

Computatietheorie

Lucien Sina

12.6.2025

© 2025 door Lucien Sna. Alle rechten voorbehouden.

ISBN: 9789403843087

Gedrukt door Bookmundo

Inhoudsopgave

1	Inleiding	1
2	Formele talen	1
2.1	Inleiding	1
2.2	Probleemcodering	2
2.2.1	Inleiding	2
2.3	Métro-graaf en modellering als voorbeeld	4
2.3.1	Geabstraheerde gewogen graaf	4
2.3.2	Codering van de geabstraheerde graaf	4
2.4	Codering door middel van een adjacentielijst	5
2.5	Codering door middel van een adjacenciematrix	6
2.6	Voor- en nadelen	7
2.7	Basisbegrippen formele talen	8
2.8	Eisen aan een codering	8
2.9	Getalcoderingen	9
2.10	Opgave	9
2.11	Beslissingsproblemen	10
2.12	Woord- en taalbewerkingen	10
2.12.1	Concatenatie van woorden	10
2.12.2	Overige definities	11
2.13	Leeg woord vs. lege verzameling	12
2.14	Opgaven	12
3	Berekeningsmodellen	14
4	Reguliere talen	16
4.1	Inleiding	16
4.2	Deterministische eindige automaten	17
4.2.1	Voorbeeld: een eenvoudige DEA	18
4.2.2	Model vs. de realisatie ervan	18
4.3	Toestandsdiagrammen van DEA's	18
4.3.1	Grafische voorstellingen	19
4.4	De overgangsfunctie van een DEA	20
4.5	Voorbeeld 1: berekeningsrun van M_1 voor $w = aba$	20

4.6	Geïtereerde overgangsfunctie	22
4.7	Toepassing: patroondetectie (deelwoord zoeken)	23
4.8	Eindige en reguliere talen	24
5	Niet-deterministische eindige automaten	26
5.1	Inleiding	26
5.2	Voorbeeld van een NEA	27
5.3	Definitie van een NEA	28
5.4	Geïtereerde overgangsfunctie van een NEA	28
5.4.1	Epsilon-sluiting	29
5.4.2	Definitie van de geïtereerde functie	29
5.4.3	Een ander voorbeeld	30
5.5	De machtsautomaten	32
5.5.1	Machtsautomaat bij de NEA en voorbeeldverloop voor <i>aba</i>	33
5.6	Correctheid van de machtsautomaat	34
5.6.1	Overgangsfunctie van de machtsautomaat	34
5.6.2	Gelijkmachtingheid van NEA en DEA	35
5.6.3	Oefening	35
6	Minimiseren van DEA's	38
6.1	De minimale automaat	39
6.1.1	Toestandsequivalentie	39
6.1.2	Aanvullende definities	39
6.1.3	De samengevouwen automaat	39
6.1.4	Welgedefinieerdheid van de samengevouwen automaat	40
6.1.5	Voorbeeld	41
6.1.6	Constructie van minimale automaten van een taal	43
6.1.7	Voorbeeld	44
6.1.8	Oefening	44
6.1.9	Eigenschappen van de Myhill-Nerode-relatie	44
6.2	De Myhill-Nerode-automaat	45
6.2.1	Welgedefinieerdheid van de Myhill-Nerode-automaat	46
6.2.2	Eigenschappen van de Myhill-Nerode-automaat	47
6.2.3	De gecomprimeerde automaat is minimaal	47
6.2.4	Stelling van Myhill-Nerode	48
7	Verdere eigenschappen van reguliere talen	49
7.1	Geslotenheid onder unie	49
7.2	Geslotenheid onder concatenatie en Kleene-ster	50
7.3	Geslotenheid onder spiegeling en complement	51
8	Reguliere expressies	52
8.1	Definitie: Reguliere expressie (RE)	52
8.2	Taal van een reguliere expressie	52
8.3	Voorbeelden	53
8.3.1	Oefeningen	53

