Kapitel 3:
Fundamentale modeller og datalogiens teoretiske paradigme

Henrik Kragh Sørensen Mikkel Willum Johansen

22. april 2022

Indhold

3.1 Datalogiens matematiske rødder

Selvom datalogi og matematik i dag er forskellige, specialiserede fagligheder med separate akademiske institutioner, er matematikken på mange måder datalogiens nærmeste nabo i det akademiske landskab. Det gælder historisk, biografisk og institutionelt, og det gælder også for en lang række af de metoder, som udgør centrale dele af datalogens videnskabelige værktøjskasse. I dette kapitel vil vi undersøge den nære forbindelse mellem datalogi og matematik ved at analysere nogle *symbiotiske* forhold, hvor de to fag lapper over og gensidigt bidrager til hinandens videnskabelige undersøgelser. I dette kapitel skal vi se på matematikkens rolle i at udvikle og formulere det, vi vil kalde *datalogiens teoretiske paradigme* igennem at diskutere, hvordan datalogiens fundamentale model for beregnelighed har et matematisk og matematikfilosofisk ophav. I kapitel **??** kommer vi til at diskutere, hvordan formelle metoder spiller en vigtig rolle i programmeringssprog og sikkerhedsspørgsmål omkring software.

Datalogiens og matematikkens historie udviser et bemærkelsesværdigt dobbelt parløb: På den ene side kan man se udviklingen inden for hardware og software som (i hvert fald delvist) motiveret af behov for beregningskraft til matematiske og matematisk fysiske problemer inden for fx tabulering, logistik, kodebrydning og simulering. På den måde kan man sige, at computeren — og læren om den — er en forlængelse af anvendt matematik. På den anden side observerer man, at store dele af datalogien (i modsætning udvikling af den fysiske computer) handler om *abstrakte* beregninger, som har taget deres definition og formulering i matematiske termer. På den måde kan man sige, at datalogien — i hvert fald store teoretiske dele af den — har et matematisk grundlag og behandler problemer, der kan siges at ligge inden for matematisk logik og diskret matematik.

Faktabox 3.1 (Historiske eksempler på symbiosen)

Tabulering: Allerede i perioden omkring Den franske Revolution — længe før de teknologiske gennembrud, vi associerer med computere — iværksattes store tabuleringsprojekter af fx matematiske funktioner, hvori to hold af *menneskelige computere* fulgte nøje beskrevne procedurer i en strengt arbejdsdelt lineær proces (se Campbell-Kelly m.fl., 2003). Siden sammenholdtes resultaterne af de to beregninger som kvalitetscheck. Undervejs benyttedes andre tidligere tabuleringer.

Logistik: Under Den anden Verdenskrig investerede den amerikanske flåde betydelige midler i udviklingen af effektive logistiske løsninger. Via et bemærkelsesværdigt møde mellem matematikerne Harold Kuhn (1925–2014) og John von Neumann (1903–1957) førte dette både til grundlæggelsen af et nyt teoretisk felt, *lineær programmering*, og til praktiske løsninger og hardware-udvikling. (Hoff Kjeldsen, 2000)

Kodebrydning: På Bletchley Park nord for London lykkedes det fra 1942 for britiske kodebrydere at knække aksemagternes kryptering, særligt Enigma- og Lorentz-koderne. Der indgik en masse logisk tænkning og automatiseret afprøvning, men man kan med rette diskutere, hvorvidt maskinen Collossus var en virkelig *computer* (i moderne forstand). Og projekt Ultra var et tidligt operationsanalytisk gennembrud, idet man måtte behandle de vundne oplysninger operationelt hensigtsmæssigt. (Kahn, 1996)

Simulering: I *Manhattan Projektet* blev der gjort banebrydende brug af ENIAC-computeren, der med en tilsvarende vis ret kan siges at være en af de allerførste (moderne) computere. Den blev brugt til omfattende simuleringer af udløsningen af atomare kædereaktioner til brug for atombomben. Denne simulering var nødvendig for at udskifte en ellers umulig (eller i hvert fald uønskelig) trial-and-error med et præcist estimat, som imidlertid ikke kunne beregnes analytisk. (Haigh, Priestley og Rope, 2014)

De fire listede symbioser har mange historiske fortilfælde, men det er ikke tilfældigt, at symbiosen bliver så meget stærkere omkring Den anden Verdenskrig, hvor teknologiske udviklinger, et akut behov og finansiering, og et bemærkelsesværdigt personligt sammenfald i von Neumann var med til at krystallisere symbiosen, så den efterfølgende blev et vigtigt videnskabeligt og politisk agenda.

Denne dobbelthed møder nogle udfordringer, som har med forholdet mellem real- og formalvidenskab som allerede behandlet i kapitel 1: For så vidt matematik handler om den åbne virkelighed, er dens deduktive slutningsformer ikke gyldige, og for så vidt dens slutninger er absolut sikre, handler de om *abstrakte* entiteter og højst approksimativt om virkelige fænomener. Derfor er et det videnskabsteoretisk vigtigt og interessant at dykke længere ind i forholdet mellem datalogien og en af dens allernærmeste nabodiscipliner, nemlig matematikken.

3.2 Turings maskiner og Hilberts program

I datalogiens historie er engelske Alan Turing (1912–1954) en central skikkelse: Han formulerede en fundamental model for beregnelighed i 1930'erne, han var involveret i kodebrydningen på Bletchley Park, og efter krigen udarbejdede han et forslag til en ny computer kaldet *ACE*, i 1950'erne tænkte han dybt over menneskelig og mekanisk intelligens og fomulerede en test deraf, og datalogiens fornemste pris er i dag opkaldt efter ham.

Hilberts program

For at forstå Turings centrale teoretiske bidrag er det nødvendigt at forstå hans udgangspunkt, som handlede om tekniske og filosofiske diskussioner omkring matematikkens grundlag, der udspandt sig i begyndelsen af 1900-tallet (se også Johansen og Sørensen, 2014, kap. 5). Et af de fremherskende synspunkter var, at matematik skulle og kunne opnå absolut sikkerhed som et formaliseret sprog med klare, logiske deduktioner. Denne position, som byggede videre på et egentligt *logicistisk program* udviklet af bl.a. Gottlob Frege (1848–1925) og Bertrand Russell (1872–1970), blev kraftfuldt fremført af den ledende tyske matematiker David Hilbert (1862–1943). Hilberts *Program* tog udgangspunkt i sikkerheden af mentale operationer i *den endelige aritmetik*. Derved kombinerede Hilbert dels Immanuel Kants (1724–1804) anskuelsesform om tid med en opfattelse af, at atomare operationer i den endelige aritmetik var sikre, nærmest mekaniske. Ovenpå dette metafysiske grundlag byggede Hilbert en programmatisk vision for matematikkens sikkerhed, der var blevet draget i tvivl i slutningen af 1800-tallet ved opdagelsen af forskellige ikke-intuitive objekter og ikke-stringente slutningsformer:

1. Matematikkens grundlag skulle sikres gennem *formalisering* og *aksiomatisering*, dvs. at Hilberts ide var at opstille syntaktiske aksiomssystemer, hvori al mening var givet implicit af aksiomerne.

2. Pointen var at opstille formelle systemer og bevise deres *uafhængighed*, *konsistens* og *fuldstændighed* igennem *endelige metoder*. Dermed ville man have bevist, at disse centrale aksiomssystemer ikke kan indeholde modstrid.

3. Derefter kunne man tilføje *ideale elementer* igennem *konservative udvidelser*, om hvilke man kunne bevise, at de ikke *tilføjede* modstrid (se også Hilbert, 1983). Dermed ville man have sikret (kunne sikre) hele matematikken, særligt den uendelige del fra mængdelæren og differentialregningen.

I en berømt forelæsning i 1931 formulerede og beviste Kurt Gödel (1906–1978) sine *ufuldstændighedssætninger*, hvoraf den mest specifikke beviste at i ethvert aksiomssystem, som er rigt nok til at indeholde de naturlige tal, kan man bevise, at hvis det er konsistent, så er det ufuldstændigt. Dermed led Hilberts generelle program skibbrud, og matematikerne har siden måttet ty til andre måder at legitimere deres grundlag på (fx gennem såkaldt *pragmatisk formalisme*).

analytisk	udsagn, hvor udsigelsen er indeholdt i det, der udsiges noget om
syntetisk	udsagn, der ikke er analytisk
syntaktisk	udsagn i kraft af deres form
semantisk	udsagn i kraft af deres mening

Figur 1: Centrale videnskabsteoretiske begrebspar (modsætninger).

Turings maskiner og mekanisk beregnelighed

Til Hilberts program hørte også spørgsmålet om *afgørbarhed*: Kan man effektivt (evt. mekanisk) afgøre om en sætning er sand eller ej? Og det var dette problem, Turing løste i sin vigtige artikel „On Computable Numbers, with an Application to the Entscheidungsproblem“ fra 1936. I artiklen formulerede Turing en *model* for, hvad det vil sige at beregne et tal. Dette kan synes som en indskrænkning i forhold til Hilberts oprindelige spørgsmål om at afgøre *bevisbarheden* af en proposition, men ved at udnytte, at de formelle sprog er endeligt frembragte og at aksiomssystemet skal indeholde aritmetikken (samme trick, som Gödel havde benyttet), kan man se, at det at afklare beregneligheden af tal er det samme som at afklare bevisbarheden af propositioner.

