Notizen zur Vorlesung
Theoretische Informatik und Logik

Zusammenfassung

Notizen zur Vorlesung https://iccl.inf.tu-dresden.de/web/Theoretische_Informatik_und_Logik_(SS2017). Die Markierungen verweisen 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	6. Juli 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 $L_1 \cup L_2$,
Schnitt $L_1 \cap L_2$,
Komplement $\overline{L} = \Sigma^* \setminus L$,
Produkt $L_1 \circ L_2 = \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$,
Potenz $L^0 = \{\epsilon\}$ und $L^{n+1} = L \circ L^n$,
Kleene-Abschluss $L^* = L^0 \cup L^1 \cup L^2 \cup \cdots = \bigcup_{i \geq 0} L^i$

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 auf das 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, siehe Savitch)

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

Die Theoretische Informatik beginnt mit der Berechenbarkeitstheorie. Hier nutzen wir das Maschinenmodell der Turingmaschine, um Aussagen über die Entscheidbarkeit von Problemen zu treffen. Die Berechenbarkeitstheorie klassifiziert Probleme demnach nach ihrer Berechenbarkeit bzw. Entscheidbarkeit.

Ab Kapitel 2.9 behandeln wir die Komplexitätstheorie. In dieser geht es um die Fragestellung, wie viel Zeit und Speicher ein entscheidbares und algorithmisch formalisierbares Problem benötigt, um von einer Maschine wie der Turingmaschine gelöst zu werden. Die Komplexitätstheorie klassifiziert also Probleme, von denen wir wissen, dass sie berechenbar/entscheidbar sind, nach ihrem Ressourcenaufwand.

2.1 Allgemeines

Surjektiv, Injektiv, Bijektiv Betrifft Abbildungen zwischen zwei Mengen. **Surjektiv** bedeutet, jedes Element in der Zielmenge wird mindestens einmal getroffen. **Injektiv** bedeutet, jedes Element in der Zielmenge wird höchstens einmal getroffen. **Bijektiv** bedeutet, die Abbildungen sind sowohl surjektiv als auch injektiv, es besteht eine eindeutige Zuordnung. *Merkhilfe: Paare „(Injektiv höchstens), (mindestens surjektiv)“ → IHMS in alphabetischer Reihenfolge*

Kardinalität, Mächtigkeit Die Menge der Elemente in einer Menge.
Eine unendliche Zielmenge ist **abzählbar**, wenn es eine bijektive Abbildung von der Menge $\mathbb{N}$ auf die Zielmenge gibt. Ist die Zielmenge endlich, ist sie natürlich ebenfalls abzählbar. **Überabzählbare** Mengen hingegen haben mehr Elemente, als Elemente in $\mathbb{N}$ sind.

2.2 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 $v\sqcup\sqcup\dots$ 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.

2.3 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**.

2.4 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

2.5 Unentscheidbare Probleme und Reduktionen

Beweis durch Diagonalisierung, Reduktionen TIL 4

Probleme der Kategorie „unentscheidbar“.

Halteproblem $\mathbf{P}_{\text{halt}}$ Frage: „Gegeben eine Turingmaschine M und ein Wort w. Wird die Turingmaschine M für die Eingabe w jemals anhalten?“. Das Halteproblem $\mathbf{P}_{\text{halt}}$ 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 $\mathbf{P}_{\text{halt}}$ und „die Vermutung stimmt nicht“ gleichzeitig lösen. Gäbe es demnach ein Programm, welches $\mathbf{P}_{\text{halt}}$ 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?“.

2.6 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.

2.7 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.

2.8 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.

2.9 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.

2.10 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 $L = NP$.

2.11 Zeit und Raum

Zusammenhänge zwischen Zeitklassen und Speicherklassen bzw. deren Beschränkungen. TIL 8
Es gilt für beliebige Funktionen $f : \mathbb{N} \to \mathbb{R}$:

- $\mathrm{DTIME}(f) \subseteq \mathrm{DSPACE}(f)$
- $\mathrm{DSPACE}(f) \subseteq \mathrm{DTIME}(2^{O(f)})$

Robustheit von Zeitklassen Setzt sich aus zwei Erkenntnissen zusammen:

- Konstante Faktoren haben keinen Einfluss auf die Probleme, die eine zeitbeschränkte Mehrband-TM lösen kann, sofern mindestens lineare Zeit erlaubt ist (Linear Speedup Theorem). Sofern nicht einmal lineare Zeit zur Verfügung stände, könnte die TM nicht einmal die Eingabe lesen!
- Die Anzahl der Bänder hat lediglich einen polynomiellen (quadratischen) Einfluss auf die Probleme, die eine zeitbeschränkte TM lösen kann.

Robustheit von Speicherklassen Weder konstante Faktoren, noch die Anzahl der Bänder haben Einfluss auf die Probleme, welche eine speicherbeschränkte TM lösen kann.

3 Prädikatenlogik

Die Prädikatenlogik erweitert die Aussagenlogik. Neben den neuen Mengen der Variablen **V**, der Konstanten **C**, der Funktionen **F** und Prädikate **P** kommen der Allquantor $\forall$ und der Existenzquantor $\exists$ hinzu. Semantisch nutzen wir Interpretationen, um für Formeln Modelle zu finden (d.h. Variablenbelegungen zu finden, für welche die Formel nach wahr ausgewertet wird).
Formeln können mit Hilfe von syntaktischen Umformungen umgeformt und vereinfacht werden. Dazu nutzen wir Algorithmen zum logischen Schließen, z.b. die Unifikation und Resolution. TIL 13

3.1 Syntax

Im Gegensatz zu der unendlichen Menge von Atomen in der Aussagenlogik gibt es in der Prädikatenlogik die vier betrachteten Mengen V, C, F und P. Diese Mengen sind abzählbar unendlich und die Elemente disjunkt. Formeln sind, ausgenommen genannter Ausnahmen, eindeutig zu klammern. Die Mengen V, C und F arbeiten mit beliebigen Werten, die Prädikate hingegen werten bei Interpretation immer nach $true$ oder $false$ aus. TIL 13

Variablen Die Menge **V**, bestehend aus $x, y, z \ldots$.
Variablen können frei oder gebunden vorkommen (oder bei mehrfachem Auftreten einer Variable in einer Formel auch beides).

Freie Variablen sind durch keinen Quantor gebunden.
Gebundene Variablen befinden sich innerhalb des „Scope“ eines Quantors.

Beispiel: in der Formel $p(x) \wedge \exists x.q(x)$ kommt x sowohl frei ($p(x)$) als auch gebunden ($q(x)$) vor.

Konstanten Die Menge **C**, bestehend aus $a, b, c, \ldots$

Funktionen Die Menge **F**, bestehend aus $f, g, h, \ldots$. Stelligkeit (Arität) ≥ 0.

Prädiktensymbole Die Menge **P**, bestehend aus $p, q, r, \ldots$. Stelligkeit ≥ 0. Bei nullstelligen Prädiktensymbolen lassen wir die leeren Klammern weg.

Quantoren Der Allquantor $\forall$ beschreibt, dass die betreffende Formel für alle möglichen Interpretationen der Variable gelten muss. Der Existenzquantor $\exists$ beschreibt, dass es mindestens eine gültige Interpretation der Variable geben muss. Wenn ein Quantor vor einer Formel mehrere Variablen betrifft, schreiben wir diese als Liste ($\forall x, y.F$ statt $\forall x.\forall y.F$).

Atom Ein prädikatenlogisches Atom ist ein Ausdruck $p(t_1, \ldots, t_n)$ für ein n-stelliges Prädiktensymbol $p \in \mathbf{P}$ und **Terme** $t_1, \ldots, t_n$. Hierbei gilt entweder $t_1, \ldots, t_n \in \mathbf{V} \cup \mathbf{C}$ oder $t_n = f(t_1, \ldots, t_i)$ mit $f \in F$ i-stelliges Funktionssymbol und $t_1, \ldots, t_i$ wieder Terme.

Formel Jedes Atom ist eine Formel. Wenn nun $x \in \mathbf{V}$ und F und G Formeln, dann sind auch $\neg F$, $(F \wedge G)$, $(F \vee G)$, $(F \rightarrow G)$, $(F \leftrightarrow G)$, $\exists x.F$ und $\forall x.F$ Formeln. Die äußersten Klammern von Formeln dürfen weggelassen werden. Klammern innerhalb von mehrfachen Konjunktionen oder Disjunktionen dürfen weggelassen werden. Hat eine Formel keine freie Variablen ist sie **geschlossen** und wird **Satz** genannt, ansonsten ist sie eine **offene** Formel.

Teilformel Teilformeln einer Formel sind alle Teilausdrücke einer Formel, welche selbst Formeln sind.

3.2 Semantik

Der Wahrheitswert von Formeln ergibt sich aus den Wahrheitswerten der Atome in dieser Formel. TIL 13

Interpretation Interpretation $\mathcal{I}$ ist ein Paar $\langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$.
Die nichtleere Menge $\Delta^{\mathcal{I}}$ wird auch **Domäne** genannt.
Die Funktion $\cdot^{\mathcal{I}}$ heißt **Interpretationsfunktion**. Diese bildet

- jede Konstante $a \in \mathbf{C}$ auf ein Element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$,
- jedes n-stellige Funktionssymbol $f \in F$ auf eine n-stellige Funktion $f^{\mathcal{I}} : (\Delta^{\mathcal{I}})^n \rightarrow \Delta^{\mathcal{I}}$ und
- jedes n-stellige Prädiktensymbol $p \in \mathbf{P}$ auf eine Relation $p^{\mathcal{I}} \in (\Delta^{\mathcal{I}})^n$ ab.