8.4	Stelling van Kleene	54
8.4.1	Van reguliere expressies naar DEA's	54
9	Niet-regulariteit	57
9.1	Voorbeeld	57
9.2	Pompingeigenschap	58
9.2.1	Definitie: Pompgelijkheid	58
9.3	Regulierend Pumpinglemma	59
9.3.1	Niet-pompbaarheid	60
9.3.2	Bewijs van niet-regulierheid	60
9.4	Voorbeelden en Oefeningen	61
9.5	Voorbeeld van een pompbare, maar niet-reguliere taal	62
10	Contextvrije talen	64
10.1	Pushdown-automaten	64
10.1.1	Formele definitie	65
10.1.2	Werking	66
10.1.3	Verdere definities	67
10.1.4	Taal van een pushdown-automaat	67
10.1.5	Contextvrije talen	67
10.1.6	Grafische weergave	68
10.1.7	Voorbeelden	68
10.2	Contextvrije Grammatica's	70
10.2.1	Voorbeeld en definities	70
10.2.2	Meer Voorbeelden	71
10.2.3	Opgave	72
10.2.4	Afleidingboom en ambiguïteit	73
10.3	Modellequivalentie	76
10.4	Grammatica's voor reguliere talen	77
10.4.1	De Chomsky-normvorm	78
10.4.2	Voorbeelden	78
10.4.3	Transformatie naar CNF	79
10.4.4	Opdracht	80
10.5	Het woordprobleem en het CYK-algoritme	82
10.6	Contextvrij Pumpinglemma	83
10.7	Bewijs van niet-contextvrijheid	85
10.8	Afsluitingseigenschappen van contextvrije talen	87
11	Beslisbare en herkenbare talen	90
11.1	Inleiding	90
11.2	Definitie: Deterministische Turingmachine	91
11.3	Beslisbare en herkenbare talen	91
11.4	Varianten van Turingmachines en de Church-Turing-stelling	92
11.5	Vooruitblik	93
11.6	Voorbeeld: Turingmachine voor een niet-contextvrije taal	93
11.7	Varianten van Turingmachines	94

11.7.1 Meerband-Turingmachines	95
11.7.2 Halfband-Turingmachines	96
11.8 Lineair Beperkte Automaten	98
11.8.1 Informele beschrijving	98
11.8.2 Formele definitie	99
11.8.3 Acceptatie	99
11.8.4 Relatie tot grammatica-klassen	99
11.8.5 Voorbeeld	99
11.9 Niet-deterministische Turingmachines	100
11.9.1 Mogelijkheden van NTMs	102
11.9.2 Turingmachines voor functies	103
11.10 De Church-Turing-stelling	105
11.10.1 Redenen voor acceptatie / Evidentie	105
11.10.2 Gebruikelijke, gelijkwaardige modellen	106
11.10.3 Beperkingen en varianten van de stelling	106
11.10.4 Conclusie	106
11.11 Tellenbare talen	107
11.11.1 Voorbeeld en oefening	108
11.12 co-enumerabele talen	110
11.13 Turingmachine-coderingen	111
11.13.1 Waarom dergelijke coderingen nuttig zijn	113
11.14 Universele Turingmachines	114
11.14.1 Werking van een universele Turingmachine M_u	114
11.14.2 Formeel: Bestaan van een universele machine	115
11.15 Andere talen	117
11.16 Andere belangrijke talen	118
11.17 Suintingseigenschappen van A en B	119
12 Onbeslisbare talen	121
12.1 Bestaan van niet-opsombare talen	122
12.2 Een niet-beslisbare taal	123
12.3 Een niet-opnoembare taal	124
12.4 Verificatie-algoritmen	125
12.5 Het Halteprobleem	125
12.6 Het universele woordprobleem	126
12.6.1 Plaatsing in de hiërarchie	127
12.7 Het reductieprincipe	128
12.8 De stelling van Rice	131
12.9 Het equivalentieprobleem voor Turingmachines	132
13 Afsluitende woorden	135
13.1 Dank en afsluitende aanmoediging	136