Den model af en maskine, som Turing formulerede, var karakteriseret ved følgende egenskaber:

1. Den havde endeligt mange tilstande $q_1, \ldots, q_R$, som han kaldte „m-konfigurationer“.
2. Den havde et „bånd“ inddelt i „kvadrater“, som hver kan indeholde et „symbol“ (fra et endeligt alfabet).
3. Til ethvert tidspunkt er der kun et kvadrat „i maskinen“ (det „scannede kvadrat“), fx det r'te kvadrat med symbolet $\mathfrak{S}(r)$.
4. Maskinen er kun „direkte opmærksom“ på det scannede symbol, men ved at ændre m-konfigurationen kan maskinen faktisk huske nogle af de symboler, den har „set“.
5. Parret $(q_n, \mathfrak{S}(r))$ kaldte han for „konfigurationen“, og det bestemmer maskinens opførsel:
   - Maskinen kan skrive et symbol i et tomt kvadrat,
   - Maskinen kan slette et symbol i et kvadrat,
   - Maskinen kan ændre det kvadrat, der scannes, men kun ved at flytte en plads til højre eller venstre,
   - Og maskinen kan desuden ændre sin m-konfiguration.

Ved at starte sådan en 'maskine' ville det *tal*, som den skrev på sit bånd siges at være beregnet. Det var muligt, at maskinen gik i et loop og måske blev ved med at skrive tegn på båndet; så betegnede Turing den (lidt kontraintuitivt, se nedenfor) „cirkelfri“.

Det var Turings formening, at modellen ovenfor indfangede præcist, hvad det vil sige at *beregne* et tal igennem en proces, der kunne siges at være mekanisk eller automatisk, men uden at han på nogen måde havde en fysisk realisering af maskinen i tankerne:

> It is my contention that these operations include all those which are used in the computation of a number. (Turing, 1936, p. 232)

Deri ligger en grundantagelse, som også er blevet formuleret af andre nogenlunde samtidig, og som i dag går under betegnelsen *Church-Turing-tesen*: Alt hvad der kan siges at være beregneligt, kan beregnes af en maskine, som den Turing beskrev i 1936, hvilket Turing gjorde eksplicit i en artikel fra 1948 (se også Copeland, 2008). På det tidspunkt kaldte han

sine teoretiske maskiner, dvs. det vi i dag kalder Turing-maskiner, for „logical computing machines":

> It is found in practice that L.C.M.s [i.e. logical computing machines] can do anything that could be described as 'rule of thumb' or 'purely mechanical'. (Turing, 2004, p. 414)

På den måde hævdede Turing, at han havde indfanget indholdet af 'tankeløs handlen' i sine maskiner.

Den universelle Turingsmaskine og Halting-problemet

Udover at have indfanget det centrale i *beregnelighed* tillod Turings model ham også at ræsonnere omkring beregninger ved at tilbyde et meta-niveau, ligesom Hilberts formalistiske program også udgjorde en *metamatematik*. Særligt var Turing interesseret i, om han kunne afgøre, om en given maskine var *cirkelfri* eller ej. Og dertil udviklede og udnyttede han en anden central ide, nemlig at betragte maskiner (programmer) som strenge (data), hvilket ledte ham til det, vi i dag kalder *den universelle Turing-maskine*.

Turing havde udviklet en systematisk måde at angive maskinerne på ved at liste konfigurationernei tabeller, der ud fra tilstanden og symbolet på båndet angav handlingen i form af hvad der skulle skrives på båndet, hvordan hovedet skulle flytte sig, og hvilken tilstand, der skulle skiftes til. Dette tillod ham at hævde, at

> A computable sequence γ is determined by a description of a machine which computes γ. (Turing, 1936, p. 239)

Og idet han kunne systematisere tabellerne, der beskriver maskiner, (til *standard descriptions*) og oversætte dem til tal (*description numbers*) fik han en afbildning fra beskrivelsestallet n til maskinen $\mathcal{M}(n)$. Nu gælder det, at til enhver beregnelig sekvens hører mindst et beskrivelsestal, men et beskrivelsestal kan højst svare til en sekvens. Derfor er mængden af beregnelige sekvenser tællelig.

Nu fik Turing den revolutionerende ide, at ved at indkode instruktionstabellen i tilstande, kunne man konstruere en *universel* maskine:

> It is possible to invent a single machine which can be used to compute any computable sequence. (Turing, 1936, p. 241)

Hvis maskinen $\mathcal{U}$ forsynes med et bånd med standardbeskrivelsen af en maskine $\mathcal{M}$, så vil $\mathcal{U}$ beregne den samme sekvens som $\mathcal{M}$. Man kan altså sige, at Turings maskine $\mathcal{U}$ *emulerer* opførslen af maskinen $\mathcal{M}$, som er dens input. Programmeringssprog, instruktionssæt og automater siges at være *Turing-komplette*, hvis de kan simulere en Turingmaskine. Deres (beregningsmæssige) 'udtrykskraft' er altså den samme.

Denne ide satte Turing i stand til at ræsonnere omkring udtrykskraften af Turingmaskiner og i særdeleshed at løse det såkaldte *Haltingproblem*, der er forbundet med afgørbarhedstesen. Turing antog, at der findes en maskine $\mathcal{Q}$ således, at hvis den gives en standardbeskrivelse af en vilkårlig anden maskine $\mathcal{M}$, så vil $\mathcal{Q}$ markere „u", hvis $\mathcal{M}$ er „cirkulær", og markere „s", hvis $\mathcal{M}$ er „cirkelfri". Nu kombinerede han så $\mathcal{Q}$ og $\mathcal{U}$ og fik derved en maskine $\mathcal{H}$, der beregner β', dvs. tallet hvis n'te ciffer er $\phi_n(n)$, hvor $\phi_n(m)$ er det m'te ciffer i den n'te beregnelige sekvens α_n. Udførelsen af maskinen $\mathcal{H}$ er opdelt i

sektioner. I den N'te sektion tester $\mathcal{Q}$ tallet N. Og hver sektion af beregningen er endelig, så $\mathcal{H}$ er cirkelfri.

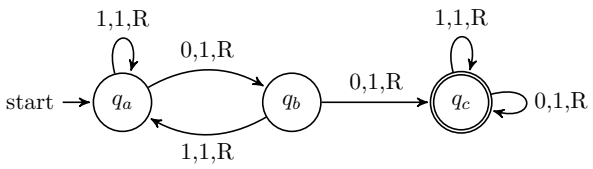
Lad så K være beskrivelsestallet for $\mathcal{H}$. Hvad skal $\mathcal{H}$ stille op i den K'te sektion af sin beregning? Eftersom K er beskrivelsestallet for $\mathcal{H}$, og $\mathcal{H}$ er cirkelfri, så kan svaret ikke være „u". Men svaret kan heller ikke være „s": Hvis svaret skal være „s", så skal maskinen beregne de første $R(K-1)+1 = R(K)$ cifre beregnet af maskinen hørende til K og skrive det $R(K)$'te ciffer som en del af den sekvens, som $\mathcal{H}$ beregner. Derved opstår en regress, og det $R(K)$'te ciffer vil aldrig blive fundet. Derfor er $\mathcal{H}$ cirkulær. Altså har vi opnået en modstrid, og maskinen $\mathcal{Q}$ er umulig.

Dette tekniske argument viser altså, at det ikke er muligt at have en generel Turingmaskine $\mathcal{Q}$, der afgør om en vilkårlig anden Turingmaskine er cirkelfri eller ej. Andre teoretiske dataloger har arbejdet videre med Turings ideer og for eksempel indført en standardiseret notation for *endelige automater* (se figur 2). Og i 1952 lykkedes det datalogen Stephen Cole Kleene (1909–1994) at formulere Turings resultat i termer af *terminerende* maskiner. Derved beviste Kleene, at der ikke kan findes en generel maskine, der afgør om en vilkårlig maskine terminerer. Antag nemlig, at der findes en procedure (en Turingmaskine) **HALTS**(P, I) som giver `true` hvis P terminerer på input I og `false` ellers. Betragt så proceduren **Z**, som også er en Turingmaskine:

```
Z(string x) {
  if (HALTS(x,x)) {
    loop forever
  } else {
    halt
  }
}
```

Nu må **Z**(Z) enten terminere eller ej. Men begge dele leder hurtigt til modstrid.

Turings resultat afgør *Entscheidungsproblemet* afkræftende: Der kan ikke gives en generel, effektiv (beregnelig) algoritme, der afgør, om en streng er en sætning i et givet formelt system. Ligesom Gödels ufuldstændighedssætninger, var Turings bevis bygget op omkring selvreference, og de deler også det forhold, at de begge er *forpligtende* for matematikere og dataloger: Beviserne er interne og baseret på grundlæggende antagelser, så man kan ikke tvivle på konklusionerne uden at drage antagelserne i tvivl. Samtidig viser Turings (og Kleenes) bevis også vejen til mange andre uafgørbare problemer ved *reduktion*: Hvis X var afgørbart, så kan vi konstruere en afgørbar procedure til **HALTS**; det kan ikke lade sig gøre, så X er også uafgørbar (se også Davis, 1982). Og det fører til problemkomplekser og kompleksitetsteori (se næste afsnit), herunder til en udviklet teori om såkaldte 'automater' (se figur 2) og spørgsmålet om $P = NP$.



Figur 2: Moderne repræsentation af en *endelig automat* med alfabetet $\{0, 1\}$. Den er givet et bånd med en streng af tegn, og hvis den når til tilstanden q_c (markeret som en *halting-tilstand*), siger vi, at den har *genkendt* (eller accepteret) strengen. Hvis man fx står

i tilstanden q_a og læser 0 på båndet, så skriver man 1 på båndet, flytter hovedet til højre (R) og skifter til tilstanden q_b.