Zuweisung Zuweisung $\mathcal{Z}$ für eine Interpretation $\mathcal{I}$ ist eine Funktion $\mathcal{Z} : \mathbf{V} \rightarrow \Delta^{\mathcal{I}}$, sie bildet also Variablen auf Elemente der Domäne ab. Bei $x \in \mathbf{V}$ und $\delta \in \Delta^{\mathcal{I}}$ schreiben wir für die Zuweisung von x auf δ und für alle $y \neq x$ auf $\mathcal{Z}(y)$: $\mathcal{Z}[x \mapsto \delta]$.

Wahrheitsbestimmung Die Wahrheitsbestimmung von Atomen und Formeln unter einer Interpretation und einer Zuweisung werden rekursiv aufgelöst.

- Für Konstanten c benötigen wir nur die Interpretation: $c^{\mathcal{I},\mathcal{Z}} = c^{\mathcal{I}}$
- Für Variablen x benötigen wir nur die Zuweisung: $x^{\mathcal{I},\mathcal{Z}} = \mathcal{Z}(x)$
- Für einen Funktionsterm $t = f(t_1, \ldots, t_n)$ definieren wir: $t^{\mathcal{I},\mathcal{Z}} = f^{\mathcal{I}}(t_1^{\mathcal{I},\mathcal{Z}}, \ldots, t_n^{\mathcal{I},\mathcal{Z}})$
- Für Prädikate/Atome $p(t_1, \ldots, t_n)$ setzen wir nun rekursiv:
  $p(t_1, \ldots, t_n)^{\mathcal{I},\mathcal{Z}} = 1$ wenn $\langle t_1^{\mathcal{I},\mathcal{Z}}, \ldots, t_n^{\mathcal{I},\mathcal{Z}} \rangle \in p^{\mathcal{I}}$ bzw.
  $p(t_1, \ldots, t_n)^{\mathcal{I},\mathcal{Z}} = 0$ wenn $\langle t_1^{\mathcal{I},\mathcal{Z}}, \ldots, t_n^{\mathcal{I},\mathcal{Z}} \rangle \notin p^{\mathcal{I}}$

Für eine Formel gilt nun:
eine Interpretation $\mathcal{I}$ und eine Zuweisung $\mathcal{Z}$ **erfüllen** eine Formel F, geschrieben „$\mathcal{I}, \mathcal{Z} \models F$“, wenn die Rekursion mit Atomen, Operationen und Quantoren zu Wahr auflöst.

3.3 Semantische Grundbegriffe

Wir wollen in der Prädikatenlogik wenn möglich nur mit Sätzen arbeiten, d.h. mit geschlossenen Formeln ohne ungebundene Variablen. TIL 14

Modelltheorie Wir unterscheiden grob zwischen der Prädikatenlogik mit und ohne offenen Formeln. Bei der Prädikatenlogik mit Sätzen können wir auf Zuweisungen verzichten. Formeln sind Behauptungen, die wahr oder falsch sein können. Modelle sind mögliche Welten (prädikatenlogische Interpretationen und ggf. Zuweisungen), in denen manche Behauptungen gelten und andere nicht. (**Intuition**)

Typen von Formeln Siehe hierzu auch die Graphen „Modelle $\models$ Formeln“ in TIL 14

- allgemeingültig (tautologisch): Eine Formel, die in allen Modellen wahr ist
- widersprüchlich (inkonsistent): Eine Formel, die in keinem Modell wahr ist
- erfüllbar: Eine Formel, die in einem Modell wahr ist
- widerlegbar: Eine Formel, die in einem Modell falsch ist

Logisches Schließen Bei der Analyse von Modellen für Formeln und andersherum können in Wechselwirkung Konsequenzen hergestellt werden.

a) Wenn $\mathcal{I}$ die Formel F erfüllt, also $\mathcal{I} \models F$, dann ist $\mathcal{I}$ ein Modell für F.

b) $\mathcal{I}$ kann mehrere Formeln erfüllen, d.h. sie kann Modell für eine Formelmenge $\mathcal{T}$ sein, wenn $\mathcal{I}$ alle Formen in $\mathcal{T}$ erfüllt.

c) Eine Formel F ist nun eine **logische Konsequenz** aus einer Formel bzw. Formelmenge G, d.h. $G \models F$, wenn jedes Modell $\mathcal{I}$ von G auch ein Modell von F ist, d.h. $\mathcal{I} \models G \implies \mathcal{I} \models F$.
Sonderfall: Ist F eine Tautologie, dann schreiben wir nur $\models F$.

Beispiel 1: Gegeben sind vier Modelle $\mathcal{I}_i$ und vier Formeln F_j. $\mathcal{I}_2$ und $\mathcal{I}_3$ sind alle erfüllenden Modelle für F_3. $\mathcal{I}_2$ und $\mathcal{I}_3$ sind aber u.a. auch Modelle für F_2. Das bedeutet, wenn F_3 erfüllt ist, ist auch immer F_2 erfüllt. Es gilt $F_3 \models F_2$.

Beispiel 2: Im Beispiel der Logelei „Wir sind alle vom gleichen Typ“ haben wir fünf Formeln gegeben. Drei davon ergeben sich aus den gegebenen Aussagen („gegebene Theorie“) und die anderen beiden sind Allquantor-Behauptungen für „alle sagen die Wahrheit“ bzw. „alle lügen.“. Wir können anhand der Modelle „LL“, „WL“ und „WW“ und der Theorie Konsequenzen erstellen und somit über das Modell „WW“ die Behauptung „$\forall x.W(x)$“ als logische Konsequenz für unsere Theorie identifizieren.

d) Zwei Formelmengen F und G können auch semantisch äquivalent sein, d.h. $F \equiv G$, wenn sie genau die gleichen Modelle haben ($\mathcal{I} \models F$ gdw. $\mathcal{I} \models G$ für alle Modelle $\mathcal{I}$).

Semantische Äquivalenz Eine Äquivalenzrelation $\equiv$ ist reflexiv, symmetrisch und transitiv. Alle Tautologien sind semantisch äquivalent. Alle unerfüllbaren Formeln sind semantisch äquivalent.
Äquivalenz $F \equiv G$ gdw. $F \models G$ und $G \models F$.

Problem logischen Schließens in der Prädikatenlogik Die zwei Fragen „Model checking“ (Überprüfung eines Modells auf Erfüllung einer Formel) und „Logische Folgerung (Entailment)“ (Überprüfung ob zwei Formeln oder Formelmengen eine logische Konsequenz sind) sind in der Prädikatenlogik schwerer zu lösen als in der Aussagenlogik.

Monotonie und Tautologie Aus der Definition von $\models$ folgt die Monotonie: je mehr Sätze in einer logischen Theorie gegeben sind, desto weniger Modelle können die gesamte Theorie erfüllen und desto mehr Schlussfolgerungen kann man aus der logischen Theorie ziehen. D.h. mehr Annahmen führen zu mehr Schlussfolgerungen. Extremfälle sind hierbei Tautologien (sind in jedem Modell wahr und daher logische Konsequenz jeder Theorie) und unerfüllbare Formeln (sind in keinem Modell wahr und haben daher alle anderen Sätze als Konsequenz).

Gleichheit Es gibt ein spezielles Gleichheitsprädikat $\approx$. In Interpretationen $\mathcal{I}$ gilt $\approx^{\mathcal{I}} = \{\langle\delta, \delta\rangle | \delta \in \Delta^{\mathcal{I}}\}$. Dies kann z.B. zum Erzwingen von gleicher Interpretation von Konstanten verwendet werden. Auch gibt es $\not\approx$, Definition $\forall x, y.(x \not\approx y \leftrightarrow \neg x \approx y)$. Man kann aber mit Hilfe anderer Definitionen der Prädikatenlogik sowohl Gleichheit als auch Ungleichheit einsparen. TIL 14 TIL 15

3.4 Prädikatenlogik als Universalsprache

Die Entwicklung der Logik hat ein zentrales Motiv: Logik als eine universelle, präzise Sprache. Die Entwicklung begann bei Aristoteles als Grundlage der philosophischen Argumentation, ging in Leibniz Sinne in Richtung „rechnen“ und wurde von Hilbert und Russell schließlich zusammen mit der Mathematik formalisiert. Wenn nun die Mathematik in logischen Formeln formuliert wird, wird logisches Schließen zur Kernaufgabe der Mathematik. Eine zentrale Frage des Schließens ist hierbei die Überprüfung auf Erfüllbarkeit einer Formel bzw. einer Formelmenge. TIL 15

Strukturelle Induktion Diese Induktion kann man über jede induktiv definierte syntaktische Struktur durchführen (z.B. Formeln, Terme, Programme,...).

- In der „klassischen Induktion“ wird eine Eigenschaft E untersucht, wobei (1) „0 hat E“ geprüft und darauf aufbauend (2) für alle $n > 0$ im Falle von „$n - 1$ hat E“ geprüft wird.
- In der **strukturellen Induktion auf Formeln** prüfen wir nun ob (1) alle atomaren Formeln E haben und (2) alle nicht-atomaren Formeln F ebenfalls E haben, wenn alle ihre echten Teilformeln E haben.

Im Beispiel „Induktion auf der Insel der Wahrheitssager und Lügner. Ein Einwohner verkündet: 'Was ich jetzt sage, das habe ich schon einmal gesagt.' Welchen Typ hat er?“ muss der Einwohner ein Lügner sein, da er mindestens beim ersten Mal lügt.

3.5 Unentscheidbarkeit des logischen Schließens

Erinnerung: F ist logische Konsequenz von G ($F \models G$), wenn alle Modelle von F auch Modelle von G sind. (1) Es ist nicht offensichtlich, wie man das überprüfen sollte, denn es gibt unendliche viele Modelle. (2) Ebenso schwer erscheinen die gleichwertigen Probleme der Erfüllbarkeit und Allgemeingültigkeit.

Intuition: prädikatenlogisches Schließen ist unentscheidbar. Beweis durch Reduktion eines bekannten unentscheidbaren Problems, z.B. Halteproblem, PCP, Äquivalenz kontextfreier Sprachen u.a.

