Notizen zur Vorlesung
Theoretische Informatik und Logik

Zusammenfassung

Notizen zur Vorlesung https://iccl.inf.tu-dresden.de/web/Theoretische_Informatik_und_Logik_(SS2017).

Ich versuche möglichst ohne formelle Symbole und Definitionen zu arbeiten, daher verweisen die Markierungen jeweils auf die Vorlesungsnummer in FS bzw. TIL. Obwohl der Schwerpunkt auf TheoLog liegt, habe ich ein paar Definitionen aus Formale Systeme mit einbezogen, da TheoLog diese weiterverwendet.

Einige Formulierungen habe ich aus den hervorragenden Folien von Prof. Krötzsch geliehen. Quellen dieser Folien sind auf Github zu finden unter https://github.com/mkroetzsch und sind unter der Lizenz CC BY 3.0 DE verwendbar. Für diese gilt: „(C) Markus Krötzsch, https://iccl.inf.tu-dresden.de/web/TheoLog2017, CC BY 3.0 DE“.

Inhaltsverzeichnis

Autor	Dominik Pataky
Dozent	Prof. Markus Krötzsch
Ort	Fakultät Informatik, TU Dresden
Zeit	Sommersemester 2017
Letztes Update	1. Juni 2017
Lizenz	CC BY-SA 4.0

1 Formale Systeme

1.1 Sprachen und Automaten

(formale) Sprache Menge von Wörtern/Symbolen/Tokens, z.B. Programmiercode oder natürliche Sprache. Zusätzliche Begriffe: Konkatenation, Präfix/Suffix/Infix, leeres Wort FS 1

Symbol Token der Sprache, z.B. if/else, +/-, True/False, "Hello World"-String

Alphabet nichtleere, endliche Menge von Symbolen

Wort endliche Sequenz von Symbolen

Grammatik formelle Spezifikation einer Sprache. Aus einer Grammatik kann man widerum eine Sprache erzeugen FS 2

Rechenoperationen Vereinigung, Schnitt, Komplement, Produkt, Potenz, Kleene-Abschluss

Abschlusseigenschaft Beispiel: Wenn Sprache A und Sprache B regulär sind, wäre dann auch der Schnitt der beiden Sprachen wieder regulär? FS 5

Automat Beginnt von einem Startzustand und folgt je nach Eingabe seinen Übergängen in die jeweiligen Zustände. Akzeptiert, wenn letzter Zustand ein akzeptierter Endzustand.

Deterministischer endlicher Automat (DFA) erkennen reguläre Sprachen FS 3

nichtdeterministischer endlicher Automat (NFA) „rät" die richtigen Übergänge, arbeitet parallel. Nichtdeterminismus sinnvoll? Kompaktere Darstellungen, Start für Entwicklung DFA, kann bei Untersuchung Komplexität/Berechenbarkeit helfen FS 4

Kellerautomat (PDA) PDA erweitern endliche Automaten um einen unbeschränkt großen Speicher, der aber nur nach dem LIFO-Prinzip verwendet werden kann. PDAs erkennen genau die kontextfreien Sprachen. FS 15

Turingmaschine (TM) liefert allgemeines Modell der Berechnung. Liest und schreibt in einem Schritt, hat unendlichen Speicher, kann beliebig auf Speicher zugreifen (im Gegensatz zu LIFO bei PDA). Kann ein Band oder mehrere Bänder haben. Kann deterministisch (DTM) oder nichtdeterministisch (NTM) sein. Alle Varianten der TM können die selben Funktionen berechnen (Probleme lösen) - einzig der Aufwand ist unterschiedlich (NTM kann DTM darstellen, NTM kann durch DTM simuliert werden etc.). Siehe auch Church-Turing-These. FS 18

Kardinalität Unterscheidung abzählbar (mit natürlichen Zahlen) und überabzählbar

Chomsky-Hierarchie Kategorische Einteilung von Sprachen je nach Komplexität ihrer Grammatik. Hierarchie 0 > 1 > 2 > 3. These: „Die meisten Sprachen können nicht mit Grammatiken beschrieben werden (abzählbar viele Grammatiken vs. überabzählbar viele Sprachen)". FS 2

Typ 0 beliebige Grammatiken (Turingmaschinen)

Typ 1 kontextsensitive Grammatiken

Typ 2 kontextfreie Grammatiken (CYK, Kellerautomaten)

Typ 3 reguläre Grammatiken (DFA, NFA, Pumping Lemma)

Probleme Probleme formulieren Berechnungsfragen.

