Prof. Dr. Heinz Schmitz
Fachbereich Informatik
Hochschule Trier

Theoretische Informatik
Wintersemester 2018/2019
20. September 2018

Inhalt

1. Einführung

   1.1 Warum Theoretische Informatik?
   Gründe für die Unverzichtbarkeit.

   1.2 Über diese Lehrveranstaltung
   Überblick über den Inhalt, Lernziele Wissen / Anwenden / Beweisen, Prüfung, unterstützende Angebote, Vorbereitung.

   1.3 Sätze und Beweise
   Formulierung von Sätzen, Implikation, Voraussetzung, Prämisse, Konklusion, Quantoren, Genau-dann-wenn-Aussagen, Beweisformen: direkter Beweis, Kontraposition, Widerspruchsbeweis, Beweis durch Gegenbeispiel.

2. Induktion

   2.1 Vollständige Induktion über $\mathbb{N}$
   Algorithmus `s`, PYTHON-Einführung, Korrektheitsnachweis, Induktionsprinzip, (IA), (IS), (IV), erweitertes Induktionsprinzip, Beispiele.

   2.2 Induktive Definitionen
   Induktive Struktur von $\mathbb{N}$, induktive Definitionen, $Expr$, Prinzip der strukturellen Induktion, Beispiele.

   2.3 Rekursive Funktionen
   Schema für rekursive Funktionen über $\mathbb{N}$, fib, rekursive Funktionen auf induktiv definierten Mengen.

3. Wörter und Sprachen

   3.1 Alphabete und Wörter
   Alphabet, Wort über Σ, leeres Wort ε, Länge $|w|$, Konkatenation von Wörtern, w^n, Präfix, Suffix, Teilwort, Einführung in PYTHON-Strings.

3.2 Formale Sprachen
Σ^k, Σ^*,Σ^+, Binärdarstellung, formale Sprache über Σ, Entscheidungsproblem.

3.3 Operationen auf Sprachen
Mengenoperationen auf Sprachen, Alphabetswechsel, Besonderheit Komplement, Konkatenation von Sprachen, L^k, Iteration L^*, L^+, Spiegelung w^R, L^R, Bindungsstärke, Einführung in PYTHON-Mengen, Unterschiede zu den theoretischen Konzepten.

4. WHILE-Programme

4.1 Berechnungsmodelle
Eigenschaften von Algorithmen, Berechnungsmodell, Vorgehensweise.

4.2 Syntax und Semantik
Bedeutung von Syntax und Semantik, Bezeichner, Konstanten, Variablen, Ausdrücke `(a+b)`, `(a-b)`, Bedingungen, Anweisungsblöcke, Zuweisungen, Hintereinanderausführung, **`if`**, **`while`**, **`for`**, Funktionsdeklarationen, WHILE-Programm, von P berechnete Funktion φ_P auf $\mathbb{Z}$, Ursachen für Nicht-Terminierung, **`print`**, Kommentare.

4.3 Beispiele
prodZ, `prodZ`, divZ, `divZ`, Bestimmung der berechneten Funktionen.

5. Berechenbarkeit

5.1 Algorithmusbegriff
Algorithmus, WHILE-berechenbare Funktionen, **WHILE**, Berechnungsmodelle, Hauptsatz der Algorithmentheorie, Turing-Vollständigkeit, Church-Turing-These, berechenbare Funktionen.

5.2 Existenz nicht berechenbarer Funktionen
Mächtigkeitsrelation $\approx$, endliche Mengen, abzählbare und überabzählbare Mengen, Satz: Σ^* ist abzählbar unendlich; **WHILE** ist abzählbar unendlich; Satz: Es gibt überabzählbar viele totale, einstellige Funktionen auf $\mathbb{Z}$, Diagonalisierungsbeweis, Existenz nicht berechenbarer Funktionen.

5.3 (Un-)Entscheidbare Mengen
Charakteristische Funktion c_A, Entscheidbarkeit, Klasse **REC**, Abschlusseigenschaften; Halteproblem, Äquivalenzproblem und Korrektheitsproblem sind unentscheidbar (o.B.).

6. Laufzeitanalyse

   6.1 Vorgehensweise
   Anforderungen an Laufzeitaussagen, drei Schritte zur Ermittlung von Schrittzahlfunktion, Laufzeitfunktion bzw. Ordnungsklasse; Arten von Kostenmodellen, uniformes Kostenmodell für WHILE-Programme.