Voorwoord

Dit boek *Computatietheorie* is bedoeld als een compacte, doch rigoureuze inleiding in de fundamentele concepten van berekenbaarheid, formele talen en automatenleer. Het doel is u — studenten van informatica, wiskunde en aanverwante vakken, evenals alle technisch geïnteresseerden — de belangrijkste modellen, bewijstechnieken en grenzen van het berekenbare op een begrijpelijke manier na te brengen.

Centraal staan de klassieke rekenmodellen (eindige automaten, kellerautomaten (pushdownautomaten), Turingmachines), de bijbehorende taalklassen (regulier, contextvrij, beslisbaar / opsombaar) en de kernstellingen die ons laten zien wat berekenbaar is en waar fundamentele grenzen beginnen (onder andere Myhill–Nerode, pumping-lemma, Kleene-stelling, diagonalisatie, Rice-stelling). Naast formele definities en bewijzen vindt u talrijke voorbeelden, oefenopgaven en modeloplossingen — de nadruk ligt op bewijstechniek en intuïtie, niet op formele pedanterie.

Voor het begrijpen van dit boek zijn basiskennis van discrete wiskunde, propositielogica en predikaatlogica evenals grondbeginselen van algoritmetheorie nuttig; deze kunt u, indien nodig, via mijn andere boeken over deze onderwerpen verwerven. Voorafgaande kennis van formele talen en automaten is handig maar niet verplicht: de behandeling is zodanig opgezet dat ook lezers die deze onderwerpen voor het eerst grondig bestuderen, stapsgewijs worden meegenomen.

De opbouw van het boek is bewust modulair: eerst behandelen we reguliere talen en automaten, daarna contextvrije grammatica's en kellerautomaten, vervolgens Turingmachines en de klassen van beslisbare respectievelijk opsombare talen. Tenslotte onderzoeken we onvertaalbaarheid (onbeslisbaarheid), het reductieprincipe en de stelling van Rice. Elk hoofdstuk bevat motiverende voorbeelden, grafische weergaven, volledige bewijzen en oefenopgaven met modeloplossingen — hierdoor leent het boek zich zowel voor zelfstudie als als begeleidend tekstboek bij colleges.

Tijdens het schrijven vond ik het belangrijk een balans te bewaren tussen wiskundige strengheid en een heldere, intuïtieve presentatie. Veel bewijzen zijn volledig uitgewerkt; waar technische details de leesbaarheid sterk zouden schaden, heb ik bewust teruggehouden en de intuïtie benadrukt.

Ik ontvang graag opmerkingen, correcties en suggesties — die helpen de volgende druk nog duidelijker en nuttiger te maken.

Veel lees- en bewijsplezier!

Hoofdstuk 1

Inleiding

Een *algoritme* is een precieze en eindige reeks instructies die een gegeven probleem voor willekeurige geldige invoer in een eindig aantal stappen oplost. Het kenmerkt zich door de volgende eigenschappen:

- **Eenduidigheid:** Elke stap is duidelijk gedefinieerd en eenduidig uitvoerbaar.
- **Eindigheid:** Het algoritme bestaat uit een eindig aantal instructies en beëindigt na een eindig aantal stappen.
- **Determinisme:** Bij gelijke toestand en gelijke invoer leidt dezelfde stap altijd tot dezelfde volgende actie (bij niet-deterministische modellen wordt dit overeenkomstig uitgebreid).
- **Invoer en uitvoer:** Het neemt een goed gedefinieerde invoer aan en levert na voltooiing een goed bepaalde uitvoer.
- **Effectiviteit:** Elke stap kan mechanisch worden uitgevoerd (met pen en papier of door een geïdealiseerde machine).