Der Beweis in der Vorlesung zeigt die Reduktion vom CFG-Schnittproblem. Hierfür werden Wörter ω aus der Modellmenge (Modellstruktur) $\mathcal{I}$ als Ketten von binären Relationen kodiert und untersucht, ob das Wort ω in der Schnittmenge zweier kontextfreier Grammatiken G_1 und G_2 vorkommt.
Beispiel: wir haben auf der Insel z.B. das Modell mit Kombination „LLWWW“ (drei sagen die Wahrheit, zwei lügen), und wir wollen wissen ob $F \models G$. Wir kodieren die erfüllenden Modelle der Formeln F und G wie o.g. und erhalten G_1 und G_2. Nach Kodierung müssten also in beiden Grammatiken die Übergänge $\langle L_1, L_2\rangle, \langle L_2, W_1\rangle, \langle W_1, W_2\rangle, \langle W_2, W_3\rangle$ vorkommen. Ist dies der Fall, dann erfüllt dieses Modell beide Formeln. (*Vergleich und Notation nicht nach VL!*)
Zusammenfassend lassen sich demnach logische Konsequenzen auf diese Probleme reduzieren und da CFG unentscheidbar gilt auch: Logisches Schließen (Erfüllbarkeit, Allgemeingültigkeit, logische Konsequenz) in der Prädikatenlogik ist unentscheidbar. TIL 15

3.6 Gödel

Gödelscher Vollständigkeitssatz und Unvollständigkeitssätze. TIL 15

Gödelscher Vollständigkeitssatz „Es gibt ein konsistentes Verfahren, das alle Konsequenzen einer prädikatenlogischen Theorie effektiv beweisen kann." (1) Alle wahren Sätze können endlich bewiesen werden. (2) Prädikatenlogisches Schließen ist semi-entscheidbar.

1. Gödelscher Unvollständigkeitssatz „Es gibt kein konsistentes Verfahren, das alle Konsequenzen der elementaren Arithmetik effektiv beweisen kann." (1) Für jedes Verfahren gibt es Sätze über elementare arithmetische Zusammenhänge, die weder bewiesen noch widerlegt werden können. (2) Die Wahrheit elementarer arithmetischer Zusammenhänge ist nicht semi-entscheidbar.

3.7 Syntaktische Umformungen

Äquivalenzen mit Quantoren Es gelten die folgenen Beziehungen: TIL 16

- Negation von Quantoren: $\neg\exists.F \equiv \forall x.\neg F$ und $\neg\forall x.F \equiv \exists x.\neg F$
- Kommutativität: $\exists x.\exists y.F \equiv \exists y.\exists x.F$, selbiges für $\forall$
- Distributivität: $\exists x.(F \vee G) \equiv (\exists x.F \vee \exists x.G)$, selbiges für $\forall/\wedge$

Wichtig: andere Kombinationen funktionieren **nicht** ohne dass die Semantik verändert wird.

Negationsnormalform (NNF) Enthält nur Quantoren und die Junktoren $\wedge$, $\vee$ und $\neg$.
Der Negator $\neg$ befindet sich nur noch direkt vor Atomen (Literalen).
Zum Umformeln beginnen wir zuerst mit der Ersetzung von $\rightarrow$ und $\leftrightarrow$:
$(F \rightarrow G) \equiv (\neg F \vee G)$ und $(F \leftrightarrow G) \equiv (\neg F \vee G) \wedge (\neg G \vee F)$.
Folgend wird NNF(F) rekursiv umgeformelt. Hierbei können z.B. Quantoren, die in ihrem Scope keine freie Variable binden, entfernt werden.

Bereinigte Formel, Variablenumbenennung Gebundene und freie Variablen in Formeln können umbenannt werden, so lange die neue Bezeichnung nicht bereits in der Formel vorkommt. Eine Formel ist **bereinigt**, wenn in ihr (1) keine Variable sowohl ungebunden als auch gebunden vorkommt und (2) keine Variable von mehr als einem Quantor gebunden wird. Beispiel:
Die Formel $\forall y.p(x,y) \rightarrow \exists x.(r(y,x) \wedge \forall y.q(x,y))$ wird zu $\forall y.p(x,y) \rightarrow \exists z.(r(y,z) \wedge \forall v.q(z,v))$.

Pränexform In der Pränexform stehen alle Quantoren am Anfang einer Formel, d.h. $Q_1x_1.Q_2x_2.\ldots Q_nx_n.F$, wobei Q_nx_n Quantor mit Variable. Die Umformung einer Formel in Pränexform geschieht nach NNF und Bereinigung, da wir dann ohne Komplikationen alle Quantoren aus der Formel herausziehen können (da jede Variable nur an maximal einem Quantor gebunden ist). TIL 17

Skolemisierung Die Skolemisierung baut auf der Pränexform auf. Nach erfolgreicher Umformung sind alle Existenzquantoren eliminiert und das Vorkommen der entsprechenden Variable durch einen Funktionsterm (Skolemterm) ersetzt.
Sei $\forall x_1 \ldots \forall x_n.\exists y.F$ eine Formel in Pränexform. Dann erstellen wir die neue Formel $\forall x_1 \ldots \forall x_n.F'$ mit $F' = F\{y \mapsto f(x_1,\ldots,x_n)\}$. Die Variable y wird also durch den Skolemterm f, eine n-stellige Skolemfunktion mit bisher unverwendetem Bezeichner, ersetzt. Die Parameter der Funktion sind die Variablen der Allquantoren vor dem eliminierten Existenzquantor. Mehrere Existenzquantoren werden von links nach rechts aufgelöst.

Beispiel: $\forall x.\exists y.\forall z.\exists v.p(x,y,z,v) \longrightarrow \forall x.\forall z.\exists v.p(x,f(x),z,v) \longrightarrow \forall x.\forall z.p(x,f(x),z,g(x,z))$

Skolemisierung kann die Semantik einer Formel verändern, jedoch bleibt die Erfüllbarkeit erhalten.

Konjunktive Normalform (KNF) Eine Formel ist in konjunktiver Normalform, wenn sie eine Konjunktion von Diskunktionen von Literalen ist: $(L_{1,1} \vee L_{1,2} \vee \ldots) \wedge \cdots \wedge (L_{n,1}, L_{n,2}, \ldots)$.
Zum Umformeln muss eine Formel (1) bereinigt, (2) in NNF umgeformt, (3) in Pränexform gebracht und (4) skolemisiert werden.
Zum Abschluss wird noch die Ersetzung $F \vee (G \wedge H) \mapsto (F \vee G) \wedge (F \vee H)$ angewandt.

Klauselform Hierfür wird die KNF nochmals vereinfacht.

- Allquantoren werden weggelassen.
- Klauseln werden als Mengen von Literalen geschrieben.
- Konjunktionen von Klauseln werden als Mengen von Mengen von Literalen geschrieben.

3.8 Algorithmen zum logischen Schließen

Substitution In der Substitution werden freie Variablen $x \in V$ durch Terme $t \in T$ ersetzt. Eine Substitution wird durch σ o.ä. definiert, z.B. $\sigma = \{x_1 \mapsto t_1, \dots\}$. Wird diese Substitution dann auf eine Formel A angewandt, d.h. $A\sigma$, nennt man dies **Instanz** von A unter σ. Man kann mehrere Substitutionen hintereinander ausführen. Dann gilt $A(\sigma \circ \theta) = (A\sigma)\theta$.

Unifikation Ein Unifikationsproblem ist eine endliche Menge von Gleichungen der Form $G = \{s_1 \doteq t_1, \dots, s_n \doteq t_n\}$. Eine Substitution σ ist ein Unifikator für G falls $s_i\sigma = t_i\sigma$ für alle $i \in \{1, \dots, n\}$ gilt. TIL 18

Es kann mehrere Substitutionen geben, die diese Anforderung erfüllen. Dann kann durch Vergleich ein **allgemeinster Unifikator** gefunden werden.

Eine Substitution σ ist **allgemeiner** als eine Substitution θ, in Symbolen $\sigma \preceq \theta$, wenn es eine Substitution λ gibt, so dass $\sigma \circ \lambda = \theta$. Der allgemeinste Unifikator für ein Unifikationsproblem G ist ein Unifikator σ für G, so dass $\sigma \circ \theta$ für alle Unifikatoren θ für G. Die englische Bezeichnung des allgemeinsten Unifikators ist **most general unifier (mgu)**.

Ein Unifikationsproblem $G = \{x_1 \doteq t_1, \dots, x_n \doteq t_n\}$ ist in **gelöster Form**, wenn $x_1, \dots, x_n$ paarweise verschiedene Variablen sind, die nicht in den Termen $t_1, \dots, t_n$ vorkommen. In diesem Fall definieren wir eine Substitution $\sigma_G := \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$. Dann ist σ_G ein allgemeinster Unifikator für G.

Algorithmus:

- Löschen (überflüssige $\doteq$ löschen, z.B. $\{f(x) \doteq f(x)\}$)
- Zerlegung (Parameter gleicher Funktionen auflösen, z.B. $\{g(a, f(x)) \doteq g(b, f(x))\}$ wird $\{a \doteq b, f(x) \doteq f(x)\}$)
- Orientierung (Variablen auf die linke Seite)
- Eliminierung (gegebene Variablen durch Wert ersetzen, z.B. $\{x \doteq f(a), g(x) \doteq g(y)\}$ wird $\{x \doteq f(a), g(f(a)) \doteq g(y)\}$).

Resolution Mit dem Resolutionsalgorithmus versuchen wir aus einer gegebenen Klauselmenge (d.h. eine Formel in Klauselform) eine leere Klausel zu erzeugen (abzuleiten). Diese leere Klausel wäre eine unerfüllbare Behauptung, d.h. sobald wir eine solche leere Klausel finden haben wir die Unerfüllbarkeit der Formel bewiesen. Die Erfüllbarkeit ist hierbei eine „zentrale Frage des Schließens“.

Algorithmus: gegeben eine Klauselmenge, welche nummeriert angeordnet sind. Nachfolgend nehmen wir immer zwei Klauseln, welche in Kombination wahre und falsche Aussagen resolvieren. Hierbei müssen die Variablen der neu erzeugten Klauseln umbenannt werden, es entstehen **Varianten**. TIL 15 TIL 16 TIL 19

Beispiel: gegeben sind (1) und (2), neu erzeugt wird (3)
(1) $\{W(x_1), L(x_2)\}$ (2) $\{\neg W(a)\}$ (3) $\{L(x_2')\}$ (1) + (2), $\{x_1 \mapsto a\}$

Herbrand Herbranduniversum, Herbrandinterpretationen und Herbrandmodelle. TIL 19
Das Herbranduniversum ist eine Erzeugung einer „Semantik aus Syntax“, einer Konstruktion von Modellen direkt aus Formeln.

Sei a eine beliebige Konstante. Das Herbranduniversum Δ_F für eine Formel F ist die Menge aller variablenfreien Terme, die man mit Konstanten und Funktionssymbolen in F und der zusätzlichen Konstante a bilden kann:

- $a \in \Delta_F$
- $c \in \Delta_F$ für jede Konstante aus F
- $f(t_1, \dots, t_n) \in \Delta_F$ für jedes n-stellige Funktionssymbol aus F und alle Terme $t_1, \dots, t_n \in \Delta_F$

Anmerkung: Das Herbrand-Universum ist immer abzählbar, manchmal endlich und niemals leer.
Beispiel: Für die Formel $F = p(f(x), y, g(z))$ ergibt sich $\Delta_F = \{a, f(a), g(a), f(f(a)), f(g(a)), \dots\}$.

Eine Herbrandinterpretation für eine Formel F ist eine Interpretation $\mathcal{I}$ für die gilt:

- $\Delta^{\mathcal{I}} = \Delta_F$ ist das Herbrand-Universum von F
- Für jeden Term $t \in \Delta_F$ gilt $t^{\mathcal{I}} = t$

$\mathcal{I}$ ist ein Herbrandmodell für F wenn zudem gilt $\mathcal{I} \models F$.

4 Übungen

Visuelle Hilfen, die in einigen Lösungen verwendet werden, versuche ich schriftlich zu beschreiben. Hier hilft es ggf., die gegebenen Lösungen nochmals selber aufzuschreiben um die logischen Schritte besser nachvollziehen zu können.

Übungsblatt 1 (Berechenbarkeitstheorie)

Aufgabe 1

$|\mathbb{N}|$ bezeichnet die Kardinalität (Mächtigkeit) der Menge $\mathbb{N}$.

a) $|\mathbb{N}| = |\mathbb{N} \times \mathbb{N}|$: wir vergleichen hier die Kardinalität zweier Mengen. Um die Gültigkeit der Gleichung zu zeigen, betrachten wir also quasi $|A| = |B|$. Wir vergleichen also die Menge $\{0, 1, 2, 3, \dots\}$ mit der Menge $\{(0,0), (1,0), (0,1), (2,0), \dots\}$ und stellen fest, dass wir $1 \mapsto (0,0), 2 \mapsto (1,0), \dots$ „mappen" können.
Als Hilfe kann man eine Matrix verwenden, die in x- bzw. y-Richtung x bzw. y im (x,y)-Tupel hochzählt. Die Elemente aus $|\mathbb{N}|$ werden nun diagonal darübergelegt. Zu zeigen ist, dass zwischen den Mengen $|A|$ und $|B|$ eine Bijektive Abbildung existiert.

- Injektiv: die Aufzählung ist ohne Wiederholung, d.h. in der Matrix steht an verschiedenen Stellen unterschiedliche Tupel
- Surjektiv: jedes Tupel der Menge existiert an einer Position in der Matrix.

b) $|\mathbb{N}| = |\mathbb{Q}|$: wie a), nur dass in $\mathbb{Q}$ negative Zahlen ebenfalls mit in die Matrix einbezogen werden. D.h. x besteht nun aus $\{1, -1, 2, -2, \dots\}$. In dieser Menge erscheinen nun Duplikate, z.B. $\frac{1}{1}$ und $\frac{2}{2}$, welche wir einfach löschen. Die Abbildung $f(n)$ trifft also das n-te Element der Aufzählung nach Streichen von sich wiederholenden Elementen. Damit ist f sowohl surjektiv als auch injektiv, da der Bruch $\frac{i}{j}$ spätestens an der Position (i, j) vorkommt.

c) $|\mathbb{N}| = |\mathbb{R}|$: wir zeigen hier nun Überabzählbarkeit. Dazu verwenden wir $|(0,1]| > |\mathbb{N}|$ und nehmen das Gegenteil an, d.h. $|(0,1]| = |\mathbb{N}|$. Dann gäbe es eine Aufzählung $r_0, r_1, r_2, \dots$ von $(0,1]$.
Sei $r_i = 0, b_{i1}b_{i2}b_{i3}$ (wobei $b_{ij} \in \{0, \dots, 9\}$) die Dezimaldarstellung von r_i, die nicht schließlich 0 ist. Nun nehmen wir eine Tabelle und verwenden die Diagonalisierung.

r_0	$0, b_{11}b_{12}b_{13}\dots$
r_1	$0, b_{21}b_{22}b_{23}\dots$
r_2	$0, b_{31}b_{32}b_{33}\dots$
$\dots$	$\dots$

Betrachten wir nun die Zahl $\overline{r_i} = 0, \overline{b_{11}b_{22}b_{33}}$, wobei $\overline{b_{ij}} = \begin{cases} 1 \text{ wenn } b_{ij} \neq 1 \\ 2 \text{ sonst} \end{cases}$

Dann ist $\overline{r} \neq r_i$ für alle i, da sie sich in der i-ten Stelle unterscheiden.

Aufgabe 2

Sei M eine Menge. Zeigen Sie, dass es keine surjektive Funktion $f : M \to \mathcal{P}(M)$ gibt. Folgern Sie daraus, dass stets $|M| < |\mathcal{P}(M)|$ gilt.
Lösung: Wir verwenden hierzu eine Matrix und zeigen anhand der Menge $D = \{x \in M | x \notin f(x)\}$ einen Widerspruch. Zuerst die beliebig gefüllte Matrix:

M	m_0	m_1	m_2	m_3
$f(m_0)$	x			x
$f(m_1)$			x	
$f(m_2)$		x		
$f(m_3)$	x			x
D		x	x	

Die x an Stellen $(m_i, f(m_i))$ bedeuten, dass das jeweilige m_i in der abgebildeten Menge von $f(m_i)$ enthalten ist. Die konstruierte Menge D besteht nun zum Schluss aus genau den gegenteiligen Elementen

zu jedem m_i. Angenommen, $f : M \to \mathcal{P}(M)$ sei surjektiv. Dann gäbe es ein $y \in M$ mit $f(y) = D$. Es gilt dann $y \in D \Rightarrow y \notin f(y) = D$ und $y \notin D \Rightarrow y \in f(y) = D$. Widerspruch, es kann f nicht geben.

Aufgabe 3

Konstruieren Sie eine Turing-Maschine $\mathcal{A}_{mul}$, welche die Multiplikation zweier natürlicher Zahlen implementiert. Dabei sollen sowohl die Eingaben als auch die Ausgabe unär kodiert sein.
Lösung: Die Maschine erhält eine Eingabe, z.B. $000\#0000$ und soll demnach $3 * 4 = 12$ berechnen. Dafür kopiert sie pro Symbol vor dem $\#$ den unären Wert hinter dem Symbol auf das Band dahinter und löscht dann die ursprüngliche Eingabe.
$000\#0000\sqcup\sqcup \mapsto^* \sqcup\sqcup 0\#0000\sqcup 000000000\sqcup \mapsto^* \sqcup 000000000000$

$q_0, 0, q_1, \sqcup, R$	Auftrag: kopiere zweiten Faktor an das Ende
$q_1, 0, q_1, 0, R$	überspringe ersten Faktor
$q_1, \#q_2, \#, R$	Ende des ersten Faktors erreicht
$q_2, 0, q_3, \hat{0}, R$	Markiere das zu kopierende Zeichen
$q_3, 0, q_3, 0, R$	0 überspringen
$q_3, \sqcup, q_4, \sqcup, R$	Trenner hinter zweiten Faktor
$q_4, 0, q_4, 0, R$	Ans Ende des Zwischenergebnisses
$q_4, \sqcup, q_5, 0, L$	Ende erreicht, Null schreiben und zurück
$q_5, 0/\sqcup, q_5, 0/\sqcup, L$	Zurück zum nächsten zu kopierenden Zeichen
$q_5, \hat{0}, q_2, 0, R$	Nächstes Zeichen kopieren
$q_2, \sqcup, q_6, \sqcup, L$	Zweiten Faktor fertig kopiert, zurück zum ersten
$q_6, 0, q_6, 0, L$	Zweiten Faktor überspringen
$q_6, \sqcup, q_0, \sqcup, R$	Anfang erreicht, bearbeite nächste Ziffer
$q_0, \#, q_7, \sqcup, R$	Erster Faktor aufgebraucht
$q_7, 0, q_7, \sqcup, R$	Löschen des zweiten Faktors
$q_7, \sqcup, q_f, \sqcup, N$	Alles gelöscht, Berechnung abgeschlossen

Daraus ergibt sich die TM: $\mathcal{A}_{mul} = (\{q_0, \ldots, q_f\}, \{0, \#\}, \Sigma \cup \{\hat{0}, \sqcup\}, \delta, q_0, \{q_f\})$

Aufgabe 4

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 auf der leeren Eingabe halten.