Hartmanis' paradigme

En af de dataloger, der medvirkede centralt til at udfolde Turings indsigter om beregnelighed var den litauiske datalog Juris Hartmanis. Hans egen karriere er interessant, for efter Den anden Verdenskrig bragte den politiske situation ham ud på en omtumlet uddannelse, som han selv beskriver som en „random walk" (Shustek, 2015). Men den endte med at føre ham til USA, hvor han fik en karriere i industrien, hvor han udviklede de ideer, der i 1993 gav ham Turingprisen for hans (grund)forskning i kompleksitetsteori. I den forbindelse reflekterede Hartmanis over sine egne bidrag og deres udspring (Hartmanis, 1994). Han beskriver, hvordan han var inspireret af Turings 1936-artikel og af Claude Shannons (1916–2001) kommunikationsteori fra 1948. Sammen med Richard E. Stearns ledte det ham til i 1965 at formulere det centrale spørgsmål:

> Could there be quantitative laws that determine for each problem how much computing effort (work) is required for its solution and how to measure and determine it? (Hartmanis, 1994, p. 38)

Og svaret på deres spørgsmål fandt de i Turings maskiner, der derved, efter at de havde modificeret dem til at tillade flere bånd, blev til deres *fundamentale modeller*:

> To capture the quantitative behavior of the computing effort and to classify computations by their intrinsic computational complexity, which we were seeking, we needed a robust computing model and an intuitively satisfying classification of the complexity of problems. The Turing machine was ideally suited for the computer model, and we modified it to the multi-tape version. (Hartmanis, 1994, p. 38)

Nu kunne problemer så klassificeres, og fx studerede de klassen $C_{n^2} = \text{TIME}[n^2]$, som er alle de problemer, hvis instanser af længde n kan løses i n^2 skridt på en Turingmaskine med flere bånd. Derved fik man et abstrakt mål på kompleksiteten (målt på forskellige parametre som fx tid og plads) af beregninger, og man kunne efterhånden vise, at visse problemer var *hårdere* at løse end andre.

Foruden denne retro- og delvist introspektive forklaring bevægede Hartmanis sig i sin pristale også ud i filosofiske refleksioner over datalogiens universalitet og relation til matematik:

> The digital computer is a universal device and can perform in principle any computation (assuming the Church-Turing thesis, it captures all computations) and, in particular, any mathematical procedure in an axiomatized formal system. Thus in principle the full power of mathematical reasoning, which has been civilization's primary scientific tool, can be embodied in our computers from numerical computations and simulation of physical processes to symbolic computations and logical reasoning to theorem proving. (Hartmanis, 1994, p. 40)

Man fornemmer, at Hartmanis er, hvad Bolter (1986) har kaldt en „Turing's Man": Han anskuer verden i termer af beregninger, og Turingmaskinen udgør den centrale indfangning

af, hvad det vil sige at beregne noget. Man ser nærmest, at HARTMANIS ophøjer Turingmaskinen til det centrale definerende paradigme, og denne fornemmelse bestyrkes af en endnu bredere videnskabsfilosofisk betragtning fra samme tale:

> Thinking about the previously mentioned (and other) theoretical work in computer science, one is led to the very clear conclusion that theories do not compete with each other for which better explains the fundamental nature of information. Nor are new theories developed to reconcile theory with experimental results that reveal unexplained anomalies or new, unexpected phenomena as in physics. In computer science there is no history of critical experiments that decide between the validity of various theories, as there are in physical sciences.
>
> The basic, underlying mathematical model of digital computing is not seriously challenged by theory or experiments. The ultimate limits of effective computing, imposed by the theory of computing, are well understood and accepted. There is a strong effort to define and prove the feasible limits of computation, but even here the basic model of computation is not questioned. (Hartmanis, 1994, p. 40)

HARTMANIS står altså som eksponent for en grundlæggende position, at datalogi handler om beregninger, og beregninger er, hvad der kan indfanges af Den universelle Turingmaskine. Denne position er bundet til den teoretiske datalogi, idet den udelukkende behandler abstrakte Turingmaskiner (med fx ubegrænsede bånd) og ikke tager hensyn til de konkrete kørselstider, men betragter problemers og algoritmers kompleksitet.

3.3 Algoritmebegreb(er)

En stor del af datalogi handler om *algoritmer*, men hvad dækker dette begreb egentlig over, og hvilke typer viden kan man have om dem? Det er spørgsmålene i dette afsnit.

I moderne sprogbrug fylder *algoritmer* sommetider meget i den offentlige debat: Facebooks timeline-algoritme eller Netflix's anbefalelsesalgortime kritiseres for eksempel for at føre til en identitets-boble og et *confirmation bias*, hvor vi risikerer kun at blive præsenteret for nyheder og informationer, som vores venner og bekendte allerede har syntes godt om (Pariser, 2011; Zuboff, 2019). Dette er et eksempel på en algoritme betragtet som et socio-teknisk object, hvor teknologien indgår i og påvirker sociale relationer. 'Algoritmen' er et objekt i verden, som nogen ejer og nogen bruger, og som påvirker sociale relationer og i sidste ende kan siges at forme virkeligheden (Seaver, 2017). Hvis man går skridtet videre og tilføjer, at man også kan *kommunikere med* en sådan algoritme, træder den næsten ud og får karakter af et socialt objekt. Hvis man fx klikker systematisk (og meget!), vil man kunne påvirke 'algoritmens' fremtidige opførsel, og det er jo ikke så anderledes end at man kommunikerer med fx børn ved at belønne ønsket adfærd. Så i den forstand kan man faktisk godt tillægge algoritmen *agens* i den forstand, at det ikke bare er metaforisk, når vi siger, at algoritmer *ser*, *vælger* eller *siger*, selvom algoritmer (forstået som teknologiske systemer) jo teknisk set ikke besidder sanser, bevidsthed eller sprog (se fx Thomas, Nafus og Sherman, 2018).

Der findes også et snævrere algoritme-begreb, som typisk er det, dataloger møder og arbejder med. Det har sidst udspring i mekaniserede udførselse af fx regning og matematisk problemløsning, og dets historie kan spores tilbage til meget længe før computerens opfindelse. Fra den mesopotamiske kulturkreds (ca. 1800 f.v.t.) har vi overlevet en lang række

små lertavler, hvori er indprentet løsninger på matematiske opgaver, hvoraf vi vil fortolke nogle som løsninger af andengradsligninger (for mere om matematikhistorisk fortolkning, se Sørensen, 2016a). Et eksempel er lertavlen YBC 6967, som i tekstnær dansk oversættelse indeholder følgende tekst:

> *Igibûm* overstiger *igûm* med 7. *Igûm* og *igibûm* hvad? Du: 7 hvormed *igibûm* overstiger *igûm*, til to bræk: 3,5. 3,5 sammen med 3,5 lad holde: 12,25. Til 12,25, som fremkommer for dig, 60 fladen tilføj: 72,25. Ligesiden af 72,25 hvad? 8,5. 8,5 og 8,5, dets modstykke, indtegn. 3,5, det som holder, fra den ene udriv, til den anden tilføj. Den ene bliver 12, den anden bliver 5. 12 er *igibûm*, 5 er *igûm*. (Høyrup, 1998, s. 49–50)[1]

Dette ligner en 'opskrift' på at løse et bestemt problem, men der er nogle observationer, som vi hurtigt gør os: For det første ser det ud til, at der er to efterspurgte størrelser (*igibûm* og *igûm*), men kun en relation givet imellem dem. Imidlertid ved vi af andre grunde, som har med positionstalsystemet at gøre, at de to begreber er hinandens *reciprokke*, så *igibûm* ganget med *igûm* giver 60. For det andet noterer vi os ved at følge opskriften, at det ser ud til at den udregner størrelsen, der i algebraisk notation kan skrives som

$$\sqrt{\frac{7}{2}\times\frac{7}{2}+60}\pm 3\tfrac{1}{2}, \tag{1}$$

hvilket netop svarer til at løse ligningssystemet $x-y=7, xy=60$. Men den hjælpsomme algebraiske notation er ikke at finde i den mesopotamiske kilde og blev først opfundet 3000 år senere. Men at skrive en formel som (1) er også at angive en opskrift, og vi lærer i skolen at afkode og udføre den. For det tredje noterer vi også, at opskriften arbejder med helt konkrete tal; både forskellen mellem *igibûm* og *igûm* og alle andre afledte størrelser er givet i konkrete talværdier. Der er altså ikke et *variabelbegreb*, men der er spor af en teknik til præcist at henvise til tidligere skridt som „hvormed *igibûm* overstiger *igûm*" eller „det som holder". Opskriften er altså *konkret*, men har en mulighed for at kunne bringes i anvendelse også i andre situationer, selvom det er svært at forestille sig, hvordan den mesopotamiske skriver har lært sig, hvilke tal, der henviser til andre, og hvilke der ikke gør. Endelig, for det fjerde, er der i opskriften ikke nogen erklæring af, hvad den skal opnå eller hvorfor den virker. Der er dog spor af det i det geometriske sprog, hvor vi taler om at „brække" og „tilføje", som vi mener henviser til en bagvedliggende geometrisk konstruktion, som vi kender fra senere kulturer som at 'fuldende kvadratet'.

Vi kan nu sammenfatte eksemplet og fremdrage nogle aspekter, som vi finder relevante:

1. 'Opskriften' består af en rækkefølge af handlinger, som skal udføres. Hver handling er (i hvert fald efter tillæring) klar og har et veldefineret resultat. Opskriften kan derfor også ses at give et korrekt resultat.