Wortproblem Wortproblem für eine Sprache über einem Alphabet ist die Bestimmung der Ausgabe „ja, Wort ist in Sprache" oder „nein, Wort ist nicht in Sprache", für die Eingabe eines Wortes gebildet aus dem Alphabet FS 3

Leerheitsproblem (DFA, NFA) Entscheidung für „ja, Automat erzeugt Sprache" oder „nein, durch den Automaten erzeugte Sprache ist leer" (es wird nie ein Endzustand erreicht). FS 10

Inklusionsproblem (DFA, NFA) Entscheidung für „ja, Sprache A eines Automaten ist eine Teilmenge der Sprache B eines anderen Automaten" oder „nein, Sprache A ist keine Teilmenge der Sprache B". FS 10

Äquivalenzproblem (DFA, NFA) Entscheidung für „ja, Sprache A eines Automaten ist gleich der Sprache B eines anderen Automaten" oder „nein, Sprache A unterscheidet sich von Sprache B". FS 10

Endlichkeitsproblem (DFA, NFA) Entscheidung für „ja, erzeugte Sprache eines Automaten ist endlich“ oder „nein, erzeugte Sprache ist nicht endlich“ (z.B., wenn Zyklen auf dem Pfad von Start- zu Endzustand existieren). FS 10

Universalitätsproblem (DFA, NFA) Entscheidung für „ja, erzeugte Sprache eines Automaten ist Σ^*“ oder „nein, erzeugte Sprache ist nicht Σ^*“ (heißt, Komplement der erzeugten Sprache ist leer). FS 10

Halteproblem (TM) Entscheidet, ob eine Turingmaschine für eine Eingabe jemals hält oder nicht. Unentscheidbar. FS 19

Church-Turing-These Die Turingmaschine kann alle Funktionen berechnen, die intuitiv berechenbar sind. Auch: „Eine Funktion ist genau dann im intuitiven Sinne berechenbar, wenn es eine Turingmaschine gibt, die für jede mögliche Eingabe den Wert der Funktion auf das Band schreibt und anschließend hält.“ FS 18

Entscheidbarkeit Eine Sprache L ist entscheidbar / berechenbar / rekursiv, wenn es eine Turingmaschine gibt, die das Wortproblem der Sprache L entscheidet. D.h. die Turingmaschine ist Entscheider und die Sprache L ist gleich der durch die Turingmaschine erkannten Sprache. Andernfalls heißt die Sprache L unentscheidbar.
Die Sprache L ist semi-entscheidbar / Turing-erkennbar / rekursiv aufzählbar, wenn es eine Turingmaschine gibt, deren erkannte Sprache zwar L ist, jedoch die Turingmaschine kein Entscheider sein muss. FS 19

Satz von Rice Sei E eine Eigenschaft von Sprachen, die für manche Turing-erkennbare Sprachen gilt und für manche Turing-erkennbare Sprachen nicht gilt (= „nicht-triviale Eigenschaft“).
Dann ist das folgende Problem unentscheidbar: die Eingabe besteht aus einer Turingmaschine. Wir wollen prüfen, ob die durch diese Turingmaschine erkannte Sprache die Eigenschaft E besitzt. Der Beweis für die Unentscheidbarkeit dieses Problems ist eine Reduktion vom Halteproblem. FS 20

1.2 Aussagenlogik

Die Aussagenlogik untersucht logische Verknüpfungen von atomaren Aussagen. FS 21

Atomare Aussage Behauptungen, die wahr oder falsch sein können.
Auch: aussagenlogische Variablen, Propositionen, Atome

Operatoren, Junktoren Verknüpfung von atomaren Aussagen.
Negation, Konjunktion, Disjunktion, Implikation, Äquivalenz.
Können auch äquivalent durch andere Junktoren ausgedrückt werden (de Morgan). FS 22
Eine Disjunktion von Literalen nennt man **Klausel**.
Eine Konjunktion von Literalen nennt man **Monom**.

Formel Jedes Atom ist eine Formel, jede durch Junktoren verknüpfte Formeln sind wieder Formeln. Diese zusammengesetzten Formeln bestehen dann wieder aus Teilformeln (auch: Unterformeln, $Sub(F)$). Eine Formel, die nur aus einem Atom besteht, nennt man auch **Literal**. Literale können die Form x oder $\neg x$ (für x Atom) annehmen.

unerfüllbar Formel hat keine Modelle

erfüllbar Formel hat mindestens ein Modell

allgemeingültig alle Interpretationen sind Modelle für Formel. Auch: **Tautologie**, $\models F$

widerlegbar Formel ist nicht allgemeingültig

Syntax „Sprache einer Logik“ (Formeln mit logischen Operatoren). Wichtig: Klammerung.

Semantik Definition der Bedeutung. Wertzuweisung von Wahrheitswerten zu Atomen mit Hilfe der Interpretation. „Die Bedeutung einer Formel besteht darin, dass sie uns Informationen darüber liefert, welche Wertzuweisungen möglich sind, wenn die Formel wahr sein soll.“

Interpretation eine Funktion w, die von einer Menge Atome auf die Menge {0, 1} abbildet.

Wahrheitstabelle Schrittweise Auflösung einer Formel durch Lösen ihrer Teilformeln.

Modell eine Interpretation, dessen Abbildung eine Formel nach 1 löst.

Logische Konsequenz eine Formel G ist eine logische Konsequenz einer Formel F ($F \models G$), wenn jedes Modell von F auch ein Modell für G ist.

