eindeutige) korrespondierende Folge zur Rechnung von M auf w enthält deshalb zumindest einen Lösch- oder Abschlussdomino, denn sonst wäre der untere String länger als der obere.
- In dieser Folge erscheint deshalb der Zustand $\overline{q}$, da dieser Zustand auf allen Löschdominos und dem Abschlussdomino abgebildet ist.
- Die auf den Dominos dargestellte Rechnung von M auf w terminiert.

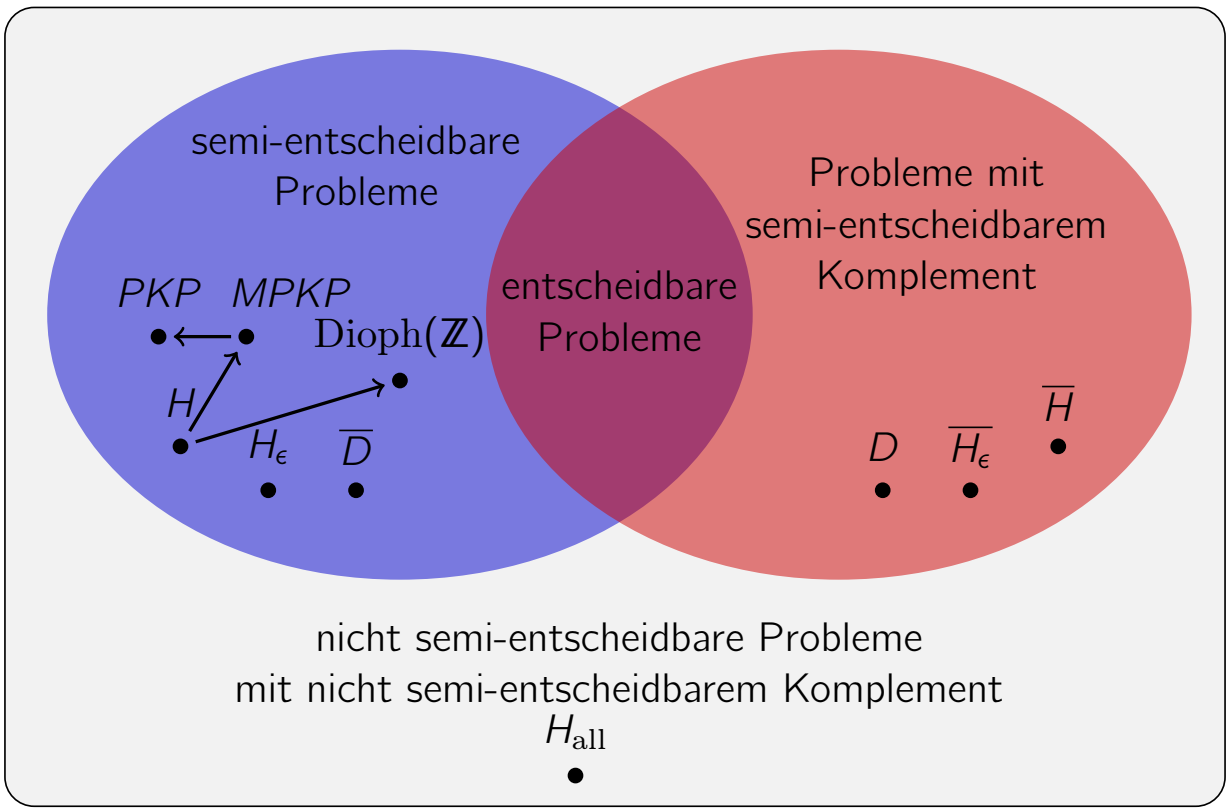
Dies widerspricht jedoch unserer Voraussetzung, dass M nicht auf w hält. Also gilt $K \notin MPKP$. □

Eingeschränkte Versionen des Korrespondenzproblems

Wie ist die Komplexität für eingeschränkte Varianten des Problems?

- Falls nur kurze Wörter erlaubt sind:
  - Wenn die Wörter auf den Dominos höchstens Länge 1 haben, so ist das Problem entscheidbar.
  - Schon für Wortlänge höchstens 2 ist das Problem unentscheidbar.
- Falls nur wenige Dominos erlaubt sind:
  - Für 2 Dominos ist das Problem entscheidbar.
    [Ehrenfeucht, Rozenberg] (1981)
  - Für 3 und 4 Dominos ist die Komplexität ungeklärt.
  - Für 5 Dominos ist das Problem unentscheidbar. [Neary] (2015)
  - Für 7 Dominos oder mehr ist das Problem unentscheidbar.
    [Matijasevič, Sénizergues] (1996)
  - Für unbeschränkt viele Dominos im Allgemeinen unentscheidbar.
    [Post] (1947)

Berechenbarkeitslandschaft

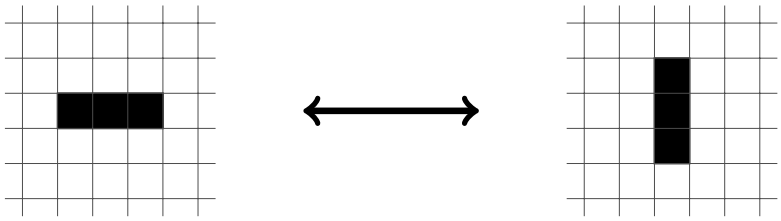


Conways Spiel des Lebens

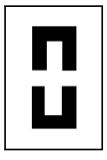
Conways Spiel des Lebens ist ein zellulärer Automat auf einem unendlichen 2-dimensionalen Gitter.
Zu jedem Zeitpunkt ist eine Zelle lebend oder tot.

Ausgehend von einer Anfangskonfiguration entwickelt sich die Zellenkonfiguration Schritt für Schritt folgendermaßen:

- Eine tote Zelle mit genau drei lebenden Nachbarn ist im nächsten Schritt lebendig.
- Lebende Zellen mit weniger als 2 lebenden Nachbarn sterben ab.
- Lebende Zellen mit mehr als 3 lebenden Nachbarn sterben ab.
- Alle anderen Zellen bleiben unverändert.



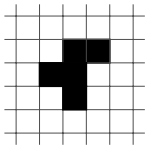
Einige Startkonfigurationen

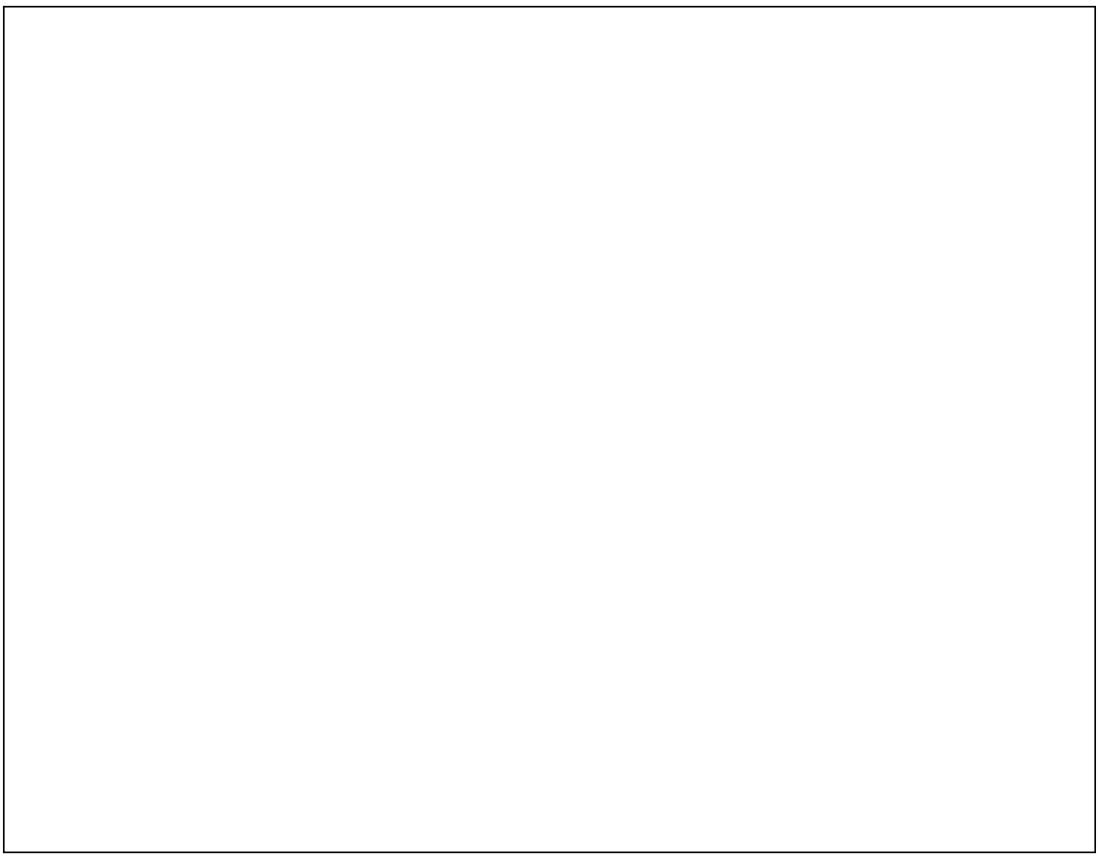
Das U: 

Der Gleiter:

Einige Startkonfigurationen

Das f-Pentomino.





User:Stummi/Wikimedia Commons/CC-BY-SA-3.0

Entscheidbarkeit im Spiel des Lebens

Satz

Das Spiel des Leben ist Turing-vollständig.

Dies impliziert, dass diverse Probleme unentscheidbar sind.

- Zum Beispiel ist unentscheidbar, ob eine gegebene Anfangskonfiguration ausstirbt.

Gospers Gleiterkanone

User:Kieff/Wikimedia Commons/CC-BY-SA-3.0

Vorlesung 11
WHILE-Programme

Wdh.: Das Postsche Korrespondenzproblem

Das Postsche Korrespondenzproblem (PKP) ist eine Art Puzzle aus Dominos.
Eine Instanz ist zum Beispiel

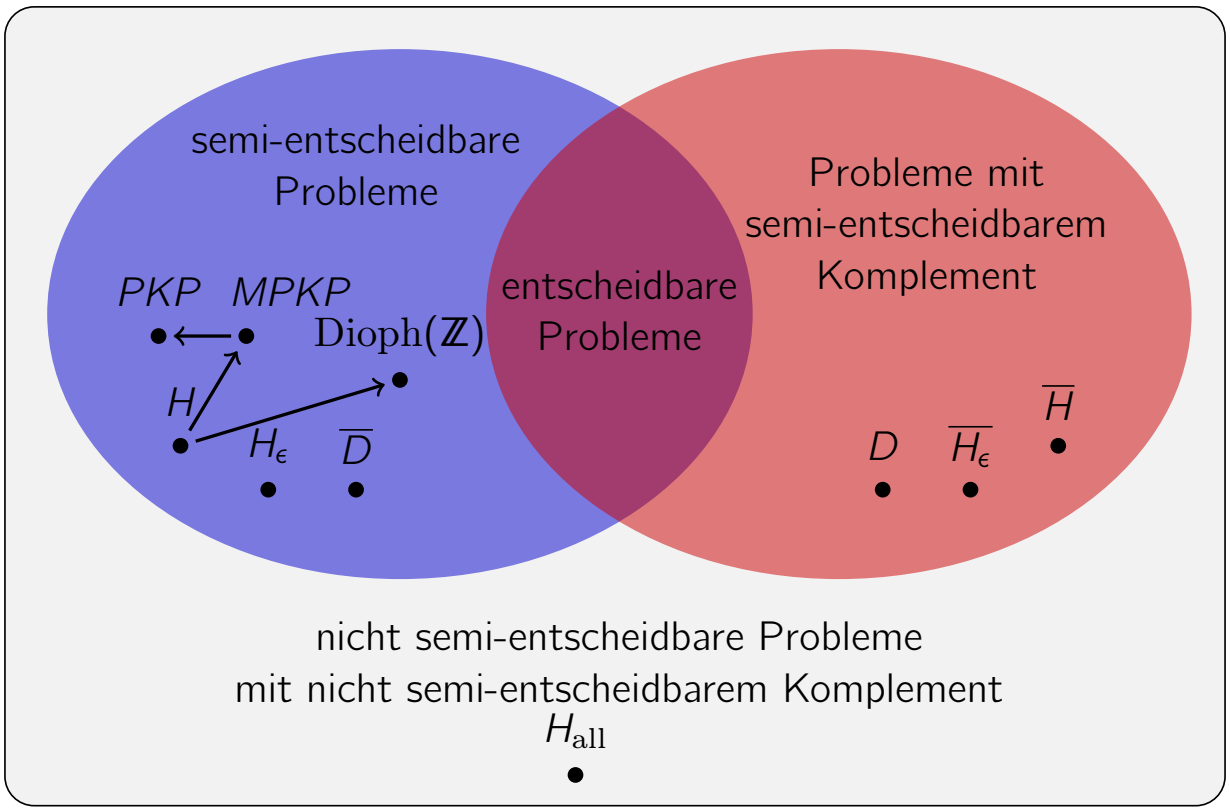
$$K = \left\{ \left[\frac{b}{ca}\right], \left[\frac{a}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{abc}{c}\right] \right\} .$$

Eine Lösung ist

$$\left[\frac{a}{ab}\right] \left[\frac{b}{ca}\right] \left[\frac{ca}{a}\right] \left[\frac{a}{ab}\right] \left[\frac{abc}{c}\right] .$$

Satz
Das PKP ist nicht entscheidbar.

Wdh.: Berechenbarkeitslandschaft



Wdh.: Simulation einer TM durch Dominos

δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	$(\overline{q}, 1, N)$
q_1	$(q_2, 0, R)$	$(q_1, 1, R)$	$(\overline{q}, 1, N)$
q_2	$(q_2, 0, R)$	$(q_2, 1, R)$	(q_2, B, R)

Wird simuliert durch:

Startdomino: $\left[\frac{\#}{\#\#q_0 0011\#}\right]$. Kopierdominos: $\left[\frac{0}{0}\right], \left[\frac{1}{1}\right], \left[\frac{B}{B}\right], \left[\frac{\#}{\#}\right]$.

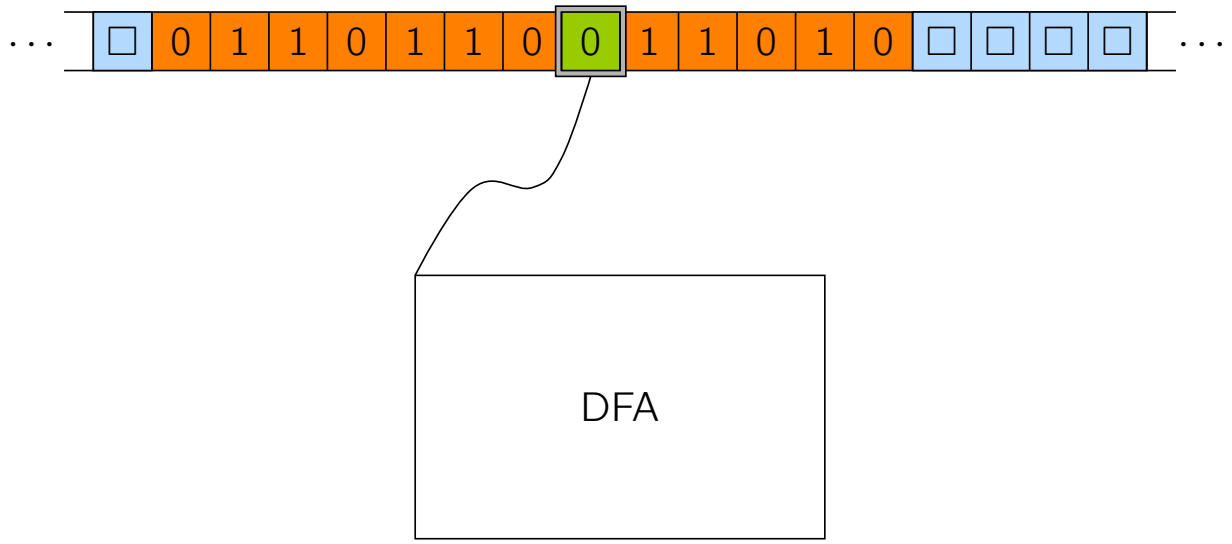
Überführungsdominos: $\left[\frac{q_0 0}{0 q_0}\right], \left[\frac{q_0 1}{1 q_1}\right], \left[\frac{q_0 B}{\overline{q}1}\right], \left[\frac{q_1 0}{0 q_2}\right], \left[\frac{q_1 1}{1 q_1}\right], \left[\frac{q_1 B}{\overline{q}1}\right], \left[\frac{q_2 0}{0 q_2}\right], \left[\frac{q_2 1}{1 q_2}\right], \left[\frac{q_2 B}{B q_2}\right]$

Spezielle Überführungsdominos: $\left[\frac{q_0 \#}{\overline{q}1\#}\right], \left[\frac{q_1 \#}{\overline{q}1\#}\right]$

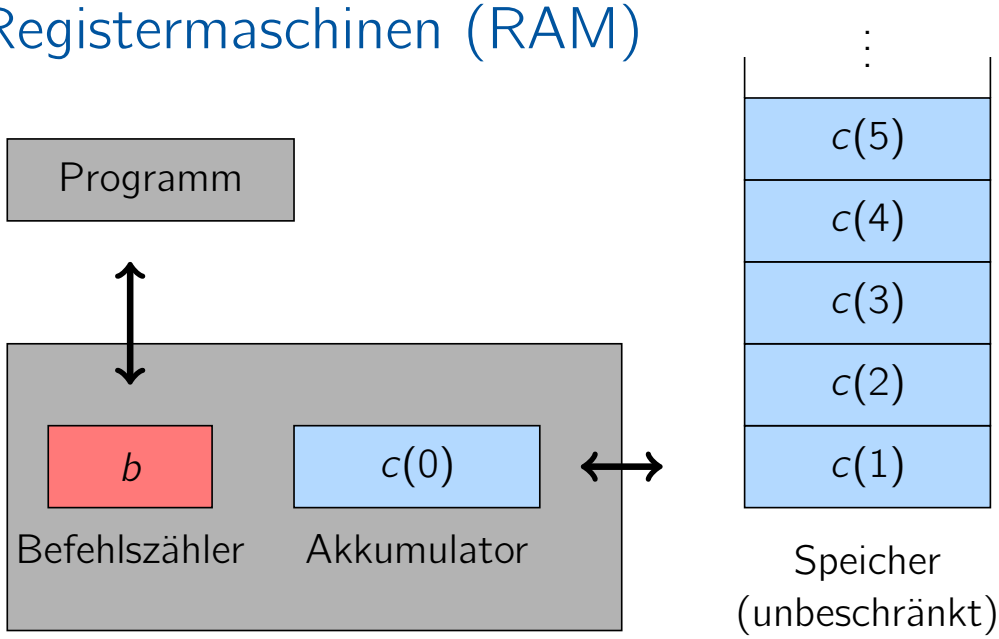
Löschdominos: $\left[\frac{\overline{q}0}{\overline{q}}\right], \left[\frac{\overline{q}1}{\overline{q}}\right], \left[\frac{\overline{q}B}{\overline{q}}\right], \left[\frac{0\overline{q}}{\overline{q}}\right], \left[\frac{1\overline{q}}{\overline{q}}\right], \left[\frac{B\overline{q}}{\overline{q}}\right]$

Abschlussdomino: $\left[\frac{\#\overline{q}\#\#}{\#}\right]$

Wdh.: Deterministische Turingmaschine (TM bzw. DTM)



Wdh.: Registermaschinen (RAM)



Befehlssatz:

LOAD, STORE, ADD, SUB, MULT, DIV
INDLOAD, INDSTORE, INDADD, INDSUB, INDMULT, INDDIV
CLOAD, CADD, CSUB, CMULT, CDIV
GOTO, IF c(0)?x THEN GOTO j (wobei ? aus {=,<,<=,>,>=} ist),
END

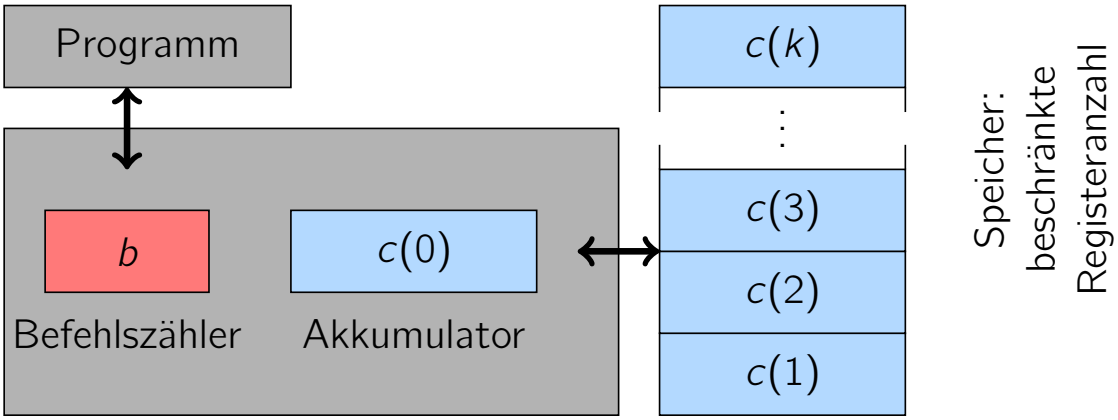
Turing-mächtige Rechnermodelle

Definition

Ein Rechnermodell wird als Turing-mächtig bezeichnet, wenn jede Funktion, die durch eine TM berechnet werden kann, auch durch dieses Rechnermodell berechnet werden kann.

- Da die Registermaschine die Turingmaschine simulieren kann, ist sie Turing-mächtig.
- Ebenso kann man zeigen, dass das Spiel des Lebens Turing-mächtig ist.

Eingeschränkte Registermaschine (eingeschränkte RAM)



Befehlssatz:

LOAD, STORE

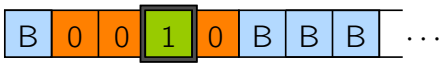
CLOAD, CADD, CSUB

GOTO,
IF c(0) $\neq$ 0 THEN GOTO j,
END

Man kann zeigen, dass die eingeschränkte RAM Turing-mächtig ist.

Wdh.: Simulation TM durch RAM

simulierte Turingmaschine M

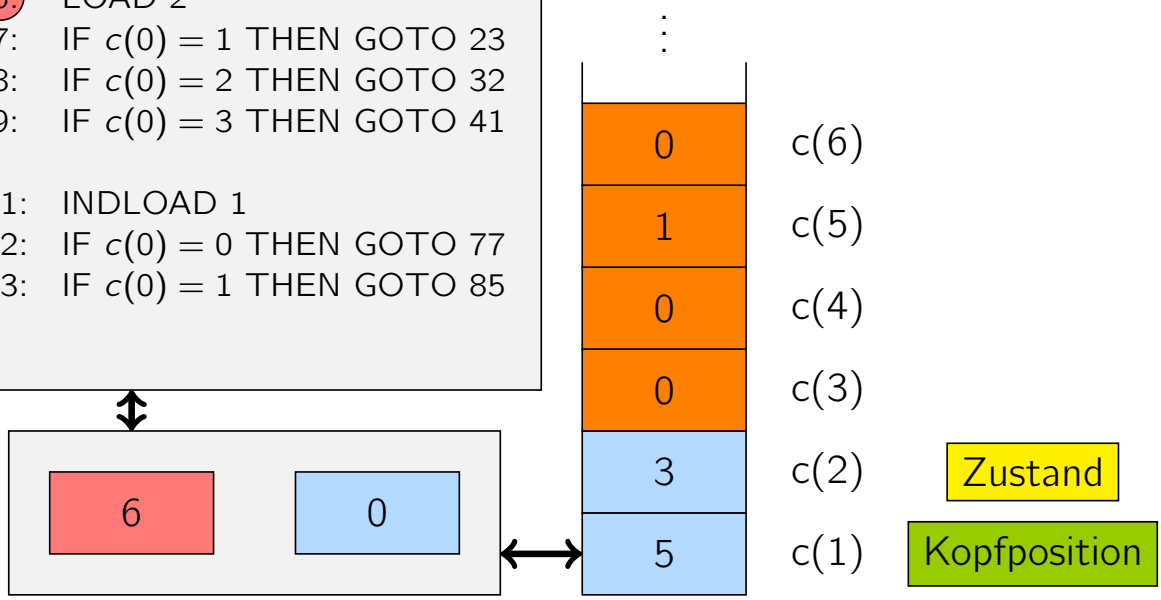


δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

q_3

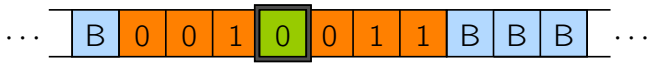
simulierende Registermaschine

```
6:  LOAD 2
7:  IF c(0) = 1 THEN GOTO 23
8:  IF c(0) = 2 THEN GOTO 32
9:  IF c(0) = 3 THEN GOTO 41

41: INDLOAD 1
42: IF c(0) = 0 THEN GOTO 77
43: IF c(0) = 1 THEN GOTO 85
```



Simulation TM durch eingeschränkte RAM

simulierte Turingmaschine M

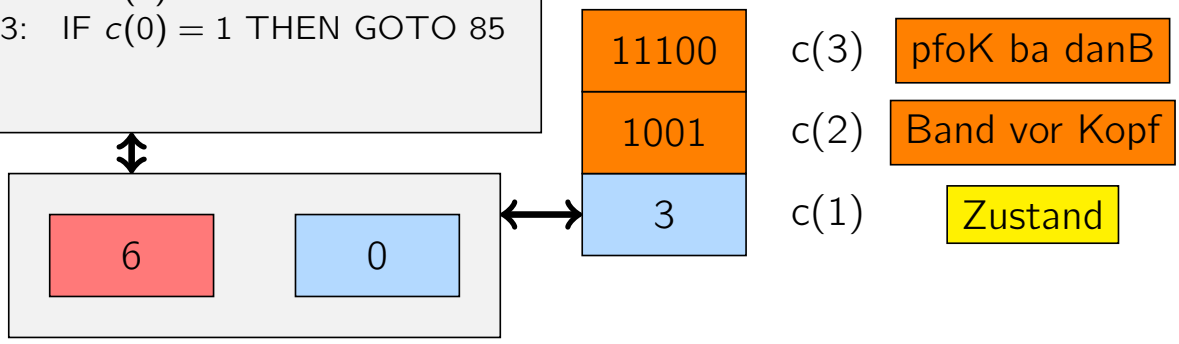


δ	0	1	B
q_1			
q_2			
q_3		$(q_2, 0, R)$	

q_3

simulierende Registermaschine

```
6:  LOAD 2
7:  IF c(0) = 1 THEN GOTO 23
8:  IF c(0) = 2 THEN GOTO 32
9:  IF c(0) = 3 THEN GOTO 41

41: INDLOAD 1
42: IF c(0) = 0 THEN GOTO 77
43: IF c(0) = 1 THEN GOTO 85
```



Turing-mächtige Programmiersprachen

Definition

Eine Programmiersprache wird als Turing-mächtig bezeichnet, wenn jede Funktion, die durch eine TM berechnet werden kann, auch durch ein Programm in dieser Programmiersprache berechnet werden kann.

Welche Elemente benötigt eine Programmiersprache, um Turing-mächtig zu sein?

Die Programmiersprache WHILE – Syntax

Elemente eines WHILE-Programms

- Variablen x_0 x_1 x_2 ...
- Konstanten -1 0 1
- Symbole ; := + $\neq$
- Schlüsselwörter WHILE DO END

Die Programmiersprache WHILE – Syntax

Induktive Definition – Induktionsanfang

Zuweisung

Für jedes $c \in \{-1, 0, 1\}$ ist die Zuweisung

$$x_i := x_j + c$$

ein WHILE-Programm.

Die Programmiersprache WHILE – Syntax

Induktive Definition – Induktionsschritte:

Hintereinanderausführung

Falls P_1 und P_2 WHILE-Programme sind, dann ist auch

$$P_1; P_2$$

ein WHILE-Programm.

WHILE-Konstrukt

Falls P ein WHILE-Programm ist, dann ist auch

$$\text{WHILE } x_i \neq 0 \text{ DO } P \text{ END}$$

ein WHILE-Programm.

Die Programmiersprache WHILE – Semantik

Ein WHILE-Programm P berechnet eine k-stellige Funktion der Form $f : \mathbb{N}^k \to \mathbb{N}$.

- Die Eingabe ist in den Variablen $x_1, \ldots, x_k$ enthalten.
- Alle anderen Variablen werden mit 0 initialisiert.
- Das Resultat eines WHILE-Programms ist die Zahl, die sich am Ende der Rechnung in der Variable x_0 ergibt.

- Programme der Form $x_i := x_j + c$ sind Zuweisungen des Wertes $x_j + c$ an die Variable x_i (wobei $0 + (-1) = 0$).
- In einem WHILE-Programm

$$P_1; P_2$$

  wird zunächst P_1 und dann P_2 ausgeführt.
- Das Programm WHILE $x_i \neq 0$ DO P END hat die Bedeutung, dass P solange ausgeführt wird, bis x_i den Wert 0 erreicht.

Beispiel eines WHILE-Programms

Was berechnet dieses WHILE-Programm?

WHILE $x_2 \neq 0$ DO
    $x_1 := x_1 + 1$;
    $x_2 := x_2 - 1$
END;
$x_0 := x_1$

Die Programmiersprache WHILE – Mächtigkeit

Satz

Die Programmiersprache WHILE ist Turing-mächtig.

Beweis:

Wir zeigen, dass jede Funktion, die durch eine eingeschränkte RAM berechnet werden kann, auch durch ein WHILE-Programm berechnet werden kann.

Da die eingeschränkte RAM Turing-mächtig ist, ist somit auch die Programmiersprache WHILE Turing-mächtig.

Sei Π ein beliebiges Programm der eingeschränkten RAM. Sei ℓ die Anzahl der Zeilen in Π und k die Anzahl der verwendeten Register.

Beweis Turing-Mächtigkeit von WHILE-Programmen

Wir speichern den Inhalt von Register $c(i)$ für $0 \leq i \leq k$ in der Variable x_i des WHILE-Programms.

In der Variable x_{k+1} speichern wir zudem den Befehlszähler b der eingeschränkten RAM ab.

Die Variable x_{k+2} verwenden wir als Hilfsvariable, die immer mit dem Wert 0 initialisiert ist.

Die einzelnen RAM-Befehle werden nun in Form von konstant vielen Zuweisungen der Form $x_i := x_j + c$ mit $c \in \{0, 1\}$ implementiert.

Beweis Turing-Mächtigkeit von WHILE-Programmen

RAM vs. **WHILE**

- LOAD, STORE
- CLOAD, CADD, CSUB, GOTO
- IF c(0) $\neq$ 0 GOTO
- END

- $x_i := x_j + c$ für $c \in \{-1, 0, 1\}$
- $P_1; P_2$
- WHILE $x_i \neq 0$ DO P END

Der RAM-Befehl LOAD i wird beispielsweise ersetzt durch

$$x_0 := x_i + 0;\ x_{k+1} := x_{k+1} + 1$$

Beweis Turing-Mächtigkeit von WHILE-Programmen

RAM vs. **WHILE**

- LOAD, STORE ✓
- CLOAD, CADD, CSUB, GOTO
- IF c(0) $\neq$ 0 GOTO
- END

- $x_i := x_j + c$ für $c \in \{-1, 0, 1\}$
- $P_1; P_2$
- WHILE $x_i \neq 0$ DO P END

Der RAM-Befehl CLOAD i wird ersetzt durch

$$x_0 := x_{k+2} + 0;\ \underbrace{x_0 := x_0 + 1;\ \ldots;\ x_0 := x_0 + 1;}_{i \text{ mal}}\ x_{k+1} := x_{k+1} + 1$$

Beweis Turing-Mächtigkeit von WHILE-Programmen

RAM vs. **WHILE**

- LOAD, STORE ✓
- CLOAD, CADD, CSUB, GOTO ✓
- IF c(0) $\neq$ 0 GOTO
- END

- $x_i := x_j + c$ für $c \in \{-1, 0, 1\}$
- $P_1; P_2$
- WHILE $x_i \neq 0$ DO P END

Den RAM-Befehl IF $c(0) \neq 0$ GOTO j ersetzen wir durch das WHILE-Programm:

$x_{k+1} := x_{k+1} + 1;$ $(b := b + 1)$
$x_{k+3} := x_0 + 0;$ $(help := c(0))$
WHILE $x_{k+3} \neq 0$ DO (while $help \neq 0$)
  $x_{k+1} := x_{k+2} + 0;\ \underbrace{x_{k+1} := x_{k+1} + 1; \cdots + 1;}_{j \text{ mal}}$ $(b := j)$
  $x_{k+3} := x_{k+2} + 0$ $(help := 0)$
END (end of while)

Beweis Turing-Mächtigkeit von WHILE-Programmen

RAM vs. **WHILE**

- LOAD, STORE ✓
- CLOAD, CADD, CSUB, GOTO ✓
- IF c(0) $\neq$ 0 GOTO ✓
- END

- $x_i := x_j + c$ für $c \in \{-1, 0, 1\}$
- $P_1; P_2$
- WHILE $x_i \neq 0$ DO P END

Den RAM-Befehl END ersetzen wir durch das WHILE-Programm

$$x_{k+1} = 0$$

Beweis Turing-Mächtigkeit von WHILE-Programmen

Jede Zeile des RAM-Programms wird nun wie oben beschrieben in ein WHILE-Programm transformiert.

Das WHILE-Programm für Zeile i bezeichnen wir mit P_i.

Aus P_i konstruieren wir nun ein WHILE-Programm P'_i mit der folgenden Semantik:

Falls $x_{k+1} = i$ dann führe P_i aus.

Übungsaufgabe: Implementiere das WHILE-Programm P'_i mit Unterprogramm P_i.

Beweis Turing-Mächtigkeit von WHILE-Programmen

Nun fügen wir die WHILE-Programme $P'_1, \ldots, P'_\ell$ zu einem WHILE-Programm P zusammen:

$x_{k+1} := 1;$

WHILE $x_{k+1} \neq 0$ DO

$P'_1; \ldots; P'_\ell$

END

P berechnet die gleiche Funktion wie Π. □

Ausblick: Die Programmiersprache LOOP

Syntax

Änderung im Vergleich zu WHILE-Programmen:

Wir ersetzen das WHILE-Konstrukt durch ein LOOP-Konstrukt der folgenden Form:

$$\text{LOOP } x_i \text{ DO } P \text{ END} \quad ,$$

wobei die Variable x_i nicht in P vorkommen darf.

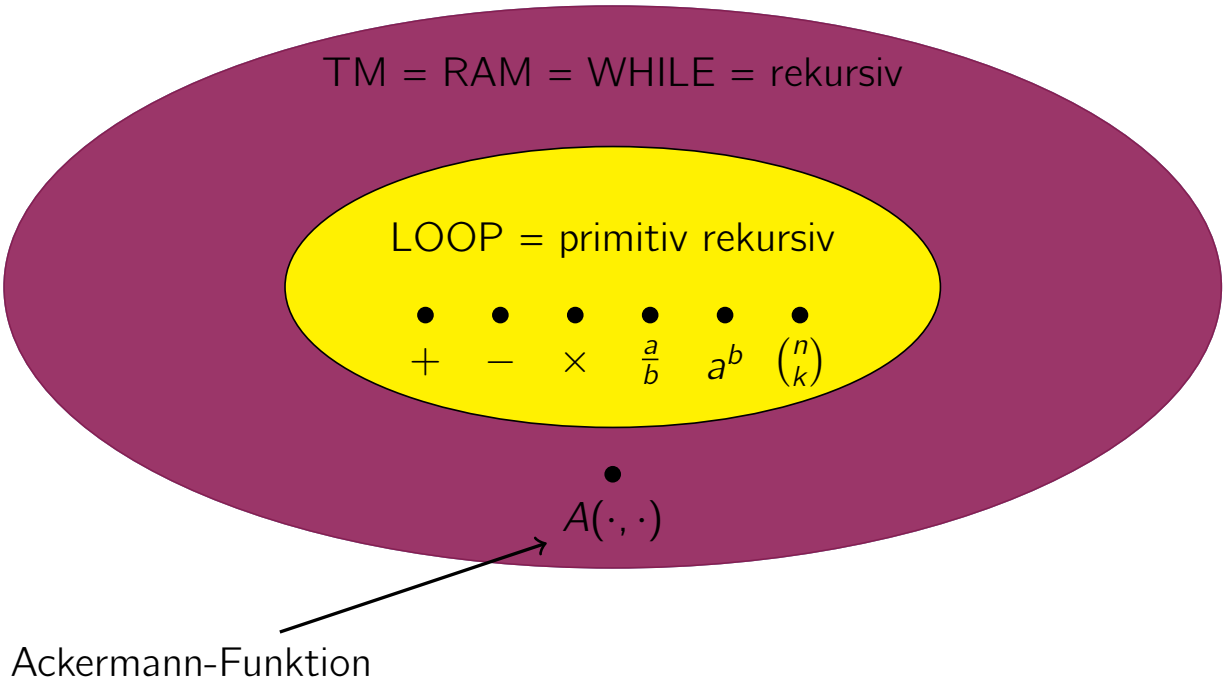
Semantik

Das Programm P wird x_i mal hintereinander ausgeführt.

Frage

Sind LOOP-Programme Turing-mächtig?

Ausblick: Mächtigkeit von LOOP-Programmen



Ausblick: Ackermann-Funktion

Definition

Die Ackermann-Funktion $A : \mathbb{N}^2 \to \mathbb{N}$ ist folgendermaßen definiert:

$$\begin{array}{lcll} A(0,n) & = & n+1 & \text{für } n \geq 0 \\ A(m+1,0) & = & A(m,1) & \text{für } m \geq 0 \\ A(m+1,n+1) & = & A(m,A(m+1,n)) & \text{für } m,n \geq 0 \end{array}$$

Ackermann-Funktion – Beispiele

$$\begin{array}{lcll} A(0,n) & = & n+1 & \text{für } n \geq 0 \\ A(m+1,0) & = & A(m,1) & \text{für } m \geq 0 \\ A(m+1,n+1) & = & A(m,A(m+1,n)) & \text{für } m,n \geq 0 \end{array}$$

Ein paar Beispiele:

$A(1,0) = A(0,1) = 2$

$A(1,1) = A(0,A(1,0)) = A(1,0) + 1 = 3$

$A(1,2) = A(0,A(1,1)) = A(1,1) + 1 = 4$

Allgemein: $A(1,n) = 2 + n$

$A(2,0) = A(1,1) = 3$

$A(2,1) = A(1,A(2,0)) = A(2,0) + 2 = 5$

Allgemein: $A(2,n) = 2n + 3$

$A(3,0) = A(2,1) = 5$

$A(3,1) = A(2,A(3,0)) = 2A(3,0) + 3$

Allgemein: $A(3,n) = 8 \cdot 2^n - 3$

Ausblick: Ackermann-Funktion

Wenn man den ersten Parameter fixiert ...

- $A(1, n) = 2 + n$,
- $A(2, n) = 2n + 3$,
- $A(3, n) = 8 \cdot 2^n - 3$,
- $A(4, n) = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^2}}}}}_{\substack{n+2 \text{ viele} \\ \text{Potenzen}}} - 3$,

Bereits $A(4, 2) = 2^{65536} - 3$ ist größer als die (vermutete) Anzahl der Atome im Weltraum.

Vorlesung 12
LOOP-Programme

Wdh.: Turing-mächtige Programmiersprachen

Definition
Eine Programmiersprache wird als Turing-mächtig bezeichnet, wenn jede Funktion, die durch eine TM berechnet werden kann, auch durch ein Programm in dieser Programmiersprache berechnet werden kann.

Satz
Die Programmiersprache WHILE ist Turing-mächtig.

Wdh.: Beispiel eines WHILE-Programms

Was berechnet dieses WHILE-Programm?