2. I kilden er input helt konkret, men med et passende variabelbegreb kan man gøre opskriften anvendelig på en række beslægtede problemer (andre talværdier), og den bliver mere *generel* men har stadig et anvendelsesdomæne, selvom det ikke er angivet.

3. Med et moderne, matematisk/datalogisk syn på algoritmebegrebet ville vi sige, at kilden svarer til en *afvikling* (*kørsel*) af en (bagvedliggende, skjult) mere generel opskrift,

[1] Vi har her, ligesom i Høyrup (1998), oversat selve taltegnene fra kileskrift til det moderne talsystem, og desuden har vi konverteret fra det mesopotamiske 60-tals positionssystem til moderne decimaltal.

som vi kunne kalde et *program* eller *kode*, og som også er indfanget i (1). Denne er igen en konkretisering af en yderligere bagvedliggende *algoritme*, som er det begreb, vi ønsker at indkredse.

Selve navnet „algoritme“ stammer fra den islamiske matematiker Muhammad ibn Mūsā al-Khwārizmī (ca. 780–ca. 850), der i slutningen af 900-tallet beskrev det talsystem, vi i dag bruger og kalder for de hindu-arabiske tal. Da al-Khwārizmīs værk blev oversat til latin, blev hans navn latiniseret som *Algorismus*, og deraf udledtes *algoritme* som navn for den sekvens af operationer, han beskrev for at regne med de nye cifre i 10-tals positionssystemet. Det er værd at bemærke, at simpel regning, for eksempel multiplikation af 3-cifrede tal, netop består af en 'opskrift', som skal følges, og nogle primitiver, som skal kendes på forhånd (de små additions- og multiplikationstabeller).

Det er i sig selv bemærkelsesværdigt, at selvom matematikere og dataloger arbejder meget med algoritmer, er selve algoritmebegrebet svært at indkredse. Det algoritmebegreb, som vi kredser om, kan man forsøge at indfange ved hjælp af følgende definition:

> En *algoritme* er en endelig, abstrakt, effektiv, sammensat kontrolstruktur, som er givet imperativt, og som opnår et givet formål under givne forhold. (Hill, 2016, s. 47)

Definitionen omfatter to dele, nemlig først en indkredsning af, hvad en opskrift-delen af algoritmen skal opfylde, og dernæst en (ikke helt standard) insisteren på, at algoritmer har formål og afgrænsede anvendelsesdomæner.

Det er centralt, at Robin K. Hill definerer algoritmens essens som en *kontrolstruktur*, dvs. som en struktureret beskrivelse, der foreskriver og kontrollerer den række handlinger, der skal udføres. Det er i den forstand, at den skal være imperativt givet — dvs. udtrykt i bydeform, der kræver visse handlinger udført. Men begrebet omfatter også, at der er tale om en kontrolleret række af handlinger, hvor rækkefølgen bestemmes (kontrolleres) af den aktuelle tilstand, kørslen befinder sig i. Man kan med andre ord beskrive, at visse handlinger skal gentages et vist antal gange (som bestemmes undervejs), eller at den næste handling skal være afhængig af informationer, vi aktuelt har tilgængelige.

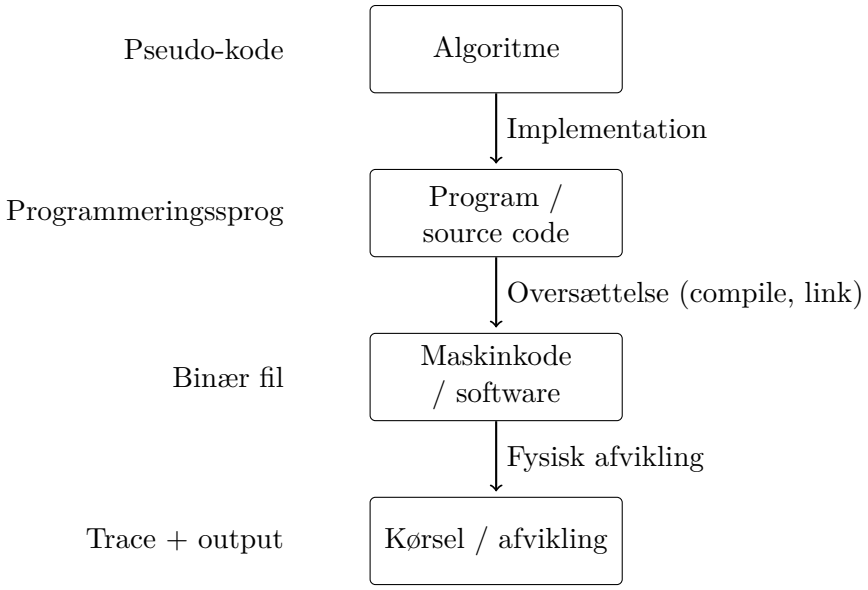
At denne kontrolstruktur skal være endelig, betyder ikke, at den ikke kan kontrollere en uendelig proces, men at selve beskrivelsen skal være endelig. Så der ligger altså ikke i kravet om endelighed noget om, at algoritmen skal terminere. Den skal også være abstrakt og dermed dække forskellige instanser i modsætning til konkrete udførelser (som fx YBC 6967). Endvidere skal den være effektiv (eng.: *effective*) i den forstand, at der ikke må skulle indgå udefrakommende valg, fx truffet ud fra menneskelig forståelse eller lignende. Algoritmen skal altså være fuldt ud præciseret, og skal kunne udføres 'tankeløst' eller 'mekanisk', hvilket dog ikke kræver nogen egentlig mekanisme, hvorfor Hill ikke ønsker at bruge dette begreb.

Ud fra Hills definition og hentydning til 'forudbestemthed' (eng.: *predetermined*) er fx randomiserede og andre ikke-deterministiske beskrivelser faktisk udelukket fra begrebet *algoritme*. Tilsvarende overraskende er det, at Hill påpeger, at *rekursive definitioner* faktisk ikke er at betragte som algoritmer, fordi de ikke er givet imperativt, selvom de kan formuleres i en kontrolstruktur. Alle disse afgrænsninger kan — og er blevet — diskuteret indgående, når dataloger og filosoffer har forsøgt at indkredse det ellers så intuitivt plausible algoritmebegreb.

Men det måske allermest bemærkelsesværdige ved Hills definition er faktisk en udvidelse af andre klassiske algoritmebegreber, idet hun kræver, at algoritmen opnår givne formål under givne forhold. Det er jo måske ikke så underligt (det betyder, at algoritmen

virker) — men det interessante er, at hun inddrager det som en del af selve definitionen af en algoritme. Dermed er en beskrivelse af en kontrolstruktur uden en beskrivelse af dens formål og antagelser altså slet ikke nogen algoritme i denne definition.

Man kan godt tænke om fysisk afvikling, maskinkode, program, og algoritme som gradvise lag af abstraktioner: Algoritmen er mere abstrakt end programmet, idet den fx ikke bundet til et bestemt programmeringssprog med fx bestemt syntaks og en bestemt måde at tilgå hukommelsen på. Og programmet er mere abstrakt end maskinkoden, hvortil det er blevet oversat og linket. Og endelig er maskinkoden mere abstrakt end kørslen, da programmet er en tekstuel repræsentation, hvorimod kørslen er en fysisk og tidslig proces, der godt nok resulterer i noget, der igen kan indfanges tekstuelt (fx et 'trace' og et 'output').



Figur 3: Relationen mellem algoritme, program, maskinkode og kørsel som en kæde af abstraktioner og instanser.

Denne måde at opfatte programmering af en computer på åbner for, at vi kan betragte algoritmer, programmer og maskinkode som 'immaterielle objekter', dvs. som ting, hvis egentlige eksistens ikke er som fysiske ting, selvom de kan repræsenteres fysisk som fx printouts af kildekode eller en USB-key med en binær fil. Endvidere kan vi opfatte processen fra algoritme til program til maskinkode som *oversættelser* der tager en given intention (formål) beskrevet på et abstraktionsniveau og omformer det til et andet abstraktionsniveau.

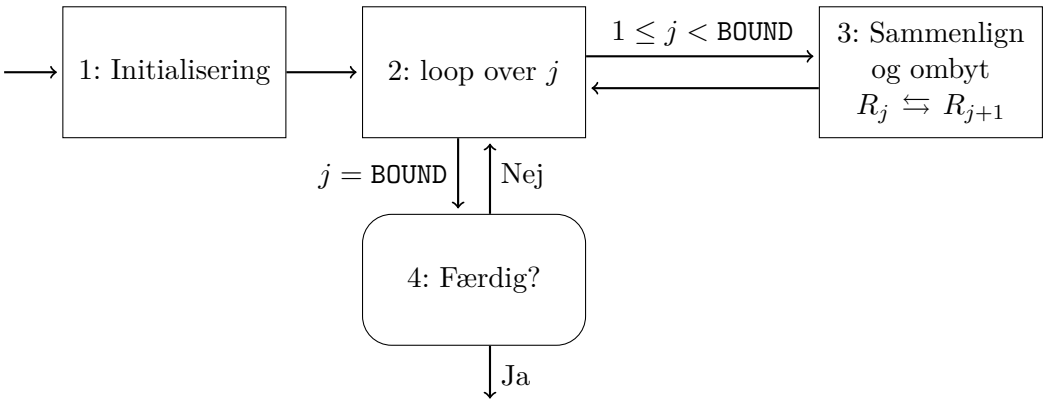
Hvis vi ønsker at sikre os, at en computer eller et program udfører *det rigtige*, dvs. at vi står foran en *korrekt* oversættelse, så har vi basalt set to muligheder: 1. Vi kan gå empirisk til værks og opfatte hele processen i figur 3 som en fysisk afvikling. Det er det, vi gør, når vi *tester* programmer (se mere i kapitel **??**). 2. Men hvis vi afgrænser os til de tre øverste niveauer og oversættelserne, så kan vi faktisk betragte dem alle som matematiske objekter, og derfor kan vi også gå formalvidenskabeligt til værks og *bevise*, at et underliggende niveau er en *formelt korrekt* instans af det overliggende niveaus specifikation.