Deze formele opvatting van een algoritme vormt de basis voor alle volgende beschouwingen over berekenbaarheid en de analyse van de benodigde hulpbronnen.

De algoritmeleer houdt zich bezig met de vraag welke problemen überhaupt met behulp van algoritmen op te lossen zijn en hoeveel rekeninspanning daarvoor nodig is. In dit boek volgen we twee centrale leidende vragen:

1. **Berekenbaarheid:** Welke problemen zijn principieel algoritmisch oplosbaar, en welke niet?
2. **Complexiteit:** Voor die problemen die berekenbaar zijn, hoe kunnen de daarvoor benodigde hulpbronnen — in het bijzonder tijd en geheugen — worden gekwantificeerd en vergeleken?

Om deze vragen systematisch te beantwoorden, voeren we eerst formele rekenmodellen in, waarmee algoritmische processen nauwkeurig beschreven en geanalyseerd kunnen worden. We behandelen onder andere:

- Turingmachines als standaardmodel van berekenbaarheid,
- registtermachines en RAM-modellen om praktische kosten in termen van hulpbronnen te beoordelen,
- evenals de basisbegrippen van formele talen en grammatica's.

Een centraal resultaat van de berekenbaarheidstheorie is dat er problemen bestaan die zelfs met willekeurig krachtige (theoretische) rekenmodellen niet oplosbaar zijn. Een klassiek voorbeeld is het halteprobleem (stopprobleem): er bestaat geen algoritme dat voor elke willekeurige combinatie van programma en invoer beslist of het programma stopt of oneindig blijft doorgaan.

Zelfs wanneer een probleem berekenbaar is, kan de benodigde inspanning zo groot zijn dat een praktische oplossing onhaalbaar blijft. In de complexiteitstheorie classificeren we problemen daarom naar hun moeilijkheidsgraad in complexiteitsklassen zoals **P**, **NP**, **PSPACE** en verder.

Samen met mijn vervolgbok over complexiteitstheorie vormt dit een tweedelige boekenreeks:

Deel I Berekenbaarheidstheorie: grondslagen, rekenmodellen en onbeslisbaarheidsbewijzen.

Deel II Complexiteitstheorie: analyse van hulpbronnen, inleiding in complexiteitsklassen en centrale onopgeloste vragen.

Het doel van dit boek is u zowel het theoretische fundament als de methodische instrumenten te bieden om zelf berekenbaarheids- en complexiteitsvraagstukken nauwkeurig te formuleren en te beantwoorden.

Hoofdstuk 2

Formele talen

2.1 Inleiding

In de informatica dienen formele talen als een krachtig en precies middel om complexe structuren, regels en feiten om te zetten in eenduidig gedefinieerde tekenreeksen. In plaats van informele beschrijvingen in natuurlijke taal bieden formele talen en de bijbehorende formalisering de volgende voordelen:

- **Eenduidigheid:** Elke uitspraak en elke expressie is goed gedefinieerd en vrij van ambiguïteit.
- **Abstractie:** Essentiële eigenschappen worden benadrukt, onnodige details worden weggelaten.
- **Analyseerbaarheid:** Wiskundige en algoritmische procedures kunnen direct op formele beschrijvingen worden toegepast.
- **Automatisering:** Hulpmiddelen zoals parsers, compilers en bewijssystemen kunnen formele specificaties machinaal verwerken.
- **Controleerbaarheid:** Correctheidsbewijzen en complexiteitsanalyses kunnen systematisch worden uitgevoerd.

Een formele taal L over een alfabet Σ is een verzameling woorden, waarbij elk woord een eindige opeenvolging van symbolen uit Σ voorstelt. Formeel schrijven we

$$L \subseteq \Sigma^*,$$

waarin Σ^* alle mogelijke eindige combinaties van symbolen uit Σ omvat. Een formele taal bepaalt daarmee welke tekenreeksen als 'geldig' gelden en vormt de basis voor precieze definities van syntaxis, herkennings- en beoordelingsprocedures.