WHILE $x_2 \neq 0$ DO
    $x_1 := x_1 + 1$;
    $x_2 := x_2 - 1$
END;
$x_0 := x_1$

Die Programmiersprache LOOP – Syntax

Elemente eines LOOP-Programms

- Variablen x_0 x_1 x_2 ...
- Konstanten -1 0 1
- Symbole ; $:=$ $+$ $\neq$
- Schlüsselwörter LOOP DO END

Die Programmiersprache LOOP – Syntax

Induktive Definition – Induktionsanfang

Zuweisung

Für jedes $c \in \{-1, 0, 1\}$ ist die Zuweisung

$$x_i \;:=\; x_j + c$$

ein LOOP-Programm.

Statt $x_i := x_j + 0$ schreiben wir einfach $x_i := x_j$.

Die Programmiersprache LOOP – Syntax

Induktive Definition – Induktionsschritte:

Hintereinanderausführung

Falls P_1 und P_2 LOOP-Programme sind, dann ist auch

$$P_1; P_2$$

ein LOOP-Programm.

LOOP-Konstrukt

Falls P ein LOOP-Programm ist, dann ist auch

$$\text{LOOP } x_i \text{ DO } P \text{ END}$$

ein LOOP-Programm, wobei x_i nicht in P vorkommen darf.

Die Programmiersprache LOOP – Semantik

Ein LOOP-Programm P berechnet eine k-stellige Funktion der Form $f : \mathbb{N}^k \to \mathbb{N}$.

- Die Eingabe ist in den Variablen $x_1, \ldots, x_k$ enthalten.
- Alle anderen Variablen werden mit 0 initialisiert.
- Das Resultat eines LOOP-Programms ist die Zahl, die sich am Ende der Rechnung in der Variable x_0 ergibt.

- Programme der Form $x_i := x_j + c$ sind Zuweisungen des Wertes $x_j + c$ an die Variable x_i.
- In einem LOOP-Programm $P_1; P_2$ wird zunächst P_1 und dann P_2 ausgeführt.
- Das Programm LOOP x_i DO P END hat folgende Bedeutung: P wird x_i mal hintereinander ausgeführt.

LOOP-Programme – Beispiele

Beispiel 1

$x_0 := x_1 + 0;$
LOOP x_2 DO $x_0 := x_0 + 1$ END

Diese Programm berechnet die Addition $x_1 + x_2$.

Beispiel 2

LOOP x_1 DO
    LOOP x_2 DO $x_0 := x_0 + 1$ END
END

Diese Programm berechnet die Multiplikation $x_1 \cdot x_2$.

LOOP-Programme – Beispiele

Beispiel 3

Seien P_1 und P_2 LOOP-Programme, in denen x_1, x_2 und x_3 nicht vorkommen.

$x_2 := x_2 + 1;$
LOOP x_1 DO $x_2 := x_3$; $x_3 := x_3 + 1$ END;
LOOP x_2 DO P_1 END;
LOOP x_3 DO P_2 END

Dieses Programm entspricht:
IF $x_1 = 0$ THEN P_1 ELSE P_2 END

LOOP-berechenbare Funktionen

Beobachtung

Alle LOOP Programme halten bei jeder Eingabe an, weil sie keine Endlosschleifen enthalten können.

LOOP Programme berechnen also immer totale Funktionen.

Die durch LOOP-Programme berechenbaren (totalen) Funktionen bezeichnen wir als LOOP-berechenbar.

Anmerkung

LOOP-berechenbare Funktionen entsprechen genau den sogenannten primitiv rekursiven Funktionen. Das sind (vereinfacht ausgedrückt) Funktionen, deren Wert an der Stelle n rekursiv aus den Werten an Stellen $k < n$ definiert wird.

Die Turmfunktion

Die Turmfunktion $T : \mathbb{N} \to \mathbb{N}$ ist rekursive definiert durch

$$T(0) := 1, \qquad T(n+1) := 2^{T(n)}.$$

Also

$$T(n) = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \Big\} n \text{ mal}$$

Einige Werte:

n	0	1	2	3	4	5
$T(n)$	1	2	4	16	65536	2^{65536}

Lemma

Die Turmfunktion ist LOOP-berechenbar.

Ein Programm für die Turmfunktion

- Ein Programm P_1 für die Funktion $f_1(x) = 2x$:
  LOOP x_1 DO $x_0 := x_0 + 1; x_0 := x_0 + 1$ END

- Ein Programm P_2 für die Funktion $f_2(x) = 2^x$:
  $x_0 := x_0 + 1;$
  LOOP x_1 DO
  $x_0 := 2x_0$
  END
  LOOP x_1 DO
  $x_2 := x_3;$
  LOOP x_0 DO $x_2 := x_2 + 1; x_2 := x_2 + 1$ END;
  $x_0 := x_2$
  END
- Ein Programm P_3 für die Turmfunktion:
  $x_0 := x_0 + 1$
  LOOP x_1 DO
  $x_0 := 2^{x_0}$
  END

Laufzeit von LOOP-Programmen

LOOP-Programme halten immer an, weil sie keine Endlosschleifen enthalten. Schleifen können aber trotzdem s e h r l a n g e laufen.

Beispiel

$x_2 := T(x_1)$;
LOOP x_2 DO P END

Ist LOOP Turing-mächtig?

Vermutung von Hilbert (1926):

Die Klasse der primitiv-rekursiven Funktionen (also LOOP-berechenbaren Funktionen) stimmt mit der Klasse der berechenbaren (totalen) Funktionen überein.

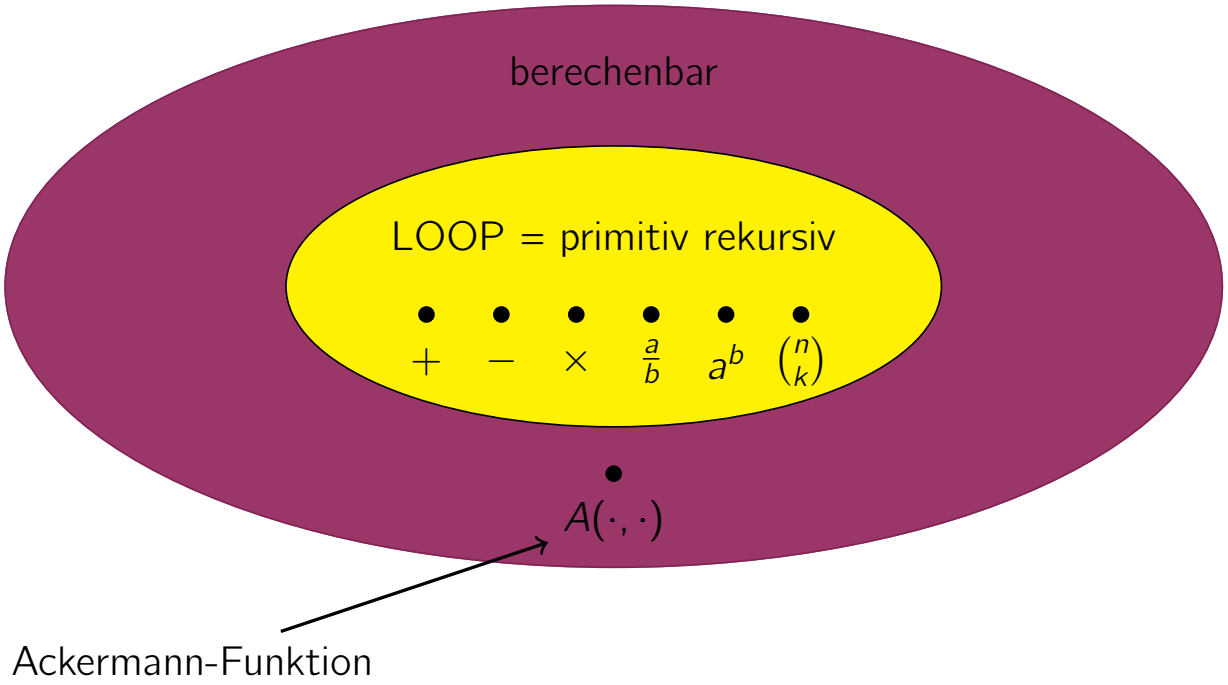
Hilbert hatte allerdings noch keinen formalen Begriff von Berechenbarkeit, sondern bezog sich auf einen intuitiven Begriff von berechenbaren (oder rekursiven) Funktionen.

Ackermann (1929)

Hilberts Vermutung ist falsch!

Ackermann gab eine Funktion an, die intuitiv klar berechenbar ist, und bewies, dass sie nicht primitiv rekursiv ist.

Berechenbare Totale Funktionen



Die Ackermann-Funktion – Definition

Definition

Die Ackermannfunktion $A\colon \mathbb{N}^2 \to \mathbb{N}$ ist folgendermaßen definert:

$$\begin{array}{lcll} A(0,n) & = & n+1 & \text{für } n \geq 0 \\ A(m+1,0) & = & A(m,1) & \text{für } m \geq 0 \\ A(m+1,n+1) & = & A(m,A(m+1,n)) & \text{für } m,n \geq 0 \end{array}$$

Einige Werte der Ackermann-Funktion

$m\backslash n$	0	1	2	3	4	5	6
0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	3	5	7	9	11	13	15
3	5	13	29	61	125	253	509
4	13	65 533	$2^{65536}-3$				

Es ist klar, dass die Ackermann-Funktion berechenbar ist.

Wachstum für feste m

Lemma

Für alle $n \in \mathbb{N}$ gilt:

- $A(0, n) = n + 1$,
- $A(1, n) = n + 2$,
- $A(2, n) = 2 \cdot (n + 3) - 3$,
- $A(3, n) = 2^{n+3} - 3$,
- $A(4, n) = T(n + 3) - 3$,

(Ohne Beweis)

Bereits $A(4, 2) = 2^{65536} - 3$ ist größer als die (vermutete) Anzahl der Atome im Weltraum.

Monotonieeigenschaften der Ackermannfunktion

Strikte Monotonie in der zweiten Komponente

$A(m, n + 1) > A(m, n)$ für alle $m, n \in \mathbb{N}$ (Übungsaufgabe)

Diagonale Monotonie

$A(m + 1, n) \geq A(m, n + 1)$ für alle $m, n \in \mathbb{N}$

Beweis der diagonalen Monotonie

Wir beweisen zunächst $A(m,n) \geq n+1$ für alle $m,n \in \mathbb{N}$ per Induktion über m.

Induktionsanfang: $m = 0$

$$A(0,n) = n+1.$$

Induktionsschritt: $m \to m+1$

Wir beweisen $A(m+1,n) \geq n+1$ per Induktion über n.

Induktionsanfang: $n = 0$

$$A(m+1,0) = A(m,1) \geq 2$$

nach IA(m) (Induktionsannahme der Induktion über m).

Induktionsschritt: $n \to n+1$

$$\begin{aligned} A(m+1,n+1) &= A(m,A(m+1,n)) && \text{Definition } A \\ &\geq A(m+1,n)+1 && \text{IA}(m) \\ &\geq n+1+1 && \text{IA}(n) \\ &= n+2. \end{aligned}$$

Beweis der diagonalen Monotonie (Forts.)

Wir beweisen $A(m+1,n) \geq A(m,n+1)$ per Induktion über n.

Induktionsanfang: $n = 0$

$A(m+1,0) = A(m,1)$ gilt nach Definition von A.

Induktionsschritt: $n \to n+1$

Nach der Induktionsannahme gilt $A(m+1,n) \geq A(m,n+1)$.

Wir haben bereits $A(m,n+1) \geq n+2$ bewiesen.

Also gilt $A(m+1,n) \geq n+2$ und damit

$$\begin{aligned} A(m+1,n+1) &= A(m,A(m+1,n)) && \text{Def. } A \\ &\geq A(m,n+2) && \text{Mon. in 2. Komp.} \end{aligned}$$

Monotonie

Strikte Monotonie in der zweiten Komponente

$A(m, n+1) > A(m, n)$ für alle $m, n \in \mathbb{N}$ (Übungsaufgabe)

Diagonale Monotonie

$A(m+1, n) \geq A(m, n+1)$ für alle $m, n \in \mathbb{N}$

Strikte Monotonie in der ersten Komponente

$A(m+1, n) > A(m, n)$ für alle $m, n \in \mathbb{N}$

Beweis:

$$A(m+1, n) \geq A(m, n+1) > A(m, n).$$

Wachstum der Variableninhalte in einem LOOP-Programm

Definition der Funktion F_P

- Sei P ein LOOP-Programm
- Seien $x_0, x_1, \ldots, x_k$ die Variablen in P.
- Wenn die Variablen initial die Werte $a = (a_0, \ldots, a_k) \in \mathbb{N}^{k+1}$ haben, dann sei $f_P(a)$ das $(k+1)$-Tupel der Variablenwerte nach Ausführung von P.
- Sei $|f_P(a)|$ die Summe der Einträge im $(k+1)$-Tupel $f_P(a)$.
- Wir definieren nun die Funktion $F_P \colon \mathbb{N} \to \mathbb{N}$ durch

$$F_P(n) = \max \left\{ |f_P(a)| \;\middle|\; a \in \mathbb{N}^{k+1} \text{ mit } \sum_{i=0}^{k} a_i \leq n \right\}.$$

Intuitiv beschreibt die Funktion F_P das maximale Wachstum der Variablenwerte im LOOP-Programm P.

Ackermannfunktion versus F_P

Wir zeigen nun, dass $F_P(n)$ für alle $n \in \mathbb{N}$ echt kleiner ist als $A(m, n)$, wenn der Parameter m genügend groß in Abhängigkeit von P gewählt wird.

Lemma

Für jedes LOOP-Programm P gibt es eine natürliche Zahl m, so dass für alle $n \in \mathbb{N}$ gilt: $F_P(n) < A(m, n)$.

Beachte: Für ein festes Programm P ist der Parameter m eine Konstante.

Beweis durch strukturelle Induktion (Überblick)

Induktionsanfang

- Sei P von der Form $x_i := x_j + c$ für $c \in \{-1, 0, 1\}$.
- Wir werden zeigen: $F_P(n) < A(2, n)$.

Induktionsschritt (1. Art)

- Sei P von der Form $P_1; P_2$.
- Induktionsannahme: $\exists q \in \mathbb{N} : F_{P_1}(\ell) < A(q, \ell)$ und $F_{P_2}(\ell) < A(q, \ell)$.
- Wir werden zeigen: $F_P(n) < A(q + 1, n)$.

Induktionsschritt (2. Art)

- Sei P von der Form LOOP x_i DO Q END.
- Induktionsannahme: $\exists q \in \mathbb{N} : F_Q(\ell) < A(q, \ell)$.
- Wir werden zeigen: $F_P(n) < A(q + 1, n)$.

Beweis des Lemmas

Induktionsanfang

- Sei P von der Form $x_i := x_j + c$ für $c \in \{-1, 0, 1\}$.
- Dann gilt $F_P(n) \leq 2n + 1$.
- Somit folgt $F_P(n) < A(2, n)$.

Erläuterung:

- Vor Ausführung von P könnte gelten $x_j = n$ und alle anderen Variablen haben den Wert 0.
- Ferner könnte c den Wert 1 haben.
- Nach Ausführung von P gilt somit $x_i = n + 1$ und somit ist die Summe der Variableninhalte $x_i + x_j = 2n + 1$.
- Ein größeres Wachstum der Variableninhalte ist nicht möglich.

Formal:

$$\sum_{t\in\{0,\ldots,k\}} x'_t = x'_i + \sum_{t\in\{0,i-1,i+1,k\}} x'_t \leq x_j + 1 + \sum_{t\in\{0,i-1,i+1,k\}} x_t \leq n+1+n,$$

wobei x'_t der Wert der Variable x_t nach Ausführung des Programms P sei.

Beweis des Lemmas

Induktionsschritt (1. Art)

- Sei P von der Form $P_1; P_2$.
- Induktionsannahme: $\exists q \in \mathbb{N} : F_{P_1}(\ell) < A(q, \ell)$ und $F_{P_2}(\ell) < A(q, \ell)$.
- Somit gilt

$$\begin{aligned} F_P(n) &\leq F_{P_2}(F_{P_1}(n)) \\ &< A(q, A(q, n)) && \text{IA und Monotonie 2. Komp.} \\ &\leq A(q, A(q+1, n-1)) && \text{Diagonalmon. und Mon. 2. Komp.} \\ &= A(q+1, n). \end{aligned}$$

Beweis des Lemmas

Induktionsschritt (2. Art)

- Sei P von der Form LOOP x_i DO Q END.
- Induktionsannahme: $\exists q \in \mathbb{N} : F_Q(\ell) < A(q, \ell)$.
- Sei $\alpha = \alpha(n)$ derjenige Wert aus $\{0, 1, \ldots, n\}$ für x_i, der $F_P(n)$ maximiert.
- Dann gilt

$$F_P(n) \;\leq\; F_Q(F_Q(\ldots F_Q(F_Q(n-\alpha))\ldots)) + \alpha,$$

wobei die Funktion $F_Q(\cdot)$ hier α-fach ineinander eingesetzt ist.

Beweis des Lemmas

Induktionsschritt (2. Art) – Fortsetzung

- Bisher haben wir gezeigt, dass

$$F_P(n) \;\leq\; \underbrace{F_Q(F_Q(\ldots F_Q(F_Q(n-\alpha))\ldots))}_{\alpha\text{-fach verschachtelt}} + \alpha.$$

- Aus der Induktionsannahme folgt $F_Q(\ell) < A(q, \ell)$.
- Dies wenden wir auf die äußerste Funktion F_Q an und erhalten

$$F_P(n) \;<\; A(q, \underbrace{F_Q(\ldots F_Q(F_Q(n-\alpha))\ldots)}_{(\alpha-1)\text{-fach verschachtelt}}) + \alpha,$$

also

$$F_P(n) \;\leq\; A(q, \underbrace{F_Q(\ldots F_Q(F_Q(n-\alpha))\ldots)}_{(\alpha-1)\text{-fach verschachtelt}}) + \alpha - 1,$$

Beweis des Lemmas

Induktionsschritt (2. Art) – Fortsetzung

- Bisher haben wir gezeigt, dass

$$F_P(n) \leq A(q, \underbrace{F_Q(\ldots F_Q(F_Q(n-\alpha))\ldots)}_{(\alpha-1)\text{-fach verschachtelt}}) + \alpha - 1,$$

- Erneute Anwendung von $F_Q(\ell) < A(q,\ell)$ und Monotonie ergibt

$$F_P(n) < A(q, A(q, \underbrace{F_Q(\ldots F_Q(F_Q(n-\alpha))\ldots)}_{(\alpha-2)\text{-fach verschachtelt}})) + \alpha - 1,$$

  also $F_P(n) \leq A(q, A(q, \underbrace{F_Q(\ldots F_Q(F_Q(n-\alpha))\ldots)}_{(\alpha-2)\text{-fach verschachtelt}}))) + \alpha - 2,$

- Wir wiederholen das Argument und erhalten für $0 \leq i \leq \alpha$

$$F_P(n) \leq \underbrace{A(q, A(q, \ldots}_{i\text{-mal verschachtelt}}, \underbrace{F_Q(\ldots F_Q(F_Q(n-\alpha))\ldots)}_{(\alpha-i)\text{-fach verschachtelt}}))) + \alpha - i,$$

Beweis des Lemmas

Induktionsschritt (2. Art) – Fortsetzung

- Für $i = \alpha$ ergibt sich

$$F_P(n) \leq \underbrace{A(q, A(q, \ldots A(q, A(q, n-\alpha))\ldots))}_{\alpha\text{-fach verschachtelt}}$$

  Mit Hilfe der Monotonie erhält man

$$F_P(n) \leq \underbrace{A(q, A(q, \ldots A(q, A(q+1, n-\alpha))\ldots))}_{\alpha\text{-fach verschachtelt}}$$

- Der Definition der Ackermannfunktion angewandt auf die innere Verschachtelung ergibt

$$F_P(n) \leq \underbrace{A(q, A(q, \ldots A(q+1, n-\alpha+1)\ldots))}_{(\alpha-1)\text{-fach verschachtelt}}.$$

- Nach $\alpha - 2$ weiteren Anwendungen folgt $F_P(n) \leq A(q+1, n-1)$.

Beweis des Lemmas

Induktionsschritt (2. Art) – Fortsetzung

- Bisher haben wir gezeigt:

$$F_P(n) \leq A(q+1, n-1)$$

- Die Monotonie im zweiten Argument ergibt dann

$$F_P(n) < A(q+1, n)$$

□

Nicht-LOOP-Berechenbarkeit der Ackermannfunktion

Satz

Die Ackermannfunktion ist nicht LOOP-berechenbar.

Beweis:

- Angenommen, es gibt ein LOOP-Programm, das die Ackermannfunktion berechnet.
- Dann gibt es auch ein LOOP-Programm, das die Funktion $B(n) = A(n, n)$ berechnet. Sei P dieses LOOP-Programm.
- Aus dem Lemma über LOOP-Programme folgt: es gibt $m \in \mathbb{N}$, so dass für jedes $n \in \mathbb{N}$ gilt: $F_P(n) < A(m, n)$.
- Wenn P mit Eingabe m aufgerufen wird, so berechnet P den Funktionswert $B(m)$. Somit gilt $B(m) \leq F_P(m)$.
- Es folgt

$$B(m) \leq F_P(m) < A(m, m) \overset{\text{Def. von } B}{=} B(m).$$

- Widerspruch! Also folgt der Satz. □

Schlussfolgerung

Da die Ackermannfunktion (durch eine TM) berechenbar ist, folgt:

Korollar

Die Klasse der LOOP-berechenbaren Funktionen ist eine echte Teilmenge der berechenbaren (totalen) Funktionen.

Zur Klärung

Technisch beschränken wir uns in diesem Kapitel auf Funktionen $f : \mathbb{N}^k \to \mathbb{N}$, $k \in \mathbb{N}$. Obige Aussage gilt auch für Funktionen der Form $f : \Sigma^* \to \Sigma^*$ über einem beliebigen endlichen Alphabet Σ.

LOOP-entscheidbare Mengen

Eine Menge $L \subseteq \mathbb{N}$ ist LOOP-entscheidbar, wenn ihre charakteristische Funktion $\chi_L : \mathbb{N} \to \{0, 1\}$, definiert durch

$$\chi_L(x) = \begin{cases} 1 & \text{fall } x \in L, \\ 0 & \text{sonst,} \end{cases}$$

LOOP-berechenbar ist.

Theorem

Es gibt eine entscheidbare Menge, die nicht LOOP-entscheidbar ist.

Der Beweis lässt sich mittels eines Diagonalisierungsarguments führen.

Zur Klärung

- Eine Menge $L \subseteq \mathbb{N}$ nennen wir entscheidbar, wenn die Menge $\{\mathsf{bin}(n) \mid n \in L\} \subseteq \{0, 1\}^*$ entscheidbar ist.
- Umgekehrt können wir den Begriff der LOOP-Entscheidbarkeit auch auf Sprachen $L \subseteq \Sigma^*$ über beliebigen endlichen Alphabeten Σ erweitern.

Zusammenfassung – Berechenbarkeit

Wir haben die folgenden Turing-mächtigen Rechenmodelle und Programmiersprachen kennen gelernt.

- Turingmaschine (TM)
- k-Band-TM
- Registermaschine (RAM)
- eingeschränkte RAM
- WHILE-Programme (und somit C, Java, Pascal, Postscript, etc.)

LOOP-Programme sind hingegen nicht Turing-mächtig.

Zusammenfassung – Berechenbarkeit

Church-Turing-These

Die Klasse der TM-berechenbaren Funktionen stimmt mit der Klasse der „intuitiv berechenbaren“ Funktionen überein.

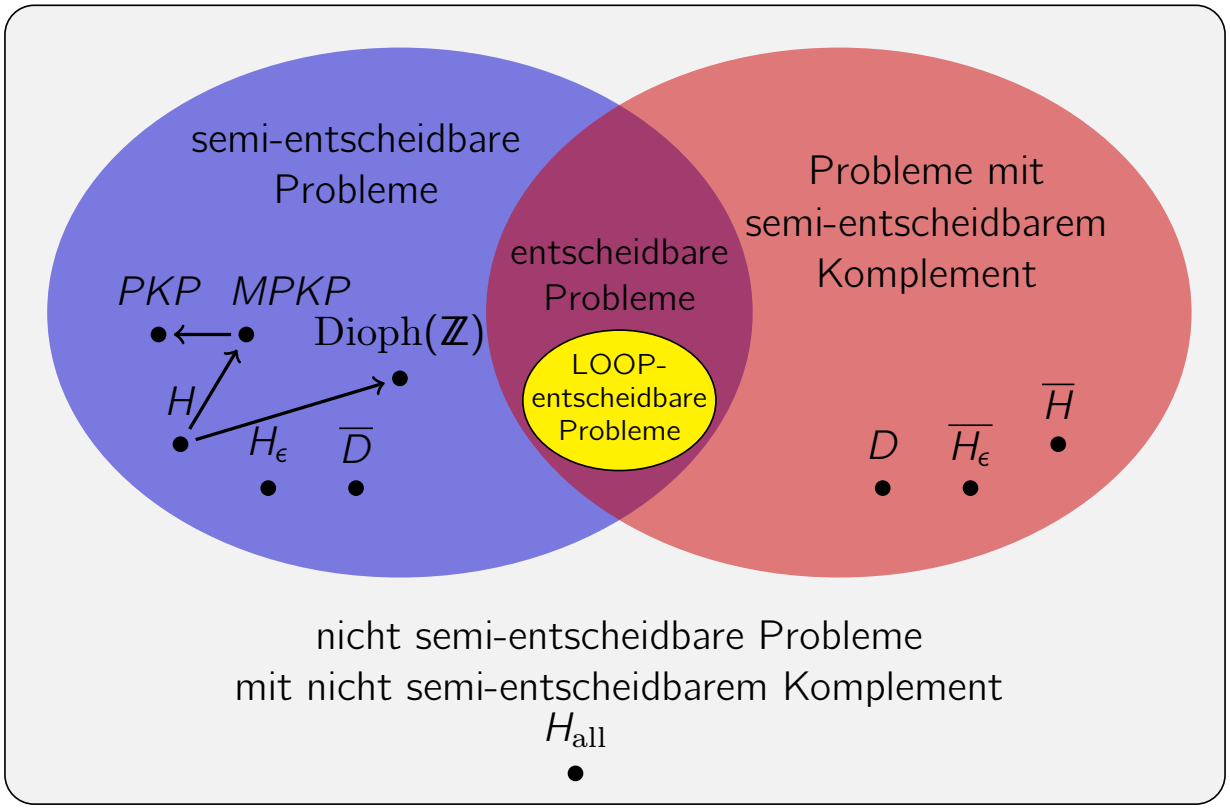
In anderen Worten:

Ein Problem kann genau dann „algorithmisch gelöst werden“, wenn es eine TM für dieses Problem gibt.

An Stelle des Begriffs „TM“ können wir auch jedes andere Turing-mächtige Rechenmodell verwenden.

Zusammenfassung – Berechenbarkeit

Berechenbarkeitslandschaft:



Zusammenfassung – Berechenbarkeit

Bedeutende nicht berechenbare Probleme:

- Halteproblem, in verschiedenen Varianten
- *Satz von Rice:* Aussagen über Eigenschaften von Funktionen, die durch eine gegebene TM berechnet werden, sind nicht entscheidbar
- *Schlussfolgerung:* Die automatische Verifikation von Programmen in einer TM-mächtigen Programmiersprachen ist nicht möglich
- Hilberts 10. Problem
- Postsches Korrespondenzproblem

Zusammenfassung – Berechenbarkeit

Methoden zum Nachweis von Nicht-Berechenbarkeit:

- Diagonalisierung
- Unterprogrammtechnik
- Satz von Rice
- Reduktionen (spezielle Variante der Unterprogrammtechnik)

Teil III
Komplexität

Vorlesung 13
Die Komplexitätsklassen P und NP

Whd.: Die Ackermann-Funktion

Definition

Die Ackermannfunktion $A\colon \mathbb{N}^2 \to \mathbb{N}$ ist folgendermaßen definert:

$$\begin{array}{lcll} A(0,n) & = & n+1 & \text{für } n \geq 0 \\ A(m+1,0) & = & A(m,1) & \text{für } m \geq 0 \\ A(m+1,n+1) & = & A(m,A(m+1,n)) & \text{für } m,n \geq 0 \end{array}$$

Wdh.: LOOP vs WHILE

Lemma

Für jedes LOOP-Programm P gibt es eine natürliche Zahl m, so dass für alle $n \in \mathbb{N}$ gilt: $F_P(n) < A(m,n)$.

Satz

Die Ackermannfunktion ist nicht LOOP-berechenbar.

Korollar

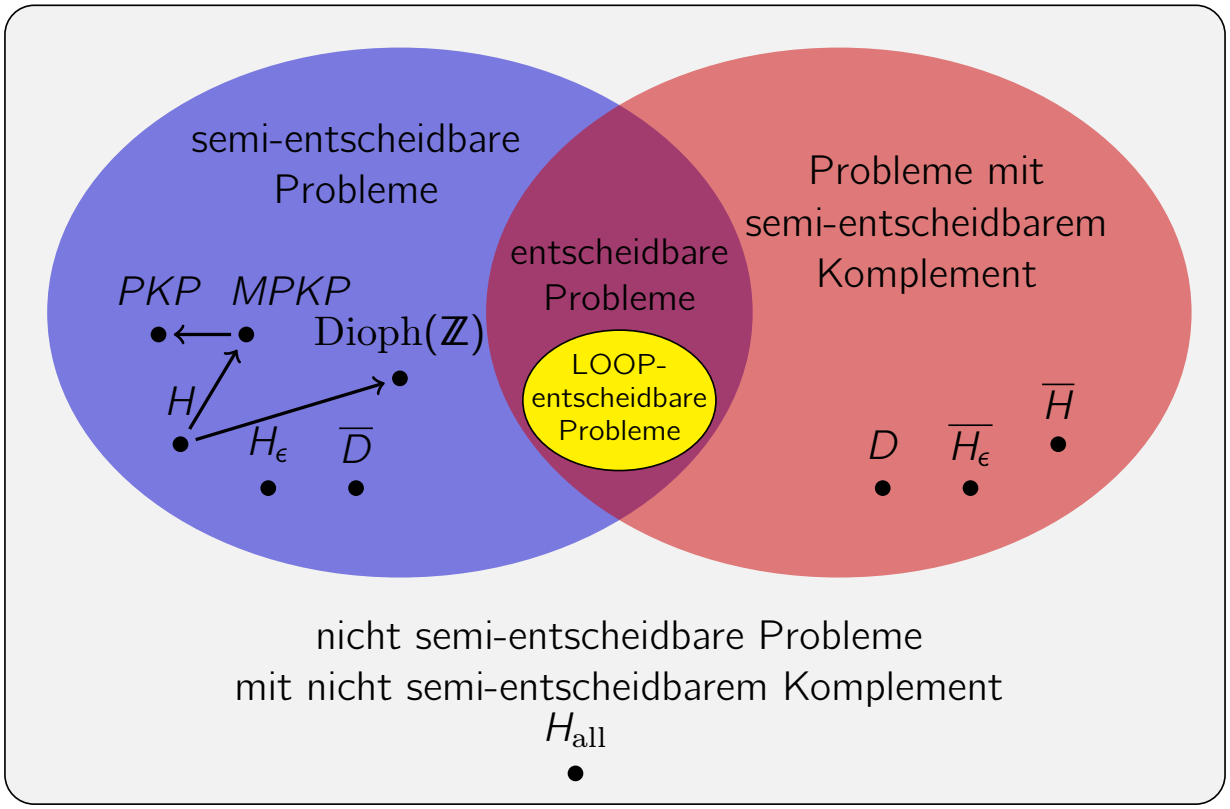
Die Klasse der LOOP-berechenbaren Funktionen ist eine echte Teilmenge der berechenbaren (totalen) Funktionen.

Bemerkung

Mit Hilfe eines Diagonalisierungsarguments lässt sich auch beweisen, dass es entscheidbare Sprachen gibt, die nicht LOOP-entscheidbar sind.

Wdh.: Berechenbarkeit

Berechenbarkeitslandschaft:



Wdh.: Kostenmodelle der RAM

Modelle für die Rechenzeit einer RAM

- Uniformes Kostenmaß: Jeder Schritt zählt eine Zeiteinheit.
- Logarithmisches Kostenmaß: Die Laufzeitkosten eines Schrittes sind proportional zur binären Länge der Zahlen in den angesprochenen Registern.

Definition von Polynomialzeitalgorithmus

Definition (worst case Laufzeit eines Algorithmus)

Die worst case Laufzeit $t_A(n)$, $n \in \mathbb{N}$, eines Algorithmus A entspricht den maximalen Laufzeitkosten auf Eingaben der Länge n bezüglich des logarithmischen Kostenmaßes der RAM.

Definition (Polynomialzeitalgorithmus)

Wir sagen, die worst case Laufzeit $t_A(n)$ eines Algorithmus A ist polynomiell beschränkt, falls gilt

$$\exists \alpha \in \mathbb{N} : t_A(n) = O(n^\alpha).$$

Einen Algorithmus mit polynomiell beschränkter worst case Laufzeit bezeichnen wir als Polynomialzeitalgorithmus.

Sortieren in Polynomialzeit

Problem (Sortieren)

Eingabe: N Zahlen $a_1, \ldots, a_N \in \mathbb{N}$
Ausgabe: aufsteigend sortierte Folge der Eingabezahlen

Anmerkung: Soweit wir nichts anderes sagen, nehmen wir an, dass Zahlen binär kodiert sind.

Sortieren in Polynomialzeit

Satz

Sortieren kann in Polynomialzeit gelöst werden.

Beweis:

- Wir lösen das Problem beispielsweise mit Mergesort.
- Laufzeit im uniformen Kostenmaß: $O(N \log N)$.
- Laufzeit im logarithmischen Kostenmaß: $O(\ell N \log N)$, wobei $\ell = \max_{1 \leq i \leq N} \log(a_i)$.
- Sei n die Eingabelänge. Es gilt $\ell \leq n$ und $\log N \leq N \leq n$.
- Somit ist die Laufzeit durch $\ell N \log N \leq n^3$ beschränkt.

□

Bemerkung: Sortieren ist kein Entscheidungsproblem.

Definition der Klasse P

Definition (Komplexitätsklasse P)

P ist die Klasse der Entscheidungsprobleme, für die es einen Polynomialzeitalgorithmus gibt.

Anmerkungen:

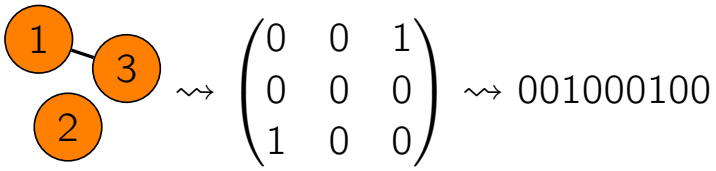
- Alternativ kann man sich auch auf die Laufzeit einer TM beziehen, da sich RAM und TM gegenseitig mit polynomiellem Zeitverlust simulieren können.
- Polynomialzeitalgorithmen werden häufig auch als „effiziente Algorithmen“ bezeichnet.
- P ist in diesem Sinne die Klasse derjenigen Probleme, die effizient gelöst werden können.

Graphzusammenhang in P

Problem (Graphzusammenhang)

Eingabe: Graph $G = (V, E)$
Frage: Ist G zusammenhängend?

Anmerkung: Bei Graphproblemen gehen wir grundsätzlich davon aus, dass der Graph in Form einer Adjazenzmatrix eingegeben wird.



Graphzusammenhang in P

Satz

Graphzusammenhang $\in$ P.

Beweis

- Wir lösen das Problem mit einer Tiefensuche.
- Laufzeit im uniformen Kostenmaß: $O(|V|^2)$,
  (sogar $O(|V| + |E|)$ bei Adjazenzlistenrepräsentation),
- Laufzeit im logarithmischen Kostenmaß: $O(|V|^2 \cdot \log |V|)$
- Die Eingabelänge ist $n = |V|^2$.
- Die Gesamtlaufzeit ist somit

$$O(|V|^2 \log |V|) \;=\; O(n \log n) \;=\; O(n^2) \;.$$

□

Weitere Beispiele für polynomiell lösbare Probleme

- Kürzester Weg
- Minimaler Spannbaum
- Maximaler Fluss
- Maximum Matching
- Lineare Programmierung
- Größter Gemeinsamer Teiler
- Primzahltest

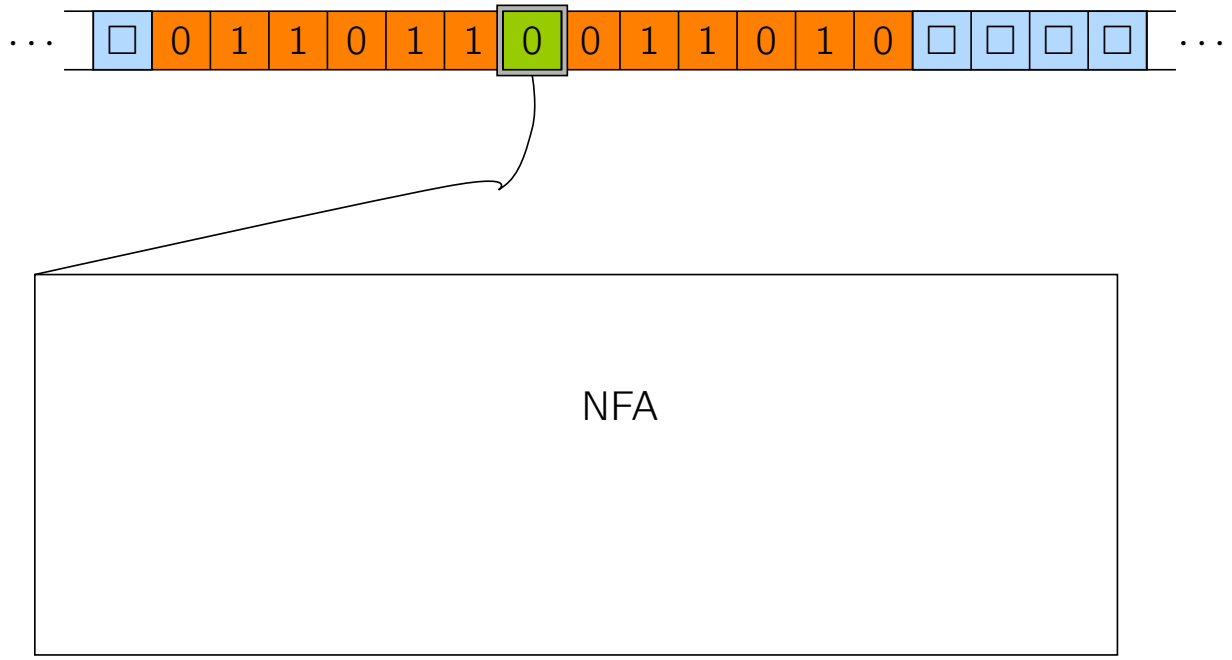
Definition von NTM

Definition (Nichtdeterministische Turingmaschine – NTM)

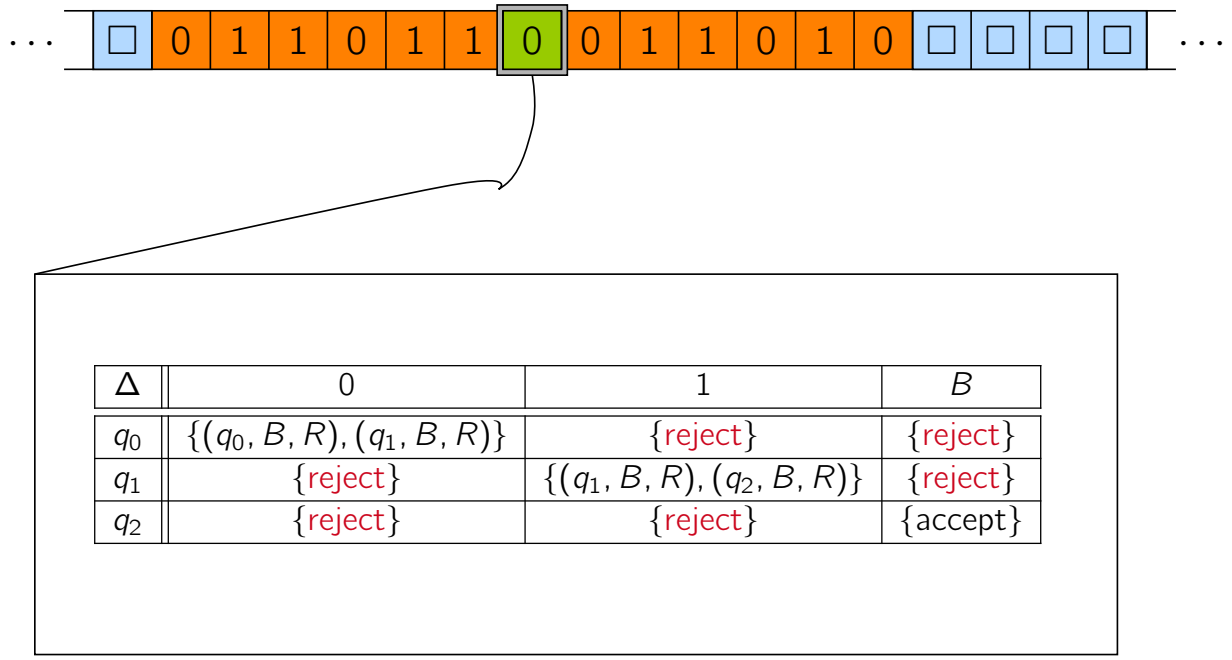
Eine nichtdeterministische Turingmaschine (NTM) ist definiert wie eine deterministische Turingmaschine (TM), nur die Zustandsübergangsfunktion wird zu einer Relation

$$\Delta \subseteq ((Q \setminus \{\bar{q}\}) \times \Gamma) \times (Q \times \Gamma \times \{L, R, N\}) \quad .$$

Nichtdeterministische Turingmaschine (NTM)



Nichtdeterministische Turingmaschine (NTM)



Δ	0	1	B
q_0	$\{(q_0, B, R), (q_1, B, R)\}$	{reject}	{reject}
q_1	{reject}	$\{(q_1, B, R), (q_2, B, R)\}$	{reject}
q_2	{reject}	{reject}	{accept}

Erläuterung der Rechnung einer NTM

- Eine Konfiguration K' ist direkter Nachfolger einer Konfiguration K, falls K' durch einen der in Δ beschriebenen Übergänge aus K hervorgeht.
- Rechenweg = Konfigurationsfolge, die mit Startkonfiguration beginnt und mit Nachfolgekonfigurationen fortgesetzt wird, bis eine Endkonfiguration im Zustand $\bar{q}$ erreicht wird.
- Der Verlauf der Rechnung ist also nicht deterministisch, d.h., zu einer Konfiguration kann es mehrere direkte Nachfolgekonfigurationen geben.