Bubble sort

Datalogen og Turing Award-vinderen Donald Knuth, som er blevet beskrevet som Silicon Valleys Yoda (Roberts, 2018), har igennem mere end et halvt århundrede arbejdet med sit store og stadigt ufuldendte opus *The Art of Computer Programming*. Deri beskriver og analyserer Knuth en lang række algoritmer, og hele bind 3 er dedikeret til søgning og sortering. Blandt de mange sorteringsalgoritmer, som han behandler, og som man også tit møder i lærebøger er den såkaldte *bubble sort*, som er en 'in-place' sortering af en liste af 'records' $R_1, \ldots, R_N$, som hver indeholder en 'key' $K_1, \ldots, K_N$ (Knuth, 1998, s. 107). Algoritmen består af fire trin:

1. Initialisering: Sæt `BOUND`:=N
2. Loop over j: Sæt $t := 0$. Udfør trin 3 for $j = 1, 2, \ldots,$ `BOUND` $- 1$ og gå så til trin 4.
3. Sammenlign og byt: Hvis $K_j > K_{j+1}$, så ombyt R_j og R_{j+1} og sæt $t := j$.
4. Færdig? Hvis $t = 0$ så terminer. Ellers sæt `BOUND` $:= t$ og gå tilbage til trin 2.

Knuth illustrerede denne tekstuelle algoritme med et flow chart, som svarer til figur 4.



Figur 4: Knuths diagrammatiske repræsentation af *bubble sort* som et *flow chart* (Knuth, 1998, s. 107). De rektangulære kasser er handlinger, mens den rundede kasse er en afgørelse.

Tallene i flow chartet (figur 4) henviser til trin i algoritmen, og det var typisk den måde, man beskrev algoritmer på som en blanding af flow chart og instruktioner for hver kasse. Men Knuth angav også et program, skrevet i maskinkode med instruktionssættet `MIX`, som han arbejdede med samtidig.

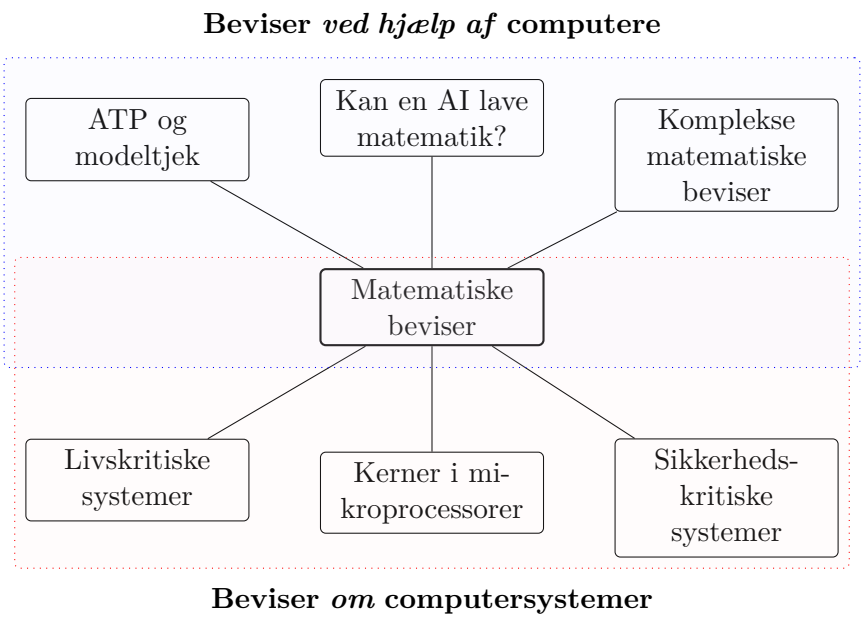
Efter at have angivet algoritmen, analyserede Knuth dens kompleksitet i termer af antal gennemløb, antal ombytninger og antal sammenligninger, og han fandt, at de sidste to var kvadratiske i N og værre end *insertion sort*, som han også havde analyseret. Endvidere foreslog han forskellige forbedringer til *bubble sort*, inden han dog nåede den lakoniske konklusion:

> In short, the bubble sort seems to have nothing to recommend it, except a catchy name and the fact that it leads to some interesting theoretical problems. (Knuth, 1998, s. 110)

For dog at trække en vigtig teoretisk pointe skal vi vende tilbage til *bubble sort* i det følgende. Og trods Knuths patos-fulde afvisning af *bubble sort*, vedbliver den at være meget udbredt, måske fordi den er nem at huske og nem at implementere (Astrachan, 2003). Den er på samme tid en lidt kluntet schweizerkniv i programmørens lomme og en datalogiens *drosophila* for den teoretiske datalog, der kender den som et udbredt og gennemstuderet modeldyr.

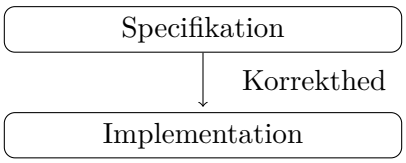
3.4 Computere og matematiske beviser

Den skotske videnskabssociolog Donald Mackenzie har interesseret sig meget for forholdet mellem det, han kalder forskellige „beviskulturer". Ved at opfatte beviser i termer af *kulturer*, flytter han fokus fra mere traditionel matematikfilosofi, der har opfattet beviser som tidsløse sandhedsgaranter, over i en mere praksis-orienteret videnskabsteori, hvor beviser er noget, *mennesker* producerer i bestemte sociale sammenhænge. En af de mest spændende analyser, som kommer ud af denne tilgang, er at betragte forskellige opfattelser af, hvad det vil sige at en computer-chip er *bevist* korrekt, når man spørger henholdsvis programmører, matematikere og jurister. Sådan en analyse var Mackenzie i stand til at gennemføre for den såkaldte „VIPER chip", som var den første kommercielle mikroprocessor, som var blevet formelt bevist korrekt (MacKenzie, 1991). Og mere generelt har Mackenzie interesseret sig for forholdet mellem matematik og computere, hvor han har undersøgt forskellige former for overlap (se figur 5). Han betragter især tre former for brug af computere i matematik: 1. Til såkaldt 'automated theorem proving (ATP)', som består i, at et stykke software med eller uden hjælp fra et menneske søger efter en formel udledning af en matematisk sætning ud fra givne præmisser (se også Johansen, Breman og Sørensen, 2019). 2. Til AI-understøttelse af matematisk kreativitet (ikke kun i form af ATP men også fx mere heuristisk); inden for de seneste tiår har man endda udviklet en *eksperimentel* gren i matematikken, som bruger computeren interaktivt og udforskende (se Sørensen, 2016b; Sørensen, 2013; Sørensen, 2010). 3. Og til understøttelse af meget store matematiske beviser med mange cases at behandle, som det for eksempel er tilfældet med Firefarvesætningen og Keplers formodning.



Figur 5: Matematiske beviser og computere, baseret på MacKenzie (2005).

Denne allerede for 20 år siden omfattende *brug* af computere og software til formalvidenskabelig vidensproduktion er vigtig i sig selv, men her er det særligt vigtigt, at den også finder anvendelse *inden for* datalogien. Mackenzie oplister tre særlige områder, hvor formelle beviser *om* computersystemer har spillet en betydelig rolle: 1. Det gælder omkring kerner og mikroprocessorer, for hvis man kan bevise, at processoren på det laveste niveau, abstraktioner oven på den, og oversættelserne imellem dem er korrekte, så er man tæt på at bevise korrektheden af computersystemet. 2. Det gælder inden for livs- og 3. sikkerhedskritiske systemer til både civil og militær anvendelse, for det vil være oplagt at søge at *bevise*, at et kritisk computersystem fx ikke kan hackes, ikke kan komme til at foretage brud på fortrolighed, eller ikke kan bringes til at crashe eller malfunktionere på anden vis. Da Mackenzie skrev for 15 år siden, var udfordringerne ved brug af formelle metoder imidlertid meget store, og primært af to slags: Dels en praktisk udfordring, som bestod i, at det var enormt tidskrævende at gennemføre formelle beviser, og dels en mere principiel udfordring, som består i, at formelle beviser aldrig kan bevise udsagn om realvidenskabelige forhold (se fx diskussionen mellem De Millo, Lipton og Perlis, 1979; Fetzer, 1988). Men siden har feltet udviklet sig enormt hurtigt, ikke mindst fordi dataloger og matematikere har fælles interesser i at udvikle platforme, der kan hjælpe med til formel verifikation. Så de oprindelige praktiske udfordringer er blevet mindre med udviklingen af værktøjer som Isabelle/HOL, og i dag findes der formelt verificerede C-compilere, operativsystemer, mikroprocessorer, konferencesystemer og meget mere. Men om alle disse systemer skal man erindre, at et formelt bevis 'blot' er et bevis for, at et niveau af software-stakken er en korrekt implementation af en specifikation på et højere niveau (se figur 6).



Figur 6: En formel verifikation er et (typisk computer-assisteret) bevis for, at et niveau af software-stakken er en korrekt implementaton af en specifikation på et højere niveau.

Beviser om algoritmer og modeller

En af de mest bemærkelsesværdige egenskaber ved algoritmer er, at vi kan komme til at betragte dem som matematiske objekter. Det er ikke helt så ligetil at skrive en algoritme ned i ord eller tegn, men dele af Hills definition er netop med for at sikre, at det kan lade sig gøre. Algoritmen er forskellige fra en konkret implementation af den, men man har udviklet forskellige mere eller mindre formaliserede måder at repræsentere *algoritmer* på, uafhængigt af specifikke aspekter ved givne programmeringssprog. Blandt de mest udbredte repræsentationsformer var *flow-diagrammer* og *pseudo-kode*, som typisk er baseret på en simplificeret form af udbredte programmeringssprog som fx *FORTRAN* (se også kap. **??**).

Formuleringen af fx *bubble sort*-algoritmen gennemgået ovenfor tillader os at *bevise* fx

1. Korrekthed af algoritmen: Vi kan *bevise*, at hvis algoritmen terminerer, så opfylder $K_i \leq K_j$ for $i \leq j$.

2. Terminering af algoritmen: Algoritmen terminerer faktisk, dvs. vi når den endelige tilstand, hvor der ikke er flere ombytninger at lave.

3. Kompleksiteten af algoritmen kan udtrykkes i $\mathcal{O}$-notation, dvs. vi kan angive asymptotiske grænser for, hvor mange hhv. gennemløb, ombytninger og sammenligninger, der foretages. Disse siger kun noget om, algoritmens effektivitet *i grænsen*, dvs. for store værdier af N, og kun hvis vores valg mellem gennemløb, ombytninger og sammenligninger faktisk indfanger den mest omkostningsfulde operation på den givne implementation.

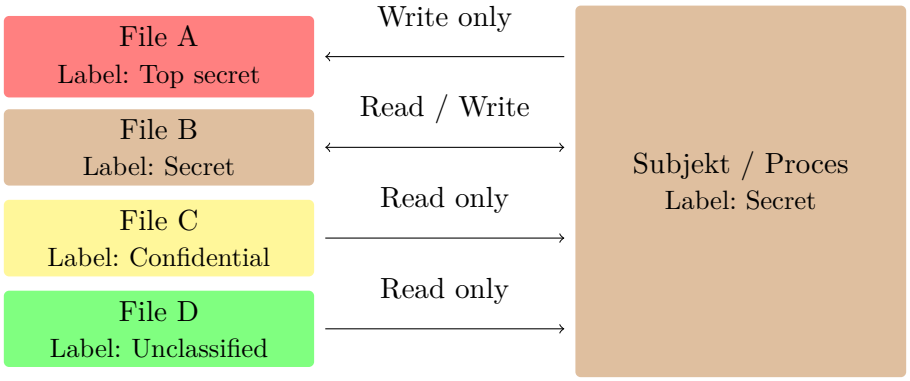
Så sammenfattende kan vi *bevise* noget om *algoritmens* korrekthed og effektivitet, og dette overfører til udsagn om de programmer, der er korrekte implementationer af algoritmen. Men det overfører ikke til viden i den fysiske verden om udkommet af en konkret beregning (som kan blive ramt af alle mulige uforudsete hændelser) eller beregningshastigheden, som sagtens kan være domineret af startomkostninger (i dette tilfælde konstantled og lineære led).

Det er måske lidt fjollet at diskutere *bubble sort* så meget, når nu Knuth og mange andre allerede har advaret os imod den algoritme, fordi den er „dårlig“. Men pointerne er principielle, vedvarende, vigtige og omkostningsfulde, sådan som vi skal se ved at betragte den såkaldte *Bell-LaPadula-model*.

Som et af resultaterne af at man under den såkaldte *softwarekrise* i slutningen af 1960'erne erkendte, at megen software var upålidelig, indførte det amerikanske forsvar retningslinjer for, hvordan den software, de skulle købe eller bygge, skulle overholde standarder for sikkerhed. Problemet var særligt udtalt, da computerne på dette tidspunkt ikke var isolerede enkeltmaskiner, men mange bruger kørte jobs på den samme mainframe ved hjælp af *time sharing*. Derfor var det en udpræget risiko, at en proces kunne komme til at læse eller skrive i hukommelse, som (også) tilhørte en anden proces. Og selvom dette ville være en generel *bug* ved et time sharing system, kunne det have kritiske konsekvenser, hvis det betød læk af fortrolig miilitær viden. Derfor satte militæret sig for at specificere en model, der skulle sikre, at informationsflow overholdt fortrolighedsprincipper — denne model blev til *Bell-LaPadula-modellen* (se Bell, 2011). Og endvidere indførte militæret den betingelse på

deres kontrakter, at software *skulle* være bevist at overholde *Bell-LaPadula-modellen*, før de kunne købe det — denne specifikation i den såkaldt 'Orange bog' måtte dog løsnes igen, da det viste sig alt for omkostningsfuldt.

Den model, som David Elliott Bell og Leonard J. LaPadula producerede og dokumenterede for det amerikanske militær blev kendetegnet ved to regler: *no write-down* og *no read-up*. Det skal forstås sådan, at alle elementer af computersystemet har en label, som angiver fortroligheden (top secret, secret, confidential og unclassified), og alle subjekter og processer, der tilgår systemet har ligeledes en af disse labels. En proces kan så ikke skrive på et niveau under sit eget og ikke læse fra et niveau over sit eget (se figur 7).



Figur 7: Bell-LaPadula-modellen, som blev udviklet for det amerikanske forsvar. Der er tale om en sikkerhedsmodel for multi-level-systemer, som sikrer *no write-down*, *no read-up*. Den brune proces (subjekt) kan læse filer på samme eller lavere niveau men kan kun skrive på sit eget eller højere sikkerhedsniveau.

Med denne specifikation kunne militæret nu kræve, at leverandører beviste, at deres systemer levede op til *Bell-LaPadula-modellen*. Men der skulle vise sig at være et problem, hvis systemet havde delte ressourcer som fx et filsystem eller en printer. For så viste det sig, at man kunne løbe ind i det, der i dag kende som en 'covert channel' — en hemmelig og utilsigtet kanal.

Antag, at vi har følgende fire operationer, som alle overholder *Bell-LaPadula-modellen*:

READ objname: Hvis objektet `objname` eksisterer, og subjektet har læseadgang, returner værdien til subjektet; ellers returner 0.

WRITE objname, value: Hvis `objname` eksisterer, og subjektet har skriveadgang, så erstat værdien med `value`; ellers gør ingenting.

CREATE objname: Hvis `objname` ikke eksisterer, så opret det på subjektets niveau; ellers gør ingenting.

DESTROY objname: Hvis `objname` eksisterer og subjektet har skriveadgang til det, så slet det; elles gør ingenting.

Hvis nu H er en agent på højt niveau, som vil kommunikere til L på et lavt niveau, så kan de indgå i en protokol (hvilket selvfølgelig kræver, at de samarbejder). Husk: H kan ikke skrive til objekter på L's niveau, og L kan ikke læse objekter på H's niveau.

1. Hvis H vil sende 0: CREATE F0; hvis H vil sende 1: gør ingenting

2. L: CREATE F0; WRITE F0, 1; READ F0; DESTROY F0

Nu ser L det samme, som H ønskede at sende: For hvis H vil sende 1 kan L godt oprette og skrive til F0, hvorfor L læser 1; omvendt hvis H vil sende 0 kan L ikke komme til at oprette F0, og derfor ikke læse fra F0, hvorfor L ser 0. Altså er information blevet sendt fra højt niveau til lavt niveau, hvilket netop strider mod *no write down*.

Covert channels er et fundamentalt problem, som peger på, at modellen er utilstrækkelig i sine antagelser. For formalvidenskabens deduktive beviser kan *kun* benyttes om *lukkede systemer*, hvor vi kender og tager højde for alle faktorer — og der er de ufejlbarlige. Men i åbne systemer, hvor agenter kan opføre sig på måder, vi ikke havde taget højde for, nytter vores matematiske beviser ikke. Sagt på en anden måde, så kan vi (forsøge at) bevise, at en *implementation* er en korrekt konkretisering af en *specifikation*. Men vi kan ikke bevise, at specifikationen i figur 6 ikke har utilsigtede *åbninger* eller sideeffekter. Dertil kræves andre tilgange — hvoraf den mest oplagte er, at vi lærer af vores tidligere fejl, sådan at vi nu indbygger potentialet af covert channels i vores antagelser. På den måde lever *Bell-LaPadula-modellen* videre — ikke længere som en uomgængelig standard, men som grundlag for læring og videreudvikling.

3.5 Videnskabelige revolutioner

Nu vender vi tilbage til Turings beskrivelse af beregninger, som Hartmanis gav udtryk for, var et grundlag for den måde, vi i datalogi måler arbejdet ved en beregning (kompleksitetsteori).

I kapitel 2 beskrev vi Thomas Kuhns (1922–1996) videnskabelige paradigmer som det fælles verdenssyn, som en gruppe videnskabsfolk er fælles om at dele. Paradigmet var konstitueret af en *disciplinær matrix*, som omfattede symbolske generaliseringer, metafysiske antagelser, værdier og eksemplarer, som gruppen er forpligtede på. Men vi forklarede ikke, hvordan Kuhn benyttede denne indfagning af den fælles sociologiske ramme til at forklare videnskabens udvikling. Nu er det på tide at vende tilbage og beskrive denne dynamik for at kunne bringe den i anvendelse til at karakterisere og perspektivere det, vi kunne kalde 'datalogiens teoretiske paradigme'.

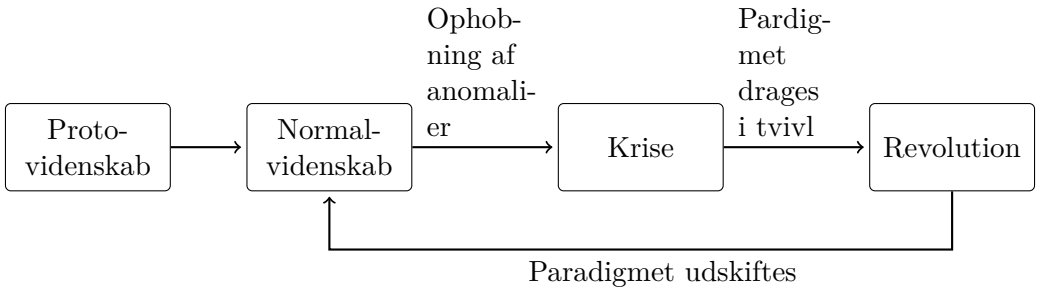
For det første bemærker man, at en enkelt videnskabsmand (m/k) i streng forstand, ifølge Kuhns teori, kun kan tilhøre et enkelt paradigme, idet et paradigme skal omfatte et *helt* verdenssyn. Så paradigmer er ikke noget, man let skifter ud, og den enkelte udøver vil være meget tilbøjelig til at forsvare og søge at opretholde sit paradigme, fordi man har 'investeret' så meget i at blive socialiseret ind i det og blive dygtig til at udøve det.

Den videnskab, som man udøver *inden for* et paradigme, kalder Kuhn for 'normalvidenskab', og den er karakteriseret ved, at udøverne så at sige *overholder* og *udforsker* paradigmet. Man søger at udvide den indsigt, som kan formuleres som svar på spørgsmål inden for paradigmet. Kuhns egen baggrund er i videnskabshistorien, nærmere bestemt fysikhistorien, og hans gennemgående eksempel er verdensbilledets udvikling, og særligt overgangen fra et geocentrisk verdensbillede, som associeres med den helenistiske astronom Klaudios Ptolemaios (ca. 100–ca. 170), til et heliocentrisk verdensbillede, som vi kalder moderne og især associerer med Nikolaus Kopernikus (1473–1543). I det geocentriske paradigme blev jorden betragtet som universets centrum, og man mente, at planeter og stjerne bevægede sig i cirkelbaner rundt om jorden. Det heliocentriske verdensbillede satte

derimod solen i centrum for universet, og forandre jordens status til en planet om solen. I første omgang beskrev man bevægelserne som cirkulære omkring solen, men med Johannes Kepler (1571–1630) blev planetbanerne raffineret til at være elliptiske. Omkring verdensbilledet forklarer Kuhn normalvidenskabelig praksis inden for fx det geocentriske verdensbillede som yderligere observationer af planeternes baner og mere og mere forfinede beregninger af fx fremtidige solformørkelser.

Imidlertid kan der i løbet af den normalvidenskabelige praksis blive afdækket såkaldte „anomalier“, dvs. uoverensstemmelser mellem paradigmets forudsigelser og empiriske fænomener. I modsætning til en radikal fortolkning af Karl Poppers (1902–1994) metode, mente Kuhn, at sådanne anomalier enten ville blive forsøgt bortforklaret inden for paradigmets begrebsverden eller simpelthen gemt væk og efterladt til senere undersøgelse. Man ville altså netop *ikke*, ifølge Kuhn, smide paradigmet (hypotesen i Poppers terminologi) overbord ved første tegn på modstand, men i stedet henlægge problemer og arbejde videre. I astronomihistorien er det mest berømte eksempel på anomalier, at man opdagede, at visse himmellegemer nogle gange bevægede sig 'baglæns' (retrograd bevægelse), hvilket ikke fik astronomer i Ptolemaios' paradigme til at opgive den geocentriske antagelse, men i stedet til at tilføje yderligere teoretiske antagelser om, at himmellegemerne bevægede sig på såkaldte „epicykler“, dvs. på cirkler omkring punkter på cirkelbevægelser omkring jorden. Andre anomalier blev simpelthen noteret men ikke forfulgt.

På et tidspunkt kan ophobningen af anomalier blive så omfattende, at nogle af udøverne begynder at drage paradigmet i tvivl, og så siger Kuhn, at paradigmet kommer i 'krise' og en vis del af den videnskabelige praksis går over i en ekstraordinær tilstand, hvor man afsøger mulige alternative formuleringer af de helt centrale antagelser. Denne krisetilstand kan (kun) løses igennem en „revolution“, hvorved det gamle paradigme forkastes og et nyt sættes i stedet. Det vil typisk ikke være en omvæltning, den enkelte udøver er i stand til at gøre, da man har investeret for meget i det gamle paradigme, og Kuhn beskriver det som, at gamle paradigmer dør ud med de sidste af deres tilhængere. Men for gruppen sættes et nyt paradigme i stedet, og krisen er blevet løst, så man nu vender tilbage til tilstanden af normalvidenskab (mono-paradigmatisk videnskab), men under et nyt paradigme. Kuhns dynamik er forklaret skematisk i figur 8.

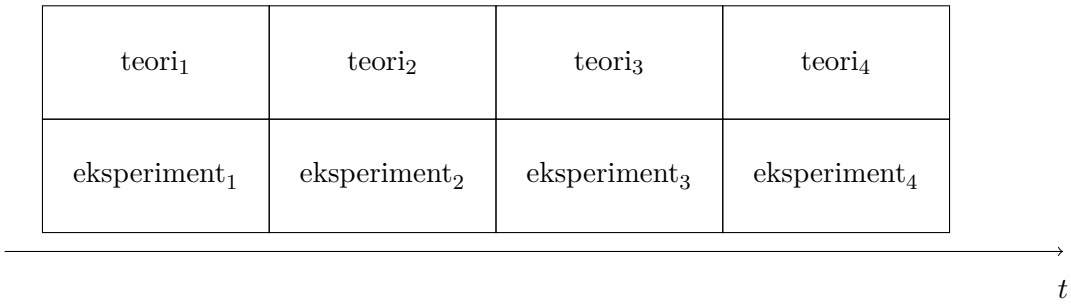


Figur 8: Skematisk repræsentation af Kuhns dynamik for videnskabens udvikling.

Hvor Popper og mange andre videnskabsteoretikere havde søgt at forklare, hvordan videnskaben *gør fremskridt*, satte Kuhn altså en cirkulær proces i stedet, hvor man kun kan tale om fremskridt inden for den normalvidenskabelige praksis. For på tværs mellem to efterfølgende paradigmer eksisterer der, ifølge Kuhn, hvad han kalder „inkommensurabilitet“. Dette begreb er en meget omdiskuteret del af Kuhns teori, og ordet betyder strengt taget 'usammenmålelighed', og det dækker over to ting: 1. Dels henviser det til, at Kuhn

mente, at det ikke var muligt at *oversætte* mellem to paradigmer, for deres grundantagelser var simpelthen inkompatible, og samme ord kunne fx betyde to helt forskellige ting, ligesom samme objekt kunne have ganske forskellig status i de to paradigmer. For eksempel var jorden jo universets centrum for PTOLEMAIOS men blot en blandt mange planeter hos KOPERNIKUS. 2. Mere radikalt henviser inkommensurabilitet også til, at der ikke findes nogen måde rationelt at *sammenligne* to paradigmer på: Man kan ikke stå uden for dem og veje dem op i mod hinanden på at udefrakommende, rationelt grundlag, for hvert paradigme vil have sine egne værdier og grunde til at hævde sine egne kvaliteter. KUHN argumenterer således for, at der ikke findes objektive kriterier at vurdere paradigmer (teorier) ud fra (Kuhn, 1995). Denne uoversættelighed og usammenlignelighed er en af de helt karakteristiske egenskaber ved en „videnskabelig revolution", sådan som KUHN fremlægger det. Den har nogle særlige konsekvenser, hvoraf vi kun kort skal berøre to, der handler om teoriladethed og interdisciplinaritet.

I KUHNS opfattelse er det ikke kun de centrale begreber og antagelser, der forandres hen over et paradigmeskift — det er så at sige også selve verden omkring os, eller i hvert fald hele vores adgang til verden omkring os. For paradigmet omfatter hele vores verdenssyn, og hvis to paradigmer ikke er oversættelige, så er al vores viden heller ikke oversættelig. Det gælder således ikke kun vores teorier om verden, men også vores empiriske adgang til den (vores eksperimenter). Dette har videnskabsteoretikeren PETER GALISON beskrevet som KUHNS „centrale metafor": At i en revolution forkastes *både* teorier og eksperimenter, for eksperimenterne er *teoriladede* og kan ikke tænkes adskilt fra den teoretiske ramme, de er udført under (se figur 9).



Figur 9: KUHNS såkaldt „centrale metafor" som introduceret i Galison (1988).

Men KUHN står over for en væsentlig udfordring, som består i, at forklare, hvor mange samtidige paradigmer, der kan findes. For eksempel kan kemikere og fysikere jo i princippet godt begge udforske naturen, men tilhører de det samme paradigme? KUHN ville nok sige nej — paradigmer er mere specialiserede end blot 'naturvidenskab' eller 'humaniora', og man kan godt tænke sig dem opdelt efter det, vi typisk kalder *discipliner*. Men hvad så med områder som fx nano-videnskab, der bevidst går på tværs mellem paradigmer som biologi, kemi og fysik? Og her har videnskabsteoretikere med udgangspunkt i KUHNS teori analyseret, hvordan sådanne „interdisciplinære" områder kan være ramt af konfliktende værdier og uoversætteligheder mellem deres indgående paradigmer.

Lad os nu vende tilbage til 'datalogiens teoretiske paradigme', ved hvilket vi vil forstå den grundantagelse (verdenssyn), som TURING og HARTMANIS tilsammen kan ses at give: Datalogien handler om studiet af beregninger (data processer) og deres kompeksitet, og alle typer beregninger kan (ultimativt) hidføres til den fundamentale teoretiske model, som vi kalder den universelle Turing-maskine.

Vi kan nu dels overveje, om denne indfangning faktisk er dækkende for vores opfattelse af datalogien, og den stemmer i hvert fald overens med mange aspekter af den karakterisering, vi i kapitel 1 uddrog fra forskellige positioner og leksika. Dernæst kan vi overveje, om den giver mening som et kuhnsk paradigme, altså som et delt verdenssyn for en gruppe af udøvere med symbolske generaliseringer, metafysiske antagelser, værdier og eksemplarer, og det er præcist det, at HARTMANIS udtrykker i citaterne, og som er blevet beskrevet som at dataloger er „Turing's Men" (Bolter, 1986) fordi de går til verden med et filter, der ser verden udelukkende i termer af beregninger.

Nu kan vi så også overveje, om karakteriseringen af datatalogi som en „vidtfavnende tværvidenskabelig disciplin" giver mening, for hvis dette er datalogiens teoretiske paradigme kan vi forudse, at der kan opstå uoversætteligheder i mødet med andre paradigmer, fx inden for naturvidenskabelige discipliner. Og netop her ligger kimen til en af datalogiens store indsigter de sidste 50 år: Hvis datalogi skal være en hjælp til andre videnskaber (og sikkert også til andre opgaver), så kræver det nok netop 'tværvidenskabelighed'. Og så kunne karakteriseringen måske udvikles til, at den form for datalogi nærmere er en *hybrid* mellem det teoretiske paradigme og andre naturvidenskabelige paradigmer, ligesom nano-videnskab er en hybrid, fordi det udvikler *sit eget* paradigme på tværs af fysik, biologi og kemi.

Og endelig kan vi overveje, om der er nogen tegn på, at det teoretiske paradigme kunne være på vej i krise og måske ikke er fuldt dækkende for det, vi vil opfatte som 'beregning'. Her er kvantecomputeren måske den meste oplagte udfordring, for hvis det lykkes at gøre den praktisk anvendelig i større stil, kunne den udfordre antagelsen om, at alle beregninger lader sig reducere til den universelle Turing-maskine. Disse bestræbelser på at opnå „quantum supremacy", dvs. bygge en kvantecomputer, der kan løse problemer, en almindelig computer ikke kan (i samme kompeksitet), er helt aktuel forskning i mange forsknings- og udviklingsnetværk, der omfatter private virksomheder som Google og Microsoft og førende universiteter verden over.

Litteratur

Astrachan, Owen (jan. 2003). „Bubble sort. An Archaeological Algorithmic Analysis". *ACM SIGCSE Bulletin*, bd. 35, nr. 1. DOI: 10.1145/792548.611918.

Bell, David Elliott (2011). „Bell–LaPadula Model". I: *Encyclopedia of Cryptography and Security*. Red. af Henk C. A. van Tilborg og Sushil Jajodia. Springer US, s. 74–79. DOI: 10.1007/978-1-4419-5906-5_811.

Bolter, J. David (1986). *Turing's Man: Western Culture in the Computer Age*. London etc.: Penguin Books.

Campbell-Kelly, Martin m.fl., red. (okt. 2003). *The History of Mathematical Tables*. Oxford University Press. DOI: 10.1093/acprof:oso/9780198508410.001.0001.

Copeland, B. Jack (2008). „The Church-Turing Thesis". I: *The Stanford Encyclopedia of Philosophy*. Red. af Edward N. Zalta.

Davis, Martin (1982). *Computability and Unsolvability*. Mineola: Dover.

De Millo, Richard A., Richard J. Lipton og Alan J. Perlis (maj 1979). „Social Processes and Proofs of Theorems and Programs". *Communications of the ACM*, bd. 22, nr. 5, s. 271–280.

Fetzer, James H. (sep. 1988). „Program Verification. The Very Idea". *Communications of the ACM*, bd. 37, nr. 9, s. 1048–1063. DOI: 10.1145/48529.48530.

Galison, Peter (1988). „History, Philosophy, and the Central Metaphor“. *Science in Context*, bd. 2, nr. 1, s. 197–212.

Haigh, Thomas, Mark Priestley og Crispin Rope (2014). „Los Alamos Bets on ENIAC. Nuclear Monte Carlo Simulations, 1947–1948“. *IEEE Annals of the History of Computing*, bd. 36, nr. 3, s. 42–63. DOI: 10.1109/MAHC.2014.40.

Hartmanis, Juris (okt. 1994). „Turing Award Lecture: On Computational Complexity and the Nature of Computer Science“. *Communications of the ACM*, bd. 37, nr. 10, s. 37–43.

Hilbert, David (1983). „On the infinite“. I: *Philosophy of mathematics. Selected readings*. Red. af Paul Benacerraf og Hilary Putnam. 2. udg. Cambridge: Cambridge University Press, s. 183–201.

Hill, Robin K. (2016). „What an Algorithm Is“. *Philosophy & Technology*, bd. 29, nr. 1, s. 35–59. DOI: 10.1007/s13347-014-0184-5.

Hoff Kjeldsen, Tinne (2000). „A Contextualized Historical Analysis of the Kuhn-Tucker Theorem in Nonlinear Programming: The Impact of World War II“. *Historia Mathematica*, bd. 27, s. 331–361.

Høyrup, Jens (1998). *Algebra på lertavler*. Matematiklærerforeningen.

Johansen, Mikkel Willum, Hester Breman og Henrik Kragh Sørensen (sep. 2019). „Human and Computerized Mathematical Practice. Experimental Mathematics and Interactive Theorem Provers“. Under udarbejdelse.

Johansen, Mikkel Willum og Henrik Kragh Sørensen (2014). *Invitation til matematikkens videnskabsteori*. København: Forlaget Samfundslitteratur.

Kahn, David (1996). *The Codebreakers. The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Revideret og opdateret. New York: Scribner.

Knuth, Donald E. (1998). *The Art of Computer Programming*. Bd. 3: *Searching and Sorting*. 2. udg. Reading (Mass) m.m.: Addison-Wesley.

Kuhn, Thomas S. (1995). „Objektivitet, værdidom og valg af teori“. I: *Videnskabens revolutioner*. København: Forlaget Fremad, s. 262–285.

MacKenzie, Donald (aug. 1991). „The fangs of the VIPER“. *Nature*, bd. 352, nr. 6335, s. 467–468. DOI: 10.1038/352467a0.

— (2005). „Computing and the cultures of proving“. *Philosophical Transactions of The Royal Society*. A, bd. 363, s. 2335–2350. DOI: 10.1098/rsta.2005.1649.

Pariser, Eli (2011). *The filter bubble. What the Internet is hiding from you*. New York: Penguin Press.

Roberts, Siobhan (dec. 2018). „The Yoda of Silicon Valley“. *New York Times*.

Seaver, Nick (2017). „Algorithms as culture. Some tactics for the ethnography of algorithmic systems“. *Big Data & Society*, s. 1–12. DOI: 10.1177/2053951717738104.

Shustek, Len (2015). „An Interview with Juris Hartmanis“. *Communications of the ACM*, bd. 58, nr. 4, s. 33–37. DOI: 10.1145/2736346.

Sørensen, Henrik Kragh (2010). „Exploratory experimentation in experimental mathematics. A glimpse at the PSLQ algorithm“. I: *PhiMSAMP. Philosophy of Mathematics: Sociological Aspects and Mathematical Practice*. Red. af Benedikt Löwe og Thomas Müller. Texts in Philosophy 11. London: College Publications, s. 341–360.

— (2013). „Er det matematiske bevis ved at dø ud?“ I: *Fremtiden*. Red. af Ole Høiris. Aarhus: Aarhus Universitetsforlag, s. 99–132.

— (apr. 2016a). *Kildecentreret matematikhistorie i praksis: En kort indføring i matematikhistorisk metode*. RePoSS: Research Publications on Science Studies 38. Aarhus: Centre for Science Studies, University of Aarhus.

Sørensen, Henrik Kragh (2016b). „'The End of Proof'? The integration of different mathematical cultures as experimental mathematics comes of age". I: *Mathematical Cultures. The London Meetings 2012–2014*. Red. af Brendan Larvor. Trends in the history of science. Birkhäuser, s. 139–160. DOI: 10.1007/978-3-319-28582-5_9.

Thomas, Suzanne L., Dawn Nafus og Jamie Sherman (2018). „Algorithms as fetish. Faith and possibility in algorithmic work". *Big Data & Society*, s. 1–11. DOI: 10.1177/2053951717751552.

Turing, A. M. (1936). „On Computable Numbers, with an Application to the Entscheidungsproblem". *Proceedings of the London Mathematical Society (series 2)*, bd. 42, s. 230–265. DOI: 10.1112/plms/s2-42.1.230.

— (2004). „Intelligent Machinery". National Physical Laboratory Report. I: *The Essential Turing. Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life* plus *The Secrets of Enigma*. Red. af B. Jack Copeland. Oxford: Clarendon Press, s. 410–432.

Zuboff, Shoshana (2019). *The age of surveillance capitalism. The fight for a human future at the new frontier of power*. London: Profile Books.