Logische Äquivalenz zwei Formeln F und G sind semantisch äquivalent ($F \equiv G$), wenn sie genau dieselben Modelle haben FS 22

Normalform jede Formel lässt sich in eine äquivalente Formel in Normalform umformen.
Für die Umformungen gibt es Algorithmen, siehe FS 22

Negationsnormalform (NNF) enthält nur UND, ODER und Negation, wobei Negation nur direkt vor Atomen vorkommt.

Konjunktive Normalform (KNF) Formel ist eine Konjunktion von Disjunktionen von Literalen.

Disjunktive Normalform (DNF) Disjunktion von Konjunktionen von Literalen.

1.3 Komplexität

Turingmaschinen sind begrenzt durch die Anzahl ihrer Speicherzellen (Speicher) und der Anzahl möglicher Berechnungsschritte (Zeit). Schranken sind Funktionen gerichtet nach der Länge der Angabe. FS 24

O-Notation charakterisiert Funktionen nach ihrem Verhalten und versteckt Summanden kleinerer Ordnung und lineare Faktoren. Beispiel: ein Polynom $n^4 + 2n^2 + 150$ wird zu $O(n^4)$.

$O(f)$-zeitbeschränkt es gibt eine Funktion $g \in O(f)$, so dass eine DTM/NTM bei beliebiger Eingabe der Länge n nach einer maximalen Anzahl Schritte $g(n)$ anhält.

$O(f)$-speicherbeschränkt es gibt eine Funktion $g \in O(f)$, so dass eine DTM/NTM bei beliebiger Eingabe der Länge n nur eine maximale Anzahl Speicherzellen $g(n)$ verwendet.

Sprachklassen Einteilung von Sprachen nach der Möglichkeit der Entscheidbarkeit.
„Klasse aller Sprachen, welche…“

- **DTIME**$(f(n))$ …durch eine $O(f)$-zeitbeschränkte DTM entschieden werden können
- **DSPACE**$(f(n))$ …durch eine $O(f)$-speicherbeschränkte DTM entschieden werden können
- **NTIME**$(f(n))$ …durch eine $O(f)$-zeitbeschränkte NTM entschieden werden können
- **NSPACE**$(f(n))$ …durch eine $O(f)$-speicherbeschränkte NTM entschieden werden können

Komplexitätsklassen erfassen Sprachklassen je nach ihrer Komplexität. Stehen untereinander in Beziehung und bilden quasi Hierarchie. FS 24

- **PTime (P)** deterministisch, polynomielle Zeit
- **ExpTime (Exp)** deterministisch, exponentielle Zeit
- **LogSpace (L)** deterministisch, logarithmischer Speicher
- **PSpace** deterministisch, polynomieller Speicher
- **NPTime (NP)** nichtdeterministisch, polynomielle Zeit
- **NExpTime (NExp)** nichtdeterministisch, exponentielle Zeit
- **NLogSpace (NL)** nichtdeterministisch, logarithmischer Speicher
- **NPSpace** nichtdeterministisch, polynomieller Speicher (gleich PSpace)

SAT Boolean Satisfiability Problem. Problem, welches ein Modell für eine Formel auf Erfüllbarkeit untersucht. In NP. Interessant für Untersuchung, da SAT ein Problem darstellt, für welches es wahrscheinlich schwierig ist eine Lösung zu finden, jedoch sehr einfach ist eine Lösung auf Korrektheit zu prüfen. FS 25

Reduktion Rückführung eines Problems auf ein anderes. Beispiel Drei-Farben-Problem ist auf SAT reduzierbar, da sich die Farb-Zustände als Formeln ausdrücken kodieren lassen. „Alle Probleme in NP können polynomiell auf SAT reduziert werden“ (**Cook, Levin**)

Härte und Vollständigkeit für P und NP FS 25

- **NP-hart** Sprache ist NP-hart, wenn jede Sprache in NP polynomiell darauf reduzierbar ist (Beispiel Halteproblem und jedes weitere unentscheidbare Problem).
- **NP-vollständig** Sprache ist NP-hart und liegt selbst in NP (Beispiel SAT).
- **P-hart** Sprache ist P-hart, wenn jede Sprache in P mit logarithmischem Speicherbedarf auf diese reduzierbar ist.
- **P-vollständig** Sprache ist P-hart und liegt selbst in P (Beispiel HornSAT).

Zusammenfassung aller Themenkomplexe, Hierarchien und Zusammenhänge in FS 26.

2 Theoretische Informatik

2.1 Turingmaschinen

Turingmaschine deterministisch als DTM oder nichtdeterministisch als NTM.
Definiert als Tupel $(Q, \Sigma, \Gamma, \delta, q_0, F)$ mit endlicher Menge von Zuständen Q, Eingabealphabet Σ, Arbeitsalphabet Γ, Übergangsfunktion δ, Startzustand q_0 und Menge von akzeptierenden Endzuständen F. Können ein oder mehrere Bänder haben. Siehe auch Church-Turing-These. FS 18 TIL 1