Erläuterung der Rechnung einer NTM

Die möglichen Rechenwege von M auf einer Eingabe $w \in \Sigma^*$ können in Form eines Berechnungsbaumes beschrieben werden:

- Die Knoten des Baumes entsprechen Konfigurationen.
- Die Wurzel des Baumes entspricht der Startkonfiguration.
- Die Kinder einer Konfiguration entsprechen den möglichen Nachfolgekonfigurationen.

Der maximale Verzweigungsgrad des Berechnungsbaumes ist

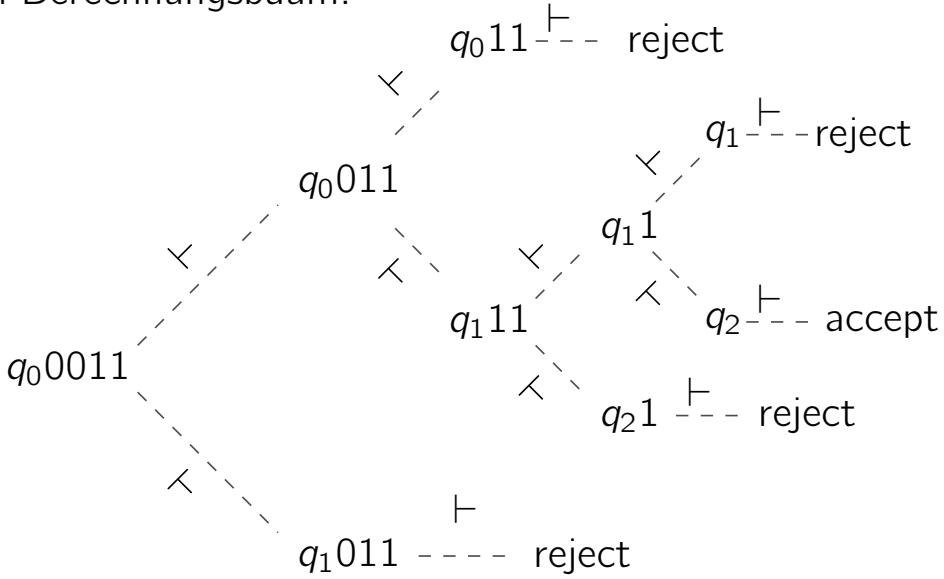
$$d = \max\left\{|\Delta(q,a)| \,\middle|\, q \in Q \setminus \{\bar{q}\}, a \in \Gamma\right\}.$$

Beachte, dass d nicht von der Eingabe abhängt, also konstant ist.

Berechnungsbaum - Beispiel

δ	0	1	B
q_0	$\{(q_0, B, R), (q_1, B, R)\}$	{reject}	{reject}
q_1	{reject}	$\{(q_1, B, R), (q_2, B, R)\}$	{reject}
q_2	{reject}	{reject}	{accept}

Die NTM mit dieser Übergangsrelation hat auf der Eingabe 0011 folgenden Berechnungsbaum:



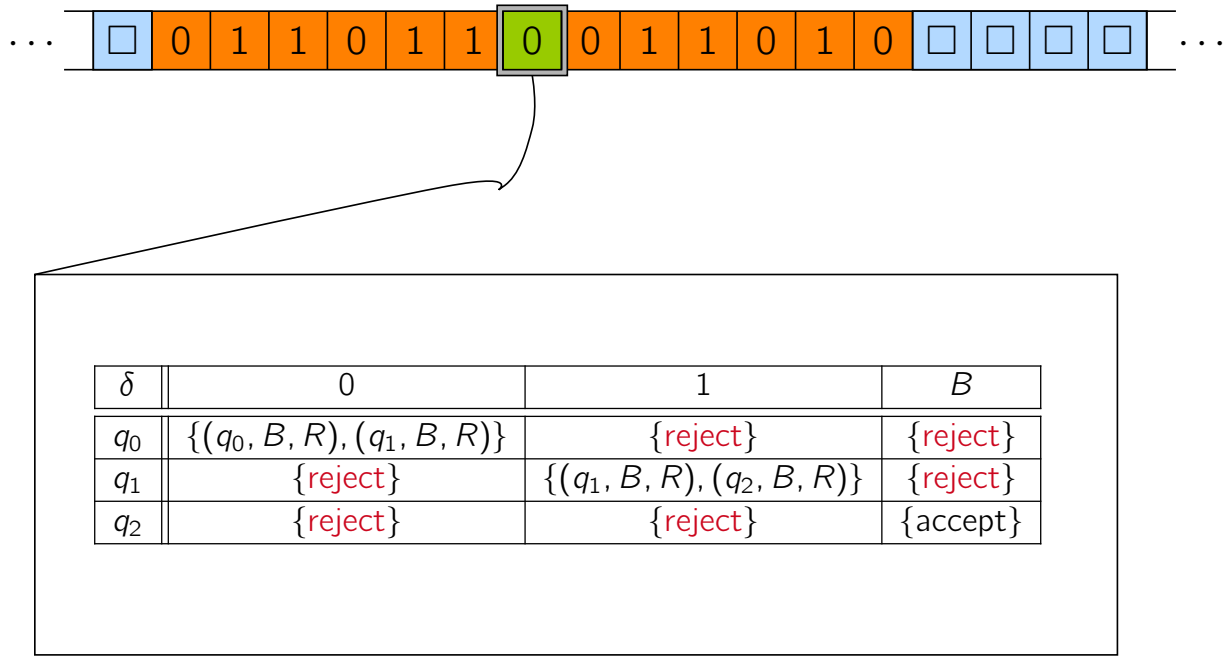
Definition des Akzeptanzverhaltens

Definition (Akzeptanzverhalten der NTM)

Eine NTM M akzeptiert die Eingabe $x \in \Sigma^*$, falls es mindestens einen Rechenweg von M gibt, der in eine Konfiguration mit akzeptierendem Zustand führt.

Die von M erkannte Sprache $L(M)$ besteht aus allen von M akzeptierten Wörtern.

Nichtdeterministische Turingmaschine (NTM)

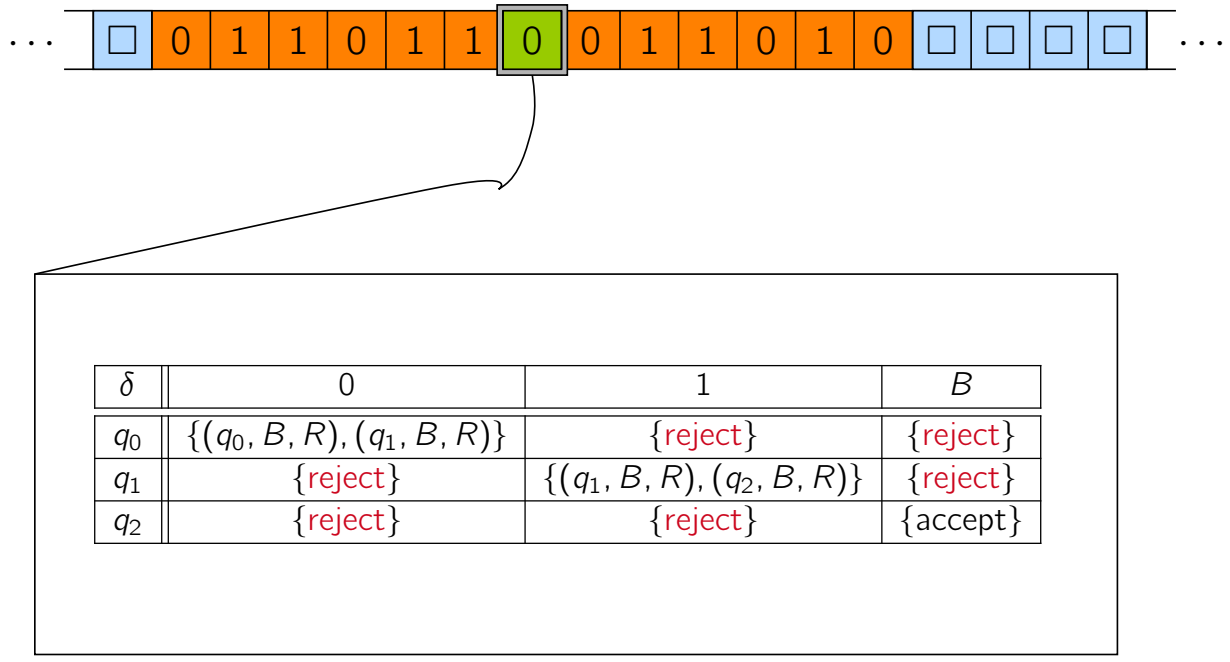


δ	0	1	B
q_0	$\{(q_0, B, R), (q_1, B, R)\}$	{reject}	{reject}
q_1	{reject}	$\{(q_1, B, R), (q_2, B, R)\}$	{reject}
q_2	{reject}	{reject}	{accept}

Welche Sprache wird von dieser NTM erkannt?

$$L(M) = \{0^i 1^j \mid i \geq 1, j \geq 1\}$$

Nichtdeterministische Turingmaschine (NTM)



δ	0	1	B
q_0	$\{(q_0, B, R), (q_1, B, R)\}$	{reject}	{reject}
q_1	{reject}	$\{(q_1, B, R), (q_2, B, R)\}$	{reject}
q_2	{reject}	{reject}	{accept}

Welche Sprache wird von dieser NTM erkannt?

$$L(M) = \{0^i 1^j \mid i \geq 1, j \geq 1\}$$

Definition der Laufzeit

Definition (Laufzeit der NTM)

Sei M eine NTM. Die Laufzeit von M auf einer Eingabe $x \in L(M)$ ist definiert als

$$T_M(x) := \text{Länge des kürzesten akzeptierenden Rechenweges von } M \text{ auf } x.$$

Für $x \notin L(M)$ definieren wir $T_M(x) = 0$.

Die worst case Laufzeit $t_M(n)$ für M auf Eingaben der Länge $n \in \mathbb{N}$ ist definiert als

$$t_M(n) := \max\{ T_M(x) \mid x \in \Sigma^n \}.$$

Definition der Klasse NP

Definition (Komplexitätsklasse NP)

NP ist die Klasse der Entscheidungsprobleme, die durch eine NTM M erkannt werden, deren worst case Laufzeit $t_M(n)$ polynomiell beschränkt ist.

NP steht dabei für nichtdeterministisch polynomiell.

Beispiel für ein Problem aus NP

Problem (Cliquenproblem – CLIQUE)

Eingabe: Graph $G = (V, E)$, $k \in \{1, \ldots, |V|\}$
Frage: Enthält G eine k-Clique?

- Für das Cliquenproblem ist kein Polynomialzeitalgorithmus bekannt.
- Die besten bekannten Algorithmen haben exponentielle Laufzeit.

Beispiel für ein Problem aus NP

Satz

$CLIQUE \in$ NP.

Beweis: Wir beschreiben eine NTM M mit $L(M) =$ CLIQUE:

1. Syntaktisch inkorrekte Eingaben werden verworfen.
2. M „rät“ einen 0-1-String y der Länge $|V|$.
3. M akzeptiert, falls
   - der String y genau k viele Einsen enthält und
   - die Knotenmenge $C = \{i \in V \mid y_i = 1\}$ eine Clique ist.

Korrektheit: Es gibt genau dann einen akzeptierenden Rechenweg, wenn G eine k-Clique enthält.

Laufzeit: Alle Schritte haben polynomielle Laufzeit. □

Die Komplexitätsklasse EXPTIME

Definition (Komplexitätsklasse EXPTIME)

EXPTIME ist die Klasse der Entscheidungsprobleme L, für die es ein Polynom q gibt, so dass sich L auf einer DTM mit Laufzeitschranke $2^{q(n)}$ berechnen lässt.

Wie verhält sich NP zu P und EXPTIME?

Wie verhält sich NP zu P und EXPTIME?

Wir setzen die Klassen P und EXPTIME mit der Klasse NP in Beziehung.

Satz

$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{EXPTIME}$

Beweis:

Es gilt $\mathrm{P} \subseteq \mathrm{NP}$, weil eine DTM als eine spezielle NTM aufgefasst werden kann.

Wir müssen noch zeigen, dass $\mathrm{NP} \subseteq \mathrm{EXPTIME}$.

Exponentielle Laufzeitschranke für Probleme aus NP

Sei $L \in \mathrm{NP}$. Sei M eine NTM mit polynomiell beschränkter Laufzeitschranke $p(n)$, die L erkennt.

Sei $w \in \Sigma^*$. Wir konstruieren eine DTM M', welche die NTM M auf Eingabe w simuliert:

- In einer Breitensuche generiert M' den Berechnungsbaum von M bis zu einer Tiefe von $p(|w|)$.
- Falls dabei eine akzeptierende Konfiguration gefunden wird, so akzeptiert M' die Eingabe; sonst verwirft M' die Eingabe.

Exponentielle Laufzeitschranke für Probleme aus NP

Korrektheit:

- Falls $w \in L$ ist, so gibt es einen akzeptierenden Rechenweg von M der Länge $p(|w|)$. M' generiert diesen Weg und akzeptiert w.
- Falls $w \notin L$ ist, so gibt es keinen akzeptierenden Rechenweg von M der Länge $p(|w|)$. In diesem Fall wird w von M' verworfen.

Laufzeit:

Sei $d \geq 2$ der maximale Verzweigungsgrad des Berechnungsbaumes.

Die Laufzeit von M' auf w ist proportional zur Anzahl der Knoten im Berechnungsbaum bis zur Tiefe $p(|w|)$. Diese Anzahl ist beschränkt durch

$$d^{p(|w|)+1} \;=\; 2^{(p(|w|)+1)\cdot \log_2 d} \;=\; 2^{O(p(|w|))}.$$

□

Die große offene Frage der Informatik

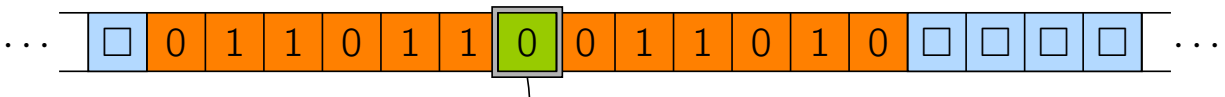
$$P = NP?$$

Diese Frage ist bedeutend, weil viele wichtige Probleme, die in der Praxis vorkommen, in NP enthalten sind, wir aber nicht wissen, ob es für diese Probleme effiziente Algorithmen (also Polynomialzeitalgorithmen) gibt.

Einige dieser Problem lernen wir in der nächsten Vorlesung kennen.

Vorlesung 14
Die Klasse NP und polynomielle Reduktionen

Wdh.: Nichtdeterministische Turingmaschine (NTM)



δ	0	1	B
q_0	$\{(q_0, B, R), (q_1, B, R)\}$	{reject}	{reject}
q_1	{reject}	$\{(q_1, B, R), (q_2, B, R)\}$	{reject}
q_2	{reject}	{reject}	{accept}

Wdh.: Komplexitätsklassen

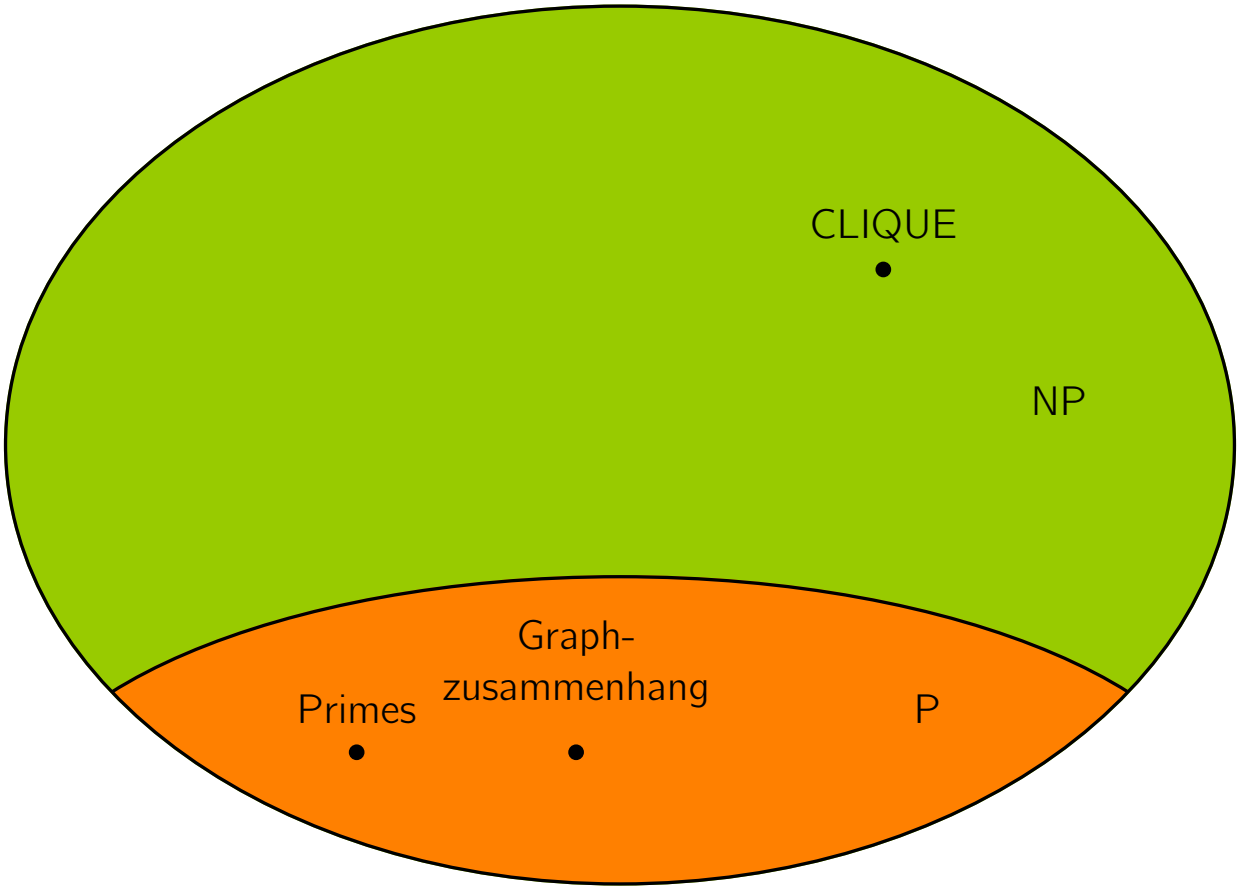
Definition (Komplexitätsklassen)

- P ist die Klasse der Entscheidungsprobleme, für die es einen Polynomialzeitalgorithmus gibt.
- NP ist die Klasse der Entscheidungsprobleme, die durch eine NTM M erkannt werden, deren worst case Laufzeit $t_M(n)$ polynomiell beschränkt ist.
- EXPTIME ist die Klasse der Entscheidungsprobleme L, für die es ein Polynom q gibt, so dass sich L auf einer DTM mit Laufzeitschranke $2^{q(n)}$ berechnen lässt.

Satz

$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{EXPTIME}$

Die Komplexitätslandschaft



Warnung: Dieser Abbildung liegt die Annahme $\mathsf{P} \neq \mathsf{NP}$ zu Grunde.

Optimierungsprobleme und ihre Entscheidungsvarianten

- Beim Rucksackproblem (KP) ist eine Menge von N Objekten mit Gewichten $w_1, \ldots, w_N$ und Nutzenwerten $p_1, \ldots, p_N$ gegeben.
- Des Weiteren ist eine Gewichtsschranke b gegeben.
- Wir suchen eine Teilmenge K der Objekte, so dass die Objekte aus K in einen Rucksack mit Gewichtsschranke b passen und dabei der Nutzen maximiert wird.

Problem (Rucksackproblem, Knapsack Problem – KP)

Eingabe: $b \in \mathbb{N}$, $w_1, \ldots, w_N \in \{1, \ldots, b\}$, $p_1, \ldots, p_N \in \mathbb{N}$

zulässige Lösungen: $K \subseteq \{1, \ldots, N\}$, *so dass* $\sum_{i \in K} w_i \leq b$

Zielfunktion: *Maximiere* $\sum_{i \in K} p_i$

Entscheidungsvariante: $p \in \mathbb{N}$ sei gegeben. Gibt es eine zulässige Lösung mit Nutzen mindestens p?

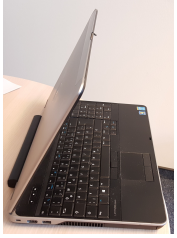
Rucksackproblem – Beispiel

maximal 10 kg

$p_1 = 20$
$w_1 = 6.5$ kg

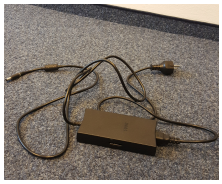
$p_3 = 0.01$
$w_3 = 0.1$ kg

$p_5 = 0.1$
$w_5 = 0.1$ kg



$p_2 = 200$
$w_2 = 3$ kg

$p_4 = 5$
$w_4 = 1$ kg



$p_6 = 30$
$w_6 = 1$ kg

Optimierungsprobleme und ihre Entscheidungsvarianten

Beim Bin Packing Problem suchen wir eine Verteilung von N Objekten mit Gewichten $w_1, \ldots, w_N$ auf eine möglichst kleine Anzahl von Behältern mit Gewichtskapazität jeweils b.

Problem (Bin Packing Problem – BPP)

Eingabe: $b \in \mathbb{N}$, $w_1, \ldots, w_N \in \{1, \ldots, b\}$

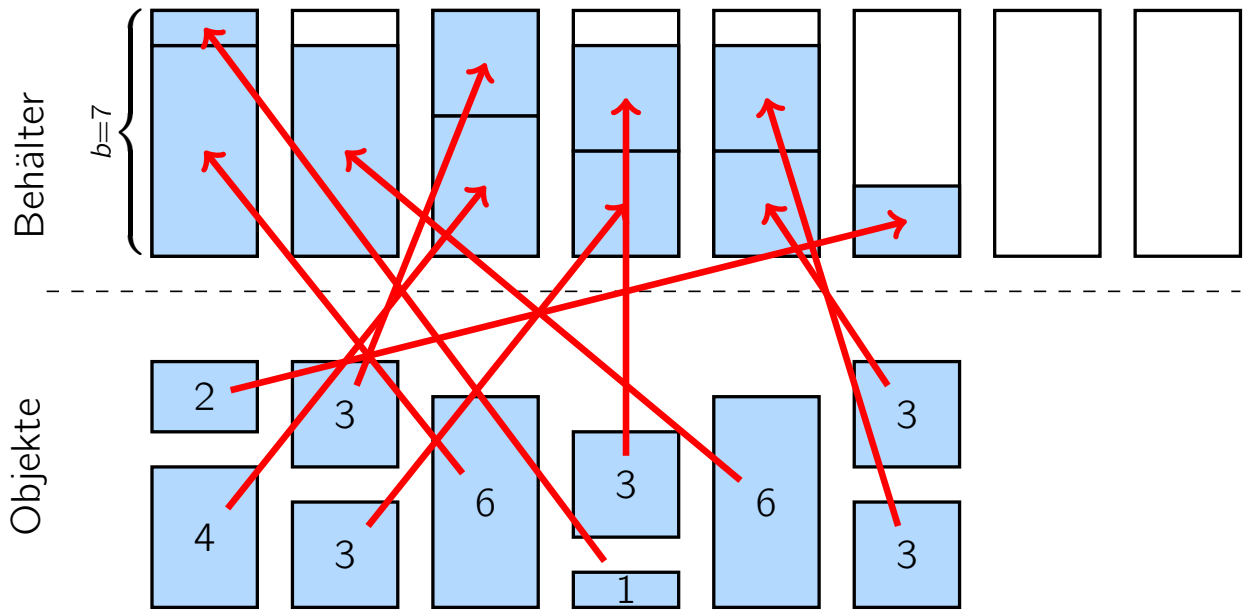
zulässige Lösungen: $k \in \mathbb{N}$ *und Funktion* $f : \{1, \ldots, N\} \to \{1, \ldots, k\}$,

$$\text{so dass } \forall i \in \{1, \ldots, k\} : \sum_{j \in f^{-1}(i)} w_j \leq b$$

Zielfunktion: *Minimiere* k *(= Anzahl Behälter)*

Entscheidungsvariante: $k \in \mathbb{N}$ sei gegeben. Passen die Objekte in k Behälter?

Bin Packing Problem – Beispiel



Eine Lösung die $k = 6$ Behälter verwendet

Optimierungsprobleme und ihre Entscheidungsvarianten

Beim Traveling Salesperson Problem (TSP) ist ein vollständiger Graph auf N Knoten (Orten) mit Kantengewichten (Kosten) gegeben. Gesucht ist eine Rundreise (ein Hamiltonkreis, eine Tour) mit kleinstmöglichen Kosten.

Problem (Traveling Salesperson Problem – TSP)

Eingabe: $c(i,j) \in \mathbb{N}$ *für* $i,j \in \{1,\ldots,N\}$ *mit* $c(j,i) = c(i,j)$

zulässige Lösungen: *Permutationen* π *auf* $\{1,\ldots,N\}$

Zielfunktion: *Minimiere* $c(\pi(N),\pi(1)) + \sum_{i=1}^{N-1} c(\pi(i),\pi(i+1))$

Entscheidungsvariante: $b \in \mathbb{N}$ ist gegeben. Gibt es eine Tour der Länge höchstens b?

Traveling Salesperson Problem – Beispiel

Man finde eine kürzeste Rundreise durch die größten deutschen Städte und zurück zum Ausgangsort.

Optimierungsproblem versus Entscheidungsproblem

- Mit Hilfe eines Algorithmus, der ein Optimierungsproblem löst, kann man die Entscheidungsvariante lösen. (Wie?)
- Häufig funktioniert auch der umgekehrte Weg. Wir illustrieren dies am Beispiel von KP.
- In den Übungen zeigen wir dasselbe auch für TSP und BPP.

Satz

Wenn die Entscheidungsvariante von KP in polynomieller Zeit lösbar ist, dann auch die Optimierungsvariante.

Beweis: Entscheidungsvariante → Zwischenvariante

Zwischenvariante: Gesucht ist nicht eine optimale Lösung, sondern nur der optimale Zielfunktionswert.

Polynomialzeitalgorithmus für die Zwischenvariante

Wir verwenden eine Binärsuche mit folgenden Parametern:

- Der minimale Profit ist 0. Der maximale Profit ist $P := \sum_{i=1}^{N} p_i$.
- Wir finden den optimalen Zielfunktionswert durch Binärsuche auf dem Wertebereich $\{0, \ldots, P\}$.
- Sei A ein Polynomialzeitalgorithmus für die Entscheidungsvariante von KP.
- In jeder Iteration verwenden wir Algorithmus A, der uns sagt in welche Richtung wir weitersuchen müssen.

Beweis: Entscheidungsvariante $\rightarrow$ Zwischenvariante

Die Anzahl der Iterationen der Binärsuche ist $\lceil \log(P+1) \rceil$.

Diese Anzahl müssen wir in Beziehung zur Eingabelänge n setzen.

Untere Schranke für die Eingabelänge:

- Die Kodierungslänge von $a \in \mathbb{N}$ ist $\kappa(a) := \lceil \log(a+1) \rceil$.
- Die Funktion κ ist subadditiv, d.h., für alle $a, b \in \mathbb{N}$ gilt $\kappa(a+b) \leq \kappa(a) + \kappa(b)$.
- Die Eingabelänge n ist somit mindestens

$$\sum_{i=1}^{N} \kappa(p_i) \;\geq\; \kappa\left(\sum_{i=1}^{N} p_i\right) \;=\; \kappa(P) \;=\; \lceil \log(P+1) \rceil \;\; .$$

Also reichen n Aufrufe von A um den optimalen Zielfunktionswert zu bestimmen.

Beweis: Zwischenvariante $\rightarrow$ Optimierungsvariante

Aus einem Algorithmus B für die Zwischenvariante konstruieren wir jetzt einen Algorithmus C für die Optimierungsvariante.

Algorithmus C

1. $K := \{1, \ldots, N\}$;
2. $p := B(K)$;
3. **for** i := 1 **to** N **do**
   if $B(K \setminus \{i\}) = p$ **then** $K := K \setminus \{i\}$;
4. **Ausgabe** K.

Laufzeit: $N+1$ Aufrufe von Algorithmus B, also polynomiell beschränkt, falls die Laufzeit von B polynomiell beschränkt ist.

Beweis: Zwischenvariante → Optimierungsvariante

Korrektheit:

Es gilt die Schleifeninvariante $B(K) = p$.

Für die ausgegebene Menge K gilt somit $B(K) = p$.

Aber ist die ausgegebene Menge K auch zulässig?

- Zum Zweck des Widerspruchs nehmen wir an, dass $\sum_{i \in K} w_i > b$.
- Dies bedeutet, dass nicht alle Objekte in den Rucksack passen.
- Dann gibt es $i \in K$, so dass $B(K \setminus \{i\}) = p$.
- Dies steht aber im Widerspruch dazu, dass der Algorithmus das Objekt i nicht aus K gestrichen hat.
- Also gilt $\sum_{i \in K} w_i \leq b$ und somit ist K zulässig.

Alternative Charakterisierung der Klasse NP

Satz

Eine Sprache $L \subseteq \Sigma^*$ ist genau dann in NP*, wenn es einen Polynomialzeitalgorithmus V (einen sogenannten Verifizierer) und ein Polynom p mit der folgenden Eigenschaft gibt:*

$$x \in L \quad \Leftrightarrow \quad \exists y \in \{0,1\}^*, |y| \leq p(|x|) : V \text{ akzeptiert } y\#x.$$

Von der NTM zu Zertifikat & Verifizierer

Es sei $L \in \mathrm{NP}$ eine Sprache.

- Sei M eine NTM, die L in polynomieller Zeit erkennt.
- Die Laufzeit von M sei beschränkt durch ein Polynom q.
- O.B.d.A. sehe die Überführungsrelation von δ immer genau zwei Übergänge vor, die wir mit 0 und 1 bezeichnen.

Konstruktion von Zertifikat und Verifizierer:

- Für die Eingabe $x \in L$ beschreibe $y \in \{0,1\}^{q(|x|)}$ den Pfad von M auf einem akzeptierenden Rechenweg.
- Wir verwenden y als Zertifikat.
- Der Verifizierer V erhält als Eingabe $y\#x$ und simuliert einen Rechenweg der NTM M für die Eingabe x.

Beweis: Von der NTM zu Zertifikat & Verifizierer

Korrektheit der Konstruktion:

- Gemäß Konstruktion gilt

$$\begin{aligned} x \in L &\Leftrightarrow M \text{ akzeptiert } x \\ &\Leftrightarrow \exists y \in \{0,1\}^{q(|x|)} : V \text{ akzeptiert } y\#x. \end{aligned}$$

- Der Verifizierer kann die durch das Zertifikat y beschriebene Rechnung mit polynomiellem Zeitverlust simulieren.
- Somit erfüllen y und V die im Satz geforderten Eigenschaften.

Beweis: Von Zertifikat & Verifizierer zur NTM

Gegeben:

Verifizierer V mit polynomieller Laufzeitschranke und Polynom p mit der Eigenschaft:

$$x \in L \quad \Leftrightarrow \quad \exists y \in \{0,1\}^*, |y| \leq p(|x|) : V \text{ akzeptiert } y\#x.$$

Konstruktion der NTM:

1. M rät das Zertifikat $y \in \{0,1\}^*, |y| \leq p(|x|)$.
2. M führt V auf $y\#x$ aus und akzeptiert genau dann, wenn V akzeptiert.

Beweis: Von Zertifikat & Verifizierer zur NTM

Korrektheit der Konstruktion:

- M erkennt die Sprache L, weil gilt

$$\begin{aligned} x \in L \quad &\Leftrightarrow \quad \exists y \in \{0,1\}^*, |y| \leq p(|x|) : V \text{ akzeptiert } y\#x \\ &\Leftrightarrow \quad \text{Es gibt einen akzeptierenden Rechenweg für } M \\ &\Leftrightarrow \quad M \text{ akzeptiert } x. \end{aligned}$$

- Die Laufzeit von M ist polynmiell beschränkt, denn
  - die Laufzeit von Schritt 1 (Zertifikat raten) entspricht der Länge des Zertifikats, und
  - die Laufzeit von Schritt 2 (Zertifikat verifizieren) entspricht der Laufzeit des Verifizierers.

□

Zertifikat & Verifizerer für KP, BPP und TSP

Satz

Die Entscheidungsvarianten von KP, BPP und TSP sind in NP.

Beweis:

Die Entscheidungsvariante eines Optimierungsproblems hat einen natürlichen Kandidaten für ein Zertifikat, nämlich zulässige optimale Lösungen.

Es muss allerdings gezeigt werden, dass

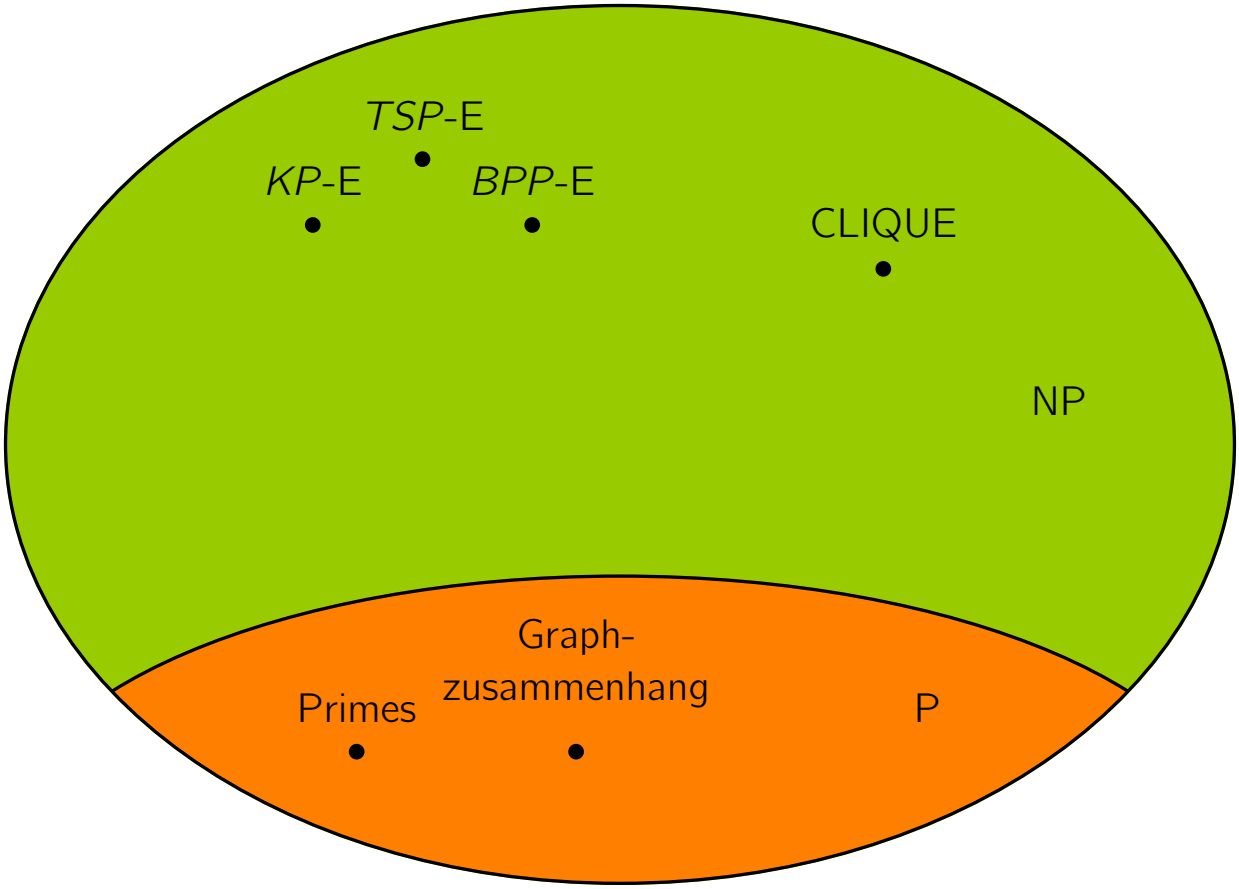
- diese Lösungen eine in der Eingabelänge polynomiell beschränkte Kodierungslänge haben, und
- ihre Zulässigkeit durch einen Polynomialzeitalgorithmus überprüft werden kann.

Zertifikat & Verifizerer für KP, BPP und TSP

- KP: Die Teilmenge $K \subseteq \{1, \ldots, N\}$ kann mit N Bits kodiert werden. Gegeben K, kann die Einhaltung von Gewichts- und Nutzenwertschranke in polynomieller Zeit überprüft werden.
- BPP: Die Abbildung $f : \{1, \ldots, N\} \to \{1, \ldots, k\}$ kann mit $O(N \log k)$ Bits kodiert werden. Gegeben f, kann die Einhaltung der Gewichtsschranken in polynomieller Zeit überprüft werden.
- TSP: Für die Kodierung einer Permutation π werden $O(N \log N)$ Bits benötigt. Es kann in polynomieller Zeit überprüft werden, ob die durch π beschriebene Rundreise die vorgegebene Kostenschranke b einhält.

□

Die Komplexitätslandschaft



Warnung: Dieser Abbildung liegt die Annahme $\mathrm{P} \neq \mathrm{NP}$ zu Grunde.

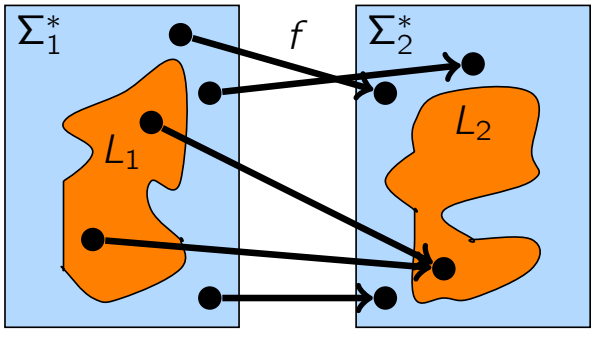
Polynomielle Reduktionen

Definition (Polynomielle Reduktion)

L_1 und L_2 seien zwei Sprachen über Σ_1 bzw. Σ_2. Dann heißt L_1 polynomiell reduzierbar auf L_2, wenn es eine Reduktion von L_1 nach L_2 gibt, die in polynomieller Zeit berechenbar ist. Wir schreiben $L_1 \leq_p L_2$.