   6.2 Asymptotische Notationen
   O-Notation, Ordnungsklassen, Ω, Θ, asymptotische Grenzen für Polynome, Logarithmus- und Exponentialfunktionen.

   6.3 Analyse
   Schrittzahlfunktion $\text{STEP}_M(x)$, Eingabelänge n für Zahlen, Satz: Zusammenhang zwischen Größe und Länge der Eingabe; Analyse von `prodZ`, Laufzeitfunktion $\text{TIME}_M(n)$.

7. PYTHON-Programme

   7.1 Berechnungsprobleme
   Berechnungsproblem, Instanz, SORT, Entscheidungsproblem, SUBWORD, SOS, Optimierungsproblem, MAXIMUM KNAPSACK, Lösungsraum $sol(x)$, Maßfunktion $m(x, y)$, $m^*(x)$, $sol^*(x)$, korrespondierendes Entscheidungsproblem, KNAPSACK.

   7.2 Erweiterung der WHILE-Programme
   Artihmetische Operationen, Kurzschreibweisen; Zeichenketten **`str`**, `subword_naiv` löst SUBWORD in $O(n^2)$, vereinfachte Laufzeitanalyse; Mengen **`set`**, Eingabelänge für Mengen, $\#A \leq |A|$.

   7.3 Weitere Datenstrukturen
   Listen **`list`**, Eingabelänge für Listen, $m \leq |l|$, `insertion_sort` löst SORT in $O(n^2)$; assoziative Listen **`dict`**, Eingabelänge für assoziative Listen.

8. Deterministische endliche Automaten

   8.1 Arbeitsweise und Definition
   Beispiel, Tupeldefinition, Transitionsdiagramm, Arbeitsweise, DEAs in PYTHON, erweiterte Überführungsfunktion $\widehat{\delta}$, akzeptierte Sprache $L(A)$, effiziente Ausführung von DEAs mit `run_dea`, die Klasse $\mathbf{DEA}_\Sigma$.

   8.2 Entscheidungsprobleme
   DEA als endliche Repräsentation einer Sprache, Problem DEA-EMPTINESS ist entscheidbar, Problem DEA-FINITENESS ist entscheidbar.

8.3 Anwendung: Suchautomaten
Preprocessing: $p_v(u)$ und Konstruktion von A'_v und A_v, Algorithmus `subword_dea` in $O(\#\Sigma \cdot |v| + |w|)$.

9. Nichtdeterministische endliche Automaten

9.1 Arbeitsweise und Definition
Beispiel, Tupeldefinition, Arbeitsweise, erweiterte Überführungsfunktion $\widehat{\delta}$, akzeptierte Sprache $L(A)$, NEAs in PYTHON, Ausführung von NEAs mit `run_nea` in $O(n^3)$, die Klasse $\mathbf{NEA}_\Sigma$.

9.2 Äquivalenz von DEA und NEA
Teilmengenkonstruktion, Beispiel, Satz: $\mathbf{DEA}_\Sigma = \mathbf{NEA}_\Sigma$.

9.3 Abschlusseigenschaften
Abschluss unter Komplement, Vereinigung, Durchschnitt, Mengendifferenz, Konkatenation, Iteration.

10. Reguläre Ausdrücke

10.1 Reguläre Ausdrücke
Syntax und Semantik regulärer Ausdrücke, Äquivalenz, Bindungsreihenfolge, Rechenregeln, die Klasse $\mathbf{REG}_\Sigma$.

10.2 Von Ausdrücken zu Automaten
Konstruktion von Automaten aus Ausdrücken, Satz: $\mathbf{REG}_\Sigma \subseteq \mathbf{NEA}_\Sigma$.

10.3 Von Automaten zu Ausdrücken
Mengen $R_{ij}^{(k)}$, Herleitung Rekursionsformel, Algorithmus `dea_to_regex` nach dem Prinzip *Dynamischen Programmierung*, Satz: $\mathbf{DEA}_\Sigma \subseteq \mathbf{REG}_\Sigma$.

11. Reguläre Sprachen

11.1 Regulär oder nicht?
Reguläre Sprachen $\mathbf{DEA}_\Sigma = \mathbf{NEA}_\Sigma = \mathbf{REG}_\Sigma$, endliche und co-endliche Sprachen sind regulär; Schubfachschluss, Pumping Lemma für reguläre Sprachen, L_{01} ist nicht regulär, Struktur von PL-Beweisen, Beispiele.