Lösung: Wir konstruieren zwei TM, erstens $\mathcal{M}_\emptyset$, welche die leere Sprache erkennt, und $\mathcal{M}$. Diese beiden werden dann in den gegebenen Algorithmus eingegeben und auf Äquivalenz getestet. Gegeben ist TM $\mathcal{K}$ die entscheidet, ob zwei TM $\mathcal{M}_1$ und $\mathcal{M}_2$ dieselbe Sprache akzeptieren. D.h.
$\boxed{\mathcal{K}(enc(\mathcal{M}_1), enc(\mathcal{M}_2)) \text{ akzeptiert} \Leftrightarrow \mathcal{L}(\mathcal{M}_1) = \mathcal{L}(\mathcal{M}_2)}$.

Gesucht ist eine TM $\mathcal{K}'$, die entscheidet, ob eine TM $\mathcal{M}$ auf ϵ hält, d.h. $\boxed{\mathcal{K}'(enc(\mathcal{M})) \text{ akzeptiert} \Leftrightarrow M \text{ hält auf } \epsilon}$.

Idee: finde für $\mathcal{M}$ zwei TM $\mathcal{M}_1$ und $\mathcal{M}_2$, so dass $\boxed{\mathcal{M} \text{ hält auf } \epsilon \Leftrightarrow \mathcal{L}(\mathcal{M}_1) = \mathcal{L}(\mathcal{M}_2)}$.
Definiere $\mathcal{M}_1$ als die TM, die alle Eingaben akzeptiert.
Definiere $\mathcal{M}_2$ als TM, die bei Eingabe ϵ die TM $\mathcal{M}$ auf ϵ simuliert und anschließend akzeptiert, und ansonsten akzeptiert.
Simuliere $\mathcal{K}(enc(\mathcal{M}_1), enc(\mathcal{M}_2))$ und gib das Ergebnis zurück.

Ablauf:

- $\mathcal{K}'$ hält auf jeder Eingabe
- Hält $\mathcal{M}$ auf ϵ, dann ist $\boxed{\mathcal{L}(\mathcal{M}_2) = \Sigma^* = \mathcal{L}(\mathcal{M}_1)}$ und $\mathcal{K}'(enc(\mathcal{M}))$ akzeptiert
- Hält $\mathcal{M}$ auf ϵ nicht, dann ist $\boxed{\mathcal{L}(\mathcal{M}_2) = \Sigma^* \setminus \{\epsilon\} \neq \mathcal{L}(\mathcal{M}_1)}$ und $\mathcal{K}'(enc(\mathcal{M}))$ verwirft

Gezeigt: Äquivalenz von TM ist nicht entscheidbar.

Übung 2 (Berechenbarkeitstheorie)

Aufgabe 1

Zeigen Sie, dass die folgenden Funktionen $f : \mathbb{N}^2 \to \mathbb{N}$ LOOP-berechenbar sind:

a) $f(x, y) := max(x - y, 0)$

```
x := x1 + 0; y := x2 + 0
LOOP y DO  x := x - 1  END
x0 := x + 0
```

b) $f(x, y) := x \cdot y$

```
x := x1 + 0; y := x2 + 0
LOOP y DO                   # Fuer jedes y
    LOOP x DO               # addiere jedes x
        res := res + 1      # Ist bei Beginn 0
    END
END
x0 := res + 0
```

c) $f(x, y) := max(x, y)$

```
x3 := x1 + 0
LOOP x2 DO  x3 := x3 - 1  END
x0 := x2 + 0
LOOP x3 DO  x0 := x1  END
```

Fallunterscheidung $x1 \leq x2$ (dann ist $x3$ gleich 0 nach der Schleife und $x0 = x2$) und $x1 > x2$ (dann $x3 \neq 0$ und $x0 = x1$).

d) $f(x, y) := ggT(x, y)$, wobei $ggT(x, y)$ den größten gemeinsamen Teiler von x und y bezeichnet.

```
x3 := max(x1, x2); x4 := min(x1, x2)
LOOP x3 DO
    x5 := max(x3-x4, x4)
    x6 := min(x3-x4, x4)
    x3 := x5
    x4 := x6
END
x0 := x3
```

Wobei $min(x, y)$ einfach $max(x, y)$ mit umgedrehter Ausgabe ist. LOOP statt WHILE, da die Durchläufe beschränkt werden müssen. $x3$ wird verwendet, da $ggT(n, 1)$ mindestens einmal laufen soll.

Aufgabe 2

Mit $kgV(x_1, x_2)$ bezeichnen wir das kleinste gemeinsame Vielfache zweier natürlicher Zahlen x_1 und x_2. Geben Sie ein WHILE-Programm an, das die Funktion $f : \mathbb{N}^2 \to \mathbb{N}, (x_1, x_2) \mapsto kgV(x_1, x_2)$ berechnet und erklären Sie seine Arbeitsweise.
Lösung: Wir nehmen hier die Funktion $kgV(x, y) = (x \cdot y)/ggT(x, y)$.

```
kgV(x1, x2) =
    IF x1 != 0 THEN
        IF x2 != 0 THEN
            x3 := x1 * x2
            x4 := ggT(x1, x2)
            x0 := div(x3, x4)
        END
    END

div(x1, x2) =
    x3 := x1 + 0
    x4 := x3 + 1          # Addiere 1, um Sonderfall x2==0 abzudecken
    x4 := x4 - x2         # Hier wuerde x2 nie dekrementiert, d.h. x4 nie 0
    WHILE x4 != 0 DO
        x0 := x0 + 1
        x3 := x3 - x2
        x4 := x3 + 1      # Wenn x2==0, dann nie x4=0 -> "Fehlercode"
        x4 := x4 - x2     #   fuer Division durch 0
    END
```

Aufgabe 3

Es sei Σ ein fest gewähltes Alphabet mit mindestens zwei Elementen. Wir betrachten eine Programmiersprache L über Σ, die in der Lage ist, Turing-Maschinen zu simulieren. Für ein Wort $w \in \Sigma^*$ ist die Kolmogorov-Komplexität $K_L(w)$ die Länge des kürzesten Programms in L, welches bei leerer Eingabe das Wort w als Ausgabe produziert. Zeigen Sie folgende Aussagen:

a) Es gibt für jede natürliche Zahl $n \in \mathbb{N}$ ein Wort $w \in \Sigma^*$ der Länge $|w| = n$, so dass $K_L(w) \geq n$.
Lösung: Gegenfrage: wie viele w mit $|w| = n$ gibt es, so dass $K_L(w) < n$?
Wir betrachten den Spezialfall $|\Sigma| = 2$, d.h. Σ hat die minimal geforderten zwei Elemente. Dann gibt es höchstens $\sum_{i=0}^{n-1} 2^i = 2^n - 1$ Wörter w mit $K_L(w) < n$. Es gibt aber 2^n Wörter w mit $|w| = n$, d.h. mindestens eines dieser w erfüllt $K_L(w) \geq n$.

b) Es gibt eine Konstante $c \in \mathbb{N}$, so dass gilt: Ist w das Ergebnis der Berechnung einer Turing-Maschine M mit Eingabe x, dann $K_L(w) \leq |\langle M, x\rangle| + c$, wobei $\langle M, x\rangle$ eine (effektive) Kodierung der Maschine M und der Eingabe x als ein Wort über Σ ist.
Lösung: Idee: simuliere M auf Eingabe x als Programm in L. Betrachte folgendes Programm in L:

- Simuliere $enc(M)$ auf $enc(x)$
- Gib das Ergebnis aus

Ist dann c die Länge des Programms ohne $enc(M)+enc(x)$, dann ist $K_L(w) \leq |enc(M)\#\#enc(x)|+c$.

c) Die Abbildung $w \mapsto K_L(w)$ ist nicht berechenbar.
Lösung: Annahme, $K_L(w)$ wäre berechenbar für alle $w \in \Sigma^*$. Dann konstruieren wir eine TM M, die für Eingaben $n \in \mathbb{N}$ Wörter w mit $|w| = n$ und $K_L(w) \geq n$ ausgibt. Wir definieren M wie folgt:
$M =$ bei Eingabe n

- zähle Wörter $w \in \Sigma^*$, $|w| = n$ auf
- falls $K_L(w) \geq n$, gib w aus

Wegen a) gibt M für jede Eingabe n ein Wort w_n zurück.
Dann ist $[n \leq K_L(w_n) \leq |enc(M)\#\#enc(n)| + c]$.
$\rightarrow n \leq K_L(w_n) = c' + |enc(n)| = c' + log_2 n$ (wobei $c' = c + |enc(M)\#\#|$). Diese Ungleichung gilt jedoch nicht für alle n. n wächst schneller als $log_2 n + c'$, stetig vs. logarithmisch.

Damit ist insbesondere gezeigt, dass es niemals einen Compiler geben kann, der ein gegebenes Programm in ein kleinstmögliches übersetzt.

Übung 3 (Berechenbarkeitstheorie)

Aufgabe 1

Zeigen Sie, dass es keine Many-One-Reduktion vom Halteproblem P_{halt} von Turing-Maschinen auf des Leerheitsproblem $P_{leer} := \{enc(M) \mid \mathcal{L}(M) = \emptyset\}$ von Turing-Maschinen gibt.

Lösung: Aussage ist demnach $P_{halt} \not\leq_m P_{leer}$. Idee zum Beweis ist ein Widerspruch.

- Wenn $A \leq_m B$, und B co-semi-entscheidbar (d.h. das Komplement des Problems ist semi-entscheidbar), dann muss auch A co-semi-entscheidbar sein
- P_{halt} ist unentscheidbar, jedoch semi-entscheidbar, demnach nicht co-semi-entscheidbar
- P_{leer} ist unentscheidbar, aber co-semi-entscheidbar

Beweis mit Annahme $P_{halt} \leq P_{leer}$. Da P_{leer} co-semi-entscheidbar ist, ist auch P_{halt} co-semi-entscheidbar. Widerspruch! Es bleibt zu zeigen, dass P_{leer} co-semi-entscheidbar ist. Dazu geben wir eine TM M an, die $\overline{P_{leer}}$ erkennt. *(Anmerkung: Es handelt sich hier um das Komplement des Problems, nicht um das Komplement einer Menge)*

M erhält Eingabe $enc(N)$, ist NTM (andere Eingaben verwerfen)
Sei $w_0, w_1, w_2, \ldots$ eine Aufzählung von Σ^*. Definiere M wie folgt:

- Für $i = 0, 1, 2, \ldots$
- Simuliere N auf $w_0, w_1, \ldots, w_i$ für i Schritte
- akzeptiere, falls N eines dieser Wörter akzeptiert

Durch das Probieren aller w_i muss irgendwann ein Wort akzeptiert werden. Entsprechend ist $\exists w \in \overline{\mathcal{L}(N)}$ bewiesen und somit das Komplement entschieden.
Simuliert M die TM N ohne die Begrenzung der Ausführungsschritte, kann es passieren, dass N in eine Endlosschleife gerät bevor M ein gültiges Wort testen konnte.

Aufgabe 2

Es sei $T := \{\ enc(M) \mid M\ ist\ eine\ TM,\ welche\ w^R\ akzeptiert,\ falls\ sie\ w\ akzeptiert\}$, wobei w^R das zu w umgekehrte Wort ist. Zeigen Sie, dass T nicht entscheidbar ist.

Lösung: Wir verwenden hier den Satz von Rice und das Wissen, dass P_{leer} unentscheidbar ist.
Sei E eine nicht-triviale Eigenschaft (d.h. eine Eigenschaft die sowohl zutreffen kann als auch nicht zutreffen kann) semi-entscheidbarer Sprachen (d.h. nicht Turingmaschinen!). Dann ist $\{enc(M) \mid \mathcal{L}(M)\ erfllt\ E\}$ unentscheidbar.
Für P_{leer} bedeutet dies: L erfüllt $E \iff L = \emptyset$
Für T: L erfüllt $E \iff (\forall w : w \in L \Leftrightarrow w^R \in L)$
E ist nicht-trivial: $L = \emptyset, L = \{a\}$ erfüllen E; $L = \{ab\}$ erfüllt E jedoch nicht

Aufgabe 3

Es sei $L := \{enc(G)\#\#enc(x) \mid G\ kontextfreie\ Grammatik\ und\ x\ Teilwort\ eines\ Wortes\ aus\ \mathcal{L}(G)\}$, wobei $enc(G)$ eine Kodierung von G ist. Zeigen Sie, dass $\mathcal{L}$ auf das Komplement des Leerheitsproblems kontextfreier Grammatiken many-one-reduziert werden kann. Hinweis: der Schnitt einer regulären (Typ 3) mit einer kontextfreien Sprache (Typ 2) ist wieder kontextfrei.

Lösung: Hier $w_1 \times w_2 \in \mathcal{L}(G) \longrightarrow \Sigma^* \times \Sigma^* \cap \mathcal{L}(G) \neq \emptyset$.
Also $enc(G)\#\#enc(x) \in \mathcal{L} \iff \Sigma^* \times \Sigma^* \cap \mathcal{L}(G) \neq \emptyset$
Reduktion: $f(enc(G)\#\#enc(x)) = enc(G')$

Aufgabe 4

Zeigen Sie, dass jede semi-entscheidbare Sprache L auf das Halteproblem P_{halt} many-one-reduziert werden kann.

Lösung: Aussage demnach: ist L semi-entscheidbar, dann gilt $L \leq_m P_{halt}$. Es gibt also eine TM M mit $\mathcal{L}(M) = L$ und für die many-one-Reduktion muss es eine berechenbare Funktion f von L nach P_{halt} geben.

Idee: $w \in \mathcal{L} \Leftrightarrow M\ akzeptiert\ w \longrightarrow M'\ hält\ auf\ w'$.
Ziel: $f(w) = enc(M')\#\#enc(w')$ so dass $w \in L \Leftrightarrow M'$ hält auf w'.
Definiere $M' =$ bei Eingabe x

- Simuliere M auf x
- Falls M akzeptiert, akzeptiere (halte)
- Ansonsten loop (Endlosschleife)

Beweis: Sei L eine semi-entscheidbare Sprache. Sei M TM mit $\mathcal{L}(M) = L$. Definiere für $w \in \Sigma^*$ den Wert $f(w) = enc(M')\#\#enc(w)$, mit M' wie oben. Dann ist f berechenbar. Es gilt $w \in L \Leftrightarrow f(w) \in P_{halt}$ und damit ist $L \leq_m P_{halt}$.

Übung 4 (Berechenbarkeitstheorie)

Aufgabe 1

Besitzen folgende Instanzen P_i des Postschen Korrespondenzproblems Lösungen oder nicht?

a) Ja, einfach zu zeigen.

b) Nein, denn der erste Stein ist der einzige, mit dem begonnen werden kann. Folgend passt nur der dritte Stein und nach diesem ebenfalls immer nur der dritte. Das untere Wort ist demnach immer länger als das obere.

c) Ja, hat Lösung mit 66 Steinen.

Aufgabe 2

Zeigen Sie, dass das Postsche Korrespondenzproblem über einem einelementigen Alphabet entscheidbar ist.
Lösung: Dies lässt sich mit Hilfe eines Algorithmus lösen, welcher die Länge der Wortpaare untersucht. Sei $P = (a^{u_1}, a^{v_1}), \ldots, (a^{u_n}, a^{v_n})$ über $\Sigma = \{a\}$. Wir schreiben nun (u_i, v_i) statt (a^{u_i}, a^{v_i}), betrachten also nur die jeweilige Länge. Fallunterscheidung:

- 1. Fall: Es gibt ein $u_i = v_i$. Dann ist Paar i die Lösung.
- 2. Fall: Alle i sind derart, dass $u_i < v_i$ bzw. $u_i > v_i$, d.h. alle oberen bzw. unteren Wörter sind länger als das jeweils andere. Dann ist P unlösbar.
- 3. Fall: Es gibt i, j mit $u_i > v_i$ und $u_j < v_j$. Eine Lösung hat dann die Form $(\overbrace{i, i, \ldots}^{\text{k-mal}}, \overbrace{j, j, \ldots}^{\text{l-mal}})$ (sofern Lösung existiert). Dann muss gelten $k \cdot u_i + l \cdot u_j = k \cdot v_i + l \cdot v_j$, also $k \cdot (u_i - v_i) = l \cdot (v_j - u_j)$. Wähle $l = (u_i - v_i)$, $k = (v_j - u_j)$. Dann ist $(k \cdot i, l \cdot j)$ tatsächlich eine Lösung.

In jedem Fall ist also entscheidbar, ob es eine Lösung gibt. Damit ist die Aussage gezeigt.

Aufgabe 3

Zeigen Sie, dass folgendes Problem unentscheidbar ist: gegeben eine Turing-Maschine M und ein $k \in \mathbb{N}$, kann die Sprache $L(M)$ durch eine Turing-Maschine mit höchstens k Zuständen erkannt werden? Zeigen Sie dazu, dass für $k = 1$ die Menge $T_k := \{\, enc(M) \mid L(M)$ wird von einer TM mit höchstens k Zuständen erkannt$\}$ nicht entscheidbar ist. Warum zeigt dies die ursprüngliche Behauptung?

Lösung: Wir setzen hier den Satz von Rice über die Unentscheidbarkeit von Eigenschaften von Sprachen an. (*Wichtig! Nicht Eigenschaften von Maschinen!*)
Wir betrachten den Fall T_1. T_1 ist nach Satz von Rice unentscheidbar.
Was ist in diesem Falle die Eigenschaft E? Definition für $L \subset \Sigma^*$.
L erfüllt $E \iff$ es gibt eine TM N mit einem Zustand, so dass $L(N) = L$. Dann ist E Eigenschaft von Sprachen. Bemerkung: ist L nicht semi-entscheidbar (benötigt für Satz von Rice!), dann gibt es keine TM N mit $L = L(M)$. Also erfüllt L die Eigenschaft E nicht.
E ist nicht-trivial: $L = \emptyset$ funktioniert, TM hat keinen Endzustand. $L = \{aa\}$ jedoch funktioniert nicht, da mit nur einem Zustand nicht gezählt werden kann. Also ist nach Satz von Rice die Maschine T_1 unentscheidbar.
Da das Problem bereits für $k = 1$ unentscheidbar ist, ist es auch für beliebige k unentscheidbar.
Der Sonderfall $k = 0$ führt zu $T_0 = \emptyset$. Ist $\emptyset$ entscheidbar? Ja, die TM lehnt einfach immer ab. Da dieser Fall jedoch trivial ist, fällt er nicht unter den Satz von Rice.

Aufgabe 4

Zeigen Sie, dass weder das Äquivalenzproblem $\mathbf{P}_{\text{äquiv}}$ für Turing-Maschinen noch dessen Komplement $\overline{\mathbf{P}_{\text{äquiv}}}$ semi-entscheidbar ist, wobei

- $\mathbf{P}_{\text{äquiv}} := \{enc(M_1)\#\#enc(M_2) | L(M_1) = L(M_2)\}$
- $\overline{\mathbf{P}_{\text{äquiv}}} := \{enc(M_1)\#\#enc(M_2) | L(M_1) \neq L(M_2)\}$

Zeigen Sie dazu, dass $\mathbf{P}_{\text{halt}} \leq_m \mathbf{P}_{\text{äquiv}}$ und $\mathbf{P}_{\text{halt}} \leq_m \overline{\mathbf{P}_{\text{äquiv}}}$ gilt. Weshalb zeigt dies die Aussage?

Lösung: Angenommen $\mathbf{P}_{\text{äquiv}}$ wäre semi-entscheidbar. Dann $\overline{\mathbf{P}_{\text{äquiv}}}$ co-semi-entscheidbar.
Da $\mathbf{P}_{\text{halt}} \leq_m \overline{\mathbf{P}_{\text{äquiv}}}$ muss auch $\mathbf{P}_{\text{halt}}$ co-semi-entscheidbar. Widerspruch! Also ist $\overline{\mathbf{P}_{\text{äquiv}}}$ nicht semi-entscheidbar.
Selbige Heransgehensweise gilt für die entsprechenden Komplemente.

Wir zeigen $\mathbf{P}_{\text{halt}} \leq_m \mathbf{P}_{\text{äquiv}}$. Dafür geben wir eine berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ an, so dass $enc(M)\#\#enc(w) \in \mathbf{P}_{\text{halt}} \Leftrightarrow f(enc(M)\#\#enc(w)) \in \mathbf{P}_{\text{äquiv}}$ für M Turingmaschine und w Eingabe. Dafür müssten wir zwei TM M_1 und M_2 finden, so dass gilt: M hält auf $w \Leftrightarrow L(M_1) = L(M_2)$. Seien also M und w wie oben.

Dafür setzen wir $M_1 =$ bei Eingabe w

- akzeptiere ($L(M_1) = \Sigma^*$)

Und $M_2 =$ bei Eingabe y

- simuliere M auf w (nimmt y, codiert dies als w)
- akzeptiere (bedeutet M hat gehalten. Andernfalls würde M nicht halten)

Bedeutet: $L(M_2)$ ist Σ^*, falls M auf w hält. Sonst $L(M_2) = \emptyset$.
Dann gilt: M hält auf $w \Leftrightarrow L(M_2) = \Sigma^* = L(M_1)$.
Daher ist $f(enc(M)\#\#enc(w)) := enc(M_1)\#\#enc(M_2)$ eine Reduktion von $\mathbf{P}_{\text{halt}}$ auf $\mathbf{P}_{\text{äquiv}}$.
Die Reduktion $\mathbf{P}_{\text{halt}} \leq_m \overline{\mathbf{P}_{\text{äquiv}}}$ verläuft analog.

Übung 5 (Komplexitätstheorie)

Aufgabe 1

Welche der folgenden Aussagen sind wahr? Begründen Sie Ihre Antwort.

a) Falls $P \neq NP$ gilt, dann auch $P \cap NP = \emptyset$.

b) Es gibt Probleme, die NP-hart, aber nicht NP-vollständig sind.

c) Polynomielle Reduzierbarkeit ist nicht transitiv.

d) Ist $L_2 \in P$ und $L_1 \leq_p L_2$, dann ist auch $L_1 \in P$.

e) Ist L_1 eine NP-vollständige Sprache und gilt $L_1 \leq_p L_2$, dann ist auch L_2 NP-vollständig.

f) Ist L_2 eine NP-vollständige Sprache und gilt $L_1 \leq_p L_2$, dann ist auch L_1 NP-vollständig.

Lösung:

a) Richtig. Derzeitiger Kenntnissstand: $P \subseteq NP$, d.h. $P \cap NP = P \neq \emptyset$.

b) Richtig. Jedes NP-Problem ist bspw. in polynomieller Zeit auf das Halteproblem $\mathbf{P}_{\text{halt}}$ reduzierbar, aber $\mathbf{P}_{\text{halt}}$ ist nicht in NP (da unentscheidbar).

c) Falsch. Reduktion ist transitiv, die Komposition von polynomiell-zeitberechenbaren Funktionen ist wieder polynomiell-zeitberechenbar. Formell: $C \leq_p B \leq_p A \Rightarrow C \leq_p A$.

d) Richtig. Ein Entscheidungsverfahren für L_1, welches in polynomieller Zeit läuft, reduziert zuerst die Eingabe w auf eine Instanz $f(w)$ für L_2 und prüft dann, ob $f(w) \in L_2$.

e) Falsch. L_2 muss nur NP-hart sein. Beispiel $\mathbf{P}_{\text{halt}}$.

f) Falsch. Beispiel $L = \emptyset, \emptyset \leq SAT$.

Aufgabe 2

Zeigen Sie, dass das Wortproblem deterministischer endlicher Automaten in L liegt: ist $\mathbf{P}_{\text{DFA}} := \{ enc(A)\#\#enc(w) \mid A \text{ ist ein DFA, der } w \text{ akzeptiert}\}$, dann gilt $\mathbf{P}_{\text{DFA}} \in L$.

Lösung: Die Klasse L ist $LogSpace$, d.h. der Automat hat zusätzlich zur Eingabe logarithmisch viel Platz für seine Berechnung. Die Klasse L ist somit die Klasse von Problemen, die mit einer konstanten Anzahl von Zählern und Zeigern gelöst werden können.

Für die Simulation von A auf w brauchen wir

- einen Zeiger, der auf den aktuellen Zustand zeigt
- einen Zeiger in die Eingabe w
- 2-3 Hilfszähler
- 1-2 Zähler, um Eingabe zu überprüfen

Wichtig: die Anzahl der Zähler/Zeiger hängt nicht von der Länge der Eingabe ab. Die Anzahl der für die Simulation benötigten Zähler und Zeiger liegt demnach in LogSpace.

Aufgabe 3

Es sei $L := \{ a^n \mid n \in \mathbb{N}\ ist\ keine\ Primzahl\}$. Zeigen Sie, dass $L \in NP$ gilt.

Lösung: Demnach ist $L = \{\epsilon, a, aaaa, aaaaaa, \dots\}$. Wir nutzen den Teiler von n als Zertifikat.
Ein nicht-deterministisches Entscheidungsverfahren für L, welches in polynomieller Zeit läuft ist folgendes: $M =$ bei Eingabe a^n:

- rate $p \in \mathbb{N}$ mit $1 < p < n$ (es gibt $\sqrt{n}$ viele p)
- prüfe ob p ein Teiler von n ist
- falls ja, akzeptiere, ansonsten verwerfe

Warum $L(M) = L$? Für jedes $a^n \in L$ gibt es mindestens einen akzeptierenden Lauf von M auf a^n und für $a^n \notin L$ verwirft sie stets. Ist M polynomiell zeitbeschränkt? Ja, denn Test lässt sich in polynomieller Zeit ausführen. Damit ist $L \in NP$. Sogar $L \in P$, wenn einfach alle Zahlen durchprobiert werden. Der Primzahltest ist in P, wird jedoch komplexer bei der Kodierung ($log\ n \to n^2$).

Aufgabe 4

Zeigen Sie: ist $P = NP$, dann gibt es einen Algorithmus, der in polynomieller Zeit für jede erfüllbare aussagenlogische Formel eine erfüllende Belegung findet.

Lösung: Idee ist die binäre Suche mit Teilformeln.
Sei φ eine aussagenlogische Formel mit Variablen $x_1, \dots, x_n$. Angenommen φ ist erfüllbar. Betrachte die Formel $\varphi[x_1 \leftarrow True]$. Ist diese Formel erfüllbar (da $P = NP$ kann hier SAT verwendet werden), setze $\beta(x_1) := True$, ansonsten setze $\beta(x_1) := False$. Berechne dann rekursiv eine erfüllende Belegung β' für $\varphi[x_1 \leftarrow \beta(x_1)]$. Dann ist β erweitert um β' eine erfüllende Belegung für φ.

Was ist die Laufzeit dieses Algorithmus? Da $P = NP$ gibt es ein Polynom $p(n)$, welches die Laufzeit für den Erfüllbarkeitstest nach oben abschätzt. Dann läuft der Algorithmus oben in Zeit $O(n \cdot p(|\varphi|)) = O(|\varphi| \cdot p(|\varphi|))$, also in polynomieller Zeit in der Größe von φ.

Wichtig: Backtracking ist hier nicht notwenig, da SAT alle weiteren Belegungen nach einer Belegung prüft. Klappt auch für 3SAT und CLIQUE.

Übung 6 (Komplexitätstheorie)

Aufgabe 1

Wir betrachten das folgende Problem K: Gegeben eine aussagenlogische Formel φ mit n Variablen. Gibt es eine erfüllbare Belegung von φ, bei der mindestens die Hälfte aller in φ vorkommenden Variablen mit $True$ belegt sind?

a) Formalisieren Sie dieses Problem als Sprache und zeigen Sie, dass $K \in NP$ gilt.

b) Zeigen Sie, dass K ein NP-hartes Problem ist.

Lösung:

a) $K = \{\ enc(\varphi) \mid \varphi$ aussagenlogische Formel, die eine erfüllende Belegung hat, die mindestens die Hälfte der in φ vorkommenden Variablen auf $True$ setzt$\}$.

Was ist NP? $NP = \{\ L \subseteq \Sigma^* \mid L$ akzeptiert von NTM in polynomieller Zeit $\}$.
Es gilt $K \in NP$, da eine entsprechende erfüllende Belegung geraten und in polynomieller Zeit geprüft werden kann.

b)

Aufgabe 2

Im folgenden Solitaire-Spiel haben wir ein Spielbrett der Größe $m \times m$ gegeben. Als Ausgangsposition liegt auf jeder der m^2 Positionen entweder ein blauer Stein, ein roter Stein, oder gar nichts. Das Spiel wird nun so gespielt, dass Steine vom Brett genommen werden bis in jeder Spalte nur noch Steine einer Farbe liegen, und in jeder Zeile mindestens ein Stein liegen bleibt. In diesem Fall ist das Spiel gewonnen. Es ist möglich, dass man ausgehend von einer Ausgangsposition das Spiel nicht gewinnen kann.

a) Formalisieren Sie das Problem, für eine gegebene Ausgangsposition im Solitaire-Spiel zu entscheiden, ob es möglich ist, das Spiel zu gewinnen, als ein Entscheidungsproblem SOLITAIRE.

b) Zeigen Sie, dass $SOLITAIRE \in NP$ gilt.

c) Zeigen Sie, dass SOLITAIRE ein NP-hartes Problem ist, indem Sie zeigen, dass 3SAT in polynomieller Zeit auf SOLITAIRE reduzierbar ist.

Aufgabe 3

Sei Σ ein Alphabet und $A, B \subseteq \Sigma^*$. Wir sagen, dass A auf B in logarithmischen Platz reduzierbar ist, und schreiben $A \leq_l B$, falls es eine Many-One-Reduktion von A nach B gibt, die in logarithmischen Platz berechenbar ist. Zeigen Sie: gilt $A \leq_l B$ und $B \leq_l C$, dann gilt auch $A \leq_l C$.

Für diese Aufgabe ist eine Musterlösung gegeben, sie wurde nicht in der Übung besprochen.

Aufgabe 4

Wir betrachten das Problem $SET - SPLITTING$, welches für eine gegebene endliche Menge S und eine Menge $C = \{C_1, \ldots, C_k\}$ von Teilmengen von S fragt, ob die Elemente von S derart mit den Farben blau oder rot gefärbt werden können, so dass niemals alle Elemente einer Menge C_i die gleiche Farbe bekommen. Zeigen Sie, dass SET-SPLITTING ein NP-vollständiges Problem ist.

5 Repetitorien

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 $\mathbf{P}_{\text{leer}} \leq_m \mathbf{P}_{\text{äquiv}}$.

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) „$\mathbf{P}_{\text{halt}}$ 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 ($\mathbf{P}_{\text{halt}} \leq_m \mathbf{P}_{\text{total}}$).
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.

Repetitorium II (02.06.2017)

Aufgabe α

a) P: Menge aller Entscheidungsprobleme, die von einer deterministischen TM in polynomieller Zeit entschieden werden können. (entschieden: der Algorithmus hält immer)

b) NP: Wie P, nur mit NTM. Menge aller Entscheidungsprobleme, so dass für Instanzen ein Zertifikat in polynomieller Zeit geraten und überprüft werden kann. (NP ist Menge aller Suchprobleme, bei denen ich weiß wann ich angekommen bin)

c) $PSpace$: Menge aller Entscheidungsprobleme, die von einer DTM in polynomiellem Platz entschieden werden können.

d) $P \subseteq NP \subseteq PSpace$: DTM „sind" auch NTM $\rightarrow P \subseteq NP$. NTP, die polynomiell-zeitbeschränkt sind, können in deterministisch polynomiellem Platz simuliert werden $\rightarrow NP \subseteq PSpace$. (Auch anhand der Anzahl möglicher Lesevorgänge begründbar). Außerdem Satz von Savitch: $NP \subseteq NPSpace \subseteq PSpace$.

e) $\mathcal{C}$-hart: ein Entscheidungsproblem ist $\mathcal{C}$-hart, wenn alle Probleme in in $\mathcal{C}$ in polynomieller Zeit auf dieses reduzierbar sind. Es ist $\mathcal{C}$-vollständig, wenn es $\mathcal{C}$-hart ist und selbst in $\mathcal{C}$ liegt. Am Beispiel von SAT sehen wir, dass SAT $\mathcal{C}$-vollständig ist, da es selbst in NP liegt und kein Problem in NP schwerer ist (und somit alle auf SAT reduzierbar sind) als SAT. Es kann vorkommen, dass mehrere Probleme $\mathcal{C}$-vollständig sind, wenn diese in polynomiell äquivalenter Zeit lösbar sind.

Aufgabe β

Zeigen, dass NP unter Kleene-Stern abgeschlossen. $\forall L \in NP : L^* \in NP$
Sei $L \in NP$ und sei M eine polynomiell-zeitbeschränkte TM, so dass $L = L(M)$.
Definiere $N =$ bei Eingabe ω

- rate Zerlegung $\omega = \omega_1, \ldots, \omega_n$ (beim leeren Wort: $n = 0$) (nicht-deterministisch)
- simuliere M auf ω_i für $i = 1, \ldots, n$ (nicht-deterministisch)
- akzeptiere, falls alle Simulationen akzeptieren

N ist polynomiell-zeitbeschränkt und $L(N) = L^*$

Aufgabe γ

Aufgabe mit Problem **K**: zwei gerichtete Graphen G_1 und G_2 sowie eine Zahl $k \in \mathbb{N}$. Gesucht: Teilmengen und Bijektion.

a) $K \in NP$ da Teilmengen $V_1^{'}$ und $V_2^{'}$ und die Zuordnung f geraten werden kann und in polynomieller Zeit überprüfbar ist ob $f : V_1 \rightarrow V_2$ eine Bijektion ist, so dass $(u, v) \in E_1 \implies (f(u), f(v)) \in E_2$.

b) Sei G ein Graph und $n \in \mathbb{N}$. Gefragt ist dann, ob G eine **CLIQUE** der Größe n als Untergraph enthält.
Sei $f(enc(G)\#\#enc(n)) := enc(G)\#\#enc(K_n)\#\#enc(n)$ wobei K_n der vollständige Graph auf n Knoten ist. Dann gilt: f ist polynomiell-zeitbeschränkt und G hat **CLIQUE** der Größe $n \iff enc(G)\#\#enc(K_n)\#\#enc(n) \in K$. Also ist f eine polynomiell-zeitbeschränkte Many-One-Reduktion von **CLIQUE** auf K und damit ist K auch NP-hart.
Liste bekannter Probleme: SAT/3SAT/CNFSAT, CLIQUE/IndepententSet/HamiltonCircle, 3-Färbbarkeit

Aufgabe δ

a) Entscheider für L_1: $N =$ bei Eingabe ω

- berechne Reduktion $f(\omega)$ (polynomielle Zeit)
- entscheide, ob $f(\omega) \in L_2$

N ist polynomiell-platzbeschränkt, da:

- f polynomiell-zeitbeschränkt
- $f(\omega) \in L_2$ kann in polynomiellem Platz entschieden werden.

N ist auch Entscheider, da es einen polynomiell-platzbeschränkten Entscheider für L_2 gibt. Also ist $L_1 \in PSpace$.

b) Sei $L \in PSpace$. Dann ist $L \leq_p L_1 \leq_p L_2$ also $L \leq_p L_2$ (transitiv). Also ist jedes Problem in PSpace auf L_2 in polynomieller Zeit reduzierbar und L_2 damit PSpace-hart.

Aufgabe ϵ

a) „Jedes PSpace-harte Problem ist NP-hart“ – Richtig, da $NP \subseteq PSpace$.

b) „Es gibt kein NP-hartes Problem, welches in PSpace liegt“ – Falsch, z.B. gilt $SAT \in PSpace$ und SAT ist NP-hart.

c) „Jedes NP-vollständige Problem liegt in PSpace“ – Richtig, da $NP \subseteq PSpace$ und alle NP-vollständigen Probleme liegen in NP.

d) „Es gilt $NP = PSpace$, wenn es ein PSpace-hartes Problem in NP gibt“ – Richtig, $NP \subseteq PSpace$ ist bekannt. Sei L ein PSpace-hartes Problem in NP. Sei $L' \in PSpace$. Dann gilt $L' \leq_p L$ und da NP unter polynomieller Zeitreduktion abgeschlossen ist, folgt $L' \in NP$. Also gilt $PSpace \subseteq NP$ und damit auch $NP = PSpace$.

e) „Wenn $P \neq NP$ gilt, dann gibt es kein NP-hartes Problem in P“ – Richtig, sonst wäre $P = NP$.

f) „Sei L ein PSpace-vollständiges Problem. Dann gilt $L \in P \iff P = PSpace$“ – Richtig.

Aufgabe ζ

Tic-Tac-Toe-Spiel. Die Beschreibung einer Gewinnstrategie erfolgt mit Hilfe eines Baumes, auf dem die möglichen Abläufe skizziert werden. Alle Möglichen Spielzüge von X und O führen zum Sieg von X.

Aufgabe η

„Zeigen Sie, dass für jedes PSpace-vollständige Problem **L** auch das Komplement $\overline{\mathbf{L}}$ ein PSpace-vollständiges Problem ist.“ $\mathbf{L} \in PSpace \rightarrow \overline{\mathbf{L}} \in PSpace$. $\overline{\mathbf{L}}$ ist PSpace-hart:

$$\begin{aligned} \mathbf{H} \in PSpace &\Longrightarrow \overline{\mathbf{H}} \in PSpace \\ &\Longrightarrow \overline{\mathbf{H}} \leq_p L \\ &\Longrightarrow \mathbf{H} \leq_p \overline{\mathbf{L}} \end{aligned}$$

Also ist $\overline{\mathbf{L}}$ PSpace-vollständig.

Aufgabe θ

„Zeigen Sie: ist $P = NP$, dann sind alle Sprachen $\mathbf{L} \in P \setminus \{\emptyset, \Sigma^*\}$ NP-vollständig.“
Sei $\mathbf{L} \in P \setminus \{\emptyset, \Sigma^*\}$. Sei $K \in NP$. Wir zeigen, dass $\mathbf{K} \leq_p \mathbf{L}$, unter der Annahme, dass $P = NP$.

Seien $x_1 \in \mathbf{L}, x_2 \in \Sigma^* \setminus \mathbf{L}$. Definiere $f(\omega) = \begin{cases} x_1 \; falls \; \omega \in \mathbf{K} \\ x_2 \; sonst \end{cases}$

Da $\mathbf{K} \in P$ ist die Abbildung f in polynomieller Zeit berechenbar. Es gilt $\omega \in \mathbf{K} \iff f(\omega) \in \mathbf{L}$. Also ist f eine polynomiell-zeitbeschränkte Many-One-Reduktion von **K** auf **L**. Also ist **L** auch NP-vollständig.