D.h. $L_1 \leq_p L_2$ gilt genau dann, wenn es eine Funktion $f : \Sigma_1^* \to \Sigma_2^*$ mit den folgenden Eigenschaften gibt:

- f ist in polynomieller Zeit berechenbar
- $\forall x \in \Sigma_1^* : x \in L_1 \;\Leftrightarrow\; f(x) \in L_2$



Polynomielle Reduktionen (Forts.)

Lemma

Angenommen $L_1 \leq_p L_2$, dann gilt: $L_2 \in P \Rightarrow L_1 \in P$.

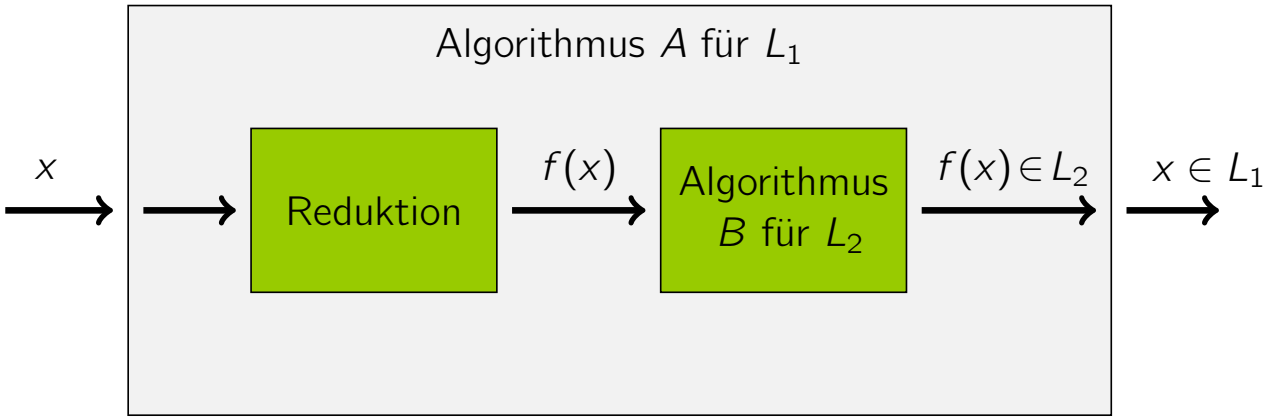
Beweis: Die Reduktion f habe die polynomielle Laufzeitschranke $p(\cdot)$.
Sei B ein Algorithmus für L_2 mit polynomielle Laufzeitschranke $q(\cdot)$.

Algorithmus A für L_1:

1. Berechne $f(x)$.
2. Simuliere Algorithmus B für L_2 auf $f(x)$ und übernehme die Antwort.

Schritt 1 hat Laufzeit höchstens $p(|x|)$. Schritt 2 hat Laufzeit höchstens $q(|f(x)|) \leq q(p(|x|) + |x|)$. □

Polynomielle Reduktionen – Illustration



COLORING $\leq_p$ SAT

Mit dem Reduktionsprinzip ist es möglich, Probleme unterschiedlichster Art aufeinander zu reduzieren. Wir demonstrieren dies an einem Beispiel.

Problem (Knotenfärbung – COLORING)

Eingabe: Graph $G = (V, E)$, Zahl $k \in \{1, \ldots, |V|\}$
Frage: Gibt es eine Färbung $c \colon V \to \{1, \ldots, k\}$ der Knoten von G mit k Farben, so dass benachbarte Knoten verschiedene Farben haben, d.h. $\forall e = \{u, v\} \in E : c(u) \neq c(v)$?

Problem (Erfüllbarkeitsproblem / Satisfiability – SAT)

Eingabe: Aussagenlogische Formel φ in KNF
Frage: Gibt es eine erfüllende Belegung für φ?

Zwei Beispiele zum Erfüllbarkeitsproblem

SAT-Beispiel 1:

$$\varphi = (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_4) \wedge (x_2 \vee x_3 \vee x_4)$$

φ ist erfüllbar, denn $x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0$ ist eine erfüllende Belegung.

SAT-Beispiel 2:

$$\varphi' = (x_1 \vee x_2) \wedge (\bar{x}_2 \vee x_1) \wedge (\bar{x}_1 \vee x_3) \wedge (\bar{x}_3 \vee \bar{x}_1)$$

φ' ist nicht erfüllbar. (Warum?)

Eine polynomielle Reduktion: COLORING $\leq_p$ SAT

Satz
COLORING $\leq_p$ *SAT.*

Beweis:
Wir beschreiben eine polynomiell berechenbare Funktion f, die eine Eingabe (G, k) für das COLORING-Problem auf eine Formel φ für das SAT-Problem abbildet, mit der Eigenschaft

$$G \text{ hat eine } k\text{-Färbung} \quad \Leftrightarrow \quad \varphi \text{ ist erfüllbar .}$$

Beispiel einer polyn. Reduktion: COLORING $\leq_p$ SAT

Beschreibung der Funktion f:

Die Formel φ hat für jede Knoten-Farb-Kombination (v, i), $v \in V$, $i \in \{1, \ldots, k\}$ eine Variable x_v^i. Die Formel für (G, k) lautet

$$\varphi \;=\; \bigwedge_{v \in V} \underbrace{(x_v^1 \vee x_v^2 \vee \ldots \vee x_v^k)}_{\text{Knotenbedingung}} \wedge \bigwedge_{\{u,v\} \in E} \bigwedge_{i \in \{1,\ldots,k\}} \underbrace{(\bar{x}_u^i \vee \bar{x}_v^i)}_{\text{Kantenbedingung}} \quad .$$

Anzahl der Literale $= O(k \cdot |V| + k \cdot |E|) = O(|V|^3)$.

Die Länge der Formel ist somit polynomiell beschränkt und die Formel kann in polynomieller Zeit konstruiert werden.

Aber ist die Konstruktion auch korrekt?

Beispiel einer polyn. Reduktion: COLORING $\leq_p$ SAT

Korrektheit:

Zu zeigen: G hat eine k-Färbung $\Rightarrow$ φ ist erfüllbar

- Sei c eine k-Färbung für G.
- Für jeden Knoten v mit $c(v) = i$ setzen wir $x_v^i = 1$ und alle anderen Variablen auf 0.
- Knotenbedingung: $(x_v^1 \vee x_v^2 \vee \ldots \vee x_v^k)$ ist erfüllt.
- Kantenbedingung: Für jede Farbe i und jede Kante $\{u, v\}$ gilt $\bar{x}_u^i \vee \bar{x}_v^i$, denn sonst hätten u und v beide die Farbe i.
- Damit erfüllt diese Belegung die Formel φ.

Beispiel einer polyn. Reduktion: COLORING $\leq_p$ SAT

Zu zeigen: φ ist erfüllbar $\Rightarrow$ G hat eine k-Färbung

- Fixiere eine beliebige erfüllende Belegung für φ.
- Wegen der Knotenbedingung gibt es für jeden Knoten v mindestens eine Farbe i mit $x_v^i = 1$.
- Für jeden Knoten wähle eine beliebige derartige Farbe aus.
- Sei $\{u, v\} \in E$. Wir behaupten: $c(u) \neq c(v)$.
- Zum Widerspruch nehmen wir an, $c(u) = c(v) = i$. Dann wäre $x_u^i = x_v^i = 1$ und die Kantenbedingung $\bar{x}_u^i \vee \bar{x}_v^i$ wäre verletzt.

□

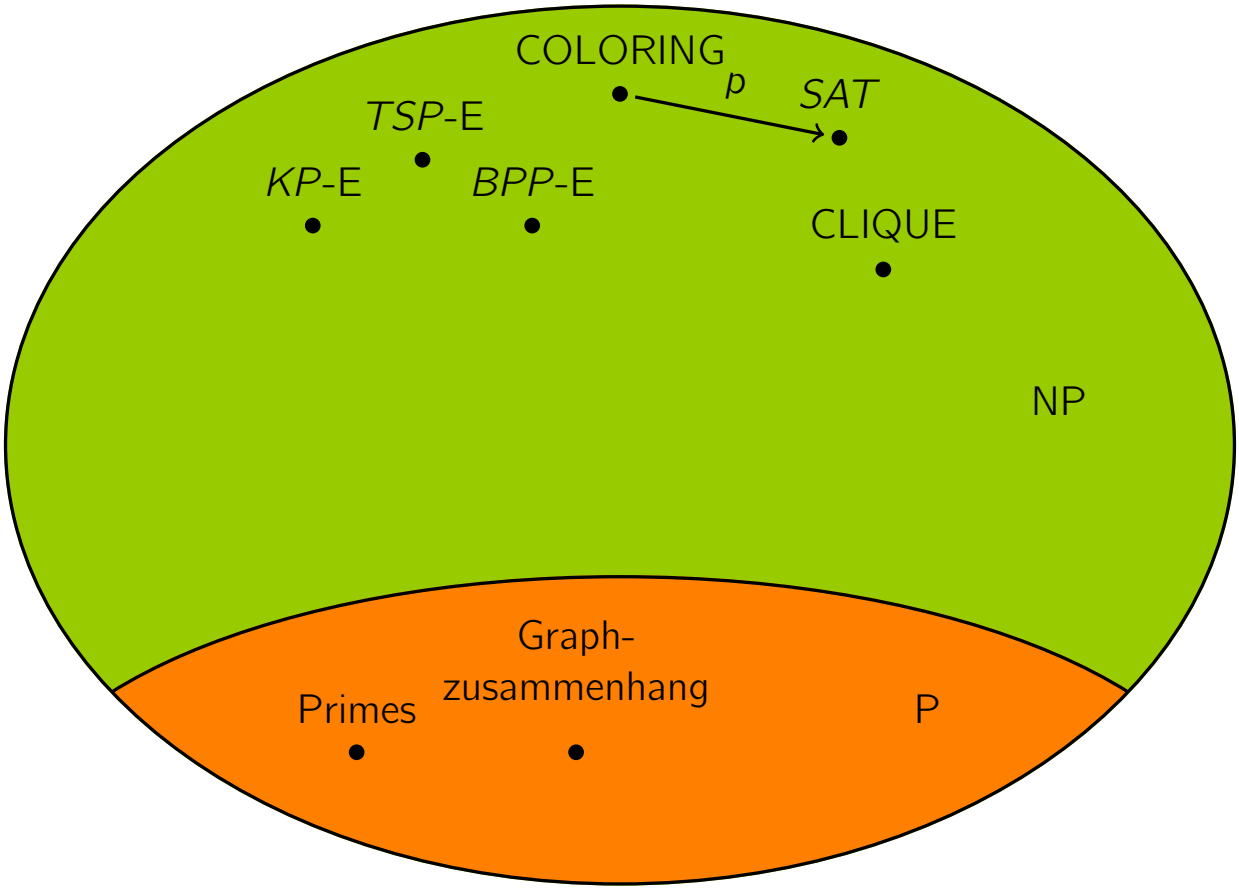
Beispiel einer polyn. Reduktion: COLORING $\leq_p$ SAT

Die Tatsache COLORING $\leq_p$ SAT impliziert das Folgende:

Korollar

Wenn SAT einen Polynomialzeitalgorithmus hat, so hat auch COLORING einen Polynomialzeitalgorithmus.

Die Komplexitätslandschaft



Warnung: Dieser Abbildung liegt die Annahme P $\neq$ NP zu Grunde.

Ausblick

Ein Problem $L \in \mathrm{NP}$ heißt NP-vollständig, wenn für jedes Problem $L' \in \mathrm{NP}$ gilt, dass $L' \leq_p L$.

Satz (Cook und Levin)

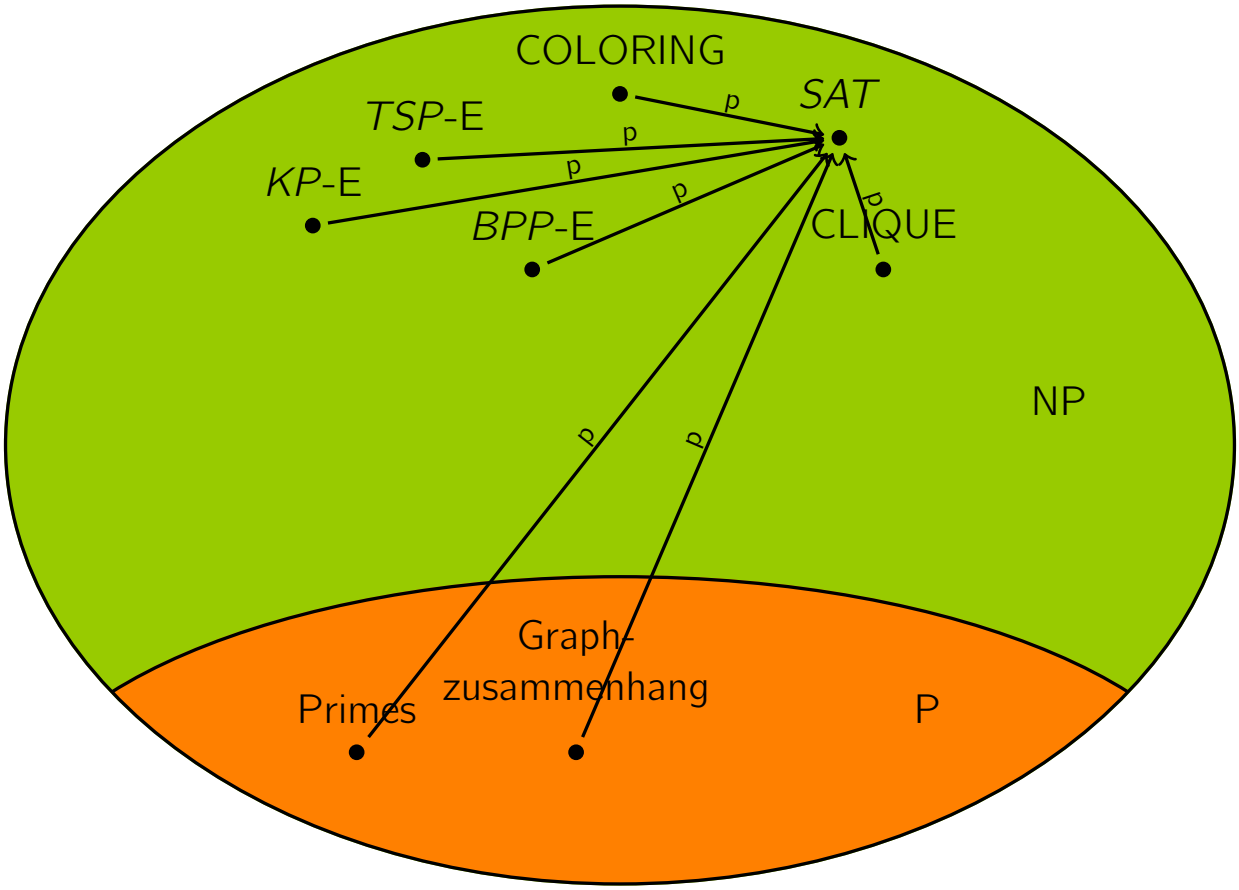
SAT ist NP*-vollständig.*

Es folgt: Wenn SAT einen Polynomialzeitalgorithmus hätte, so gäbe es auch einen Polynomialzeitalgorithmus für jedes andere Problem aus NP und somit wäre $\mathrm{P} = \mathrm{NP}$.

Im Umkehrschluss gilt:

SAT hat keinen Polynomialzeitalgorithmus, außer wenn $\mathrm{P} = \mathrm{NP}$.

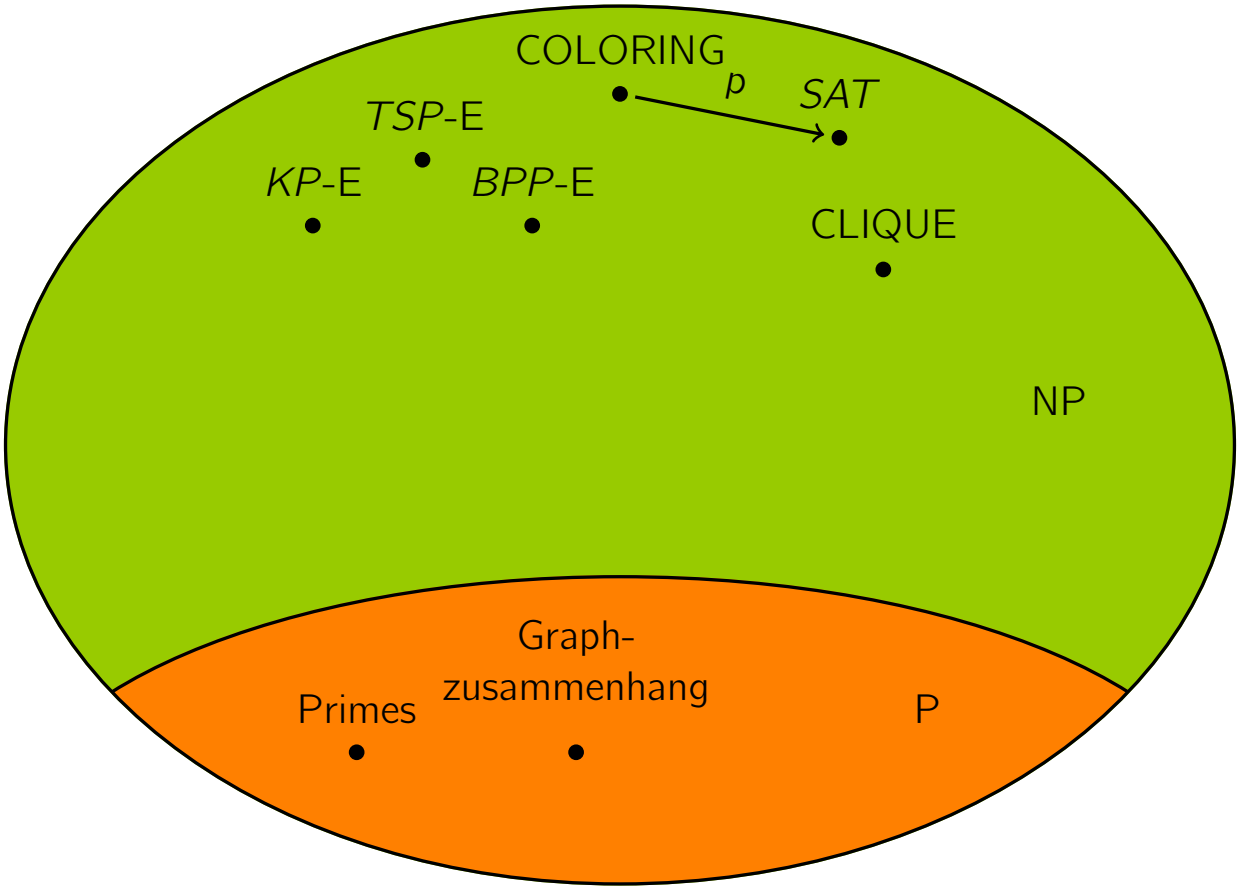
Die Komplexitätslandschaft



Warnung: Dieser Abbildung liegt die Annahme $\mathrm{P} \neq \mathrm{NP}$ zu Grunde.

Vorlesung 15
NP-Vollständigkeit

Wdh.: Die Komplexitätslandschaft



Warnung: Dieser Abbildung liegt die Annahme P $\neq$ NP zu Grunde.

Wdh.: Optimierungs- versus Entscheidungsproblem

Mit Hilfe eines Algorithmus, der ein Optimierungsproblem löst, kann man die Entscheidungsvariante lösen.

Umgekehrt gilt:

Satz

Wenn die Entscheidungsvariante von KP in polynomieller Zeit lösbar ist, dann auch die Optimierungsvariante.

Dieser Satz gilt auch für TSP und BPP.

Wdh.: Alternative Charakterisierung der Klasse NP

Satz

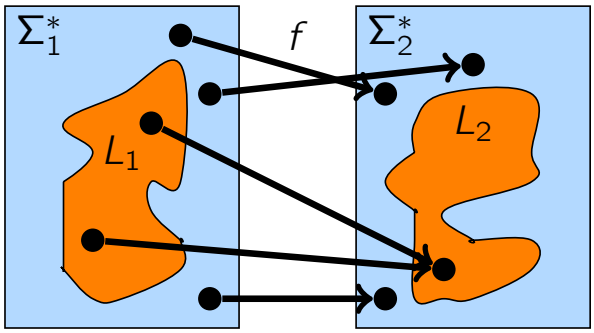
Eine Sprache $L \subseteq \Sigma^*$ ist genau dann in NP*, wenn es einen Polynomialzeitalgorithmus V (einen sogenannten Verifizierer) und ein Polynom p mit der folgenden Eigenschaft gibt:*

$$x \in L \quad \Leftrightarrow \quad \exists y \in \{0,1\}^*, |y| \leq p(|x|) : V \text{ akzeptiert } y\#x.$$

Wdh.: Polynomielle Reduktionen

Definition (Polynomielle Reduktion)

L_1 und L_2 seien zwei Sprachen über Σ_1 bzw. Σ_2. Dann heißt L_1 polynomiell reduzierbar auf L_2, wenn es eine Reduktion von L_1 nach L_2 gibt, die in polynomieller Zeit berechenbar ist. Wir schreiben $L_1 \leq_p L_2$.



Lemma

Angenommen $L_1 \leq_p L_2$, dann gilt: $L_2 \in P \Rightarrow L_1 \in P$.

Satz

COLORING $\leq_p$ SAT.

Wdh.: Das Erfüllbarkeitsproblem – SAT

Problem (Erfüllbarkeitsproblem / Satisfiability – SAT)

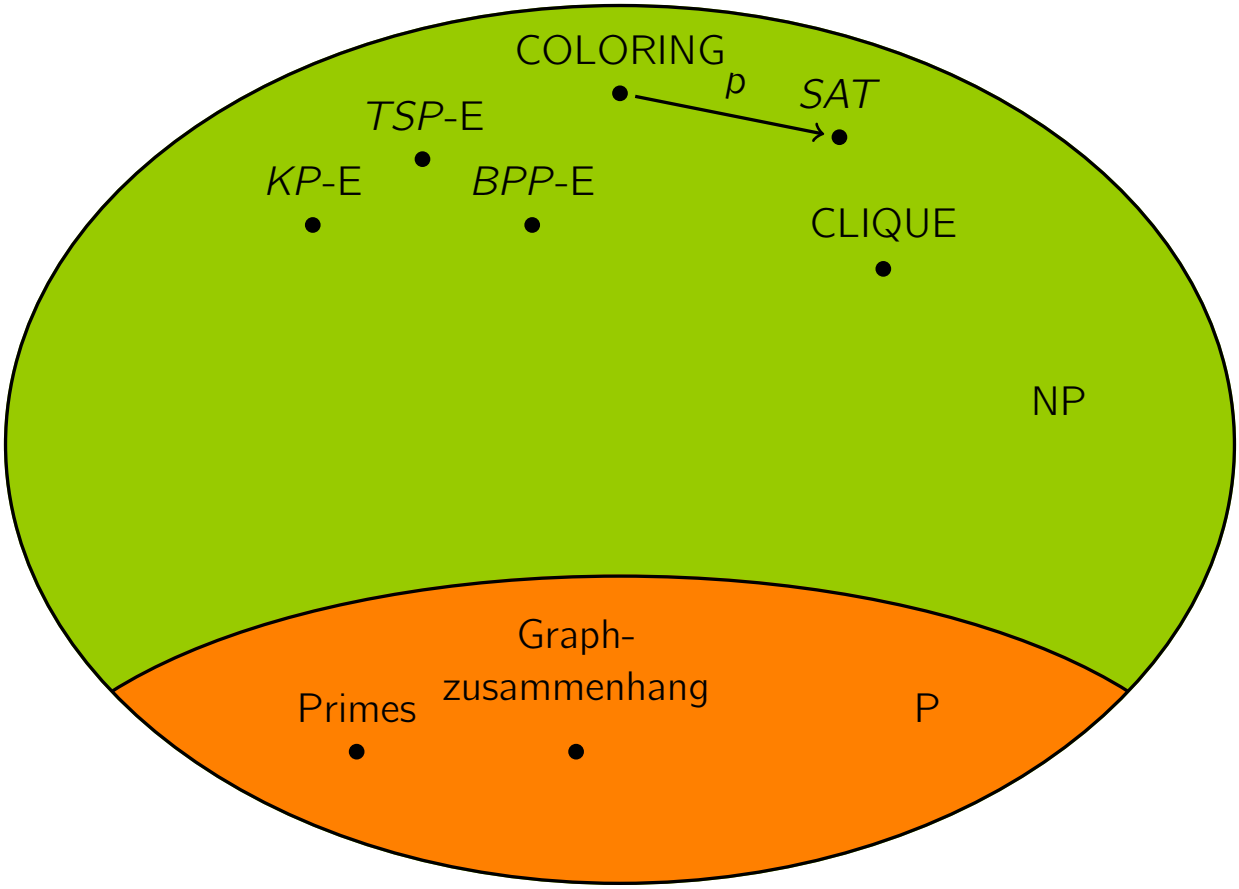
Eingabe: Aussagenlogische Formel φ in KNF
Frage: Gibt es eine erfüllende Belegung für φ?

SAT-Beispiel 1:

$$\varphi = (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_4) \wedge (x_2 \vee x_3 \vee x_4)$$

φ ist erfüllbar, denn $x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0$ ist eine erfüllende Belegung.

Wdh.: Die Komplexitätslandschaft



Warnung: Dieser Abbildung liegt die Annahme $\mathrm{P} \neq \mathrm{NP}$ zu Grunde.

NP-schwere Probleme

Definition (NP-schwer)

Ein Problem L heißt NP-schwer (engl. NP-hard), wenn gilt:

$$\forall L' \in \mathrm{NP} : L' \leq_p L.$$

Satz

Wenn L NP-schwer ist, dann gilt: $L \in \mathrm{P} \Rightarrow \mathrm{P} = \mathrm{NP}$

Beweis: Ein Polynomialzeitalgorithmus für L liefert mit der Reduktion $L' \leq_p L$ einen Polynomialzeitalgorithmus für alle $L' \in \mathrm{NP}$. □

Fazit: Für NP-schwere Probleme gibt es keine Polynomialzeitalgorithmen, es sei denn P= NP.

Def: NP-Vollständigkeit

Definition (NP-vollständig)

Ein Problem L heißt NP-vollständig (engl. NP-complete), falls gilt

1. $L \in \mathrm{NP}$, und
2. L ist NP-schwer.

Die Klasse der NP-vollständigen Probleme wird mit NPC bezeichnet.

Wir werden zeigen, dass SAT, CLIQUE, KP-E, BPP-E, TSP-E und weitere Probleme NP-vollständig sind.

Unter der Annahme, dass $\mathrm{P} \neq \mathrm{NP}$, hat also keines dieser Probleme einen Polynomialzeitalgorithmus.

NP-Vollständigkeit des Erfüllbarkeitsproblems

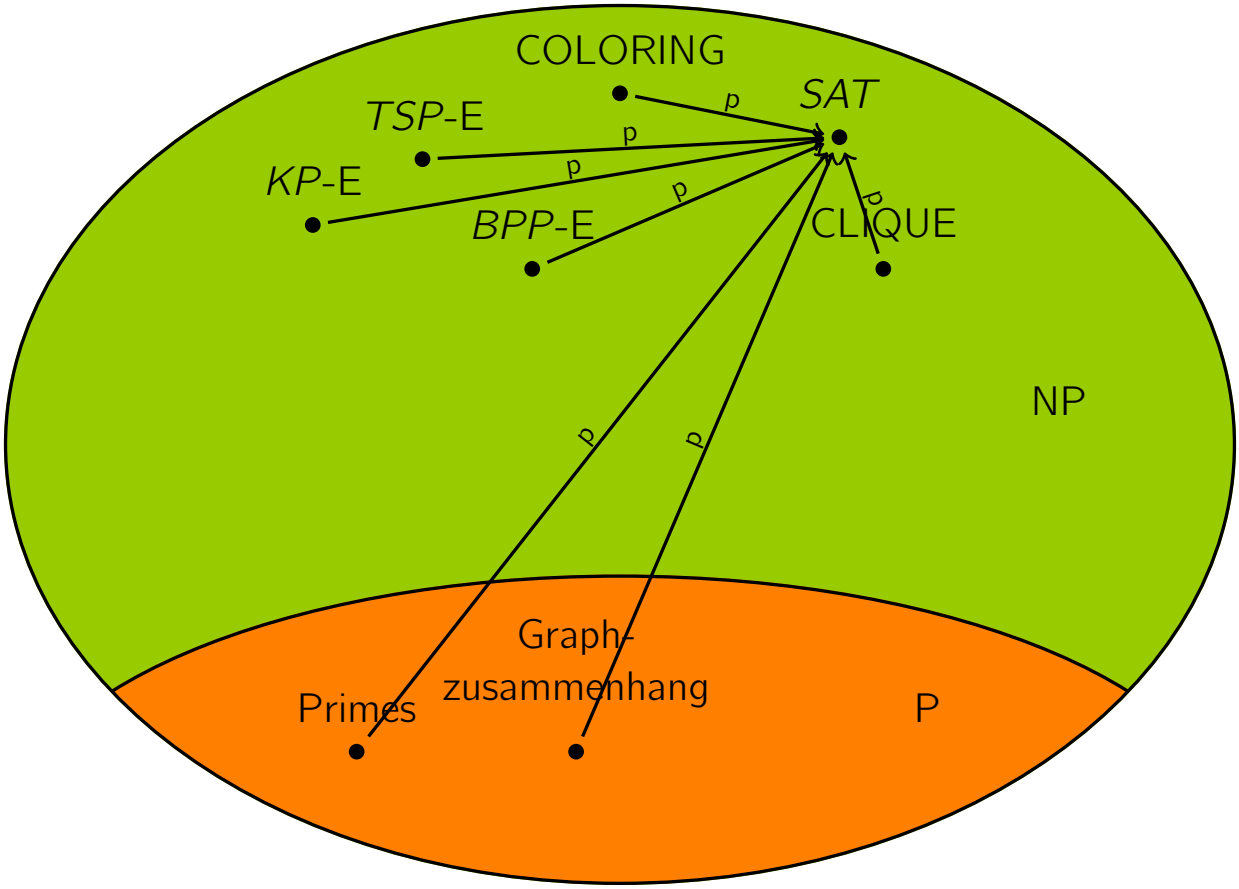
Der Ausgangspunkt für unsere NP-Vollständigkeitsbeweise ist das Erfüllbarkeitsproblem.

Satz (Cook und Levin)

SAT ist NP-vollständig.

Unter der Annahme, dass $\mathrm{P} \neq \mathrm{NP}$, hat SAT also keinen Polynomialzeitalgorithmus.

Die Komplexitätslandschaft



Warnung: Dieser Abbildung liegt die Annahme P $\neq$ NP zu Grunde.

Beweis des Satzes von Cook und Levin

- Es gilt SAT $\in$ NP, denn die erfüllende Belegung kann als Zertifikat verwendet werden.
- Wir müssen jetzt noch zeigen, dass SAT NP-schwer ist.

Sei $L \subseteq \Sigma^*$ ein Problem aus NP. Wir müssen zeigen, dass $L \leq_p$ SAT.

Dazu konstruieren wir eine polynomiell berechenbare Funktion f, die jedes $x \in \Sigma^*$ auf eine Formel φ abbildet, so dass gilt

$$x \in L \;\Leftrightarrow\; \varphi \in SAT \;\; .$$

Beweis des Satzes von Cook und Levin

M sei eine NTM, die L in polynomieller Zeit erkennt. Wir werden zeigen

$$M \text{ akzeptiert } x \;\Leftrightarrow\; \varphi \in SAT \quad .$$

Eigenschaften von M

- O.B.d.A. besuche M keine Bandpositionen links von der Startposition.
- Eine akzeptierende Rechnung von M gehe in den Zustand q_{accept} über und bleibe dort in einer Endlosschleife.
- Sei $p(\cdot)$ ein Polynom, so dass M eine Eingabe x genau dann akzeptiert, wenn es einen Rechenweg gibt, der nach $p(n)$ Schritten im Zustand q_{accept} ist, wobei n die Länge von x bezeichne.

Beweis des Satzes von Cook und Levin

Beobachtung:

Sei $K_0 = q_0 x$ die Startkonfiguration von M. M akzeptiert genau dann, wenn es einen Rechenweg, d.h. eine mögliche Konfigurationsfolge

$$K_0 \vdash K_1 \vdash \cdots \vdash K_{p(n)}$$

gibt, bei der $K_{p(n)}$ im Zustand q_{accept} ist.

Weiteres Vorgehen:

Wir konstruieren die Formel φ derart, dass φ genau dann erfüllbar ist, wenn es solch eine akzeptierende Konfigurationsfolge gibt.

Beweis des Satzes von Cook und Levin

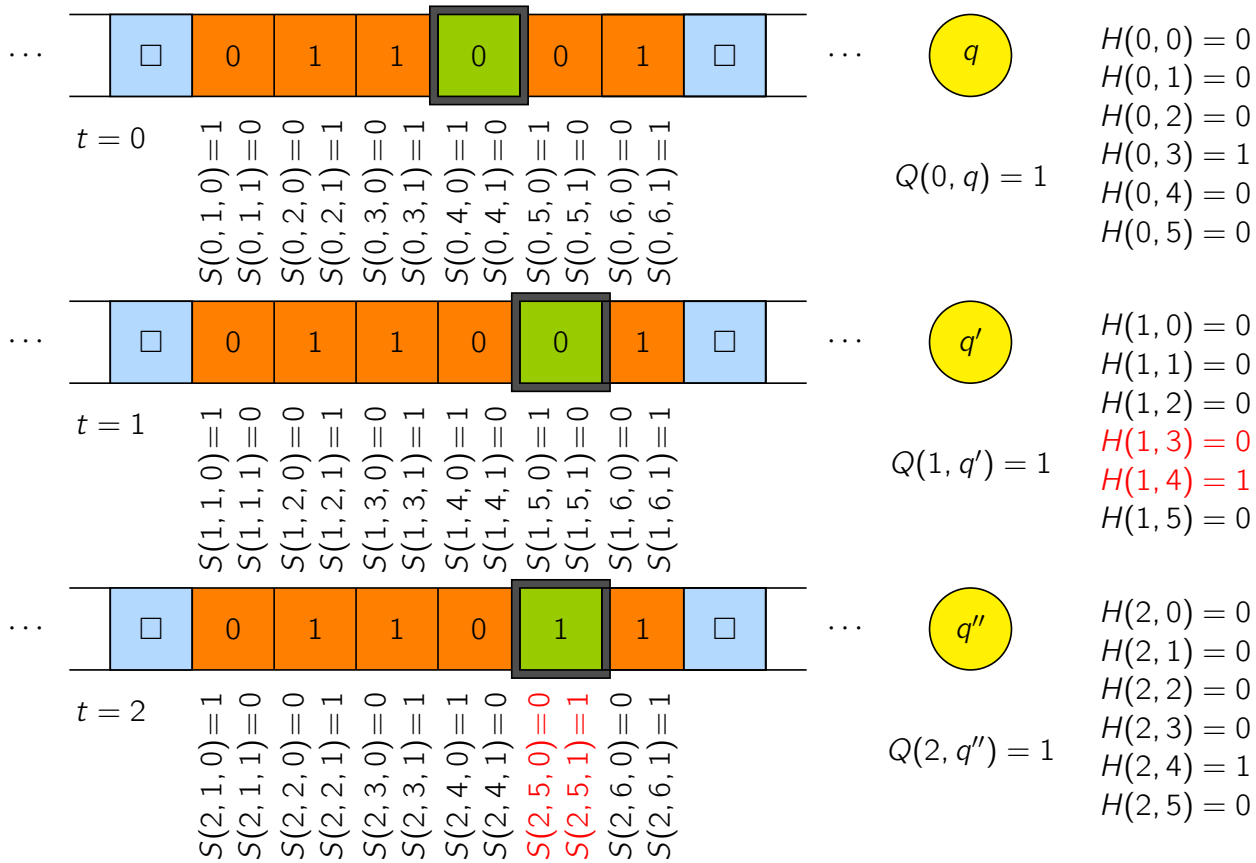
Variablen in φ

- $Q(t,k)$ für $t \in \{0,\ldots,p(n)\}$ und $k \in Q$
- $H(t,j)$ für $t,j \in \{0,\ldots,p(n)\}$
- $S(t,j,a)$ für $t,j \in \{0,\ldots,p(n)\}$ und $a \in \Gamma$

Interpretation der Variablen:

- Die Belegung $Q(t,k)=1$ soll besagen, dass sich die Rechnung zum Zeitpunkt t im Zustand k befindet.
- Die Belegung $H(t,j)=1$ steht dafür, dass sich der Kopf zum Zeitpunkt t an Bandposition j befindet.
- die Belegung $S(t,j,a)=1$ bedeutet, dass zum Zeitpunkt t an Bandposition j das Zeichen a geschrieben steht.

Beweis des Satzes von Cook und Levin – Illustration



Beweis des Satzes von Cook und Levin

Kodierung einzelner Konfigurationen in der Teilformel φ_t:

Für jedes $t \in \{0, \ldots, p(n)\}$, benötigen wir eine Formel φ_t, die nur dann erfüllt ist, wenn es

1. genau einen Zustand $k \in Q$ mit $Q(t, k) = 1$ gibt,
2. genau eine Bandposition $j \in \{0, \ldots, p(n)\}$ mit $H(t, j) = 1$ gibt, und
3. für jedes $j \in \{0, \ldots, p(n)\}$ jeweils genau ein Zeichen $a \in \Gamma$ mit $S(t, j, a) = 1$ gibt.

Beweis des Satzes von Cook und Levin

Erläuterung zur Formel φ_t:

- Für eine beliebige Variablenmenge $\{y_1, \ldots, y_m\}$ besagt das folgende Prädikat in KNF, dass genau eine der Variablen y_i den Wert 1 annimmt:

$$(y_1 \vee \ldots \vee y_m) \wedge \bigwedge_{i \neq j} (\bar{y}_i \vee \bar{y}_j)$$

- Die Anzahl der Literale in dieser Formel ist quadratisch in der Anzahl der Variablen.
- Die drei Anforderungen können also jeweils durch eine Formel in polynomiell beschränkter Länge kodiert werden.

Wir betrachten nun nur noch Belegungen, welche die Teilformeln $\varphi_0, \ldots, \varphi_{p(n)}$ erfüllen und somit Konfigurationen $K_0, \ldots, K_{p(n)}$ beschreiben.

Beweis des Satzes von Cook und Levin

Als Nächstes konstruieren wir eine Formel φ'_t für $1 \leq t \leq p(n)$, die nur für solche Belegungen erfüllt ist, bei denen K_t eine direkte Nachfolgekonfiguration von K_{t-1} ist.

Die Formel φ'_t kodiert zwei Eigenschaften:

1. Die Bandinschrift von K_t stimmt an allen Positionen außer möglicherweise der Position, an der der Kopf zum Zeitpunkt $t-1$ ist, mit der Inschrift von K_{t-1} überein.
2. Zustand, Kopfposition und Bandinschrift an der Kopfposition verändern sich gemäß der Übergangsrelation δ.

Beweis des Satzes von Cook und Levin

Die Eigenschaft, dass die Bandinschrift von K_t an allen Positionen außer möglicherweise der Position, an der der Kopf zum Zeitpunkt $t-1$ ist, mit der Inschrift von K_{t-1} übereinstimmt, kann wie folgt kodiert werden:

$$\bigwedge_{i=0}^{p(n)} \bigwedge_{z \in \Gamma} ((S(t-1,i,z) \wedge \neg H(t-1,i)) \Rightarrow S(t,i,z))$$

Dabei steht $A \Rightarrow B$ für $\neg A \vee B$. D.h., die Formel lautet eigentlich

$$\bigwedge_{i=0}^{p(n)} \bigwedge_{z \in \Gamma} (\neg(S(t-1,i,z) \wedge \neg H(t-1,i)) \vee S(t,i,z))$$

Das De Morgansche Gesetz besagt, dass $\neg(A \wedge B)$ äquivalent zu $\neg A \vee \neg B$ ist. Dadurch ergibt sich folgende Teilformel in KNF:

$$\bigwedge_{i=0}^{p(n)} \bigwedge_{z \in \Gamma} (\neg S(t-1,i,z) \vee H(t-1,i) \vee S(t,i,z))$$

Beweis des Satzes von Cook und Levin

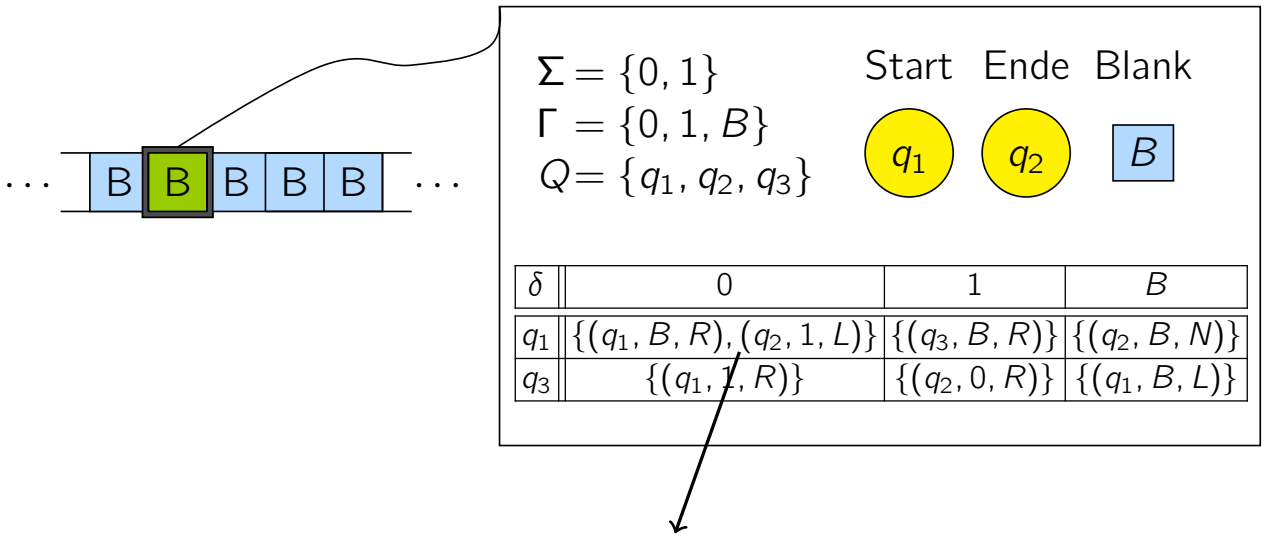
Für die Eigenschaft, dass Zustand, Kopfposition und Bandinschrift an der Kopfposition sich gemäß der Ubergangsrelation δ verändern, ergänzen wir für alle $k \in Q$, $j \in \{0, \ldots, p(n)-1\}$ und $a \in \Gamma$ die folgende Teilformel

$$(Q(t-1,k) \wedge H(t-1,j) \wedge S(t-1,j,a)) \quad \Rightarrow \bigvee_{(k,a,k',a',\kappa)\in\delta} (Q(t,k') \wedge H(t,j+\kappa) \wedge S(t,j,a')) \ ,$$

wobei κ die Werte $L=-1$, $N=0$ und $R=1$ annehmen kann.

Wie lässt sich diese Teilformel in die KNF transformieren?

Beispielübergangsklausel



δ	0	1	B
q_1	$\{(q_1, B, R), (q_2, 1, L)\}$	$\{(q_3, B, R)\}$	$\{(q_2, B, N)\}$
q_3	$\{(q_1, 1, R)\}$	$\{(q_2, 0, R)\}$	$\{(q_1, B, L)\}$

$$Q(t-1,q_1) \wedge H(t-1,j) \wedge S(t-1,j,0)$$
$$\Longrightarrow$$
$$\big(Q(t,q_1) \wedge H(t,j+1) \wedge S(t,j,B)\big) \ \vee \ \big(Q(t,q_2) \wedge H(t,j-1) \wedge S(t,j,1)\big)$$

Beweis des Satzes von Cook und Levin

Ersetzen von $A \Rightarrow B$ durch $\neg A \vee B$ und Anwenden des De Morganschen Gesetzes ergibt die Teilformel

$$\neg Q(t-1,k) \vee \neg H(t-1,j) \vee \neg S(t-1,j,a)) \vee \bigvee_{(k,a,k',a',\kappa)\in\delta} (Q(t,k') \wedge H(t,j+\kappa) \wedge S(t,j,a')) \ ,$$

wobei κ die Werte $L = -1$, $N = 0$ und $R = 1$ annehmen kann.

Jetzt müssen noch die inneren $\wedge$-Verknüpfungen „ausmultipliziert“ werden, d.h. wir ersetzen wiederholt eine Formel der Form $X \vee (A \wedge B \wedge C)$ durch eine äquivalente Formel $(X \vee A) \wedge (X \vee B) \wedge (X \vee C)$.
Wiederholte Anwendung führt zu einer Formel in KNF.

Damit ist die Beschreibung von φ'_t abgeschlossen.

Beweis des Satzes von Cook und Levin

Die Gesamtformel φ ergibt sich nun wie folgt:

$$Q(0,q_0) \ \wedge \ H(0,0) \ \wedge \ \bigwedge_{i=0}^{n} S(0,i,x_i) \ \wedge \ \bigwedge_{i=n+1}^{p(n)} S(0,i,B)$$
$$\wedge \ \bigwedge_{i=0}^{p(n)} \varphi_i \ \wedge \ \bigwedge_{i=1}^{p(n)} \varphi'_i \ \wedge \ Q(p(n),q_{\text{accept}})$$

Die Länge von φ ist polynomiell beschränkt in n, und φ ist effizient aus x berechenbar.

Gemäß unserer Konstruktion ist φ genau dann erfüllbar, wenn es eine akzeptierende Konfigurationsfolge der Länge $p(n)$ für M auf x gibt. $\square$

Kochrezept für NP-Vollständigkeitsbeweise

- Um nachzuweisen, dass SAT NP-schwer ist, haben wir in einer „Master-Reduktion“ alle Probleme aus NP auf SAT reduziert.
- Die NP-Vollständigkeit von SAT können wir jetzt verwenden, um nachzuweisen, dass weitere Probleme NP-schwer sind.

Lemma

Wenn L^* NP-schwer ist, dann gilt: $L^* \leq_p L \Rightarrow$ *L ist* NP*-schwer.*

Beweis: Gemäß Voraussetzung gilt $\forall L' \in \text{NP}: L' \leq_p L^*$ und $L^* \leq_p L$. Aufgrund der Transitivität der polynomiellen Reduktion folgt somit $\forall L' \in \text{NP}: L' \leq_p L$. □

Kochrezept für NP-Vollständigkeitsbeweise

Wie beweist man, dass eine Sprache L NP-vollständig ist?

1. Man zeige $L \in \text{NP}$.
2. Man wähle eine NP-vollständige Sprache L'.
3. Man entwerfe eine Funktion f, die Instanzen von L' auf Instanzen von L abbildet. **(Beschreibung der Reduktionsabbildung)**
4. Man zeige, dass f in polynomieller Zeit berechnet werden kann. **(Polynomialzeit)**
5. Man beweise, dass f eine Reduktion ist: Für $x \in \{0,1\}^*$ ist $x \in L'$ genau dann, wenn $f(x) \in L$. **(Korrektheit)**

NP-Vollständigkeit von 3-SAT

Eine Formel in k-KNF besteht nur aus Klauseln mit jeweils k Literalen, sogenannten k-Klauseln.

Beispiel einer Formel in 3-KNF:

$$\varphi = \underbrace{(\bar{x}_1 \vee \bar{x}_2 \vee x_3)}_{\text{3 Literale}} \wedge \underbrace{(\bar{x}_1 \vee x_2 \vee \bar{x}_3)}_{\text{3 Literale}}$$

Problem (3-SAT)

Eingabe: Aussagenlogische Formel φ in 3-KNF
Frage: Gibt es eine erfüllende Belegung für φ?

- 3-SAT ist ein Spezialfall von SAT und deshalb wie SAT in NP.
- Um zu zeigen, dass 3-SAT ebenfalls NP-vollständig ist, müssen wir also nur noch die NP-Schwere von 3-SAT nachweisen.
- Dazu zeigen wir SAT $\leq_p$ 3-SAT.

SAT $\leq_p$ 3-SAT

Lemma

SAT $\leq_p$ 3-SAT.

Beweis:

- Gegeben sei eine Formel φ in KNF.
- Wir transformieren φ in eine erfüllbarkeitsäquivalente Formel φ' in 3KNF, d.h.

$$\varphi \text{ ist erfüllbar } \Leftrightarrow \varphi' \text{ ist erfüllbar .}$$

- Aus einer 1- bzw 2-Klausel können wir leicht eine äquivalente 3-Klausel machen, indem wir ein Literal wiederholen.
- Was machen wir aber mit k-Klauseln für $k > 3$?

SAT $\leq_p$ 3-SAT

- Sei $k \geq 4$ und C eine k-Klausel der Form

$$C \;=\; \ell_1 \vee \ell_2 \vee \ell_3 \cdots \vee \ell_k \;\; .$$

- In einer Klauseltransformation ersetzen wir C durch die Teilformel

$$C' \;=\; (\ell_1 \vee \cdots \vee \ell_{k-2} \vee h) \wedge (\bar{h} \vee \ell_{k-1} \vee \ell_k) \;\; ,$$

wobei h eine zusätzlich eingeführte Hilfsvariable bezeichnet.

SAT $\leq_p$ 3-SAT

Beispiel für die Klauseltransformation:
Aus der 5-Klausel

$$x_1 \vee \bar{x}_2 \vee x_3 \vee x_4 \vee \bar{x}_5$$

wird in einem ersten Transformationsschritt die Teilformel

$$(x_1 \vee \bar{x}_2 \vee x_3 \vee h_1) \;\wedge\; (\bar{h}_1 \vee x_4 \vee \bar{x}_5) \;\; ,$$

also eine 4- und eine 3-Klausel. Auf die 4-Klausel wird die Transformation erneut angewendet. Wir erhalten die Teilformel

$$(x_1 \vee \bar{x}_2 \vee h_2) \;\wedge\; (\bar{h}_2 \vee x_3 \vee h_1) \;\wedge\; (\bar{h}_1 \vee x_4 \vee \bar{x}_5) \;\; ,$$

die nur noch 3-Klauseln enthält.

SAT $\leq_p$ 3-SAT

Nachweis der Erfüllbarkeitsäquivalenz:

φ' sei aus φ durch Ersetzen einer Klausel C durch C' entstanden.

Zu zeigen: φ erfüllbar $\Rightarrow$ φ' erfüllbar

- Sei B eine erfüllende Belegung für φ.
- B weist mindestens einem Literal aus C hat den Wert 1 zu.
- Wir unterscheiden zwei Fälle:
  1) Falls $\ell_1 \vee \cdots \vee \ell_{k-2}$ erfüllt ist, so ist φ' erfüllt, wenn wir $h = 0$ setzen.
  2) Falls $\ell_{k-1} \vee \ell_k$ erfüllt ist, so ist φ' erfüllt, wenn wir $h = 1$ setzen.
- Also ist φ' in beiden Fällen erfüllbar.

SAT $\leq_p$ 3-SAT

Zu zeigen: φ' erfüllbar $\Rightarrow$ φ erfüllbar

- Sei B' nun eine erfüllende Belegung für φ'.
- Wir unterscheiden wiederum zwei Fälle:
  - Falls B' der Variable h den Wert 0 zuweist, so muss B' einem der Literale $\ell_1, \ldots, \ell_{k-2}$ den Wert 1 zugewiesen haben.
  - Falls B' der Variable h den Wert 1 zuweist, so muss B' einem der beiden Literale ℓ_{k-1} oder k den Wert 1 zugewiesen haben.
- In beiden Fällen erfüllt B' somit auch φ.

SAT $\leq_p$ 3-SAT

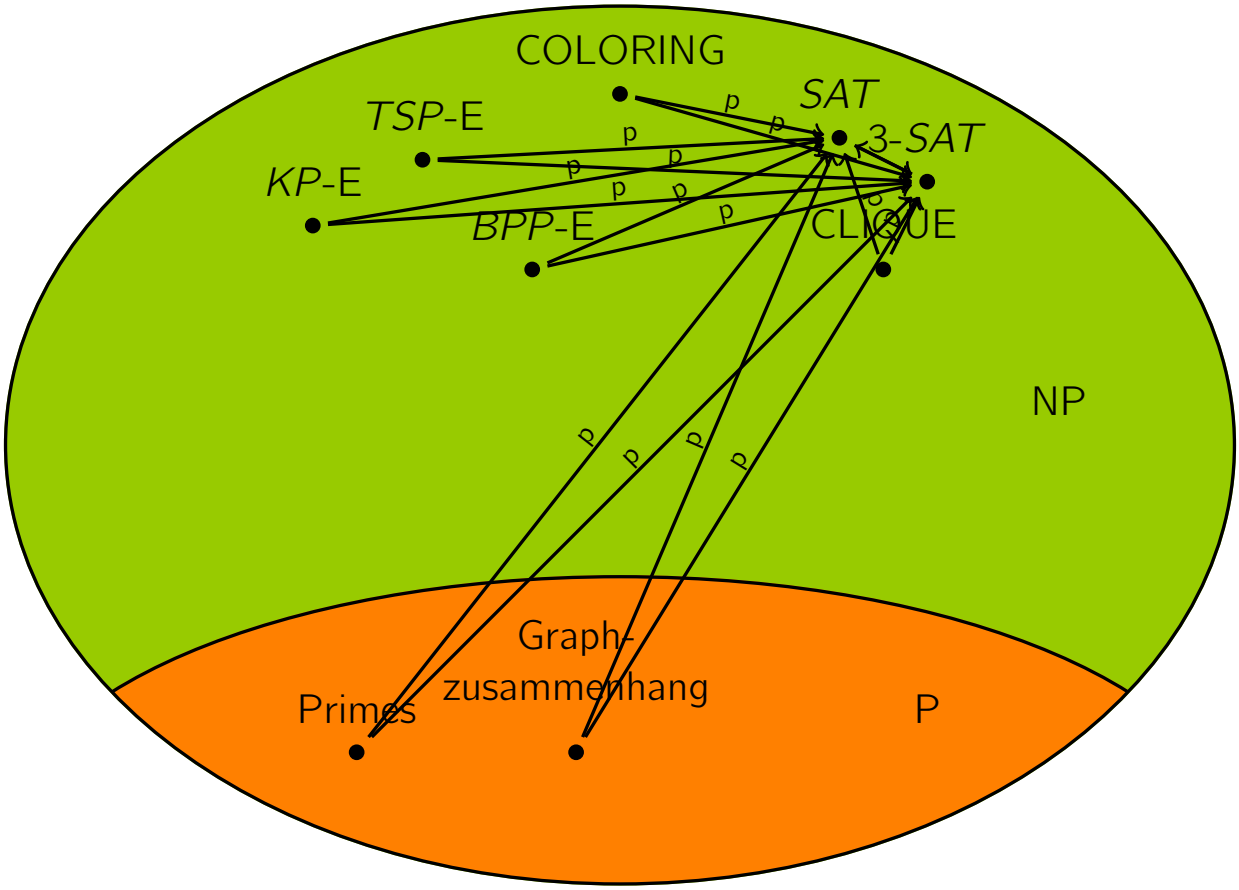
- Durch Anwendung der Klauseltransformation entstehen aus einer k-Klausel eine $(k-1)$-Klausel und eine 3-Klausel.
- Nach $k-3$ Iterationen sind aus einer k-Klausel somit $k-2$ viele 3-Klauseln entstanden.
- Diese Transformation wird solange auf die eingegebene Formel φ angewendet, bis die Formel nur noch 3-Klauseln enthält.
- Wenn n die Anzahl der Literale in φ ist, so werden insgesamt höchstens $n-3$ Klauseltransformationen benötigt.
- Die Laufzeit ist somit polynomiell beschränkt.

□

Korollar

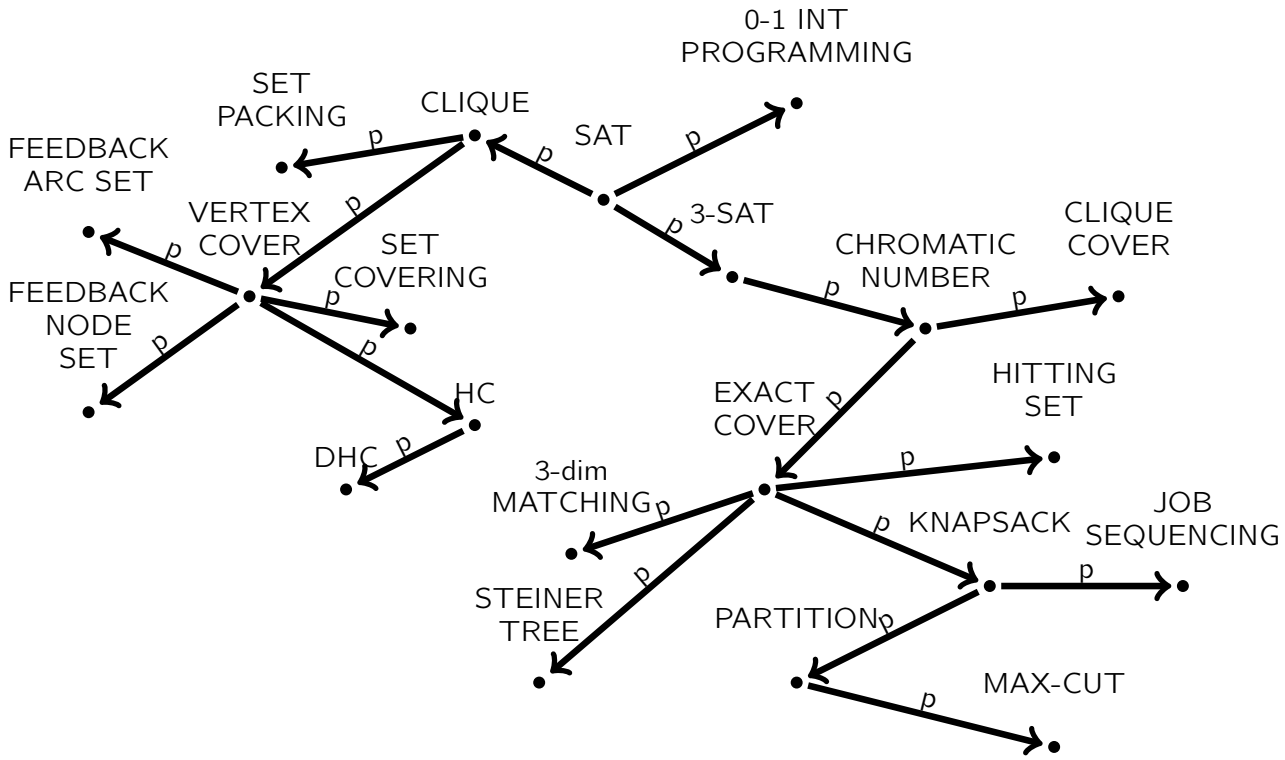
3-SAT ist NP*-vollständig.*

Die Komplexitätslandschaft



Warnung: Dieser Abbildung liegt die Annahme P $\neq$ NP zu Grunde.

Karps Liste mit 21 NP-vollständigen Problemen



Es gibt mittlerweile mehrere tausende Berechnungsprobleme verschiedenster Natur, deren NP-Vollständigkeit bekannt ist.

Vorlesung 16
NP-Vollständigkeit ausgewählter Zahlprobleme

Wdh.: NP-Vollständigkeit

Definition (NP-vollständig)

Ein Problem L heißt NP-vollständig (engl. NP-complete), falls gilt

1. $L \in \mathrm{NP}$, und
2. L ist NP-schwer.

Die Klasse der NP-vollständigen Probleme wird mit NPC bezeichnet.

Satz (Cook und Levin)

SAT ist NP-vollständig.

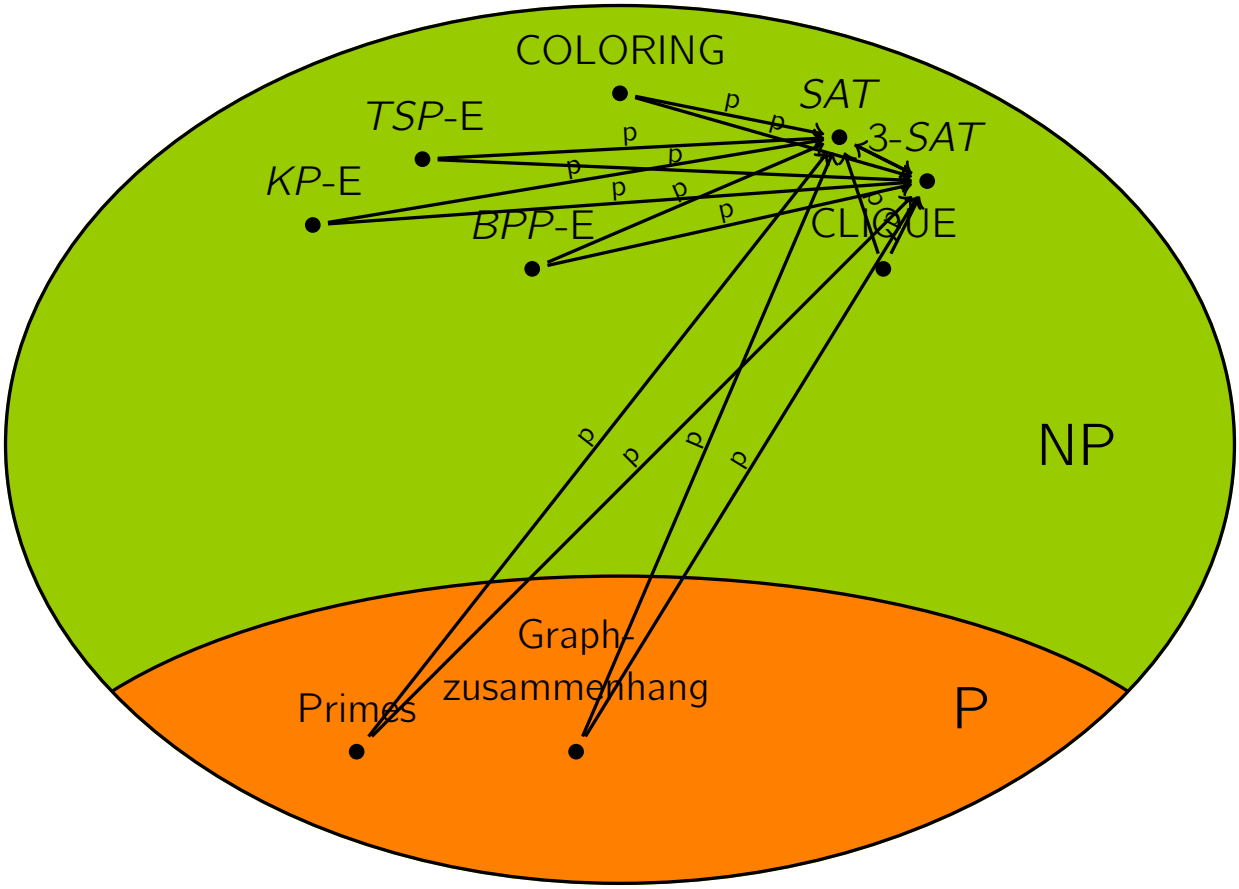
Lemma

3-SAT $\in$ NP und SAT $\leq_p$ 3-SAT.

Korollar

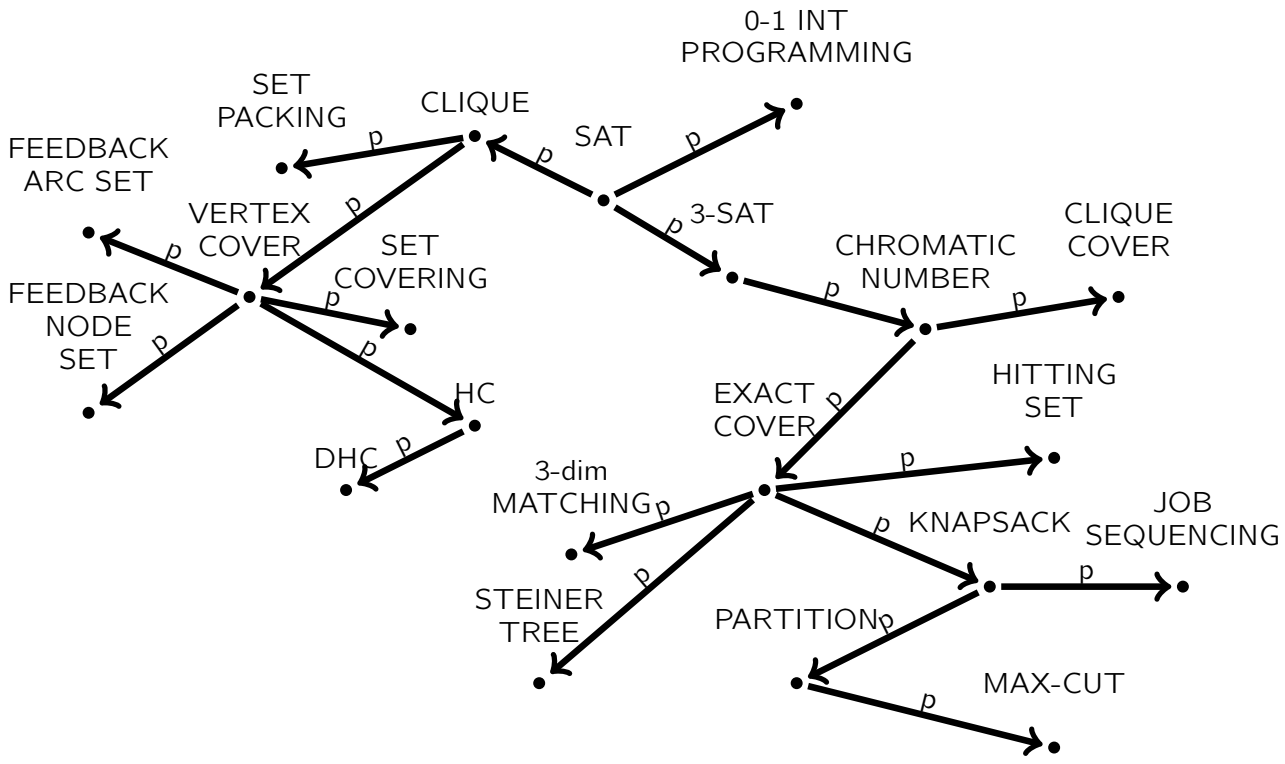
3-SAT ist NP-vollständig.

Die Komplexitätslandschaft



Warnung: Dieser Abbildung liegt die Annahme $P \neq NP$ zu Grunde.

Wdh.: Karps Liste mit 21 NP-vollständigen Problemen



Es gibt mittlerweile mehrere tausende Berechnungsprobleme verschiedenster Natur, deren NP-Vollständigkeit bekannt ist.

Wdh.: Kochrezept für NP-Vollständigkeitsbeweise

Wie beweist man, dass eine Sprache L NP-vollständig ist?

1. Man zeige $L \in \mathrm{NP}$.
2. Man wähle eine NP-vollständige Sprache L'.
3. Man entwerfe eine Funktion f, die Instanzen von L' auf Instanzen von L abbildet. **(Beschreibung der Reduktionsabbildung)**
4. Man zeige, dass f in polynomieller Zeit berechnet werden kann. **(Polynomialzeit)**
5. Man beweise, dass f eine Reduktion ist: Für $x \in \{0,1\}^*$ ist $x \in L'$ genau dann, wenn $f(x) \in L$. **(Korrektheit)**

Das SUBSET-SUM-Problem

Problem (SUBSET-SUM)

Eingabe: $a_1, \ldots, a_N \in \mathbb{N}$, $b \in \mathbb{N}$

Frage: Gibt es $K \subseteq \{1, \ldots, N\}$ *mit* $\sum_{i \in K} a_i = b$*?*

Das SUBSET-SUM-Problem ist in NP enthalten, denn die Lösung K kann als Zertifikat verwendet werden, das in polynomieller Zeit verifiziert werden kann.

NP-Vollständigkeit des SUBSET-SUM-Problems

Satz

SUBSET-SUM ist NP-vollständig.

Beweis:

1.) SUBSET-SUM $\in$ NP: ✓

2.) Um die NP-Schwere des Problems nachzuweisen, beschreiben wir eine Polynomialzeitreduktion von 3-*SAT* auf SUBSET-SUM.

3.) (Beschreibung der Reduktion) Gegeben sei eine Formel φ in 3-KNF. Diese Formel bestehe aus M Klauseln $c_1, \ldots, c_M$ über N Variablen $x_1, \ldots, x_N$.

Für $i \in \{1, \ldots, N\}$ sei

$$\begin{aligned} S(i) &= \{j \in \{1, \ldots, M\} \mid \text{Klausel } c_j \text{ enthält Literal } x_i\} \ , \\ S'(i) &= \{j \in \{1, \ldots, M\} \mid \text{Klausel } c_j \text{ enthält Literal } \bar{x}_i\} \ . \end{aligned}$$

Reduktion 3-SAT $\leq_p$ SUBSET-SUM

Aus der Formel φ in 3-KNF erzeugen wir verschiedene Dezimalzahlen mit jeweils $N + M$ Ziffern.

Die k-te Ziffer einer Zahl a bezeichnen wir dabei mit $a(k)$.

Für jede boolesche Variable x_i, $i \in \{1, \ldots, N\}$ erzeugen wir zwei Zahlen a_i und a'_i, deren Ziffern wie folgt definiert sind:

$$\begin{aligned} a_i(i) = 1 \quad \text{und} \quad \forall j \in S(i) : a_i(N + j) = 1 \ , \\ a'_i(i) = 1 \quad \text{und} \quad \forall j \in S'(i) : a'_i(N + j) = 1 \ . \end{aligned}$$

Alle anderen Ziffern setzen wir auf den Wert 0.

Diese Zahlen bezeichnen wir als *a*-Zahlen.

Reduktion 3-SAT $\leq_p$ SUBSET-SUM

Beispiel:

Gegeben sei die Formel

$$(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \bar{x_3} \vee \bar{x_4}) \ .$$

Aus dieser Formel werden folgende a-Zahlen erzeugt:

$$\begin{aligned} a_1 &= 100010 \\ a'_1 &= 100000 \\ a_2 &= 010011 \\ a'_2 &= 010000 \\ a_3 &= 001010 \\ a'_3 &= 001001 \\ a_4 &= 000100 \\ a'_4 &= 000101 \end{aligned}$$

Reduktion 3-SAT $\leq_p$ SUBSET-SUM

Zusätzlich erzeugen wir zwei sogenannte h-Zahlen h_j und h'_j für jede Klausel j. Diese Zahlen haben nur an der Ziffernposition $N + j$ eine 1, und alle anderen Ziffern sind 0.

Den Summenwert b definieren wir folgendermaßen: Wir setzen $b(k) = 1$ für $1 \leq k \leq N$ und $b(k) = 3$ für $N + 1 \leq k \leq N + M$.

Fortsetzung des Beispiels $(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \bar{x_3} \vee \bar{x_4})$:

Die h-Zahlen und der Summenwert lauten

$$\begin{aligned} h_1 &= 000010 \\ h'_1 &= 000010 \\ h_2 &= 000001 \\ h'_2 &= 000001 \\ b &= 111133 \end{aligned}$$

Reduktion 3-SAT $\leq_p$ SUBSET-SUM

Für eine Formel aus N Variablen und M Klauseln könnten sich beispielsweise die folgenden Zahlen ergeben:

	1	2	3	$\cdots$	N	$N+1$	$N+2$	$\cdots$	$N+M$
a_1	1	0	0	$\cdots$	0	1	0	$\cdots$	$\cdots$
a'_1	1	0	0	$\cdots$	0	0	0	$\cdots$	$\cdots$
a_2	0	1	0	$\cdots$	0	0	1	$\cdots$	$\cdots$
a'_2	0	1	0	$\cdots$	0	1	0	$\cdots$	$\cdots$
a_3	0	0	1	$\cdots$	0	1	1	$\cdots$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
a_N	0	0	0	$\cdots$	1	0	0	$\cdots$	$\cdots$
a'_N	0	0	0	$\cdots$	1	0	1	$\cdots$	$\cdots$
h_1	0	0	0	$\cdots$	0	1	0	$\cdots$	0
h'_1	0	0	0	$\cdots$	0	1	0	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
h_M	0	0	0	$\cdots$	0	0	0	$\cdots$	1
h'_M	0	0	0	$\cdots$	0	0	0	$\cdots$	1
b	1	1	1	$\cdots$	1	3	3	$\cdots$	3

Reduktion 3-SAT $\leq_p$ SUBSET-SUM

4.) (Polynomialzeit) Die Eingabezahlen zu SUBSET-SUM können in polynomieller Zeit erzeugt werden. (Die Zahlenwerte können aber natürlich exponentiell groß sein).

5.) (Korrektheit)

Beobachtung

Bei der Addition einer beliebigen Teilmenge der a-Zahlen und der h-Zahlen gibt es keinen Additionsübertrag von Stelle zu Stelle, weil höchstens fünf Ziffern pro Spalte den Wert 1 haben.

Anmerkung: Die Beobachtung beruht darauf, dass wir mit Dezimalziffern, d.h. zur Basis 10, rechnen. De facto wäre es auch ausreichend, wenn wir zur Basis 6 rechnen würden.

Reduktion 3-SAT $\leq_p$ SUBSET-SUM: *Korrektheit*

Zu zeigen: φ erfüllbar $\Rightarrow$ es gibt eine Teilmenge der a- und h-Zahlen, deren Summe b ist

Angenommen, es gibt eine erfüllende Belegung x^* für φ.

- Falls $x_i^* = 1$, so *wähle* a_i aus, ansonsten wähle a_i'.
- Sei A die Summe der ausgewählten a-Zahlen.
- Da für jedes $i \in \{1, \ldots, N\}$ entweder a_i oder a_i' ausgewählt wurde, gilt $A(i) = 1$.
- Zudem gilt $A(N + j) \in \{1, 2, 3\}$ für $1 \leq j \leq M$, weil in jeder Klausel mindestens ein und höchstens drei Literale erfüllt werden.
- Falls $A(N + j) < 3$, so können wir zusätzlich h_j oder h_j und h_j' auswählen, um exakt den geforderten Wert 3 an Ziffernposition $N + j$ der Summe zu erhalten.

Also gibt es eine Teilmenge mit Summenwert b.

Reduktion 3-SAT $\leq_p$ SUBSET-SUM: *Korrektheit*

Zu zeigen: es gibt eine Teilmenge der a- und h-Zahlen, deren Summe b ist $\Rightarrow$ φ erfüllbar

Sei A die Summe einer Teilmenge K_A der a-Zahlen und H die Summe einer Teilmenge der h-Zahlen, so dass gilt $A + H = b$.

In K_A ist für jedes $i \in \{1, \ldots, N\}$ genau eine der beiden a-Zahlen a_i oder a_i' enthalten, denn ansonsten wäre $A(i) \neq 1$.

Setze $x_i = 1$, falls $a_i \in K_A$, und $x_i = 0$, sonst.

Zu zeigen: x ist eine erfüllende Belegung für φ

- Es gilt $A(N + j) \geq 1$ für $1 \leq j \leq M$, denn ansonsten wäre $A(N + j) + H(N + j) \leq A(N + j) + 2 < 3$.
- Dadurch ist sichergestellt, dass in jeder Klausel mindestens eines der Literale den Wert 1 hat, so dass φ erfüllt ist.

Damit ist die Korrektheit der Reduktion nachgewiesen. □

NP-Vollständigkeit von PARTITION

Problem (PARTITION)

Eingabe: $a_1, \ldots, a_N \in \mathbb{N}$

Frage: Gibt es $K \subseteq \{1, \ldots, N\}$ *mit* $\sum_{i \in K} a_i = \sum_{i \in \{1,\ldots,N\} \setminus K} a_i$*?*

PARTITION ist ein Spezialfall von SUBSET-SUM, da die gestellte Frage äquivalent zur der Frage ist, ob es eine Teilmenge K mit Summenwert $b = \frac{1}{2} \sum_{i=1}^{N} a_i$ gibt.

NP-Vollständigkeit von PARTITION

Satz

PARTITION ist NP*-vollständig.*

Beweis:

1.) PARTITION ist offensichtlich in NP, weil es als Spezialfall von SUBSET-SUM aufgefasst werden kann.

2.) Um zu zeigen, dass PARTITION NP-schwer ist, zeigen wir SUBSET-SUM $\leq_p$ PARTITION.

Reduktion von SUBSET-SUM auf PARTITION

3.) Die Eingabe von SUBSET-SUM sei $a_1, \ldots, a_N \in \mathbb{N}$ und $b \in \mathbb{N}$.

Es sei $A = \sum_{i=1}^{N} a_i$.

Wir bilden diese Eingabe für SUBSET-SUM auf eine Eingabe für PARTITION ab, die aus $N+2$ Zahlen $a'_1, \ldots, a'_{N+2}$ bestehe.

Dazu setzen wir

- $a'_i = a_i$ für $1 \leq i \leq N$,
- $a'_{N+1} = 2A - b$, und
- $a'_{N+2} = A + b$.

In der Summe ergeben diese $N+2$ Zahlen den Wert $4A$.

Diese Zahlen bilden genau dann eine Ja-Instanz von PARTITION, wenn es eine Teilmenge der Zahlen $a'_1, \ldots, a'_{N+2}$ mit Summenwert $2A$ gibt.

Reduktion von SUBSET-SUM auf PARTITION

4.) Die Reduktion ist in polynomieller Zeit berechenbar.

5.) Wir zeigen: Es existiert eine Lösung für PARTITION $\Rightarrow$ es existiert eine Lösung für SUBSET-SUM

- Wenn es eine geeignete Aufteilung der Eingabezahlen für PARTITION gibt, so können a'_{N+1} und a'_{N+2} dabei nicht in derselben Teilmenge sein, denn $a'_{N+1} + a'_{N+2} = 3A$.
- Deshalb ergibt sich auch eine Lösung für SUBSET-SUM, denn diejenigen Zahlen aus $a'_1, \ldots, a'_N$, die sich in derselben Teilmenge wie a'_{N+1} befinden, summieren sich auf zu $2A - a'_{N+1} = b$.

Reduktion von SUBSET-SUM auf PARTITION

Wir zeigen: Es existiert eine Lösung für SUBSET-SUM $\Rightarrow$ es existiert eine Lösung für PARTITION

- Wenn es eine Teilmenge der Zahlen $a_1, \ldots, a_N$ mit Summenwert b gibt, so gibt es auch eine Teilmenge der Zahlen $a'_1, \ldots, a'_N$ mit diesem Summenwert.
- Wir können die Zahl $a'_{N+1} = 2A - b$ zu dieser Teilmenge hinzufügen und erhalten dadurch eine Teilmenge mit Summenwert $2A$. $\square$

Bin Packing ist NP-vollständig

Problem (Bin Packing Problem – BPP)

Eingabe: $b \in \mathbb{N}$, $w_1, \ldots, w_N \in \{1, \ldots, b\}$

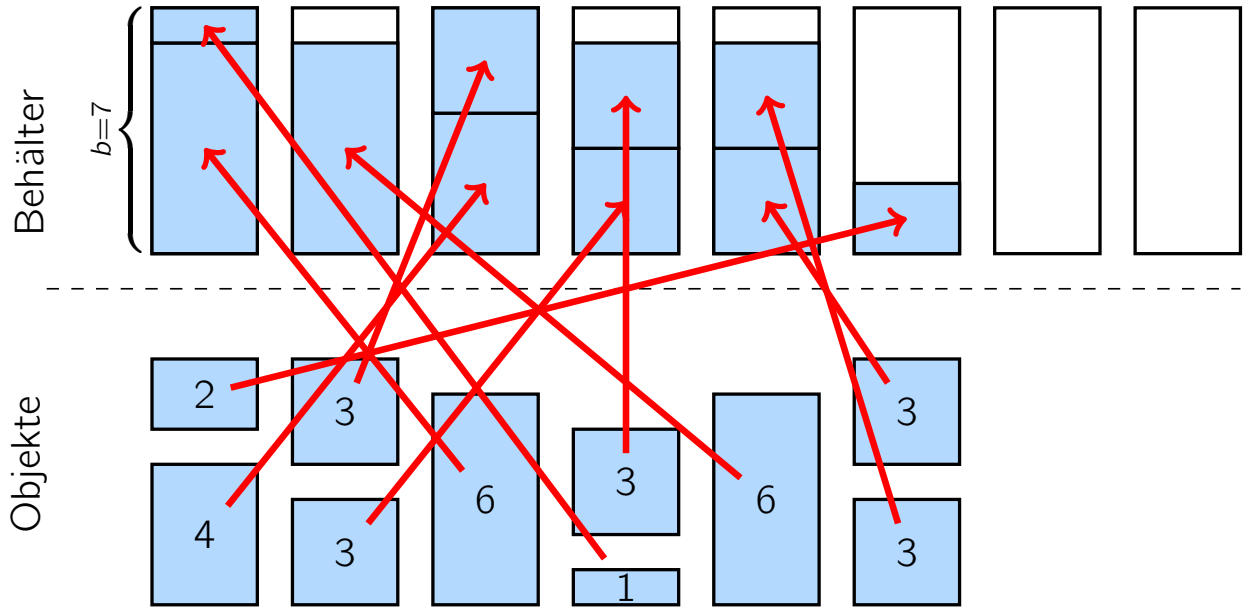
zulässige Lösungen: $k \in \mathbb{N}$ *und Funktion* $f : \{1, \ldots, N\} \to \{1, \ldots, k\}$,

$$\text{so dass } \forall i \in \{1, \ldots, k\} : \sum_{j \in f^{-1}(i)} w_j \leq b$$

Zielfunktion: *Minimiere* k *(= Anzahl Behälter)*

Entscheidungsvariante (BPP-E): Zusätzlich ist $k \in \mathbb{N}$ gegeben. Passen die Objekte in k Behälter?

Bin Packing Problem – Beispiel



Eine Lösung die $k = 6$ Behälter verwendet

Bin Packing ist NP-vollständig

Satz

BPP-E ist NP*-vollständig.*

Beweis:

1.) BPP-E $\in$ NP haben wir bereits gezeigt.

(2.-5.) Die NP-Schwere ergibt sich durch eine triviale Reduktion von PARTITION:

Setze $k = 2$, $w_i = a_i$ für $1 \leq i \leq N$ und $b = \left\lfloor \frac{1}{2} \sum_{i=1}^{N} w_i \right\rfloor$.

□

Das Rucksackproblem ist NP-vollständig

Problem (Entscheidungsvariante des Rucksackproblems – KP-E)

Eingabe: $B, P \in \mathbb{N}$, $w_1, \ldots, w_N \in \{1, \ldots, B\}$, $p_1, \ldots, p_N \in \mathbb{N}$

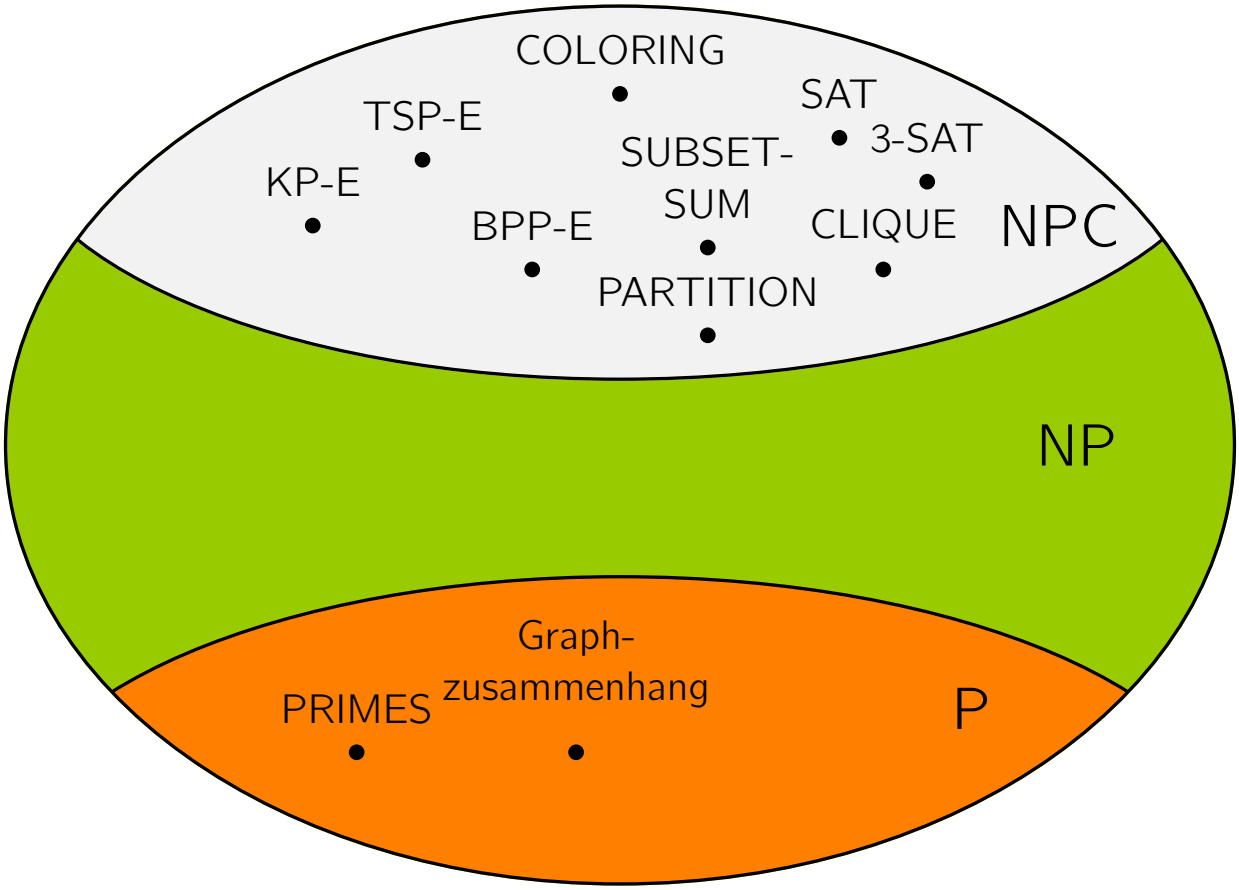
Frage: *Gibt es* $K \subseteq \{1, \ldots, N\}$ *mit* $\sum_{i \in K} w_i \leq B$ *und* $\sum_{i \in K} p_i \geq P$*?*

Korollar

KP-E ist NP*-vollständig.*

Beweis durch einfache Reduktion von SUBSET-SUM (Wie?)

Die Komplexitätslandschaft



Warnung: Dieser Abbildung liegt die Annahme P $\neq$ NP zu Grunde.

Vorlesung 17
NP-Vollständigkeit ausgewählter Graphprobleme

Wdh.: NP-Vollständigkeit von Zahlproblemen

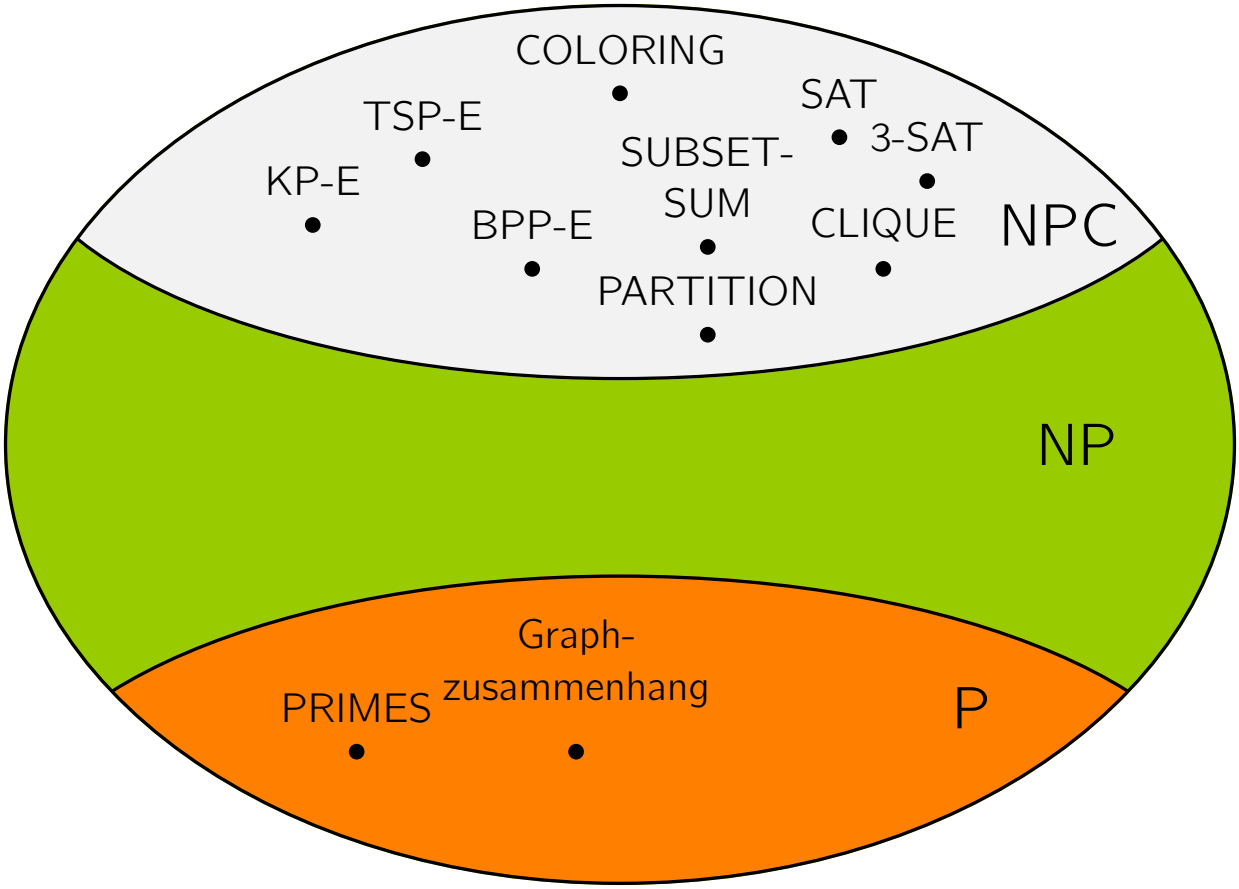
Satz
SUBSET-SUM ist NP-vollständig.

Satz
PARTITION ist NP-vollständig.

Satz
BPP-E ist NP-vollständig.

Satz
KP-E ist NP-vollständig.

Wdh.: Die Komplexitätslandschaft



Warnung: Dieser Abbildung liegt die Annahme $\mathsf{P} \neq \mathsf{NP}$ zu Grunde.

Wdh.: Kochrezept für NP-Vollständigkeitsbeweise

Wie beweist man, dass eine Sprache L NP-vollständig ist?

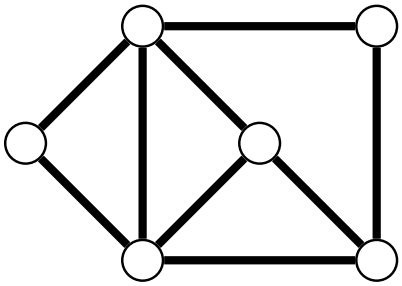
1. Man zeige $L \in \mathsf{NP}$.
2. Man wähle eine NP-vollständige Sprache L'.
3. Man entwerfe eine Funktion f, die Instanzen von L' auf Instanzen von L abbildet. (**Beschreibung der Reduktionsabbildung**)
4. Man zeige, dass f in polynomieller Zeit berechnet werden kann. (**Polynomialzeit**)
5. Man beweise, dass f eine Reduktion ist: Für $x \in \{0,1\}^*$ ist $x \in L'$ genau dann, wenn $f(x) \in L$. (**Korrektheit**)

NP-Vollständigkeit von CLIQUE

Wie erinnern uns an das Cliquenproblem.

Problem (CLIQUE)

Eingabe: *Graph* $G = (V, E)$, $k \in \{1, \ldots, |V|\}$
Frage: *Hat* G *eine* k*-Clique?*



$k = 4$

Der Graph hat keine Clique der Größe 4.

Satz

CLIQUE ist NP*-vollständig.*

Beweis der NP-Vollständigkeit von CLIQUE

1.) Da wir schon wissen, dass das Cliquenproblem in NP ist, müssen wir zum Nachweis der NP-Vollständigkeit nur noch die NP-Schwere nachweisen.

2.) Dazu zeigen wir SAT $\leq_p$ CLIQUE.

3.) Wir beschreiben eine polynomiell berechenbare Funktion f, die eine KNF-Formel φ in einen Graphen $G = (V, E)$ und eine Zahl $k \in \mathbb{N}$ transformiert, so dass gilt:

$$\varphi \text{ ist erfüllbar } \Leftrightarrow \ G \text{ hat eine } k\text{-Clique } .$$

Beweis der NP-Vollständigkeit von CLIQUE

Beschreibung der Funktion f:

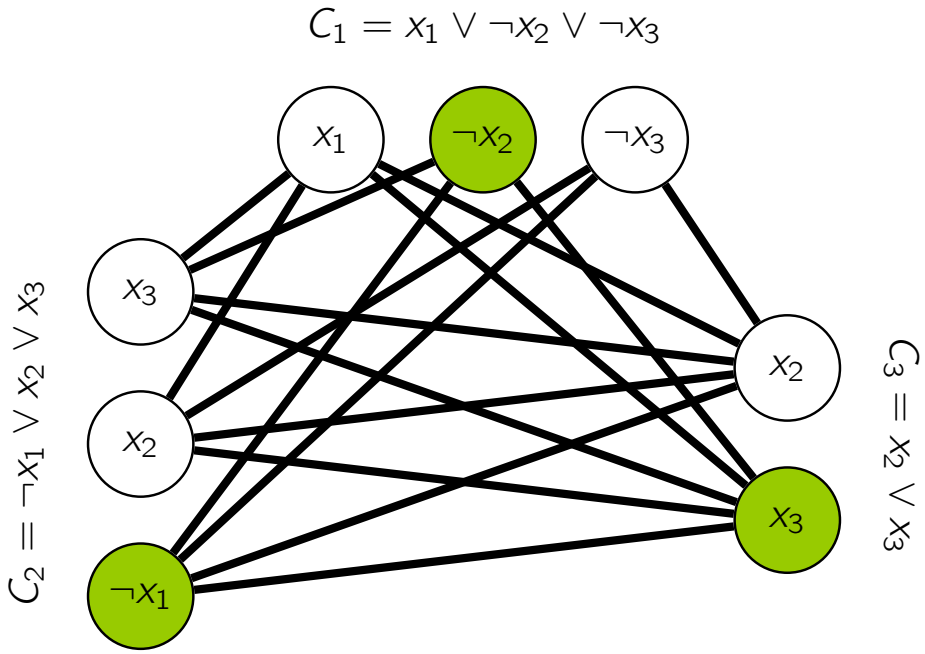
- Seien $C_1, \ldots, C_m$ die Klauseln von φ.
- Sei k_i die Anzahl an Literalen in Klausel C_i.
- Seien $\ell_{i,1}, \ldots, \ell_{i,k_i}$ die Literale in Klausel C_i.
- Identifiziere Literale und Knoten, d.h., setze

$$V \;=\; \{\ell_{i,j} \mid 1 \leq i \leq m,\; 1 \leq j \leq k_i\} \;.$$

- Zwei Knoten werden mit einer Kante verbunden, wenn
  - sie aus verschiedenen Klauseln stammen und
  - ihre Literale nicht Negationen voneinander sind.
- Setze $k = m$.

Beweis der NP-Vollständigkeit von CLIQUE – Illustration

Beispiel: $\varphi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3)$



Erfüllende Belegung: $x_1 = 0$, $x_2 = 0$, $x_3 = 1$.

Beweis der NP-Vollständigkeit von CLIQUE

5.) Korrektheit der Transformation:

Zu zeigen: φ erfüllbar $\Rightarrow$ G hat m-Clique

Betrachte eine beliebige erfüllende Belegung. Pro Klausel wähle ein erfülltes Literal beliebig aus. Sei U die Menge dieser Literale. Wir behaupten, U bildet eine m-Clique.

Begründung: Gemäß der Definition ist $|U| = m$. Seien ℓ und ℓ' zwei unterschiedliche Literale aus U. Wir müssen zeigen, dass ℓ und ℓ' durch eine Kante verbunden sind:

- Nach Konstruktion stammen ℓ und ℓ' aus verschiedenen Klauseln.
- Da ℓ und ℓ' gleichzeitig erfüllt sind, sind sie nicht Negationen voneinander.

Also gibt es eine Kante zwischen ℓ und ℓ'.

Beweis der NP-Vollständigkeit von CLIQUE

Zu zeigen: G hat m-Clique $\Rightarrow$ φ erfüllbar

- Sei U eine m-Clique in G.
- Dann gehören die Literale in U zu verschiedenen Klauseln.
- U enthält somit genau ein Literal pro Klausel in φ.
- Diese Literale können alle gleichzeitig erfüllt werden, da kein Literal positiv und negiert vorkommt.
- Also ist φ erfüllbar.

4.) Die Funktion f ist in Polynomialzeit berechenbar. $\square$

Hamiltonkreisprobleme

Problem (Hamiltonkreis – Hamiltonian Circuit – HC)

Eingabe: Graph $G = (V, E)$

Frage: Gibt es einen Hamiltonkreis in G?

Problem (Gerichteter Hamiltonkreis – Directed HC – DHC)

Eingabe: gerichteter Graph $G = (V, E)$

Frage: Gibt es einen Hamiltonkreis in G?

$HC \leq_p DHC$

Lemma

$HC \leq_p DHC$.

Beweis:

Reduktion: Für HC liege ein ungerichteter Graph G vor. Wir transformieren G in einen gerichteten Graphen G', indem wir jede ungerichtete Kante durch zwei entgegengesetzte gerichtete Kanten ersetzen.

Polynomialzeit: Diese lokale Ersetzung ist offensichtlich in polynomieller Zeit möglich.

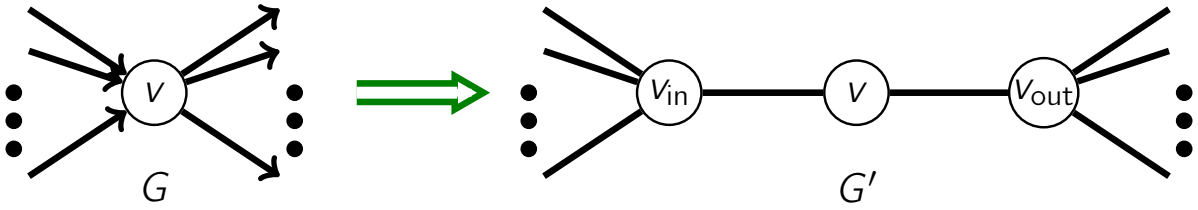
Korrektheit: Nach Konstruktion hat G genau dann einen Hamiltonkreis, wenn auch G' einen Hamiltonkreis hat. □

$DHC \leq_p HC$

Lemma
$DHC \leq_p HC$.

Beweis:

Reduktion: Gegeben sei nun ein gerichteter Graph $G = (V, E)$. Aus G konstruieren wir wieder mittels lokaler Ersetzung einen ungerichteten Graphen G':



Interpretation: v_{in} ist der Eingangsknoten für v, und v_{out} ist der Ausgangsknoten für v.

$DHC \leq_p HC$ – Fortsetzung Beweis

Korrektheit:
Wir müssen zeigen: G hat genau dann einen Hamiltonkreis, wenn auch G' einen Hamiltonkreis hat.

Jede Rundreise in G kann offensichtlich in eine Rundreise in G' transformiert werden.

Aber wie sieht es mit der Umkehrrichtung aus?

- Eine Rundreise in G', die einen Knoten v direkt nach v_{in} besucht, besucht jeden Knoten v' direkt nach v'_{in}.
- Eine Rundreise in G', die einen Knoten v direkt nach v_{out} besucht, besucht jeden Knoten v' direkt nach v'_{out}. Aber diese Rundreise können wir rückwärts ablaufen.

Also kann auch jeder Hamiltonkreis in G' in einen Hamiltonkreis in G transformiert werden. □

NP-Vollständigkeit von HC und DHC

Satz

HC und DHC sind NP*-vollständig.*

Beweis:

Beide Probleme sind offensichtlich in NP, da man die Kodierung eines Hamiltonkreises in polynomieller Zeit auf ihre Korrektheit überprüfen kann.

Da HC und DHC gegenseitig aufeinander polynomiell reduzierbar sind, genügt es die NP-Schwere eines der beiden Probleme nachzuweisen.

Wir zeigen die NP-Schwere von DHC durch eine polynomielle Reduktion von SAT auf DHC.

NP-Vollständigkeit von HC und DHC – Fortsetzung Beweis

(Reduktion:)

Wir präsentieren eine polynomiell berechenbare Funktion f, die eine KNF-Formel φ mit Variablen

$$x_1, \ldots, x_N$$

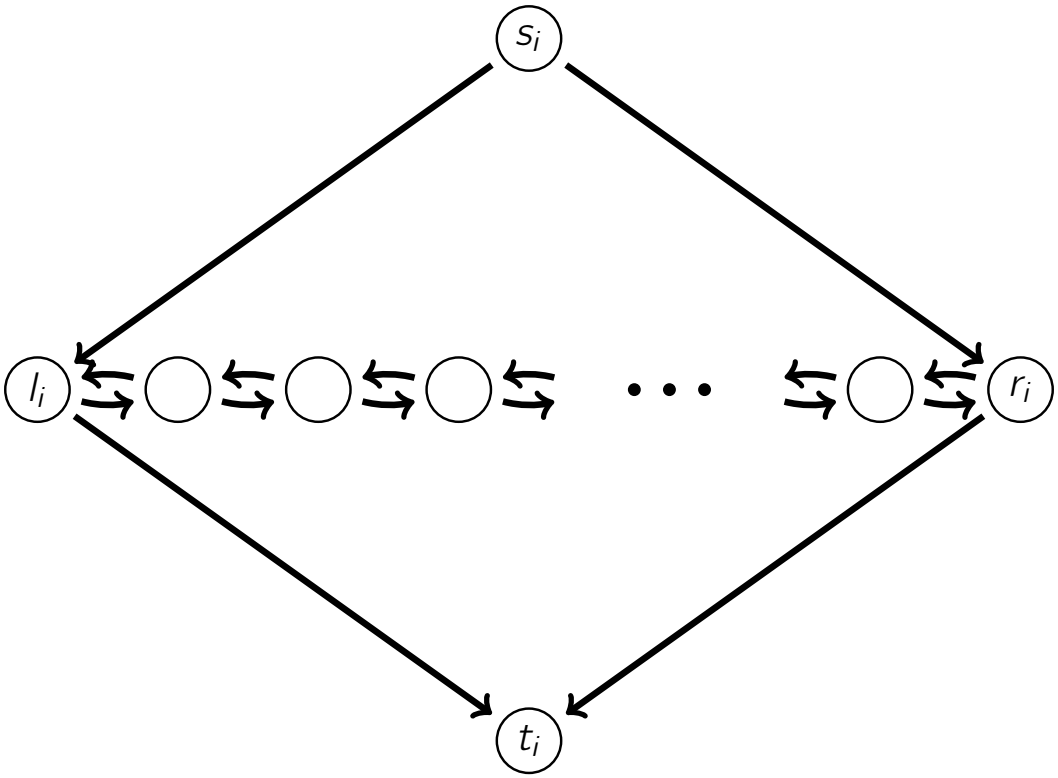
und Klauseln

$$c_1, \ldots, c_M$$

auf einen gerichteten Graphen $G = (V, E)$ abbildet, der genau dann einen Hamiltonkreis hat, wenn φ erfüllbar ist.

NP-Vollständigkeit von HC und DHC – Beweis

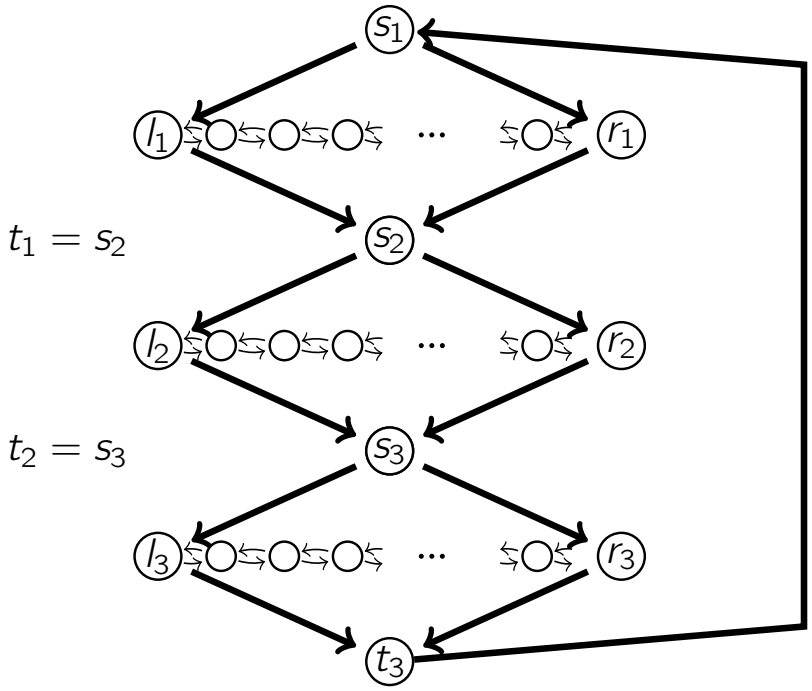
Für jede Variable x_i enthalte der Graph G die folgende Struktur G_i.



Diese Struktur heißt Diamantengadget.

NP-Vollständigkeit von HC und DHC – Fortsetzung Beweis

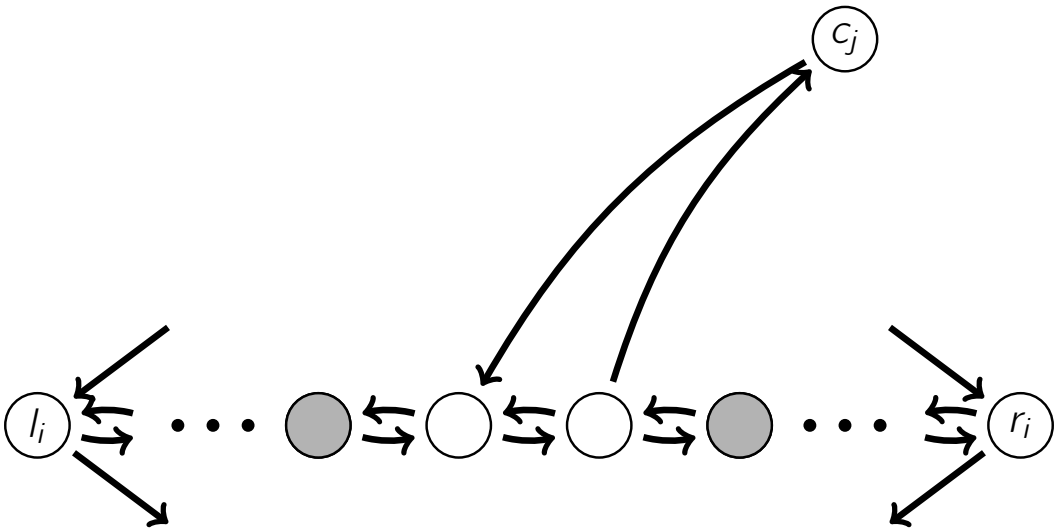
Diese N Gadgets werden miteinander verbunden, indem wir die Knoten t_i und s_{i+1} (für $1 \leq i \leq N-1$) sowie t_N und s_1 miteinander identifizieren.



NP-Vollständigkeit von HC und DHC – Fortsetzung Beweis

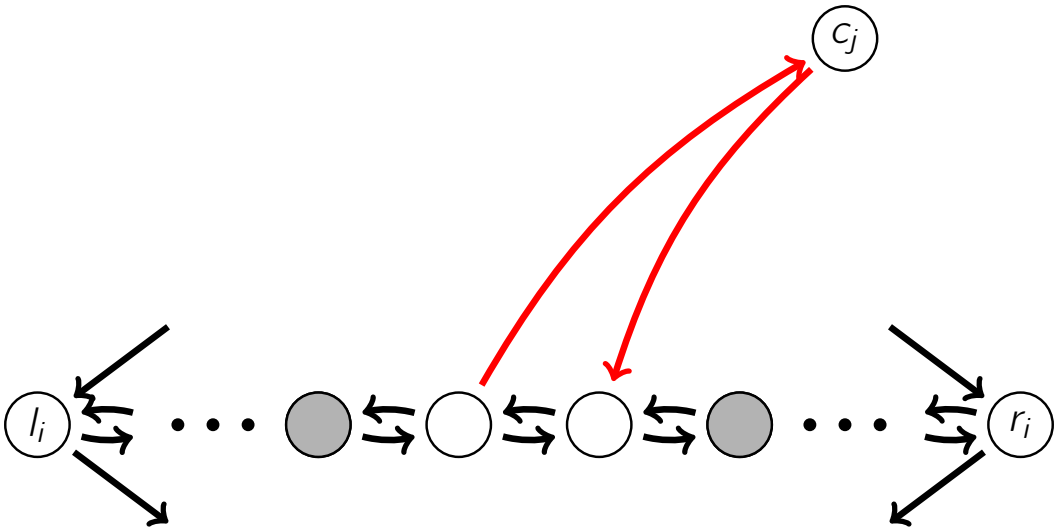
Jetzt fügen wir einen weiteren Knoten für jede Klausel c_j ein.

Falls das Literal x_i in Klausel c_j enthalten ist, so verbinden wir das Gadget G_i wie folgt mit dem Klauselknoten c_j:



NP-Vollständigkeit von HC und DHC – Fortsetzung Beweis

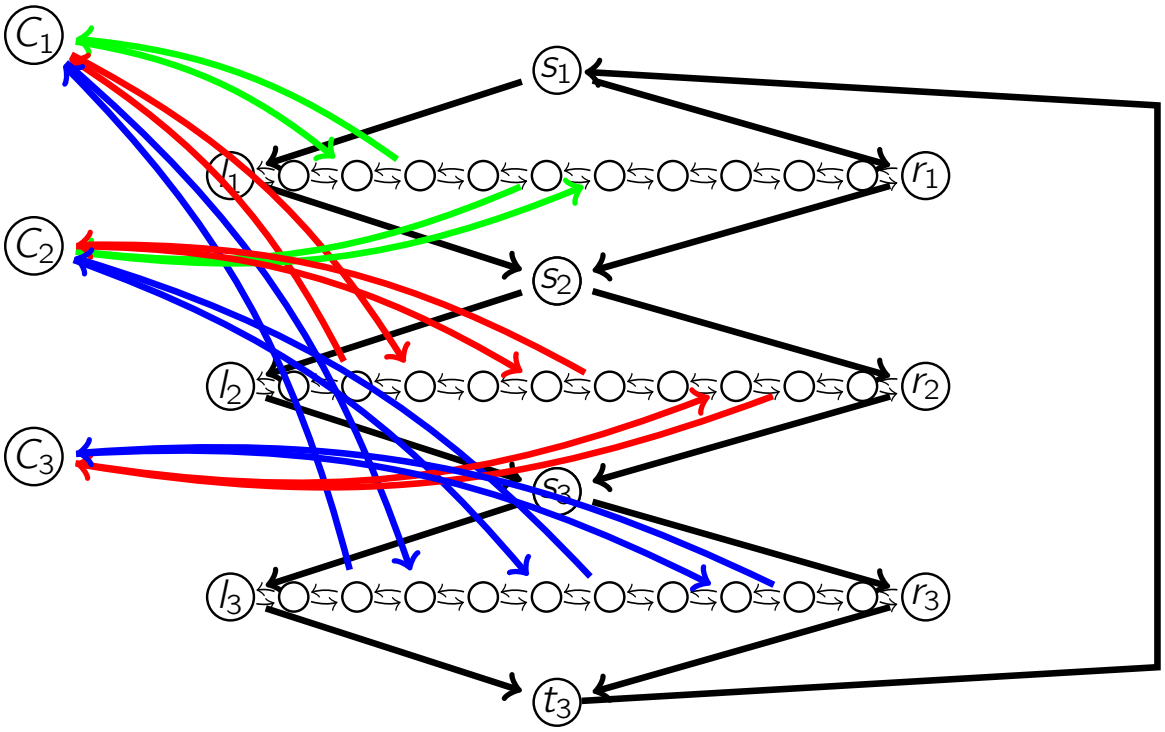
Falls das Literal $\bar{x}_i$ in Klausel c_j enthalten ist, so verbinden wir das Gadget G_i wie folgt mit dem Klauselknoten c_j:



Ist es nach Hinzunahme der Klauselknoten möglich, dass eine Rundreise zwischen den Gadgets hin- und herspringt, statt sie in der vorgesehenen Reihenfolge zu besuchen? – Nein. (Warum?)

NP-Vollständigkeit von HC und DHC – Illustration

$$\varphi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3)$$



NP-Vollständigkeit von HC und DHC – Fortsetzung Beweis

Korrektheit:

Zu zeigen: G hat einen Hamiltonkreis $\Rightarrow$ φ ist erfüllbar

- Wird ein Klauselknoten c_j aus einem Gadget G_i heraus von links nach rechts durchlaufen, so muss gemäß unserer Konstruktion die Klausel c_j das Literal x_i enthalten.
- Also wird diese Klausel durch die mit der Laufrichtung von links nach rechts assoziierten Belegung $x_i = 1$ erfüllt.
- Bei einer Laufrichtung von rechts nach links, die mit der Belegung $x_i = 0$ assoziiert ist, wird die Klausel ebenso erfüllt, weil sie in diesem Fall das Literal $\bar{x}_i$ enthält.

Also erfüllt die mit der Rundreise assoziierte Belegung alle Klauseln.

NP-Vollständigkeit von HC und DHC – Fortsetzung Beweis

Zu zeigen: φ ist erfüllbar $\Rightarrow$ G hat einen Hamiltonkreis

- Eine Belegung beschreibt, in welcher Richtung die Gadgets $G_1, \ldots, G_N$ jeweils durchlaufen werden.
- Klauselknoten c_j können wir in die Rundreise einbauen, indem wir eine der Variablen x_i auswählen, die c_j erfüllt, und c_j vom Gadget G_i aus besuchen.
- Sollte c_j für $x_i = 1$ erfüllt sein, so ist x_i positiv in c_j enthalten und somit ist ein Besuch von c_j beim Durchlaufen des Gadgets G_i von links nach rechts möglich.
- Sollte c_j hingegen für $x_i = 0$ erfüllt sein, so ist die Variable negiert in der Klausel enthalten und der Besuch von c_j kann beim Durchlaufen des Gadgets G_i von rechts nach links erfolgen.

Also können alle Klauselknoten in die Rundreise eingebunden werden. □

NP-Vollständigkeit von TSP

Satz

TSP-E ist NP*-schwer.*

Beweis: Wir zeigen, dass TSP sogar dann NP-schwer ist, wenn nur die Kantengewichte 1 und 2 verwendet werden.

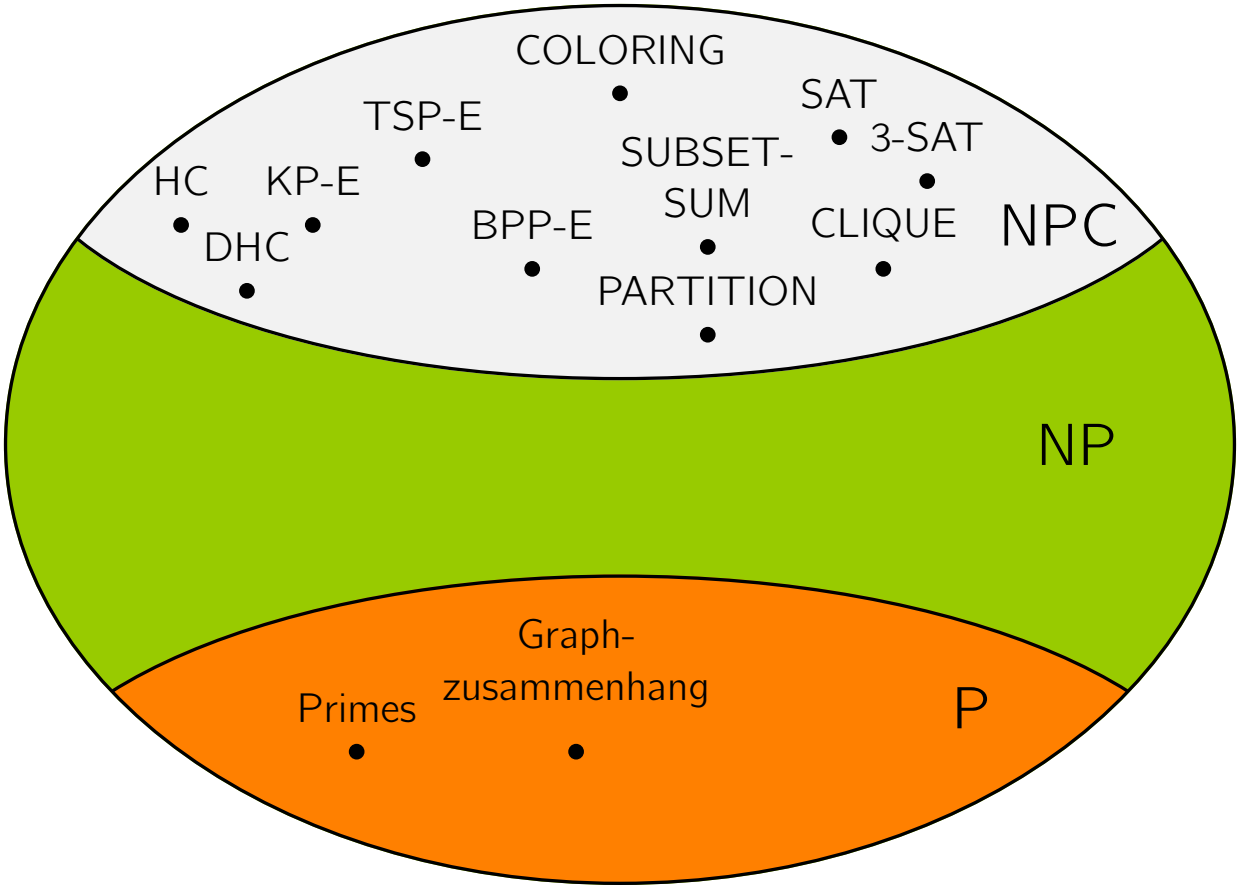
Diese eingeschränkte Variante von TSP nennen wir $\{1,2\}$-TSP. Wir zeigen HC $\leq_p$ $\{1,2\}$-TSP-E.

Aus dem Graphen $G = (V, E)$ für HC erzeugen wir einen vollständigen Graphen $G' = (V, E')$ mit Kosten

$$c(u,v) = \begin{cases} 1 & \text{falls } \{u,v\} \in E \\ 2 & \text{falls } \{u,v\} \notin E \end{cases}$$

G hat genau dann einen Hamiltonkreis, wenn G' eine TSP-Tour der Länge höchstens n hat. □

Die Komplexitätslandschaft



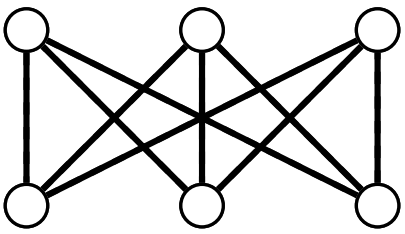
Warnung: Dieser Abbildung liegt die Annahme $P \neq NP$ zu Grunde.

Das Graphisomorphieproblem GI

Zwei Graphen G_1, G_2 sind isomorph, wenn es eine Bijektion von den Knoten von G_1 auf die Knoten von G_2 gibt, die Adjazenz und Nicht-Adjazenz erhält.