11.2 Zusammenfassung "Reguläre Sprachen"

12. Kontextfreie Sprachen

    12.1 Kontextfreie Grammatiken
    L_{pal}, Tupeldefinition kontextfreier Grammatiken, G_{pal}, G_{Expr}, Ableitung $\Rightarrow_G^*$, erzeugte Sprache $L(G)$, die Klasse $\mathbf{KFG}_\Sigma$, Eingabelänge für `tuple`, Grammatiken in PYTHON, Satz: $\mathbf{DEA}_\Sigma \subsetneq \mathbf{KFG}_\Sigma$.

    12.2 Parse-Bäume
    Parse-Baum, Links- und Rechtsableitung, Satz: Ableitung, Linksableitung, Rechtsableitung und Parse-Baum sind äquivalent; Linksableitungsschritte in PYTHON, Mehrdeutigkeit.

    12.3 Epsilon-freie Grammatiken
    ε-Regeln, verkürzende Ableitungen, ε-freie KFG, Satz: Zu jeder KFG existiert eine äquivalente ε-freie KFG; Umwandlungsverfahren, Beispiel.

13. Ein universeller Parser

    13.1 Die Chomsky-Normalform
    Definition CNF, nützliche NTs, reduzierte Grammatik, Satz: Zu jeder KFG existiert eine äquivalente KFG in CNF; Umwandlungsverfahren, Beispiel.

    13.2 Der CYK-Algorithmus
    Idee, Sonderfall ε, induktive Beschreibung der Mengen $T(i, i+k)$, Implementierung mittels Dynamischer Programmierung, Satz: `cyk` entscheidet in $O(|w|^3 \cdot \#P)$, ob $w \in L(G)$; universeller Parser, Beurteilung.

    13.3 Ein Parser für die Menge $Expr$
    Herstellung der CNF für G_{Expr}, Implementierung.

14. Die Klassen $\mathbf{P}$ und $\mathbf{NP}$

    14.1 Polynomial- und Exponentialzeit
    Definition, Komplexitätsklassen $\mathbf{P}$, $\mathbf{FP}$ und $\mathbf{EXP}$, Beispiele, Laufzeittabelle, Polynomialzeit als Grenze des praktisch Machbaren; $\mathbf{KFG}_\Sigma \subseteq \mathbf{P}$, Abschlusseigenschaften von $\mathbf{P}$, Problem PRIM, Satz: $\mathbf{P} \subsetneq \mathbf{EXP}$.

    14.2 Die Klasse $\mathbf{NP}$
    Entscheidungsproblem TSP, gemeinsame Problemstrukturen, die Klasse $\mathbf{NP}$, SOS, KNAPSACK und TSP sind in $\mathbf{NP}$, Verifikationsalgorithmus `V_sos`; Satz: $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXP}$, `sos_exhaustive`, P-NP-Frage.

15. NP-Vollständigkeit

    15.1 Reduzierbarkeit
    Polynomialzeit-Many-one-Reduzierbarkeit $\leq_m^p$, Reduktionsfunktion, SOS $\leq_m^p$ KNAPSACK, $\leq_m^p$ ist Quasiordnung, Abschluss von $\mathbf{P}$ und $\mathbf{NP}$ unter $\leq_m^p$.

    15.2 NP-vollständige Probleme
    Definition, Beispiele SOS, TSP, SAT, KNAPSACK; Eigenschaften NP-vollständiger Probleme, Konsequenzen für die Praxis.

    15.3 Klassenübersicht

Index

Literatur

[ASU99] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilerbau - Teil 1*. Oldenbourg, 1999.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.

[HMU07] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Pearson, third edition, 2007.

[Hro07] J. Hromkovič. *Theoretische Informatik – Formale Sprachen, Berechenbarkeit, Komplexitätstheorie, Algorithmik, Kommunikation und Kryptographie*. Teubner, third edition, 2007.

[MM08] C. Meinel and M. Mundhenk. *Mathematische Grundlagen der Informatik*. Teubner, third edition, 2008.

[Sch01] U. Schöning. *Theoretische Informatik kurzgefaßt*. Spektrum, 2001.

[VW04] G. Vossen and K.-U. Witt. *Grundkurs Theoretische Informatik*. Viweg, third edition, 2004.

[Wag03] K. W. Wagner. *Theoretische Informatik – Eine kompakte Einführung*. Springer, second edition, 2003.

[Weg93] I. Wegener. *Theoretische Informatik – Eine algortihmenorientierte Einführung*. Teubner, 1993.

Genauere Literaturangaben für die einzelnen Lektionen finden sich im Vorlesungsskript.