Met formele talen kunnen we algoritmische problemen zo op- en beschrijven dat we

1. problemen precies formuleren en vertalen naar bewerkingen op tekens en structuren,
2. de syntaxis en semantiek van programmeertalen, dataformaten en protocollen modelleren,
3. bereikbaarheids-, bevestigbaarheids- en beslissingsvragen systematisch onderzoeken.

In het vervolg van dit hoofdstuk verdelen we de inhoud in de onderstaande secties, voordat we in het volgende hoofdstuk beginnen met de gedetailleerde codering van individuele problemen:

- § **Probleemcodering** Weergave van problemen via alfabetten, unaire en binaire codering, adjacency-lijsten, adjacency-matrices en Booleaanse functies.
- § **Beslissingsproblemen** Formele definitie van ja-/nee-vragen en de talen die deze antwoorden karakteriseren.
- § **Woord- en taalkundige operaties** Onderzoek naar operaties zoals vereniging, concatenatie en Kleene-ster en hun effect op taalklassen.
- § **Rekenmodellen** Invoering van geïdealiseerde rekenaars – eindige automaten, pushdownautomaten, Turingmachines.
- § **Looptijdanalyse** Methoden om de tijdcomplexiteit bij de verwerking van formele talen te schatten en te vergelijken.

2.2 Probleemcodering

2.2.1 Inleiding

Om algorithmische problemen in de praktijk op te lossen, moeten we reële vraagstukken eerst omzetten in wiskundige modellen. Pas op dit abstracte niveau kunnen we algoritmen ontwikkelen en hun correctheid en efficiëntie analyseren.

Voorbeeld (navigatie in een openbaar vervoersnet). We willen de kortste reis (zonder wachttijden) tussen twee haltes in een openbaar vervoersnet berekenen. In het wiskundige model representeren we het net door een *gewogen graaf* $G = (V, E, w)$, waarbij

- V de verzameling stations (knooppunten) is,
- $E \subseteq V \times V$ de lijnen/verbindingen (kanten) zijn, en
- $w: E \rightarrow \mathbb{N}$ het gewicht van elke kant aangeeft als gemiddelde reistijd in minuten.

Het oorspronkelijke vraagstuk reduceert hiermee tot het klassieke kortste-padprobleem.

Maar hoe "begrijpt" een geïdealiseerde rekenaar deze graaf? Op het laagste niveau kan een rekenaar slechts bits (0 en 1) verwerken. Daarom hebben we een *codering* nodig die de gewogen graaf als een eindige tekenreeks (woord) voorstelt.

1. **Keuze van het alfabet:** We gebruiken de tekenvoorraad

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \#\}.$$

2. **Benaming van knopen:** De knopen $V = \{v_1, v_2, \dots, v_n\}$ nummeren we van 1 tot n .

3. **Codering van kanten:** Elke kant (v_i, v_j) met gewicht w_{ij} schrijven we als

$$i \# j \# w_{ij}.$$

4. **Scheidingssymbol:** Enkelvoudige kantbeschrijvingen worden door het woord ' $\#\#$ ' van elkaar gescheiden.

Het resulterende woord

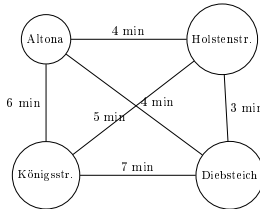
$$i_1 \# j_1 \# w_{i_1 j_1} \# \# i_2 \# j_2 \# w_{i_2 j_2} \# \# \dots \# \# i_m \# j_m \# w_{i_m j_m}$$

is een geldige codering van de graaf G . Dezelfde structuur kan in verschillende volgorde gecodeerd worden zonder dat de informatie verandert.