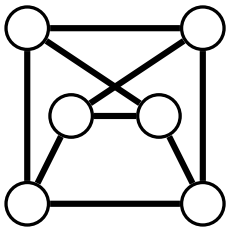
Eine solche Bijektion heißt Isomorphismus.

isomorphe Graphen



Problem (Graphisomorphieproblem GI)

Eingabe: *Zwei Graphen G_1, G_2*
Frage: *Gibt es einen Isomorphismus von G_1 nach G_2?*



Komplexität des Graphisomorphieproblems

Was ist die Komplexität des Graphisomorphieproblems?

- Es ist $GI \in \text{NP}$, da man einen Isomorphismus als Zertifikat verwenden kann.
- Damit gilt $GI \in \text{EXPTIME}$, also ist GI insbesondere berechenbar.

Gibt es einen effizienten Algorithmus für das Graphisomorphieproblem?

Folgende Fragen sind derzeit noch unbekannt:

- $GI \in P$?
- Ist GI NP-vollständig?
- $GI \in$ co-NP?

NP-intermediate

Definition (NP-intermediate)

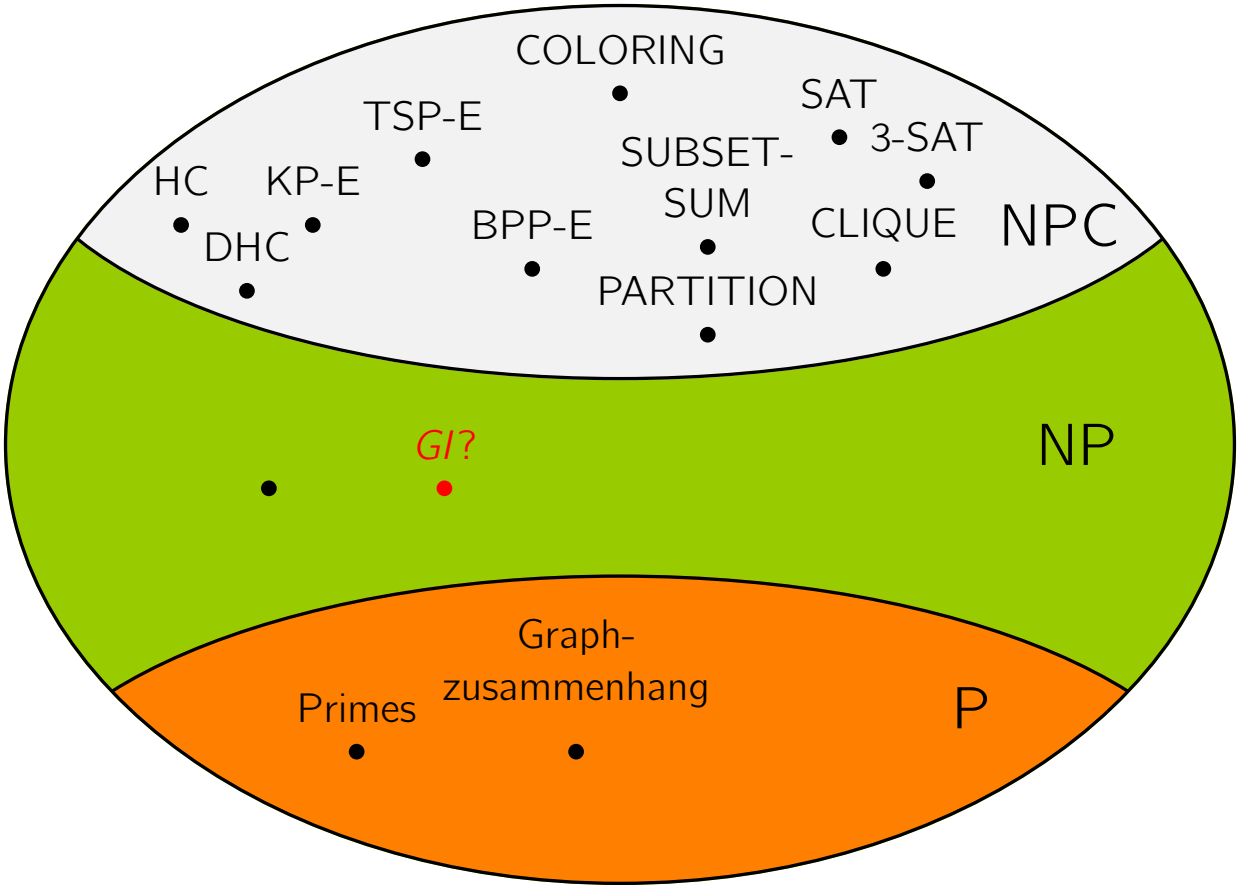
Ein Entscheidungsproblem L heißt NP-intermediate, wenn $L \in NP$ aber sowohl $L \notin \text{P}$ als auch $L \notin \text{NPC}$ gilt.

- Möglicherweise könnte GI NP-intermediate sein.

Satz von Ladner [1975]

Wenn $\text{P} \neq \text{NP}$, dann gibt es Probleme, die NP-intermediate ist.

Die Komplexitätslandschaft



Warnung: Dieser Abbildung liegt die Annahme $\mathsf{P} \neq \mathsf{NP}$ zu Grunde.

Graphisomorphieproblem – Algorithmen

- Trivialer Weise kann man *GI* in Zeit $O(n! \cdot n^2)$ lösen.
- Lange Zeit hatte der beste (bekannte) Algorithmus für *GI* eine Laufzeit von $2^{O(\sqrt{n \log n})}$. [Babai, Luks] (1983)
- Jetzt ist ein Algorithmus bekannt, der das Graphisomorphieproblem in Zeit $2^{p(\log n)}$ löst, wobei p ein Polynom ist. [Babai] (2015)
- Dieser Algorithmus verwendet die algorithmische und strukturelle Theorie der Permutationsgruppen.

Eine Laufzeit der Form $2^{p(\log n)}$ für ein Polynom p nennt man quasi-polynomiell.

Die Exponential time hypothesis – ETH

- Wenn P $\neq$ NP, dann kann man 3-SAT nicht in polynomieller Zeit lösen.
- Aber dann ist immer noch unklar, wie schnell man 3-SAT lösen kann.

Hypothese (Exponential time hypothesis – ETH)

Es gibt ein $\delta > 0$, so dass kein Algorithmus 3-SAT in Zeit $O(2^{\delta n})$ löst.

Diese Hypothese impliziert $P \neq NP$.

$2^{p(n)}$	$\sim$	exponentiell
$2^{p(\log n)}$	$\sim$	quasi-polynomiell

Beobachtung: Wenn $L \leq_p L'$ und L' in quasi-polynomieller Zeit gelöst werden kann, dann kann auch L in quasi-polynomieller Zeit gelöst werden.

Es folgt: Wenn GI NP-vollständig ist, dann ist 3-SAT in quasi-polynomieller Zeit lösbar, und damit wäre die ETH falsch.

Vorlesung 18
Ausblick: Algorithmen und Komplexität

Wdh.: NP-Vollständigkeit

Definition (NP-vollständig)

Ein Problem L heißt NP-vollständig (engl. NP-complete), falls gilt

1. $L \in \mathrm{NP}$, und
2. L ist NP-schwer.

Die Klasse der NP-vollständigen Probleme wird mit NPC bezeichnet.

Satz (Cook und Levin)

SAT ist NP-vollständig.

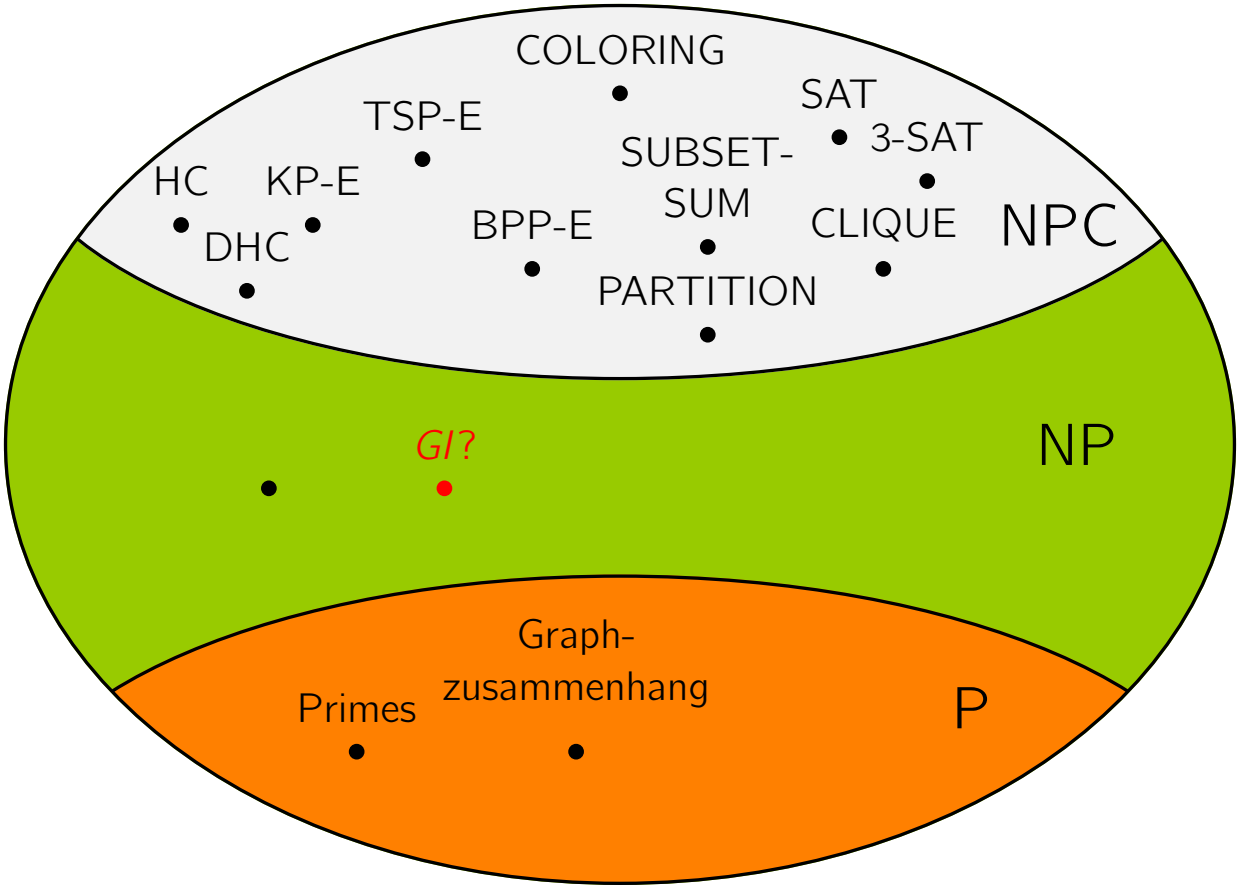
Lemma

3-SAT $\in$ NP und SAT $\leq_p$ 3-SAT.

Korollar

3-SAT ist NP-vollständig.

Wdh.: Die Komplexitätslandschaft



Warnung: Dieser Abbildung liegt die Annahme $P \neq NP$ zu Grunde.

Was tun mit NP-schweren Problemen?

Viele praxisrelevante Optmierungsprobleme sind NP-schwer, z.B. das Bin Packing Problem (BPP), das Rucksack Problem (KP) und das Traveling Salesperson Problem (TSP).

In der Praxis müssen die Probleme dennoch gelöst werden. Verschiedene Strategien können hier zum Erfolg führen; die wichtigsten sind:

- Approximationsalgorithmen
- Ausnutzen der Eingabestruktur durch spezielle Algorithmen
- Parametrisierte Algorithmen
- Randomisierte Algorithmen
- Heuristiken (ohne irgendwelche Garantien)

Ausnutzen der Eingabestruktur

Beispiel 1

- Straßennetze lassen sich durch Graphen modellieren, die (fast) planar sind.
- Es gibt effiziente Graphalgorithmen, die speziell auf diese planare Struktur ausnützen.

Beispiel 2

- Instanzen des TSP haben in der Praxis häufig eine metrische Struktur, d.h., die Distanzen sind symmetrisch und erfüllen die Dreiecksungleichung.
- Für das "metrische TSP" gibt es gute Approximationsalgorithmen.

Parametrische Analyse

In der Praxis wird es oft der Problemstellung nicht gerecht, die Effizienz eines Algorithmus allein in Abhängigkeit von der Eingabegröße zu messen. Eine verfeinerte Analyse, die andere Eingabeparameter berücksichtigt, kann zu besseren Resultaten führen.

Beispiel

Wir wollen eine Anfrage α in einer Datenbank $\mathcal{D}$ auswerten.

- Sei ℓ die Größe von α und m die Größe von $\mathcal{D}$.
  Die Eingabegröße ist dann $n = \ell + m$.
- Häufig ist m sehr groß und ℓ relativ klein. Ein Algorithmus mit einer Laufzeit von $2^\ell m$ kann deswegen durchaus gut sein, ein Algorithmen mit Laufzeiten wie m^ℓ oder gar $2^m \ell$ hingegen kaum.
- Analysieren wir die Algorithmen hingegen nur nach der Eingabegröße n, so wird dieser wichtige Unterschied nicht sichtbar.

Approximationsalgorithmen

Sei Π ein Optimierungsproblem. Für eine Instanz I von Π bezeichnen wir den optimalen Zielfunktionswert mit $opt(I)$.

- Ein α-Approximationsalgorithmus, $\alpha > 1$, für ein *Minimierungsproblem* Π berechnet für jede Instanz I von Π eine zulässige Lösung mit Zielfunktionswert höchstens $\alpha \cdot opt(I)$.
- Ein α-Approximationsalgorithmus, $\alpha < 1$, für ein *Maximierungsproblem* Π berechnet für jede Instanz I von Π eine zulässige Lösung mit Zielfunktionswert mindestens $\alpha \cdot opt(I)$.

Die Zahl α wird auch als Approximationsfaktor oder Approximationsgüte bezeichnet.

BPP – Approximationsalgorithmus

Zur Erinnerung:

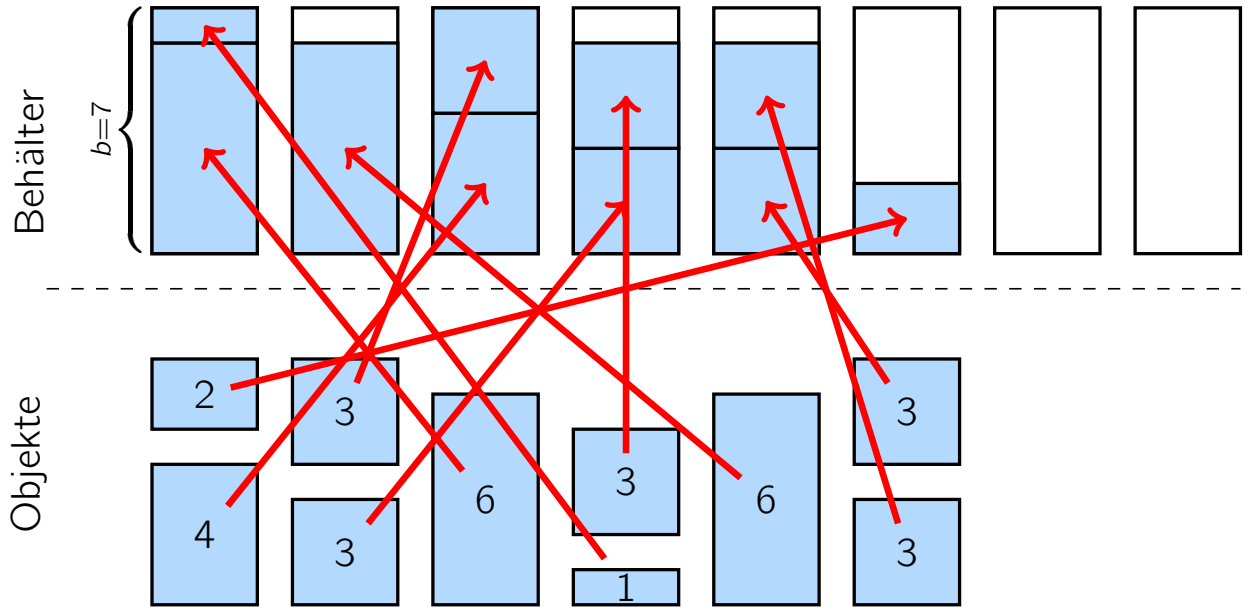
Beim Bin Packing Problem (BPP) suchen wir eine Verteilung von N Objekten mit Gewichten $w_1, \ldots, w_N$ auf eine möglichst kleine Anzahl von Behältern mit Gewichtskapazität jeweils b.

Satz

Für BPP gibt es einen effizienten 2-Approximationsalgorithmus.

Der Begriff effizienter Algorithmus wird als Synonym für Algorithmus mit polynomiell beschränkter Laufzeit verwendet.

Bin Packing Problem – Beispiel



Eine Lösung die $k = 6$ Behälter verwendet

BPP – Approximationsalgorithmus

Algorithmus FIRST FIT:

Initial seien alle Kisten *ungeöffnet*.

Betrachte die Objekte in beliebiger Reihenfolge:

- Wenn das gerade betrachtete Objekt in keine der geöffneten Kisten eingefügt werden kann, ohne die Kapazität zu überschreiten, dann *öffne* eine neue Kiste für dieses Objekt.
- Ansonsten füge das Objekt in die erste bereits geöffnete Kiste ein, in die es passt.

BPP – Approximationsalgorithmus

Analyse des Approximationsfaktors:

Wieviele geöffnete Kisten gibt es, die höchstens halb gefüllt sind?

- FIRST FIT öffnet nur dann eine neue Kiste und fügt ein Objekt mit Gewicht höchstens $b/2$ in diese Kiste ein, wenn keine geöffnete Kiste mit Gewicht höchstens $b/2$ existiert.
- Es folgt, dass es zu keinem Zeitpunkt zwei geöffnete Kisten gibt, die Gewicht höchsten $b/2$ enthalten.

Schlussfolgerung: Wenn am Ende k Kisten geöffnet sind, gibt es somit mindestens $k-1$ Kisten, die mehr als halb voll sind.

Sei $W = \sum_{i=1^N} w_i$. Es folgt $W > \frac{k-1}{2} b$.

Aus der trivialen unteren Schranke $opt \geq W/b$ folgt nun $\frac{k-1}{2} < opt$ und somit $k \leq 2opt$.

□

BPP – Untere Schranke für den Approximationsfaktor

Satz

Wenn $\mathrm{P} \neq \mathrm{NP}$*, dann gibt es für BPP keinen effizienten* α*-Approximationsalgorithmus mit* $\alpha < \frac{3}{2}$.

Beweis:

Wir zeigen, dass aus einem effizienten α-Approximationsalgorithmus mit $\alpha < \frac{3}{2}$ für BPP ein effizienter Algorithmus für PARTITION abgeleitet werden kann.

Das NP-vollständige PARTITON-Problem wäre dann in P und daraus würde $\mathrm{P} = \mathrm{NP}$ folgen.

BPP – Untere Schranke für den Approximationsfaktor

Analog zum Konzept der polynomiellen Reduktion, erzeugen wir aus einer Eingabe $a_1, \ldots, a_N$ für PARTITION eine Eingabe für BPP, indem wir $w_1 = a_1, \ldots, w_N = a_N$ und $b = (\sum_{i=1}^{N} a_i)/2$ setzen.

Es gilt:

- Falls es sich um eine JA-Instanz von Partition handelt, gibt es eine Lösung für die BPP-Instanz mit $k = 2$ Kisten.
- Im Falle einer JA-Instanz würde ein α-Approximationsalgorithmus für BPP somit eine Aufteilung auf zwei Kisten finden, weil eine Lösung mit drei oder mehr Kisten um mindestens den Faktor $\frac{3}{2} > \alpha$ vom Optimum abweichen würde.
- Falls es sich um eine NEIN-Instanz von Partition handelt, gibt es hingegen keine BPP-Lösung mit zwei Kisten.

Durch Aufruf des Approximationsalgorithmus für BPP kann man somit die JA- und NEIN-Instanzen von PARTITION unterscheiden. □

Nichtapproximierbarkeit vom allgemeinen TSP

Zur Erinnerung:

Beim Traveling Salesperson Problem (TSP) ist ein vollständiger Graph $G = (V, E)$ mit Kantenkosten $c(u, v) \in \mathbb{N}$ für $u, v \in V$ mit $c(u, v) = c(v, u)$ gegeben.

Gesucht ist eine Rundreise (ein Hamiltonkreis) mit kleinstmöglichen Kosten.

Satz

Sei $\alpha \geq 1$ beliebig gewählt. Wenn $\mathrm{P} \neq \mathrm{NP}$, dann gibt es für das TSP-Problem gibt es keinen effizienten α-Approximationsalgorithmus.

Nichtapproximierbarkeit vom allgemeinen TSP

Beweis:

Wir zeigen, dass aus einem effizienten α-Approximationsalgorithmus $\mathcal{A}$ für TSP ein effizienter Algorithmus $\mathcal{A}'$ für das Hamiltonkreisproblem (HC) abgeleitet werden könnte.

Algorithmus $\mathcal{A}'$:

- Sei $G' = (V, E')$ die Eingabe für HC.
- $\mathcal{A}'$ transformiert G' in einen gewichteten, vollständigen Graphen $G = (V, E)$, in dem nur die Kanten aus E' die Länge 1 haben, alle anderen Kanten in E erhalten die Länge αn.
- Dann ruft $\mathcal{A}'$ den Algorithmus $\mathcal{A}$ auf G auf.
- Falls $\mathcal{A}$ eine TSP-Tour der Länge höchstens αn findet, so gibt $\mathcal{A}'$ aus, dass G' einen Hamilton-Kreis enthält.
- Ansonsten meldet $\mathcal{A}'$, dass G' keinen Hamilton-Kreis enthält.

Nichtapproximierbarkeit vom allgemeinen TSP

Korrektheit von $\mathcal{A}'$:

- Zuerst nehmen wir an, G' enthält einen Hamilton-Kreis. Jeder Hamilton-Kreis in G' entspricht einer TSP-Tour der Länge n in G. $\mathcal{A}$ findet somit eine Tour der Länge höchstens αn.
- Jetzt nehmen wir an, G' enthält keinen Hamilton-Kreis. Dann haben alle TSP-Touren in G die Länge mindestens $\alpha n + n - 1 > \alpha n$.

Damit kann $\mathcal{A}'$ mit Hilfe von $\mathcal{A}$ die JA- und NEIN-Instanzen von HC unterscheiden.

Aus der Existenz von $\mathcal{A}$ würde somit $\mathrm{P} = \mathrm{NP}$ folgen. $\square$

Metrisches TSP – Definition

Beim metrischen TSP ist ein vollständiger Graph $G = (V, E)$ mit Kantenkosten $c(u, v) \in \mathbb{N}$ für $u, v \in V$ gegeben, die für beliebige Knoten u, v, w die folgenden Bedingungen erfüllen:

- $c(u, u) = 0$
- $c(u, v) = c(v, u)$ (Symmetrie)
- $c(u, w) \leq c(u, v) + c(v, w)$ (Dreiecksungleichung)

Gesucht ist eine Rundreise (ein *Hamiltonkreis*) mit kleinstmöglichen Kosten.

Metrisches TSP – NP-Schwere

Beobachtungen:

- Metrisches TSP ist ein Spezialfall des allgemeinen TSP.
- $\{1, 2\}$-TSP erfüllt die Dreiecksungleichung und ist somit ein Spezialfall des metrischen TSP.
- Aus der NP-Schwere von $\{1, 2\}$-TSP folgt somit die NP-Schwere vom metrischen TSP.

Metrisches TSP – Approximationsalgorithmus

Satz

Für das metrische TSP gibt es einen effizienten 2-Approximationsalgorithmus.

Beweis:

Algorithmus:

1. Finde einen minimalen Spannbaum T von G;
2. Verdopple die Kanten von T und erhalte dadurch den Euler-Graphen T';
3. Berechne eine Euler-Tour auf T';
4. Bereinige die Euler-Tour um wiederholt vorkommende Knoten.

Metrisches TSP – Approximationsalgorithmus

Analyse des Approximationsfaktors:

- Aus einer TSP-Tour können wir einen Spannbaum erzeugen, indem wir eine Kante löschen.
- Also ist ein minimaler Spannbaum nicht teurer als die Länge einer minimalen TSP-Tour.
- Die Kosten der berechneten Euler-Tour entsprechen den doppelten Kosten des minimalen Spannbaums, sind also höchstens zweimal so teuer wie die minimale TSP-Tour.
- Aufgrund der Dreiecksungleichung erhöht das Überspringen von mehrfach besuchten Knoten in Schritt 3 die Kosten nicht.

□

Metrisches TSP – Bemerkungen

- Der beste bekannte Approximationsalgorithmus für metrisches TSP war lange Zeit ein von Christofides im Jahr 1975 vorgestellter Algorithmus mit einem Approximationsfaktor von $\frac{3}{2}$.
- Erst 2021 wurde dieser Approximationsfaktor übertroffen: Karlin, Klein und Garan haben einen Approximationsalgorithmus für metrisches TSP mit einem Faktor $\frac{3}{2} - \epsilon$ für ein winziges

$$\epsilon > \frac{1}{1.000.000.000.000.000.000.000.000.000.000.000.000.000.000}$$

gefunden.
- Andererseits haben Karpinski, Lampis und Schmied 2015 bewisen, dass es unter der Annahme P $\neq$ NP keinen effizienten Approximationsalgorithmus mit einem Approximationsfaktor besser als $\frac{123}{122}$ gibt.

Approximationsschemata

Ein Approximationsschema ist ein Algorithmus, der es ermöglicht, für jedes vorgegebene $\epsilon > 0$ eine zulässige Lösung mit Approximationsfaktor $1 + \epsilon$ (bzw. $1 - \epsilon$) zu berechnen.

Satz

Für das Rucksackproblem (KP) gibt es ein polynomielles Approximationsschema.

Die Klasse PSPACE

- Sei PSPACE die Klasse derjenigen Probleme, die wir mit einem polynomiell beschränkten Band auf einer TM lösen können.
- Im Gegensatz zur Zeitkomplexität kann man nachweisen, dass diese Klasse sich nicht ändern würde, wenn wir sie bezüglich der Platzkomplexität auf NTM definieren würden.
  (*Satz von Savitch*)
- Wie verhält sich PSPACE zu NP? – Da sich der Kopf einer Turingmaschine pro Zeitschritt nur eine Position bewegen kann gilt

$$\text{NP} \subseteq \text{PSPACE} \ .$$

Die Klasse EXPTIME

- Die Klasse der Probleme mit einer Laufzeitschranke $2^{p(n)}$ auf einer TM für ein Polynom p bezeichnen wir als EXPTIME.
- Wie verhält sich EXPTIME zu NP? und wie zu PSPACE?
- Bei einer Speicherplatzbeschränkung der Größe $s(n)$ gibt es nur $2^{O(s(n))}$ viele verschiedenen Konfigurationen für eine Turingmaschine, so dass auch die Rechenzeit durch $2^{O(s(n))}$ beschränkt ist.
- Die Probleme in PSPACE können deshalb in Zeit $2^{p(n)}$ gelöst werden, so dass gilt

$$\text{PSPACE} \subseteq \text{EXPTIME} \ .$$

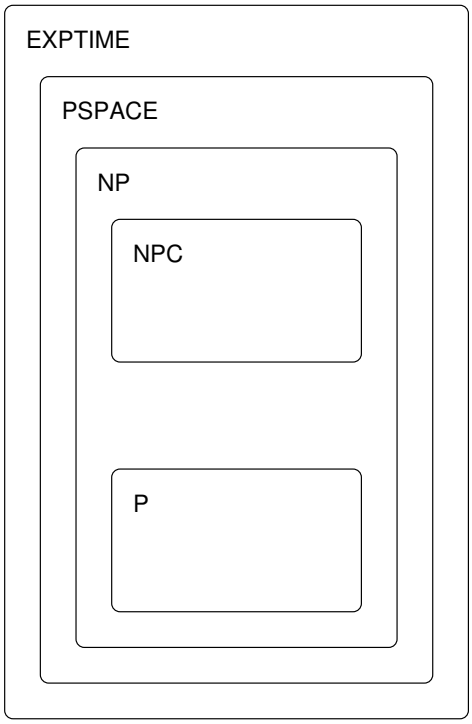
Zusammenfassung

- Wir haben gezeigt

$$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME.$$

- Es ist nicht bekannt, ob diese Inklusionen echt sind.
- Möglicherweise ist P = PSPACE oder NP = EXPTIME.
- Man weiß allerdings, dass $P \subsetneq EXPTIME$ (*Zeithierarchiesatz*).

Zusammenfassung



Mehr dazu in der Vorlesung Komplexitätstheorie.

Vorlesung 19
Zusammenfassung

Wdh.: Was tun mit NP-schweren Problemen?

Viele praxisrelevante Optimierungsprobleme sind NP-schwer, z.B. das Bin Packing Problem (BPP), das Rucksack Problem (KP) und das Traveling Salesperson Problem (TSP).

In der Praxis müssen die Probleme dennoch gelöst werden. Verschiedene Strategien können hier zum Erfolg führen; die wichtigsten sind:

- Approximationsalgorithmen
- Ausnutzen der Eingabestruktur durch spezielle Algorithmen
- Parametrisierte Algorithmen
- Randomisierte Algorithmen
- Heuristiken (ohne irgendwelche Performanzgarantien)

Wdh.: Approximationsalgorithmen

Sei Π ein Optimierungsproblem. Für eine Instanz I von Π bezeichnen wir den optimalen Zielfunktionswert mit $opt(I)$.

- Ein α-Approximationsalgorithmus, $\alpha > 1$, für ein *Minimierungsproblem* Π berechnet für jede Instanz I von Π eine zulässige Lösung mit Zielfunktionswert höchstens $\alpha \cdot opt(I)$.
- Ein α-Approximationsalgorithmus, $\alpha < 1$, für ein *Maximierungsproblem* Π berechnet für jede Instanz I von Π eine zulässige Lösung mit Zielfunktionswert mindestens $\alpha \cdot opt(I)$.

Die Zahl α wird auch als Approximationsfaktor oder Approximationsgüte bezeichnet.

Approximierbarkeit von einigen Problemen

	obere Schranke	untere Schranke	Bemerkung
BPP	2	$\frac{3}{2}$	$(1+\varepsilon)\mathrm{OPT}+1$ möglich
TSP	–	∞	
TSP metrisch	$\frac{3}{2}-\epsilon$ für ein $\epsilon > 0$	$\frac{123}{122}$	
TSP euklidisch	$1+\varepsilon$ für alle $\epsilon > 0$	–	
KP	$1+\varepsilon$ für alle $\epsilon > 0$	–	

Die unteren Schranken gelten für effiziente Approximierbarkeit unter der Annahme P $\neq$ NP.

Rückblick

In BuK haben wir uns mit den

Grenzen der (effizienten) Berechenbarkeit

beschäftigt.

Teil I
Grundlagen

Vorlesung 1
Einführung

Vorlesung 2
Berechnungsprobleme und Turingmaschinen

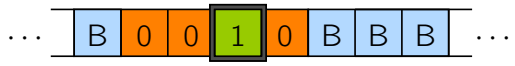
Wdh.: Kodierung von Berechnungsproblemen

3 mögliche formale Definitionen.

Als Relation:	Als Funktion	Als Sprache
▶ Primfaktor:		
$(110, 11) \in R$ $(101, 11) \notin R$ $(00110, 11) \notin R$		
▶ Multiplikation		
$(11\#10, 110) \in R$ $(11\#10, 11) \notin R$ $(1\#1\#0, 110) \notin R$	$f(11\#10) = 110$ $(f(1\#1\#0) = \perp)$	
▶ Wörter die auf 1 enden.		
$(11, 1) \in R$ $(110, 1) \notin R$ $(10, 0) \in R$	$f(11) = 1$ $f(110) = 0$	$11 \in L$ $110 \notin L$

Wdh.: Turingmaschinen

Anschauliche Definition:



δ	0	1	B
q_1	$(q_1, 1, L)$	$(q_2, 1, R)$	(q_1, B, N)
q_2	(q_3, B, R)	$(q_1, 0, L)$	(q_3, B, R)
q_3	$(q_2, 0, N)$	$(q_2, 0, R)$	(q_3, B, R)

Formale Definition:

Eine Turingmaschine ist ein 7-Tupel $(Q, \Sigma, \Gamma, B, q_0, \bar{q}, \delta)$, wobei

- Q, Σ, Γ endliche Mengen sind,
- $\Sigma \subseteq \Gamma$,
- $B \in \Gamma \setminus \Sigma$,
- $q_0, \bar{q} \in Q$ und
- $\delta \colon (Q \setminus \{\bar{q}\}) \times \Gamma \to Q \times \Gamma \times \{R, L, N\}$.

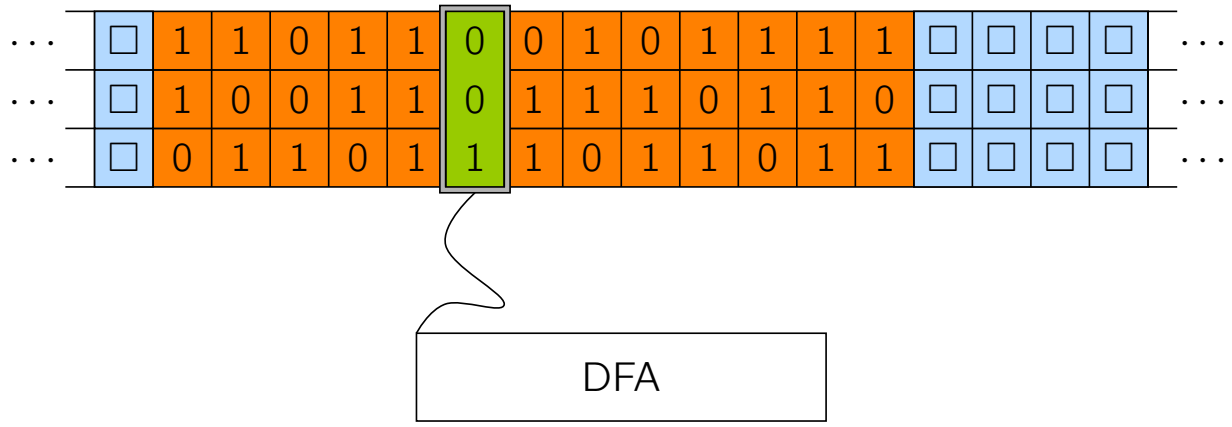
Wdh.: TM-Techniken

- Speicher im Zustandsraum:

$$Q_{\text{neu}} := Q \times \Gamma^k$$

- Mehrspurmaschinen:

$$\Gamma_{\text{neu}} := \Gamma \cup \Gamma^k$$



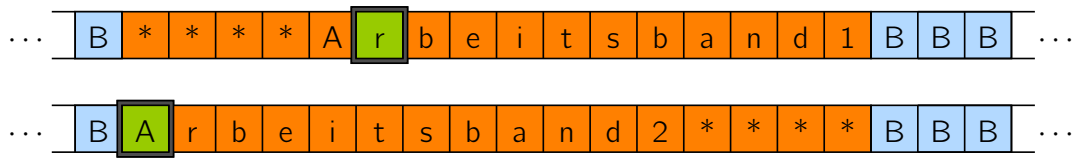
- Schleifen, Variablen, Felder (Arrays), Unterprogramme

Vorlesung 3
Mehrband-Turingmaschinen und die universelle Turingmaschine

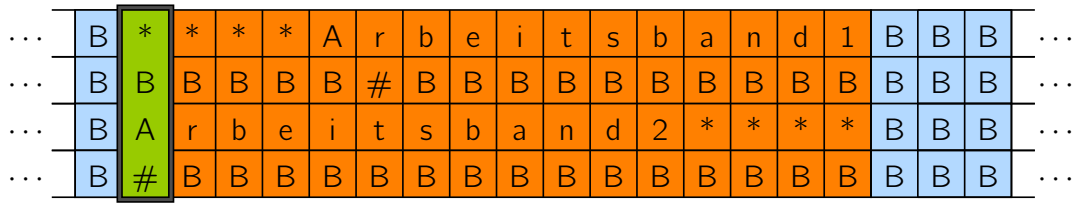
Wdh.: k-Band- vs 1-Band-TM

Satz

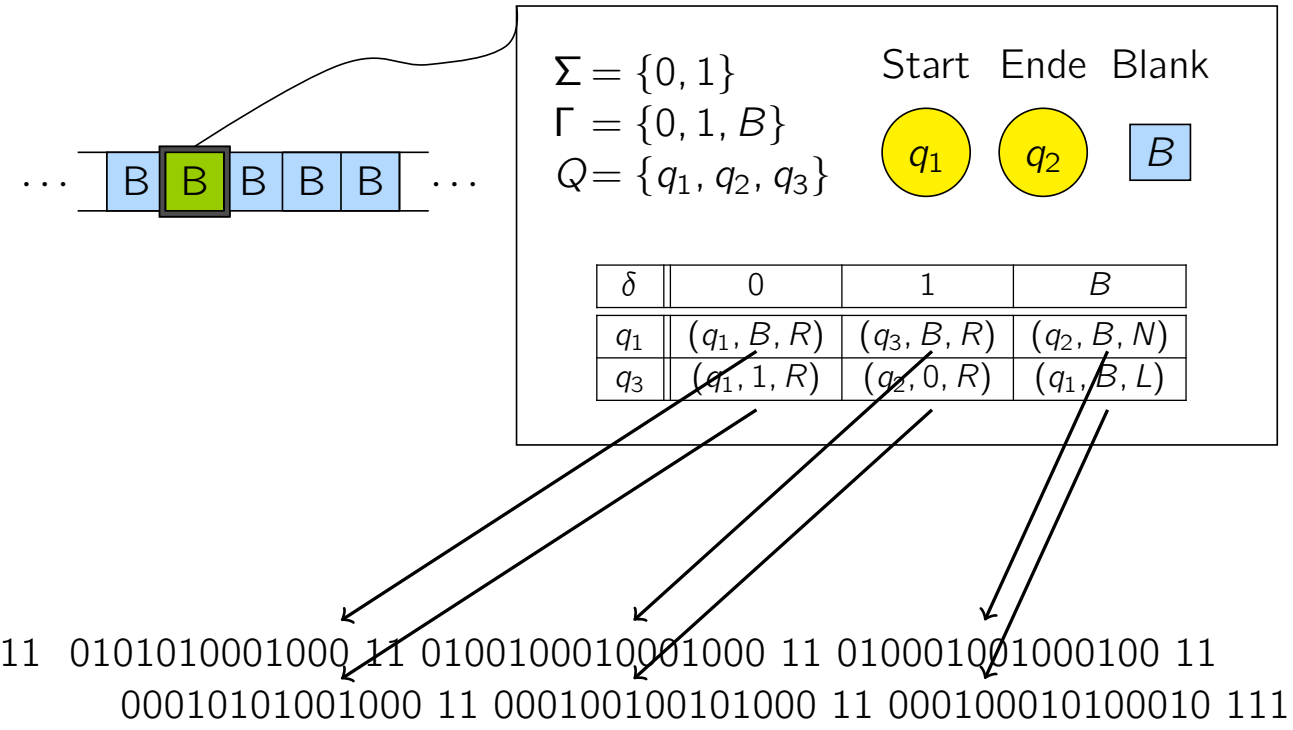
Eine k-Band-TM M, die mit Rechenzeit $t(n)$ und Platz $s(n)$ auskommt, kann von einer (1-Band-)TM M' mit Zeitbedarf $O(t^2(n))$ und Platzbedarf $O(s(n))$ simuliert werden.



Simuliert durch

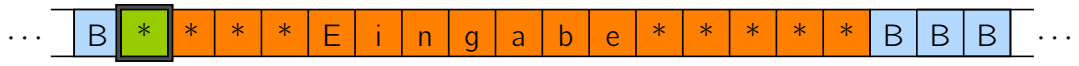


Wdh.: Gödelnummer $\langle M \rangle$

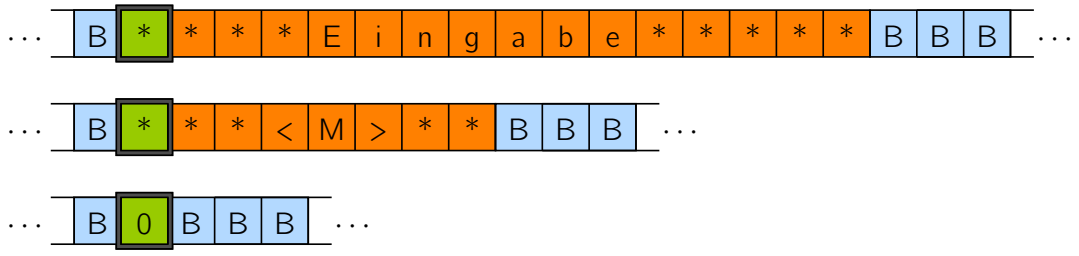


Wdh.: Universelle TM

simulierte Turingmaschine M

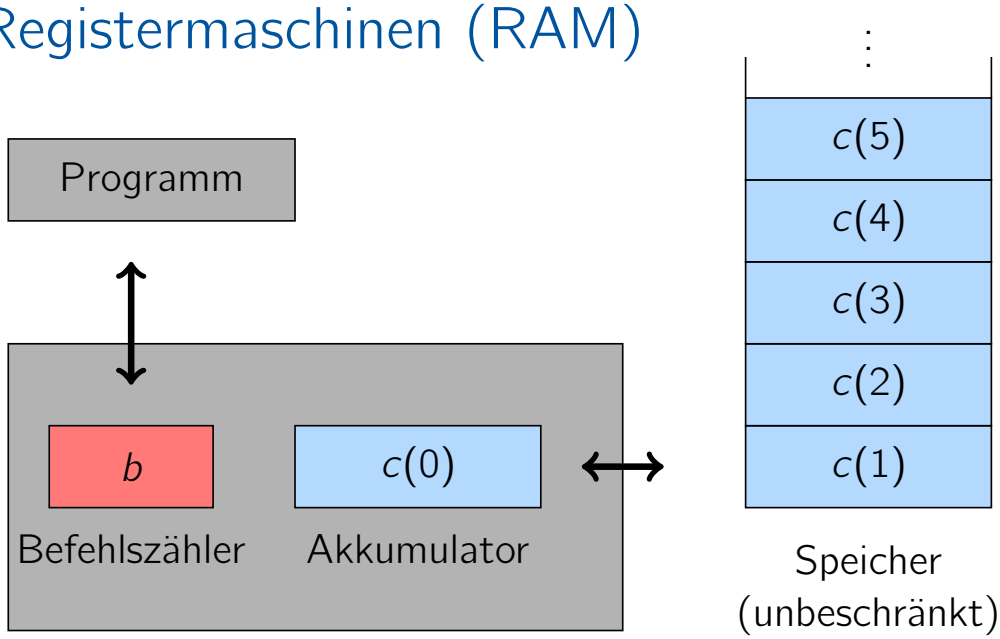


Initialisierung der universellen Maschine U



Vorlesung 4
Registermaschine (RAM), Church-Turing-These

Wdh.: Registermaschinen (RAM)



Befehlssatz:
LOAD, STORE, ADD, SUB, MULT, DIV
INDLOAD, INDSTORE, INDADD, INDSUB, INDMULT, INDDIV
CLOAD, CADD, CSUB, CMULT, CDIV
GOTO, IF c(0)?x THEN GOTO j (wobei ? aus {=,<,<=,>,>=} ist),
END

Wdh.: RAM vs. TM

Satz (RAM $\to$ TM)

Für jede im logarithmischen Kostenmaß $t(n)$-zeitbeschränkte RAM R gibt es ein Polynom q und zu diesem eine $O\big(q(n + t(n))\big)$-TM M, die R simuliert.

Satz (TM $\to$ RAM)

Jede $t(n)$-zeitbeschränkte TM kann durch eine RAM simuliert werden, die zeitbeschränkt ist durch

- $O(t(n) + n)$ *im uniformen Kostenmaß und*
- $O((t(n) + n) \cdot \log(t(n) + n))$ *im logarithmischen Kostenmaß.*

Wdh.: Die Church-Turing-These

Kein jemals bisher vorgeschlagenes „vernünftiges“ Rechnermodell hat eine größere Mächtigkeit als die TM.

Church-Turing-These

Die Klassen der TM-berechenbaren (partiellen) Funktionen und TM-entscheidbaren Sprachen stimmen mit den Klassen der “intuitiv berechenbaren” (partiellen) Funktionen bzw. “intuitiv entscheidbaren” Sprachen überein.

Wir sprechen deshalb nicht mehr von TM-berechenbaren (partiellen) Funktionen oder TM-entscheidbaren Sprachen, sondern allgemein von berechenbaren (partiellen) Funktionen bzw. entscheidbaren Sprachen.

Teil II
Berechenbarkeit

Vorlesung 5
Unentscheidbare Probleme: Diagonalisierung

Wdh.: Abzählbarkeit

Definition (Abzählbare Menge)

Eine Menge M heißt abzählbar, wenn sie leer ist oder wenn es eine surjektive Funktion $c\colon \mathbb{N} \to M$ gibt.

Abzählbare Mengen: endlichen Mengen, $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$, $\{0,1\}^*$, Menge der Gödelnummern, die Menge der endlichen Teilmengen von $\mathbb{N}$.

Satz

Die Menge $\mathcal{P}(\mathbb{N})$ ist überabzählbar.

Überabzählbare Mengen: $\mathbb{R}$, $\mathcal{P}(\mathbb{N})$, $\mathcal{P}(\{0,1\}^*)$, Menge der Berechnungsprobleme.

Schlussfolgerung: Es gibt nicht-berechenbare Probleme.

Wdh.: Unentscheidbarkeit der Diagonalsprache

Die Diagonalsprache:

$$\begin{aligned} D &= \{ w \in \{0,1\}^* \mid w = w_i \text{ und } M_i \text{ akzeptiert } w \text{ nicht}\} \\ &= \{\langle M\rangle \mid M \text{ akteptiert } \langle M\rangle \text{ nicht}\}. \end{aligned}$$

Satz

Die Diagonalsprache D ist unentscheidbar (= nicht entscheidbar).

Beweisansatz: Diagonalisierung

Wdh.: Unentscheidbarkeit der Diagonalsprache

$$A_{i,j} = \begin{cases} 1 & \text{falls } M_i \text{ das Wort } w_j \text{ akzeptiert} \\ 0 & \text{sonst} \end{cases}$$

Beispiel:

	w_0	w_1	w_2	w_3	w_4	
M_0	**0**	1	1	0	1	$\cdots$
M_1	1	**0**	1	0	1	$\cdots$
M_2	0	0	**1**	0	1	$\cdots$
M_3	0	1	1	**1**	0	$\cdots$
M_4	0	1	0	0	**0**	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		

Die Diagonalsprache lässt sich auf der Diagonale der Matrix ablesen. Es ist

$$D = \{w_i \mid A_{i,i} = 0\} .$$

Vorlesung 6
Unentscheidbarkeit des Halteproblems: Unterprogrammtechnik

Wdh.: Unentscheidbare Probleme

Die Diagonalsprache:

$$D = \{\langle M \rangle \mid M \text{ akzeptiert } \langle M \rangle \text{ nicht}\}$$

Das Diagonalsprachenkomplement:

$$\overline{D} = \{\langle M \rangle \mid M \text{ akzeptiert } \langle M \rangle\}$$

Das Halteproblem:

$$H = \{\langle M \rangle w \mid M \text{ hält auf } w\}$$

Das spezielle Halteproblem:

$$H_\epsilon = \{\langle M \rangle \mid M \text{ hält auf Eingabe } \epsilon\}$$

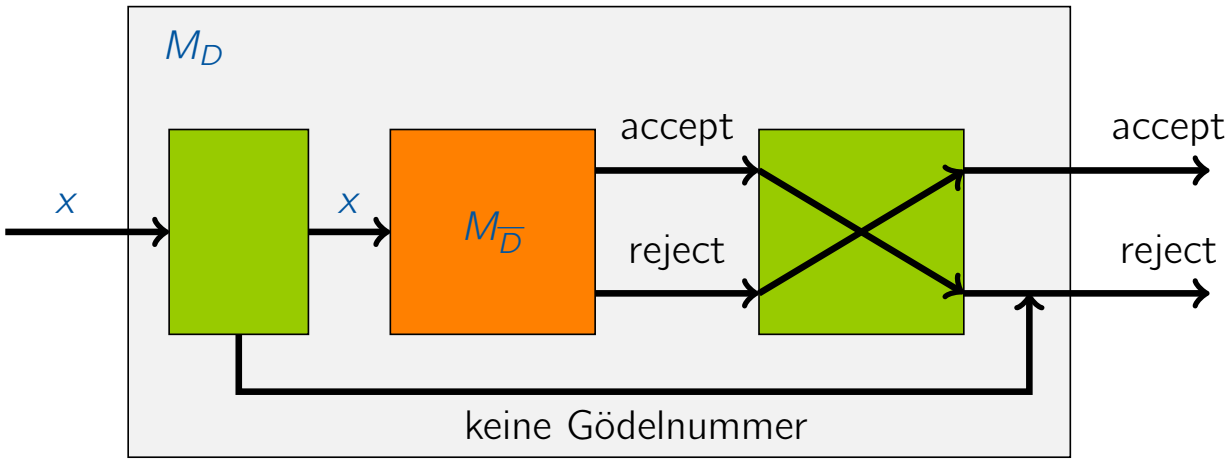
Wdh.: Beweise durch Unterprogrammtechnik

D ist unentscheidbar auf Grund eines Diagonalisierungs-Argumentes.

Die Argumentationskette war:

$$\begin{array}{ll} D \text{ ist unentscheidbar} & M_D \\ \Downarrow & \updownarrow \\ \overline{D} \text{ ist unentscheidbar} & M_{\overline{D}} \\ \Downarrow & \updownarrow \\ H \text{ ist unentscheidbar} & M_H \\ \Downarrow & \updownarrow \\ H_\epsilon \text{ ist unentscheidbar} & M_{H_\epsilon} \end{array}$$

Wdh.: Unentscheidbarkeit des Komplements der Diagonalsprache



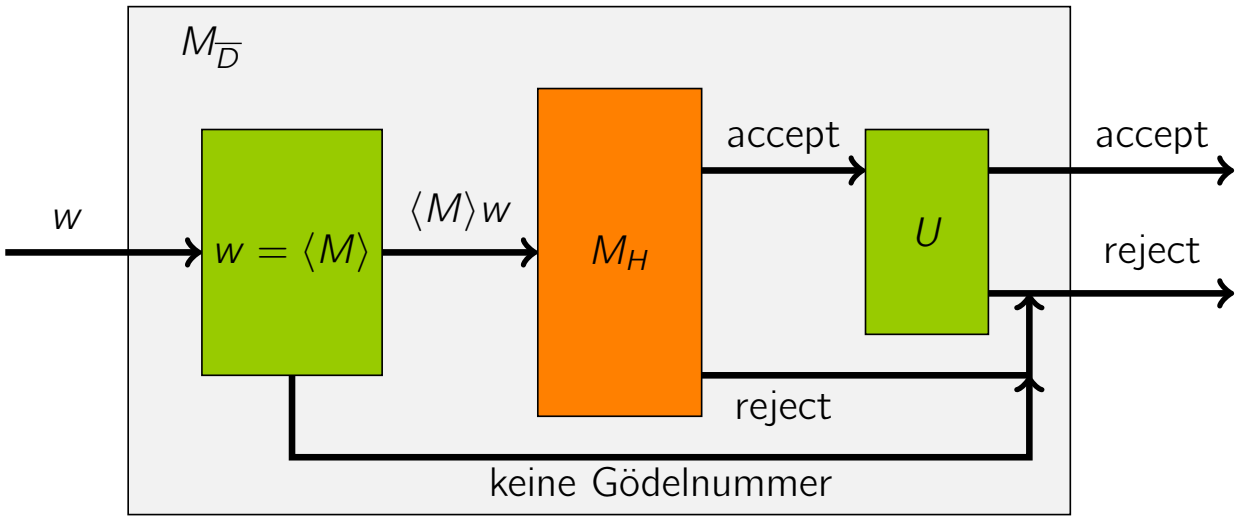
Wdh.: Beweise durch Unterprogrammtechnik

D ist unentscheidbar auf Grund eines Diagonalisierungs-Argumentes.

Die Argumentationskette war:

$$\begin{array}{ll} D \text{ ist unentscheidbar} & M_D \\ \Downarrow & \updownarrow \\ \overline{D} \text{ ist unentscheidbar} & M_{\overline{D}} \\ \Downarrow & \updownarrow \\ H \text{ ist unentscheidbar} & M_H \\ \Downarrow & \updownarrow \\ H_\epsilon \text{ ist unentscheidbar} & M_{H_\epsilon} \end{array}$$

Wdh.: Unentscheidbarkeit des Halteproblems



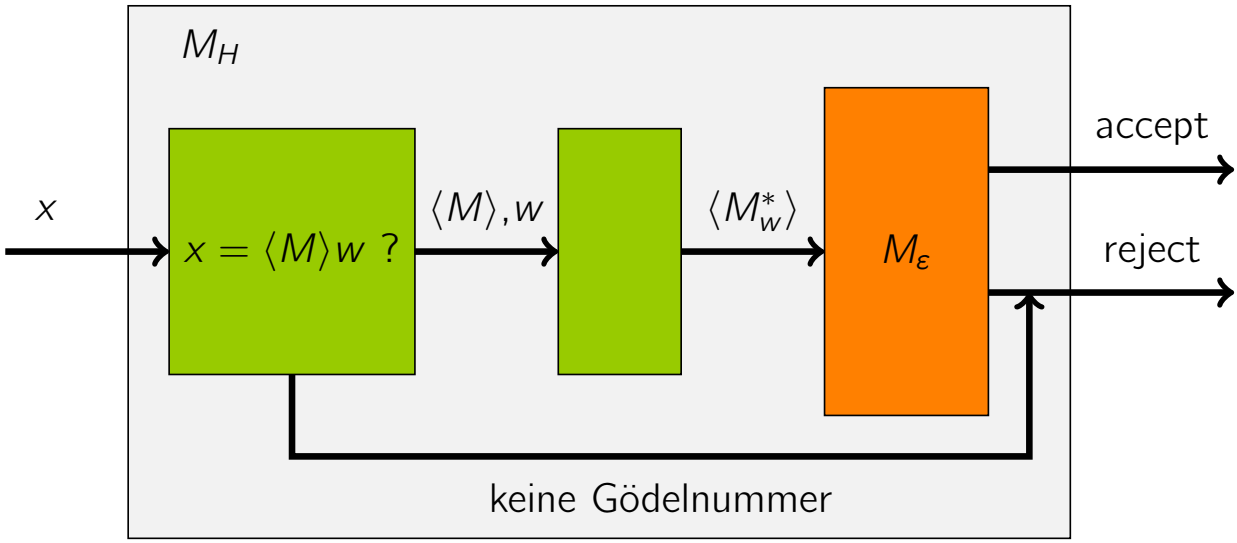
Wdh.: Beweise durch Unterprogrammtechnik

D ist unentscheidbar auf Grund eines Diagonalisierungs-Argumentes.

Die Argumentationskette war:

$$\begin{array}{ll} D \text{ ist unentscheidbar} & M_D \\ \Downarrow & \updownarrow \\ \overline{D} \text{ ist unentscheidbar} & M_{\overline{D}} \\ \Downarrow & \updownarrow \\ H \text{ ist unentscheidbar} & M_H \\ \Downarrow & \updownarrow \\ H_\epsilon \text{ ist unentscheidbar} & M_{H_\epsilon} \end{array}$$

Wdh.: Unentscheidbarkeit des speziellen Halteproblems



Vorlesung 7
Der Satz von Rice

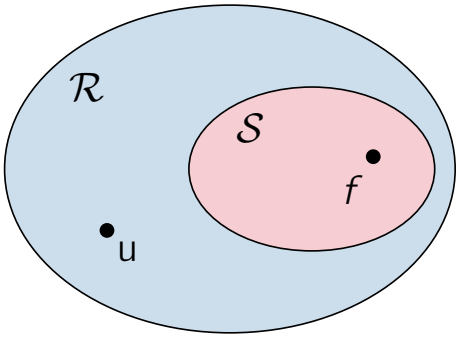
Wdh.: Der Satz von Rice

Satz

Sei $\mathcal{R}$ die Menge der von TMen berechenbaren partiellen Funktionen und $\mathcal{S}$ eine Teilmenge von $\mathcal{R}$ mit $\emptyset \subsetneq \mathcal{S} \subsetneq \mathcal{R}$. Dann ist die Sprache

$$L(\mathcal{S}) = \{\langle M \rangle \mid M \text{ berechnet eine Funktion aus } \mathcal{S}\}$$

unentscheidbar.



Wdh.: Satz von Rice – Beispiele

Beispiel

- Sei $\mathcal{S} = \{f_M \mid \forall w \in \{0,1\}^* \colon f_M(w) \neq \bot\}$.
- Dann ist

$$\begin{aligned} L(\mathcal{S}) &= \{\langle M \rangle \mid M \text{ berechnet eine Funktion aus } \mathcal{S}\} \\ &= \{\langle M \rangle \mid M \text{ hält auf jeder Eingabe}\} \end{aligned}$$

- Diese Sprache ist auch als das *allgemeine Halteproblem* H_{all} bekannt.
- Gemäß Satz von Rice ist H_{all} unentscheidbar.

Beispiel

- Sei $H_{42} = \{\langle M \rangle \mid$
  Auf jeder Eingabe hält M nach höchstens 42 Schritten$\}$.
- Über diese Sprache sagt der Satz von Rice nichts aus!
- H_{42} ist entscheidbar.

Vorlesung 8
Rekursive Aufzählbarkeit

Wdh.: Semi-Entscheidbarkeit

Eine Sprache L wird von einer TM M erkannt, wenn

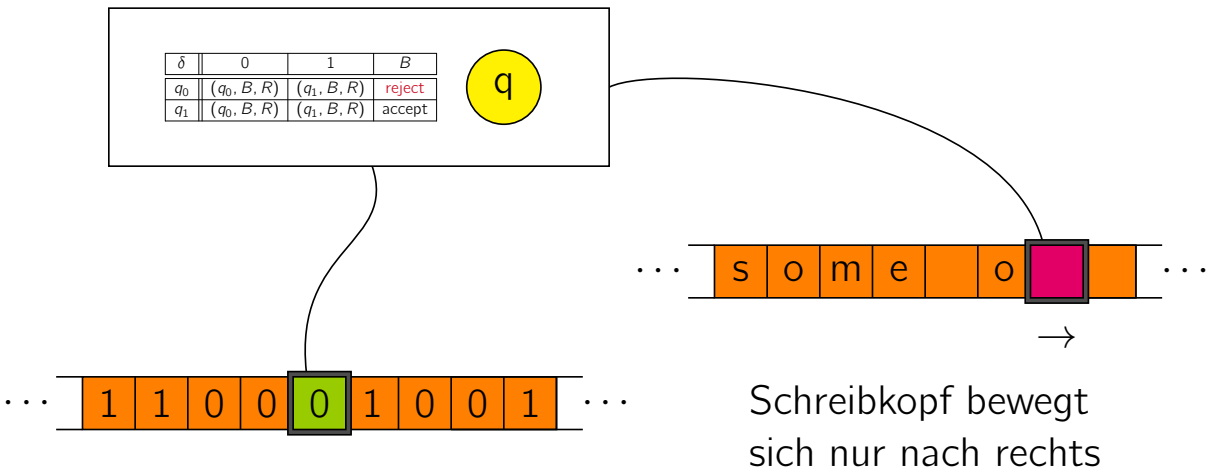
- M jedes Wort aus L akzeptiert, und
- M kein Wort akzeptiert, das nicht in L enthalten ist.

Es ist $L(M)$ die von M erkannte Sprache.

Definition
Eine Sprache L, für die eine TM existiert, die L erkennt, wird als semi-entscheidbar bezeichnet.

Beobachtung
Das Halteproblem ist semi-entscheidbar.

Wdh.: Aufzähler



Wdh.: rekursiv aufzählbar = semi-entscheidbar

Satz

Eine Sprache L ist genau dann semi-entscheidbar, wenn sie rekursiv aufzählbar ist.

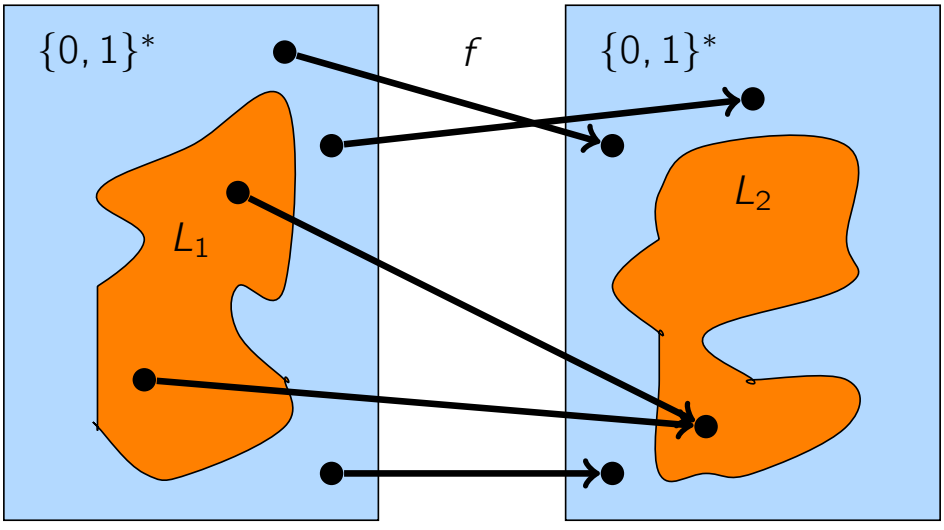
Vorlesung 9
Allgemeines Halteproblem und Hilberts 10. Problem

Wdh.: Reduktionen

Definition

Es seien L_1 und L_2 Sprachen über einem Alphabet Σ. Dann heißt L_1 auf L_2 reduzierbar, Notation $L_1 \leq L_2$, wenn es eine berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ gibt, so dass für alle $x \in \Sigma^*$ gilt

$$x \in L_1 \;\Leftrightarrow\; f(x) \in L_2 \;\; .$$



Wdh.: Komplexität des allgemeines Halteproblem

Satz

Weder $\overline{H}_{all}$ noch H_{all} sind semi-entscheidbar.

Wdh.: Hilberts zehntes Problem

Hilberts zehntes Problem

Beschreibe einen Algorithmus, der entscheidet, ob ein gegebenes Polynom mit ganzzahligen Koeffizienten eine ganzzahlige Nullstelle hat.

Die diesem Entscheidungsproblem zugrundeliegende Sprache ist

$$N = \{ p \mid p \text{ ist ein Polynom mit einer ganzzahligen Nullstelle} \} .$$

Satz von Matijasevič (1970)

Das Problem, ob ein ganzzahliges Polynom eine ganzzahlige Nullstelle hat, ist unentscheidbar.

Vorlesung 10
Das Postsche Korrespondenzproblem

Wdh.: Das Postsche Korrespondenzproblem

Das Postsche Korrespondenzproblem (PKP) ist eine Art Puzzle aus Dominos.
Eine Instanz ist zum Beispiel

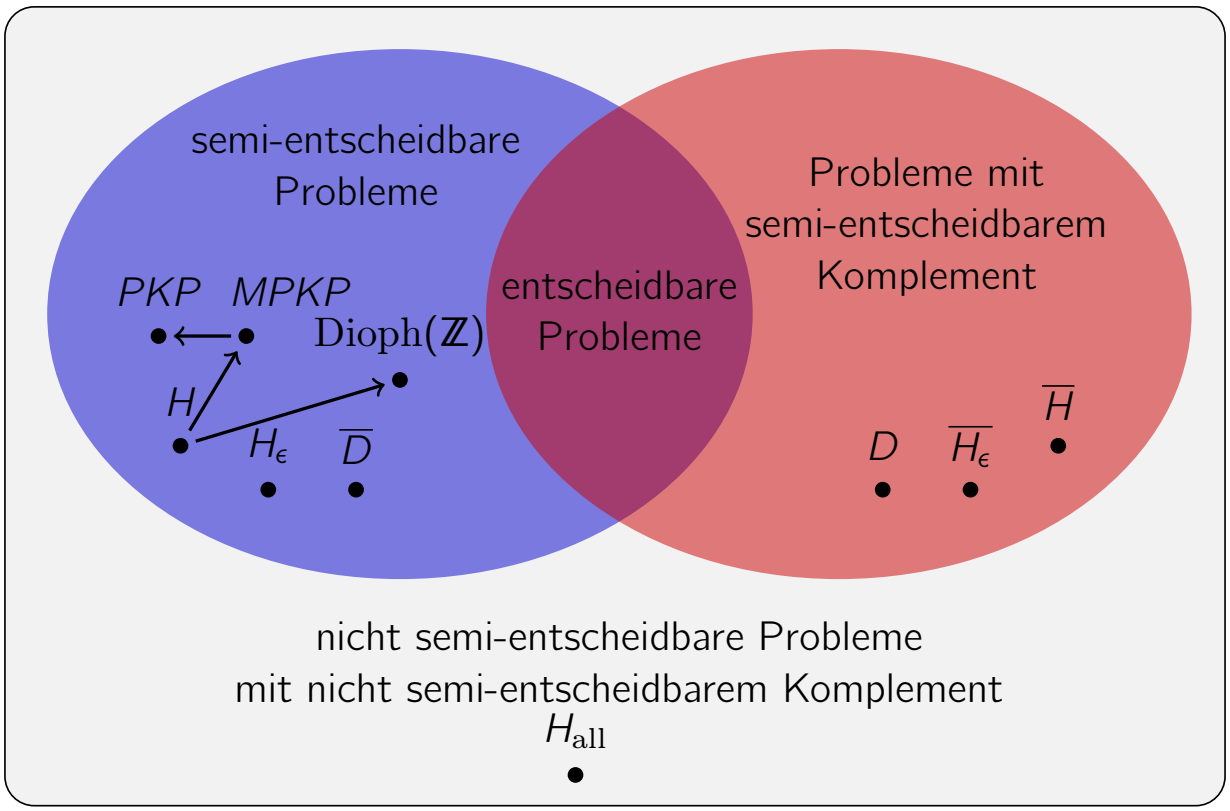
$$K = \left\{ \left[\frac{b}{ca}\right], \left[\frac{a}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{abc}{c}\right] \right\} .$$

Eine Lösung ist

$$\left[\frac{a}{ab}\right] \left[\frac{b}{ca}\right] \left[\frac{ca}{a}\right] \left[\frac{a}{ab}\right] \left[\frac{abc}{c}\right] .$$

Satz
Das PKP ist nicht entscheidbar.

Wdh.: Berechenbarkeitslandschaft



Vorlesung 11
WHILE-Programme

Wdh.: Turing-mächtige Programmiersprachen

Definition

Eine Programmiersprache wird als Turing-mächtig bezeichnet, wenn jede Funktion, die durch eine TM berechnet werden kann, auch durch ein Programm in dieser Programmiersprache berechnet werden kann.

Satz

Die Programmiersprache WHILE ist Turing-mächtig.

Vorlesung 12
LOOP-Programme

Whd.: Die Ackermann-Funktion

Definition

Die Ackermannfunktion $A\colon \mathbb{N}^2 \to \mathbb{N}$ ist folgendermaßen definert:

$$\begin{array}{lcll} A(0,n) & = & n+1 & \text{für } n \geq 0 \\ A(m+1,0) & = & A(m,1) & \text{für } m \geq 0 \\ A(m+1,n+1) & = & A(m,A(m+1,n)) & \text{für } m,n \geq 0 \end{array}$$

Wdh.: LOOP vs WHILE

Lemma

Für jedes LOOP-Programm P gibt es eine natürliche Zahl m, so dass für alle $n \in \mathbb{N}$ gilt: $F_P(n) < A(m,n)$.

Satz

Die Ackermannfunktion ist nicht LOOP-berechenbar.

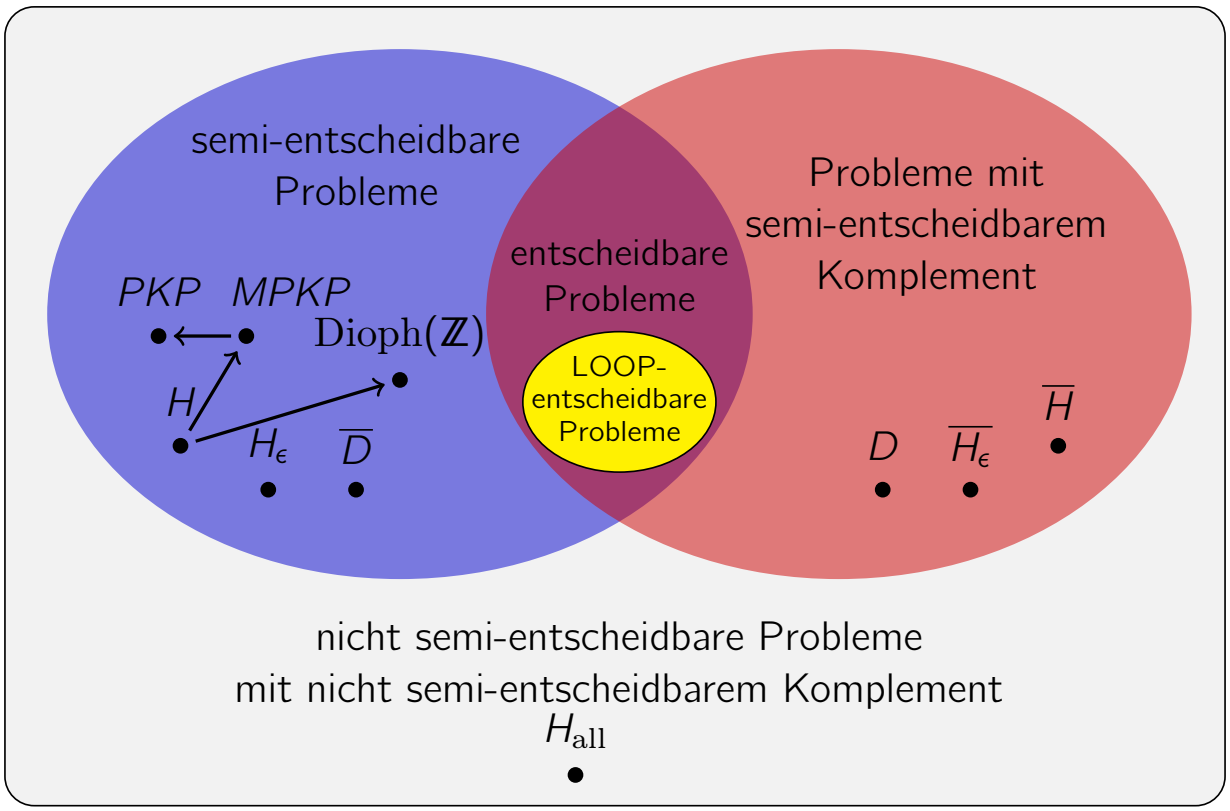
Korollar

Die Klasse der LOOP-berechenbaren Funktionen ist eine echte Teilmenge der berechenbaren (totalen) Funktionen.

Bemerkung

Mit Hilfe eines Diagonalisierungsarguments lässt sich auch beweisen, dass es entscheidbare Sprachen gibt, die nicht LOOP-entscheidbar sind.

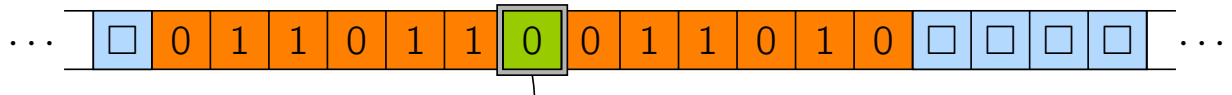
Wdh.: Berechenbarkeitslandschaft



Teil III
Komplexität

Vorlesung 13
Die Komplexitätsklassen P und NP

Wdh.: Nichtdeterministische Turingmaschine (NTM)



δ	0	1	B
q_0	$\{(q_0, B, R), (q_1, B, R)\}$	{reject}	{reject}
q_1	{reject}	$\{(q_1, B, R), (q_2, B, R)\}$	{reject}
q_2	{reject}	{reject}	{accept}

Wdh.: Komplexitätsklassen

Definition (Komplexitätsklassen)

- P ist die Klasse der Entscheidungsprobleme, für die es einen Polynomialzeitalgorithmus gibt.
- NP ist die Klasse der Entscheidungsprobleme, die durch eine NTM M erkannt werden, deren worst case Laufzeit $t_M(n)$ polynomiell beschränkt ist.
- EXPTIME ist die Klasse der Entscheidungsprobleme L, für die es ein Polynom q gibt, so dass sich L auf einer DTM mit Laufzeitschranke $2^{q(n)}$ berechnen lässt.

Satz

$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{EXPTIME}$

Vorlesung 14
Die Klasse NP und polynomielle Reduktionen

Wdh.: Optimierungs- versus Entscheidungsproblem

Mit Hilfe eines Algorithmus, der ein Optimierungsproblem löst, kann man die Entscheidungsvariante lösen.

Umgekehrt gilt:

Satz

Wenn die Entscheidungsvariante von KP in polynomieller Zeit lösbar ist, dann auch die Optimierungsvariante.

Dieser Satz gilt auch für TSP und BPP.

Wdh.: Alternative Charakterisierung der Klasse NP

Satz

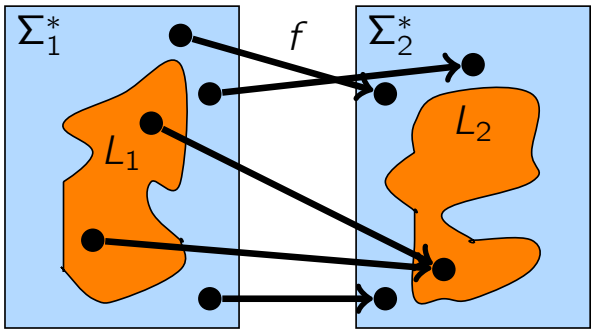
Eine Sprache $L \subseteq \Sigma^*$ ist genau dann in NP*, wenn es einen Polynomialzeitalgorithmus V (einen sogenannten Verifizierer) und ein Polynom p mit der folgenden Eigenschaft gibt:*

$$x \in L \quad \Leftrightarrow \quad \exists y \in \{0,1\}^*, |y| \leq p(|x|) : V \text{ akzeptiert } y\#x.$$

Wdh.: Polynomielle Reduktionen

Definition (Polynomielle Reduktion)

L_1 und L_2 seien zwei Sprachen über Σ_1 bzw. Σ_2. Dann heißt L_1 polynomiell reduzierbar auf L_2, wenn es eine Reduktion von L_1 nach L_2 gibt, die in polynomieller Zeit berechenbar ist. Wir schreiben $L_1 \leq_p L_2$.



Lemma

Angenommen $L_1 \leq_p L_2$, dann gilt: $L_2 \in P \Rightarrow L_1 \in P$.

Satz

COLORING $\leq_p$ SAT.

Wdh.: Das Erfüllbarkeitsproblem – SAT

Problem (Erfüllbarkeitsproblem / Satisfiability – SAT)

Eingabe: Aussagenlogische Formel φ in KNF
Frage: Gibt es eine erfüllende Belegung für φ?

SAT-Beispiel 1:

$$\varphi = (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_4) \wedge (x_2 \vee x_3 \vee x_4)$$

φ ist erfüllbar, denn $x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0$ ist eine erfüllende Belegung.

Vorlesung 15
NP-Vollständigkeit

Wdh.: NP-Vollständigkeit

Definition (NP-vollständig)

Ein Problem L heißt NP-vollständig (engl. NP-complete), falls gilt

1. $L \in \mathrm{NP}$, und
2. L ist NP-schwer.

Die Klasse der NP-vollständigen Probleme wird mit NPC bezeichnet.

Satz (Cook und Levin)

SAT ist NP-vollständig.

Lemma

3-SAT $\in$ NP und SAT $\leq_p$ 3-SAT.

Korollar

3-SAT ist NP-vollständig.

Vorlesung 16
NP-Vollständigkeit ausgewählter Zahlprobleme

Wdh.: NP-Vollständigkeit von Zahlproblemen

Satz
SUBSET-SUM ist NP-vollständig.

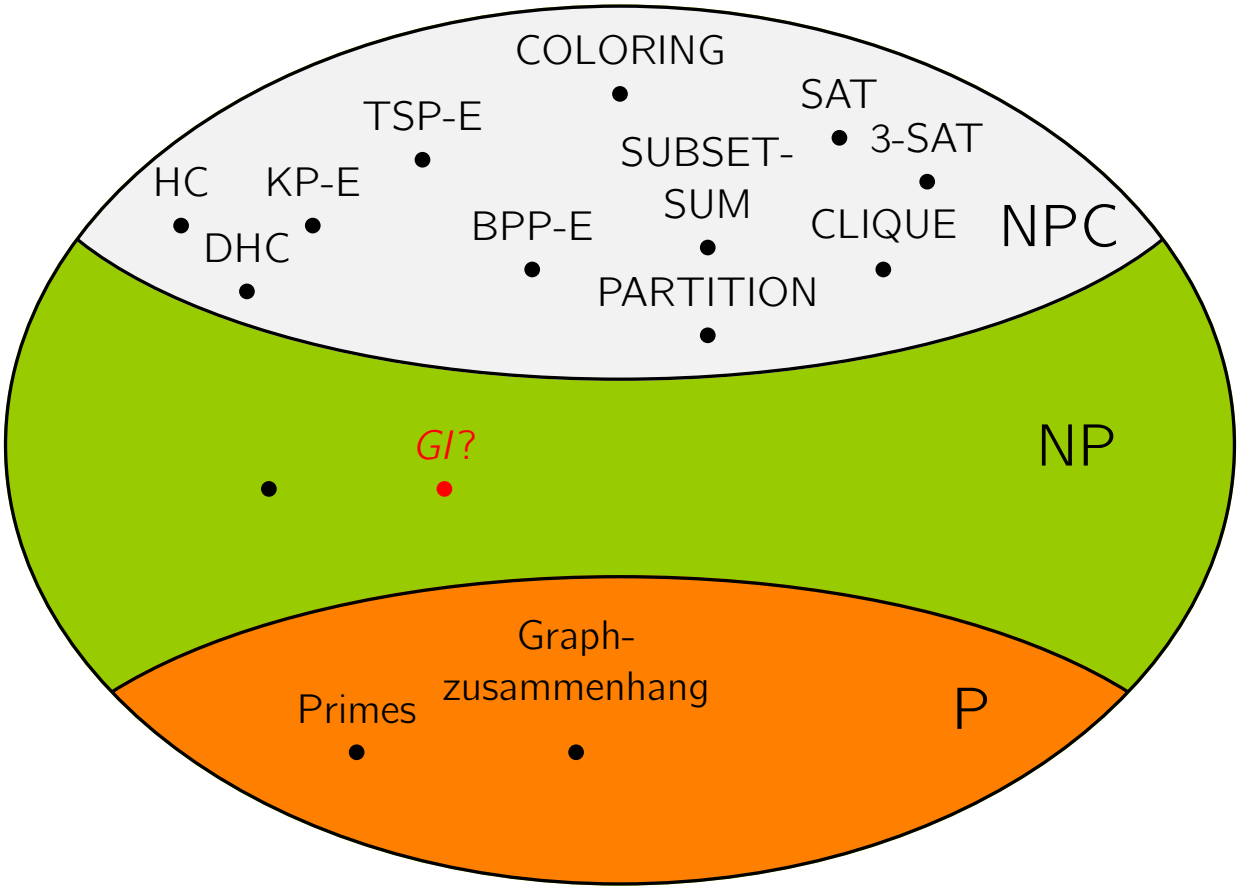
Satz
PARTITION ist NP-vollständig.

Satz
BPP-E ist NP-vollständig.

Satz
KP-E ist NP-vollständig.

Vorlesung 17
NP-Vollständigkeit ausgewählter Graphprobleme

Wdh.: Die Komplexitätslandschaft



Warnung: Dieser Abbildung liegt die Annahme $P \neq NP$ zu Grunde.

Vorlesung 18
Ausblick: Algorithmen und Komplexität

Vorlesung 19
Zusammenfassung

Berechenbarkeits- und Komplexitätslandschaft