- **Funktion** Turingmaschine kann eine Funktion von Eingaben auf Ausgabewörter definieren. Wenn eine TM bei Eingabe w anhält und die Ausgabe der Form $v\sqcup\sqcup\ldots$ entspricht, hat diese TM die Funktion berechnet.
- **Sprache** die von einer Turingmaschine erkannte Sprache ist die Menge aller Wörter, die von dieser TM akzeptiert werden (d.h. in einem Endzustand hält).
- **Konfiguration** der „Gesamtzustand" einer TM, bestehend aus Zustand, Bandinhalt und Position des Lese-/Schreibkopfs; geschrieben als Wort (Bandinhalt), in dem der Zustand vor der Position des Kopfes eingefügt ist. Beispiel $\sqcup\sqcup q_0 aaba\sqcup\sqcup$.
- **Übergangsrelation** Beziehung zwischen zwei Konfigurationen wenn die TM von der ersten in die zweite übergehen kann (deterministisch oder nichtdeterministisch)
- **Lauf** mögliche Abfolge von Konfigurationen einer TM, beginnend mit der Startkonfiguration; kann endlich oder unendlich sein
- **Halten** Ende der Abarbeitung, wenn die TM in einer Konfiguration keinen Übergang mehr zur Verfügung hat.
- **Transducer** Ausgabe der Turingmaschine ist Inhalt des Bandes, wenn TM hält, ansonsten undefiniert. Endzustände sind irrelevant.
- **Entscheider** Ausgabe der Turingmaschine ist „Akzeptiert", wenn TM in Endzustand hält, ansonsten „verwirft" (beinhaltet auch „TM hält nicht"). Bandinhalt ist irrelevant.
- **Aufzähler** ist eine DTM, die bei Eingabe des leeren Bandes immer wieder (d.h. bis zum letzten Wort bei endlichen Sprachen) einen Zustand $q_{Ausgabe}$ erreicht, in welchem das aktuelle Band ein Wort aus der Sprache dieser DTM ist. Die Sprache dieser DTM ist dann die Menge der so erzeugten Wörter. Diese DTM muss nicht halten, die Sprache kann unendlich sein. Wörter dürfen mehrfach ausgegeben werden.

Berechenbarkeit bezogen auf Funktionen. Eine Funktion F heißt berechenbar, wenn es eine DTM gibt, die F berechnet. Ist durch geeignete Kodierung (z.B. binär) erweiterbar auf natürliche Zahlen, Wörterlisten und andere Mengen. TIL 2

- **rekursiv** eine berechenbare totale Funktion ist rekursiv.
- **partiell rekursiv** eine berechenbare partielle Funktion ist partiell rekursiv.

Entscheidbarkeit bezogen auf Sprachen. TIL 2

- **entscheidbar / berechenbar / rekursiv** es existiert eine Turingmaschine, die das Wortproblem der Sprache entscheidet. D.h. die Turingmaschine ist Entscheider und die Sprache ist gleich der Sprache der TM.
- **semi-entscheidbar / Turing-erkennbar / Turing-akzeptierbar / rekursiv aufzählbar** es existiert eine Turingmaschine, deren erzeugte Sprache gleich der Sprache ist, jedoch die TM kein Entscheider ist.
  Eine Sprache ist genau dann semi-entscheidbar, wenn es einen Aufzähler für diese Sprache gibt.
- **unentscheidbar** sonst.
  „Es gibt Sprachen und Funktionen, die nicht berechenbar sind." Beweis anhand der abzählbaren Menge von Turingmaschinen im Vergleich zur Überabzählbarkeit der Menge der Sprachen über jedem Alphabet.

Probleme der Kategorie „Unentscheidbar bzw. unberechenbar, nicht berechenbar". TIL 2

- **Busy-Beaver-Funktion** ist nicht berechenbar und wächst sehr schnell. Die Funktion nimmt eine natürliche Zahl n und gibt die maximale Anzahl x-Symbole, welche eine DTM mit n Zuständen und dem Arbeitsalphabet $\{x, \sqcup\}$ bis zu ihrem Halt schreiben kann, zurück.

2.2 LOOP und WHILE

LOOP und WHILE sind eine Erfindung von Schöning und sind quasi eine pädagogische Brücke zwischen den Ultra-low-level Turingmaschinen und High-level Programmiersprachen. WHILE baut auf LOOP auf. TIL 3

LOOP Besteht aus Variablen, Wertzuweisungen und Schleifen. Die Eingabe einer Menge von natürlichen Zahlen wird in $x_1, x_2, \dots$ gespeichert. Die Ausgabe ist eine natürliche Zahl, gespeichert in x_0. Alle weiteren Variablen haben den Wert 0. LOOP terminiert immer in endlich vielen Schritten. Berechnet eine totale Funktion.

Variablen Menge $\{x_0, x_1, \dots\}$ oder auch $\{x, y, myVariable\}$. Haben als Wert eine natürliche Zahl.

Wertzuweisungen in der Form $x := y + n$ oder $x := y - n$, wobei n eine natürliche Zahl ist. Eine Wertzuweisung ist bereits ein LOOP-Programm.

Schleifen in der Form LOOP x DO P END, wobei P wieder ein LOOP-Programm ist. Der Wert der Variable x kann in P nicht geändert werden. Daher terminiert ein LOOP-Programm immer in endlich vielen Schritten.

Hintereinanderausführung wenn P_0 und P_1 LOOP-Programme, dann auch $P_0; P_1$.

Syntax-Erweiterung Die Syntax lässt sich zur Vereinfachung erweitern.

Wertzuweisung $x := y$ $x := y + 0$

Rücksetzen $x := 0$ LOOP x DO $x := x - 1$ END

Wertzuweisung Zahl $x := n$ $x := 0; x := x + n$. Alternativ $x := null + n$

Variablen-Addition $x := y + z$ $x := y$; LOOP z DO $x := x + 1$ END

Bedingung **IF** $x \neq 0$ **THEN** LOOP x DO $y := 1$ END ; LOOP y DO P END

Berechenbarkeit eine Funktion heißt LOOP-berechenbar, wenn es ein LOOP-Programm gibt, welches die Funktion berechnet. Auch hier ist mit geeigneter Kodierung wieder mehr machbar, als nur die natürlichen Zahlen in Betracht zu ziehen (Beispiel Wortproblem, Probleme in NP, gängige Algorithmen). Es gibt berechenbare totale Funktionen, die nicht LOOP-berechenbar sind (vgl. Ackermannfunktion).

WHILE Basiert auf LOOP und erweitert dieses. Jedes LOOP-Programm ist auch ein WHILE-Programm.

Schleifen in der Form WHILE $x \neq 0$ DO P WHEN, wobei P wieder WHILE-Programm. Im Gegensatz zu LOOP kann in WHILE der Wert von x in P zur Laufzeit geändert werden. Es kann also passieren, dass das Programm nicht terminiert wenn x nie auf 0 gesetzt wird.

Konvertierung LOOP-Schleifen können in WHILE-Schleifen konvertiert werden. Eine DTM kann WHILE-Programme simulieren und ein WHILE-Programm DTMen simulieren.

Berechenbarkeit Eine partielle Funktion heißt WHILE-berechenbar, wenn es ein WHILE-Programm gibt, welches bei einem definierten $f(n_0, n_1, \dots)$ terminiert und bei einem nicht definierten Wertebereich nicht terminiert. Wenn eine partielle Funktion WHILE-berechenbar ist, ist sie **Turing-berechenbar**.

2.3 Universalität

Universalmaschine U eine Turingmaschine, die andere TM als Eingabe kodiert erhält und diese simuliert. Die Kodierung ist dabei z.B. binär, mit dem Trennsymbol $\#$. Hat vier Bänder: Eingabeband von U mit kodierter TM und kodierter Eingabe w, Arbeitsband von U, Band 3 mit aktuellem Zustand der simulierten TM und Band 4 als Arbeitsband der simulierten TM.
Für die Arbeitsweise siehe TIL 4

2.4 Unentscheidbare Probleme und Reduktionen

Beweis durch Diagonalisierung, Reduktionen TIL 4

Probleme der Kategorie „unentscheidbar“.

Halteproblem $\mathbf{P}_{Halt}$ Frage: „Gegeben eine Turingmaschine M und ein Wort w. Wird die Turingmaschine M für die Eingabe w jemals anhalten?“. Das Halteproblem $\mathbf{P}_{Halt}$ der Turingmaschine M für das Wort w kann formal kodiert werden als $enc(M)\#\#enc(w)$ und einer universellen Turingmaschine zur Überprüfung übergeben werden. Beweise für Unentscheidbarkeit anhand Diagonalisierung und Reduktion in TIL 4

Goldbachsche Vermutung Beispiel für ein auf das Halteproblem reduzierbares Problem. Besagt, dass jede gerade Zahl $n \geq 4$ die Summe zweier Primzahlen ist. Zum Beispiel ist $4 = 2 + 2$ und $100 = 47 + 53$. Lässt man nun eine Turingmaschine diese Vermutung systematisch beginnend bei 4 testen, würde ein Anhalten bei Misserfolg $\mathbf{P}_{Halt}$ und „die Vermutung stimmt nicht“ gleichzeitig lösen. Gäbe es demnach ein Programm, welches $\mathbf{P}_{Halt}$ lösen kann (entscheidet), wäre eine separate Überprüfung der Goldbachschen Vermutung nicht nötig. Die Frage der Goldbachschen Vermutung wäre sofort beantwortet.

ϵ **-Halteproblem** „Gegeben sei eine Turingmaschine. Wird diese TM für die leere Eingabe ϵ jemals anhalten?“. Unentscheidbar.

Beweismethoden zum Nachweis der Unentscheidbarkeit.

Kardinalität Beweis von Aussagen anhand der unterschiedlichen Kardinalitäten.

Diagonalisierung Berechenbarkeit annehmen und einen paradoxen Algorithmus für das Problem konstruieren.

Reduktion Rückführung eines unentscheidbaren Problems auf das gegebene. Die Reduktion ist ein Entscheidbarkeitsalgorithmus. Siehe auch *1.3 Komplexität*. TIL 4

Turing-Reduktion Ein Problem **P** ist Turing-reduzierbar auf ein Problem **Q** (in Symbolen: $\mathbf{P} \leq_T \mathbf{Q}$), wenn man **P** mit einem Programm lösen könnte, welches ein Programm für **Q** als Unterprogramm (auch: Subroutine) aufrufen darf. Das Programm für **Q** muss hierbei nicht existieren.

Many-One-Reduktion Eine berechenbare totale Funktion $f : \Sigma^* \to \Sigma^*$ ist eine Many-One-Reduktion von einer Sprache **P** auf eine Sprache **Q** (in Symbolen: $\mathbf{P} \leq_m \mathbf{Q}$), wenn für alle Wörter $w \in \Sigma^*$ gilt: $w \in \mathbf{P}$ gdw. $f(w) \in \mathbf{Q}$.
Schwächer als Turing-Reduktion, jede Many-One-Reduktion kann als Turing-Reduktion ausgedrückt werden (dies gilt jedoch nicht andersherum).

Satz von Rice Siehe *1.1 Sprachen und Automaten*. TIL 5
„Praktisch alle interessanten Fragen zu Sprachen von Turingmaschinen sind unentscheibar“. Eingabe ist eine Turingmaschine, Ausgabe „hat die Sprache der TM die Eigenschaft?“.

2.5 Semi-Entscheidbarkeit

Hinweis: Hierzu gibt es im Schöning gute graphische Darstellungen. TIL 5

Komplement einer Sprache L: $\overline{L} = \{w \in \Sigma^* \mid w \notin L\}$ (Achtung: auf Kontext achten. Komplement des Halteproblems ist z.B. anderer Form). Die Turing-Reduktionen $\overline{L} \leq_T L$ bzw. $L \leq_T \overline{L}$ sind mit einer Turingmaschine überprüfbar. Für eine Eingabe w entscheidet diese, ob $w \in L$ und invertiert das Ergebnis.

Semi-Entscheidbarkeit Beispiel anhand des Halteproblems: simuliere eine Turingmaschine und deren Eingabe, kodiert als $enc(M)\#\#enc(w)$. Wenn M hält, hält auch die universelle Turingmaschine und akzeptiert. Eine Sprache L ist entscheidbar, wenn sowohl L als auch $\overline{L}$ semi-entscheidbar sind.

Co-Semi-Entscheidbarkeit Wenn eine Sprache L unentscheidbar, jedoch semi-entscheidbar ist, kann $\overline{L}$ nicht semi-entscheidbar sein.

2.6 Postsches Korrespondenzproblem

Auch: **PCP**. Ein unentscheidbares Problem ohne direkten Bezug zu einer Berechnung. TIL 5

PCP Bei diesem Problem nimmt man eine Reihe von 2-Tupeln (anschaulich vergleichbar mit Dominosteinen) mit je einem Wert oben und einem unten. Ziel der Lösung ist nun, die gegebenen Tupel so anzuordnen, dass oben und unten die gleiche Wortkette entsteht. Beispiel: wir haben die drei Tupel (AB, B), (B, BBB) und (BB, BA). Eine Anordnung mit zehn Tupeln ergibt dann die Lösung. Es kann vorkommen, dass das Problem keine Lösung besitzt.

MPCP Hilfskonstruktion. Wir nutzen MPCP, um das Halteproblem auf MPCP zu reduzieren. Folgend reduzieren wir MPCP auf PCP. Beim MPCP wird PCP verwendet, jedoch das Start-Tupel vorgegeben. Die Lösung eines MPCP ist auch eine Lösung des entsprechenden PCP, welche mit dem gegebenen Start-Tupel beginnt.

2.7 Unentscheidbare Probleme formaler Sprachen

In diesem Kapitel wird wieder auf *1.1 Sprachen und Automaten* zurückgegriffen. Eine durch eine Grammatik G erzeugte Sprache wird als $L(G)$ bezeichnet. Für Beweise der folgenden Sätze siehe Vorlesung. Siehe auch Chomsky-Hierarchie in *1.1 Sprachen und Automaten*. TIL 6

- Das Schnittproblem regulärer Grammatiken (Typ 3) ist entscheidbar.
- Das Schnittproblem kontextfreier Grammatiken (Typ 2, **CFG**) ist unentscheidbar. Beweis durch Many-One-Reduktion vom PCP.
- Das Leerheitsproblem für kontextfreie Grammatiken ist entscheidbar.
- Kontextfreie Sprachen sind unter Vereinigung abgeschlossen.
- Deterministische kontextfreie Sprachen sind unter Komplement abgeschlossen.
- Das Äquivalenzproblem kontextfreier Grammatiken ist unentscheidbar.

2.8 Komplexitätstheorie

Untersuchung von Problemkomplexitäten und Suche nach Methoden zur Bestimmung der Komplexität eines Problems. Klassierung zwischen „leicht lösbar“ bis „schwer lösbar“. Einteilung von berechenbaren Problemen entsprechend der Menge an Ressourcen, die zu ihrer Lösung nötig sind. Einführung anhand von Beispielen. TIL 7

Eulerpfad Ein Eulerpfad ist ein Pfad in einem Graphen, der jede Kante genau einmal durchquert. Ein Eulerkreis ist ein zyklischer Eulerpfad. Ein Graph hat genau dann einen Eulerschen Pfad, wenn er maximal zwei Knoten ungeraden Grades besitzt und zusammenhängend ist.

Hamiltonpfad Ein Hamiltonpfad ist ein Pfad in einem Graphen, der jeden Knoten genau einmal durchquert. Ein Hamiltonkreis ist ein zyklischer Hamiltonpfad.

Schranken von Turingmaschinen in Zeit und Raum. Siehe *1.3 Komplexität*.

Speicher Zahl der verwendeten Speicherzellen

Zeit Zahl der durchgeführten Berechnungsschritte

O**-Notation** Siehe *1.3 Komplexität*.

Linear Speedup Theorem Sei M eine Turingmaschine mit $k > 1$ Bändern, die bei Eingaben der Länge n nach maximal $f(n)$ Schritten hält. Dann gibt es für jede natürliche Zahl $c > 0$ eine äquivalente k-Band Turingmaschine M', die nach maximal $\frac{f(n)}{c} + n + 2$ Schritten hält.
Bedeutet: in der Theorie kann jedes Programm mit Hilfe mehrerer Bänder „beliebig schneller“ gemacht werden. Dies ist praktisch nicht umsetzbar, da eine Turingmaschine nicht beliebig große Datenmengen in einem Schritt lesen und nicht beliebig komplexe Zustandsübergänge in konstanter Zeit realisieren kann.

2.9 Beziehungen der Komplexitätsklassen

Siehe *1.3 Komplexität* für eine Übersicht der Klassen. TIL 7

Nichtdeterministische Klassen $NL \subseteq NP \subseteq NPSpace \subseteq NExp$

DTM auch als NTM, d.h. nichtdet. stärker $L \subseteq NL$, $P \subseteq NP$, $PSpace \subseteq NPSpace$, $Exp \subseteq NExp$

Satz von Savitch Speicherbeschränkte NTM können durch DTMs nur mit quadratischen Mehrkosten simuliert werden. Insbesondere gilt damit $PSpace = NPSpace$.

Zusammenfassend: $L \subseteq NL \subseteq P \subseteq NP \subseteq PSpace = NPSpace \subseteq Exp \subseteq NExp$.
Jedoch ist zu beachten:

- Wir wissen nicht, ob irgendeines dieser $\subseteq$ sogar $\subsetneq$ ist.
- Insbesondere wissen wir nicht, ob $P \subsetneq NP$ oder $P = NP$.
- Wir wissen nicht einmal, ob $L \subsetneq NP$ oder $L = NP$.

3 Repetitorien

3.1 Repetitorium I

Aufgabe A

Wiederholung von Begriffen Einband Turing-Maschine, Mehrband Turing-Maschine, Entscheidungsproblem, Unentscheidbarkeit, Aufzählbarkeit, Abzählbarkeit und Halteproblem.

Aufgabe B

Zeigen Sie: Wenn es möglich ist, für zwei beliebige Turing-Maschinen zu entscheiden, ob sie dieselbe Sprache akzeptieren, so ist es auch möglich, für beliebige Turing-Maschinen zu entscheiden, ob sie die leere Sprache akzeptieren. Seien K, M_1, M_2 Turingmaschinen, so dass $K(enc(M_1)\#\#enc(M_2))$ akzeptiert $\Leftrightarrow L(M_1) = L(M_2)$ und K hält auf jeder Eingabe.
Lösung: Sei M Turingmaschine und sei $M_\emptyset$ eine Turingmaschine, so dass $L(M_\emptyset) = \emptyset$.
Dann gilt $K(enc(M)\#\#enc(M_\emptyset))$ akzeptiert $\Leftrightarrow L(M) = \emptyset$, also $\mathbf{P}_{Leer} \leq_m \mathbf{P}_{\ddot{A}quiv}$.

Aufgabe C

Zeigen Sie, dass $\{1\}^*$ unentscheidbare Teilmengen besitzt.
Lösung: $\{1\}^*$ ist abzählbar unendlich, also ist $\mathfrak{P}(\{1\}^*)$ überabzählbar. Es gibt aber nur abzählbar unendlich viele entscheidbare Sprachen (auch: abzählbar viele nicht-äquivalente Turingmaschinen). Also sind einige (fast alle) dieser Sprachen unentscheidbar.

Aufgabe D

a) „Jedes LOOP-Programm terminiert." – Richtig. Definition von LOOP sagt, dass Anzahl Durchläufe nicht mehr während der Laufzeit geändert werden kann, demnach gibt es eine endliche Anzahl Durchläufe.

b) „Zu jedem WHILE-Programm gibt es ein äquivalentes LOOP-Programm." – Falsch, nicht zu jedem WHILE-Programm gibt es immer ein äquivalentes LOOP-Programm. Dies liegt daran, dass LOOP keine partiellen Funktionen verarbeiten kann.
Beispiel anhand von Division: LOOP terminiert immer, jedoch wäre Division durch 0 (ebenfalls in $\mathbb{N}$) undefiniert. Kann demnach nur mit WHILE gelöst werden (Fall $x_2 = 0$ für $div(x_1, x_2)$ landet in Endlosschleife).

c) „Die Anzahl der Ausführungen von P in der LOOP-Schleife LOOP x_i DO P END kann beeinflusst werden, indem x_i in P entsprechend modifiziert wird." – Falsch, Anzahl Schleifen kann laut Definition von LOOP nicht während Laufzeit geändert werden.

d) „Die Ackermannfunktion ist total und damit LOOP-berechenbar." – Falsch, die Ackermannfunktion ist zwar total, jedoch nicht LOOP-berechenbar (jedoch berechenbar). Die Funktion wurde gezielt gesucht und gefunden, um genau diesen Fall zu zeigen.

Aufgabe E

Geben Sie eine Turing-Maschine A_{mod2} an, die die Funktion $f : \mathbb{N} \to \mathbb{N}$ mit $f(x) = (x \ mod \ 2)$ berechnet. Stellen Sie dabei die Zahlen in unärer Kodierung dar.
Lösung: $A_{mod2} = (\{q_0, \ldots, q_f\}, \{x\}, \{x, \sqcup\}, \delta, q_0, \{q_f\})$. Die Turingmaschine liest die eingegebenen x (unäre Kodierung) und wechselt zwischen q_0 und q_1. Sobald auf ein $\sqcup$ gestoßen wird, weiß die TM, ob eine gerade oder ungerade Anzahl x eingegeben wurde. Wenn gerade, lösche alle x und ende in leerem Band. Wenn ungerade, lösche alle x und schreibe zum Abschluss ein x auf das Band.

Aufgabe F

Es sei $f : \mathbb{N} \to \mathbb{N}$ mit $f(x) = \lfloor \log_{10}(x) \rfloor$. Geben Sie ein WHILE-Programm an, welches f berechnet.
Lösung: Erst eine endlose WHILE-Schleife für die Eingabe $x = 0$. Dann Lösung mit $div(x, 10)$.

Aufgabe G

a) „Die Menge der Instanzen des Postschen Korrespondenzproblems, welche eine Lösung haben, ist semi-entscheidbar.“ – Richtig. Wenn eine Lösung existiert, kann diese (z.B. durch Breitensuche) auch gefunden werden.

b) „Das Postsche Korrespondenzproblem ist bereits über dem Alphabet $\Sigma = \{a, b\}$ nicht entscheidbar.“ – Richtig, denn Instanzen können über Σ kodiert werden, ohne die Entscheidbarkeit zu beeinflussen.

c) „Es ist entscheidbar, ob eine Turingmaschine nur Wörter akzeptiert, die Palindrome sind.“ – Falsch, Satz von Rice (die Akzeptanz von Palindromen ist eine **Eigenschaft**). Eigenschaft E ist „L besteht nur aus Palindromen“, diese Eigenschaft ist nicht-trivial: erfüllt z.B. durch $L = \emptyset$, jedoch nicht durch $L = \{ab\}$.

d) „ $\mathbf{P}_{Halt}$ ist semi-entscheidbar“ – Richtig, da es universelle Turingmaschinen gibt, die beliebige TM simulieren können.

e) „Es ist nicht entscheidbar, ob die von einer deterministischen Turing-Maschine berechnete Funktion total ist.“ – Richtig, denn sonst wäre das Halteproblem entscheidbar ($\mathbf{P}_{Halt} \leq_m \mathbf{P}_{Total}$).
Reduktion: M, w gegeben, baue Turingmaschine M' mit
$M' =$ bei Eingabe x
$\rightarrow$ simuliere M auf w
$\rightarrow$ akzeptiere mit leerem Band
M hält auf $w \Rightarrow M'$ berechnet $f(x) = \epsilon$
M hält nicht auf $w \Rightarrow M'$ berechnet Abbildung, die nirgends definiert ist.
Reduktion demnach $enc(M)\#\#enc(w) \mapsto enc(M')$.

f) „Es gibt reguläre Sprachen, die nicht semi-entscheidbar sind.“ – Falsch. Reguläre Sprachen sind immer entscheidbar, da Turingmaschinen endliche Automaten simulieren können.

Aufgabe H

Sei L eine unentscheidbare Sprache. Zeigen Sie:

a) „L hat eine Teilmenge $T \subseteq L$, die entscheidbar ist.“: $T = \emptyset$.

b) „L hat eine Obermenge $O \supseteq L$, die entscheidbar ist.“: $O = \Sigma^*$.

c) „Es gibt jeweils nicht nur eine sondern unendlich viele entscheidbare Teilmengen bzw. Obermengen wie in (a) und (b).“ – Es gilt: L ist unendlich. Dann ist die Menge der endlichen Teilmengen von L unendlich. Alles diese sind entscheidbar. Genauso für **b)**, z.B. muss $\Sigma^* \setminus L$ unendlich sein. Die Menge der endlichen Teilmengen E von $\Sigma^* \setminus L$ ist unendlich, für jede ist $\Sigma^* \setminus E$ entscheidbar.