Met deze gestandaardiseerde voorstelling kunnen we later geschikte rekenmodellen (bijv. Turingmachines) met het woord voeden en de berekenbaarheid en looptijdeigenschappen van het probleem onderzoeken.

2.3 Métro-graaf en modellering als voorbeeld

Uitsnede van de Métro-kaart: Hamburg



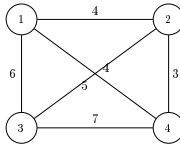
Figuur 2.1: Uitsnede van de Métro-kaart: stations en reistijden

2.3.1 Geabstraheerde gewogen graaf

We wijzen de vier stations de natuurlijke getallen $1, \dots, 4$ toe:

$$\{\text{Altona}, \text{Holstenstr.}, \text{Königsstr.}, \text{Diebsteich}\} \mapsto \{1, 2, 3, 4\}.$$

Elke directe verbinding krijgt haar reistijd als kantgewicht toegewezen.



Figuur 2.2: Geabstraheerde gewogen graaf met knopen 1-4 en kantgewichten

2.3.2 Codering van de geabstraheerde graaf

In deze sectie tonen we hoe de gewogen graaf $G = (V, E, w)$ als woord over het alfabet

$$\Sigma = \{0, 1, \dots, 9, \#\}$$

aan de abstracte rekenaar wordt doorgegeven. Elke deelreeks

$$i \# j \# w$$

representeert een kant (i, j) met het gewicht w , en het dubbele scheidingsteken ' $\#\#$ ' scheidt opeenvolgende kantenbeschrijvingen van elkaar.

Voor de hier beschouwde graaf met de knopen $\{1, 2, 3, 4\}$ en de kanten

$$(1, 2, 4), (2, 4, 3), (4, 3, 7), (3, 1, 6), (1, 4, 4), (2, 3, 5)$$

krijgt men de volgende codering:

$$1 \# 2 \# 4 \# \# 2 \# 4 \# 3 \# \# 4 \# 3 \# 7 \# \# 3 \# 1 \# 6 \# \# 1 \# 4 \# 4 \# \# 2 \# 3 \# 5.$$

Met deze tekenreeks kan de rekenaar alle kanten en hun gewichten eenduidig uitlezen en het probleem verder verwerken.

2.4 Codering door middel van een adjacentielijst

Om de gewogen graaf $G = (V, E, w)$ eveneens in een voor de geïdealiseerde rekenaar geschikte tekenreeks om te zetten, gebruiken we een *adjacentielijst*-representatie. We gaan als volgt te werk:

1. We nummeren de knopen $V = \{v_1, \dots, v_n\}$ achtereenvolgens met de natuurlijke getallen $\{1, \dots, n\}$.
2. Voor iedere knoop u definiëren we het woord

$$N_u = u \# v_1 \# w(u, v_1) \# v_2 \# w(u, v_2) \# \dots \# v_{k_u} \# w(u, v_{k_u}),$$

waarbij $v_1, \dots, v_{k_u}$ de directe burens van u zijn en $w(u, v_i)$ het gewicht van de kant (u, v_i) aangeeft.

3. De totale codering van de graaf verkrijgt men door alle deelwoorden $N_1, \dots, N_n$ met het scheidingsteken ' $\#\#$ ' aan elkaar te plakken:

$$N_1 \# \# N_2 \# \# \dots \# \# N_n.$$

Deze voorstelling bevat weliswaar elke kant tweemaal (eens in de lijst van u , eens in die van v), maar biedt een eenvoudige en systematische manier om door de rekenaar direct toegang te krijgen tot de buurinformatie van elke knoop.

Voorbeeld (Métro-Plan Hamburg): Voor onze graaf met de knopen $\{1, 2, 3, 4\}$ en de kanten

$$(1, 2, 4), (1, 3, 6), (1, 4, 5), (2, 3, 4), (2, 4, 3), (3, 4, 7)$$

ontstaan de volgende adjacentielijsten: