From Proof Theory to Machine Learning

Challenges of Responsible Software and AI

Klaus Mainzer

Emeritus of Excellence
Technical University of Munich

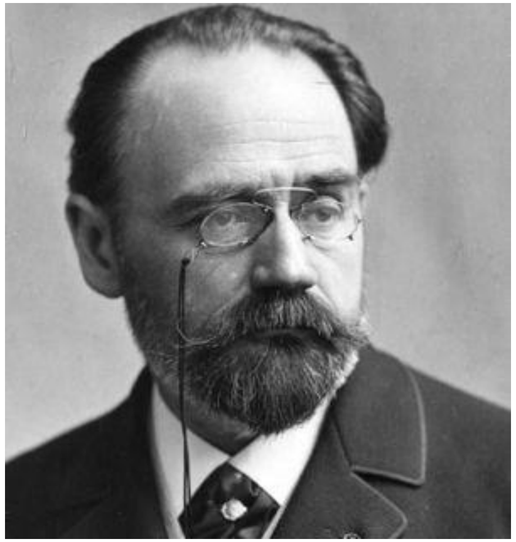
Senior Professor
Eberhard Karls University of Tübingen

G. W. Leibniz: Mathesis Universalis - Verification by Algorithms

In his *"mathesis universalis"* G. W. Leibniz (1646-1716) demanded the *<u>theory of a universal formal language</u>* (*lingua universalis*) to *represent human thinking* by *calculation procedures* (*algorithms*) and to implement them on *mechanical calculating machines*.

***<u>Mathematical theorems</u>* should be verified by "*machines"* (*ad abacos*). But also all kinds of *<u>practical problems</u>* should be *solved* by mechanical procedures for *benefits of mankind*.**

Trust & Provability in Mathematics and Society



Nowadays, ***mathematical arguments*** often have become so ***complicated*** that a ***single mathematician*** rarely can examine them in detail: They trust in the ***expertise of their colleagues***. The situation is similar to ***modern industrial labor world***: According to the French sociologist Emile Durkheim (1858-1917), modern industrial production is so ***complex*** that it must be organized on the ***principle of division of labor and trust in expertise***, but ***nobody*** has the ***total survey***.

On the background of ***critical flaws*** overlooked by the ***scientific community***, Vladimir Voevodsky (1966-2017, IAS Princeton, Fields medal) ***no longer trusted*** in the principle of "job-sharing". Humans could not keep up with the ***ever-increasing complexity of mathematics***. ***Are computers the only solution***? Thus, his foundational program of univalent mathematics is inspired by the idea of a ***proof-checking software*** to ***guarantee trust & verification in mathematics***.

Incorrectness of Programs leads to Catastrophies

Crash of Ariane 5 by software failure 1996

Killed by a machine by massive overdoses of radiation - Therac-25 1985-87

Software failure of Boing 737 Max 2019

Dramatic accidents highlight the dangers of safety-critical systems without software verification .

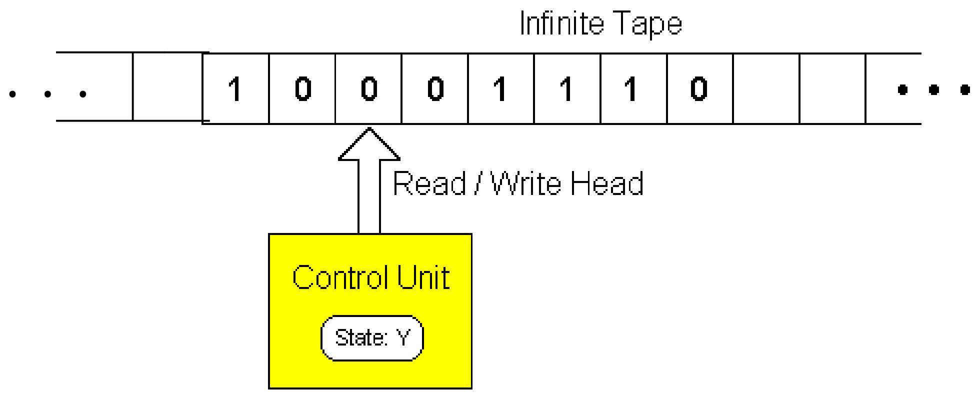
1. **Introduction: Challenges of Artificial Intelligence**
2. **Foundations of Constructive Proof Theory**
3. **From Constructive Proof Theory to Proof Assistants**
4. **Verification in Machine Learning**
5. **Verification and Trust in Mathematics, Computer Science, and Society**

1. Introduction: Challenges of Artifical Intelligence

1.1 From Digital Computer to AI
1.2 Machine Learning and Neural Nets
1.3 Machine Learning and Internet of Things
1.4 From Certification of AI-Programs to Responsible AI

1.1 From Digital Computers to AI

Turing Machine and Computing

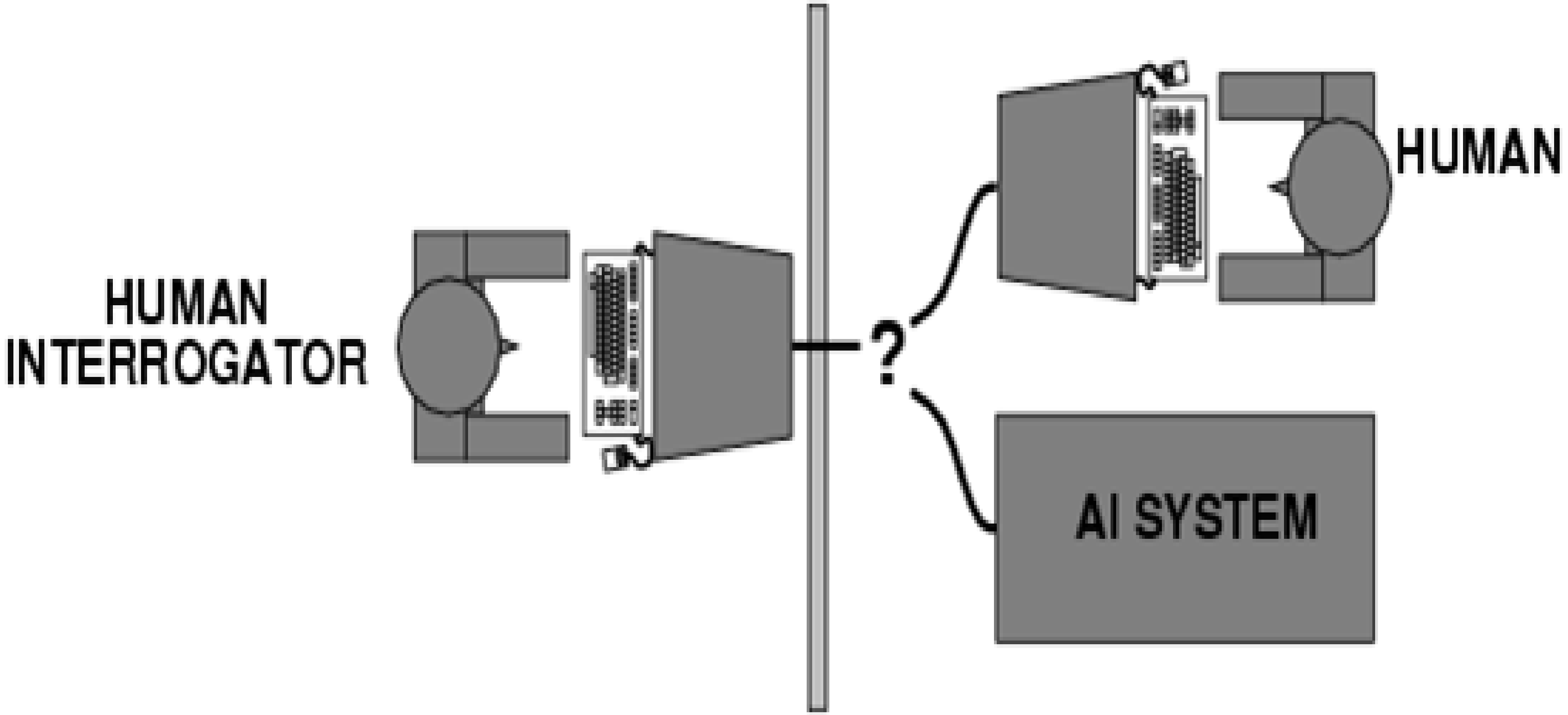


Alan M. Turing
(1912-1954)

***Every algorithm* (*computer program*) can be simulated by a *Turing machine* (*Church's thesis*).**

What is Machine Intelligence ?

Turing Test

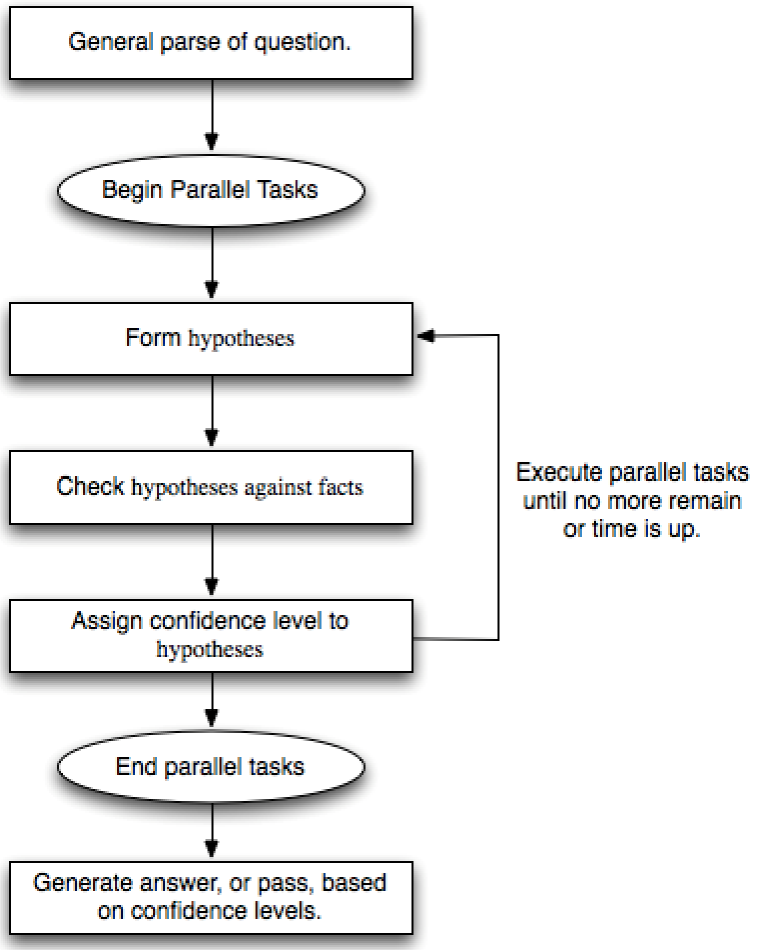


Working Definition of Artificial Intelligence

A system is called *intelligent* iff it can solve *complex problems autonomously* and *efficiently*.

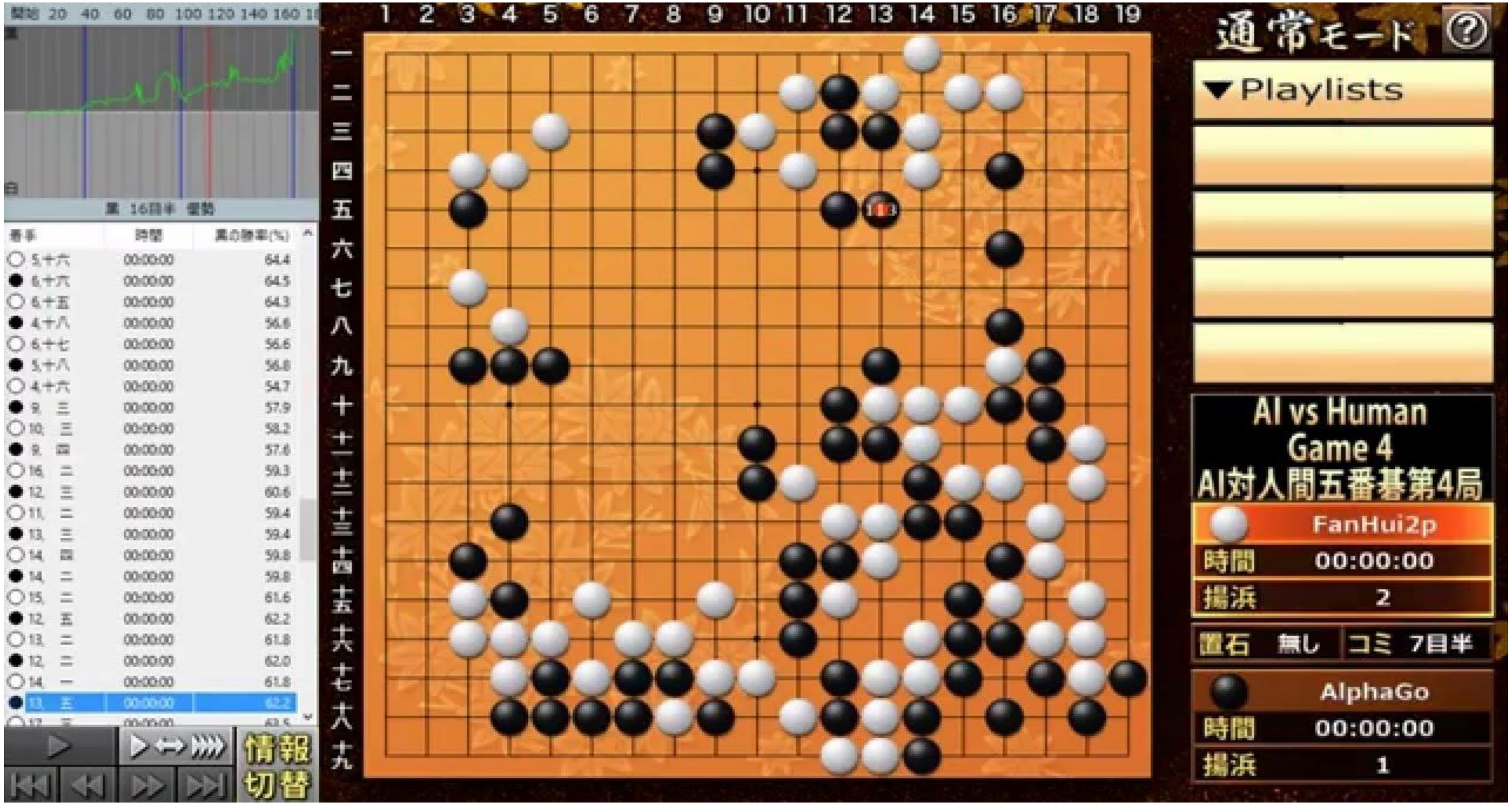
The *degree of intelligence* depends on the *degree of the autonomy* of systems, the *degree of complexity* of problems and the *degree of efficiency* of problem solving procedures.

AI defeats Humans in a Knowledge Quiz



WATSON is a *semantic search machine* (IBM) which can *understand questions* and *answers* in *natural language* by *parallel computing* of *phrases* with *linguistic algorithms* and *probabilities of answers* in *huge data bases*.

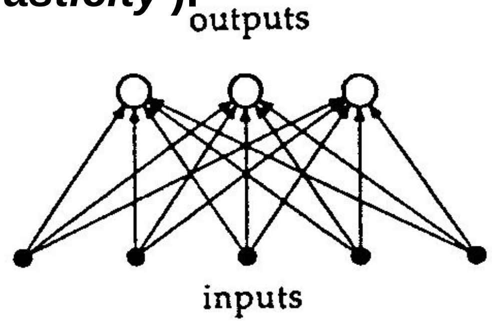
AI learns faster than Humans



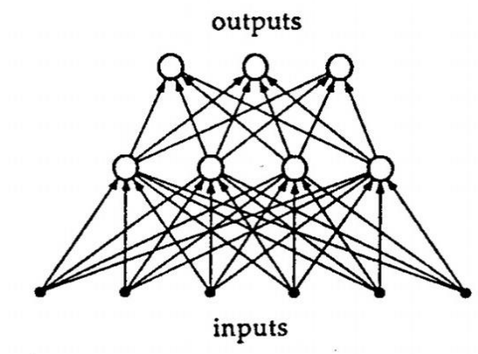
1.2 Machine Learning and Neural Networks

Neural Networks and Learning Algorithms

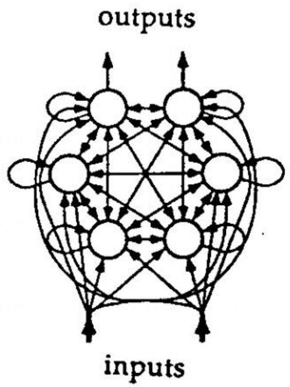
Neural networks **are** ***complex systems*** **of** ***firing*** **and** ***non-firing neurons*** **with** ***topologies*** **like brains. There is no central processor (‚mother cell'), but a** ***self-organizing information flow*** **in cell-assemblies according to rules of synaptic interaction (‚*****synaptic plasticity*****').**



Feedforward with one synaptic layer



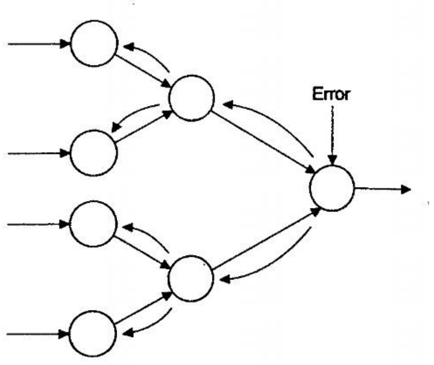
Feedforward with two synaptic layers (Hidden Units)

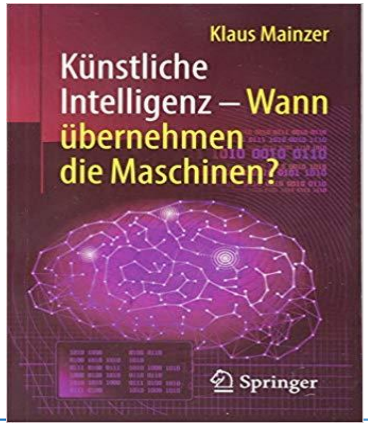


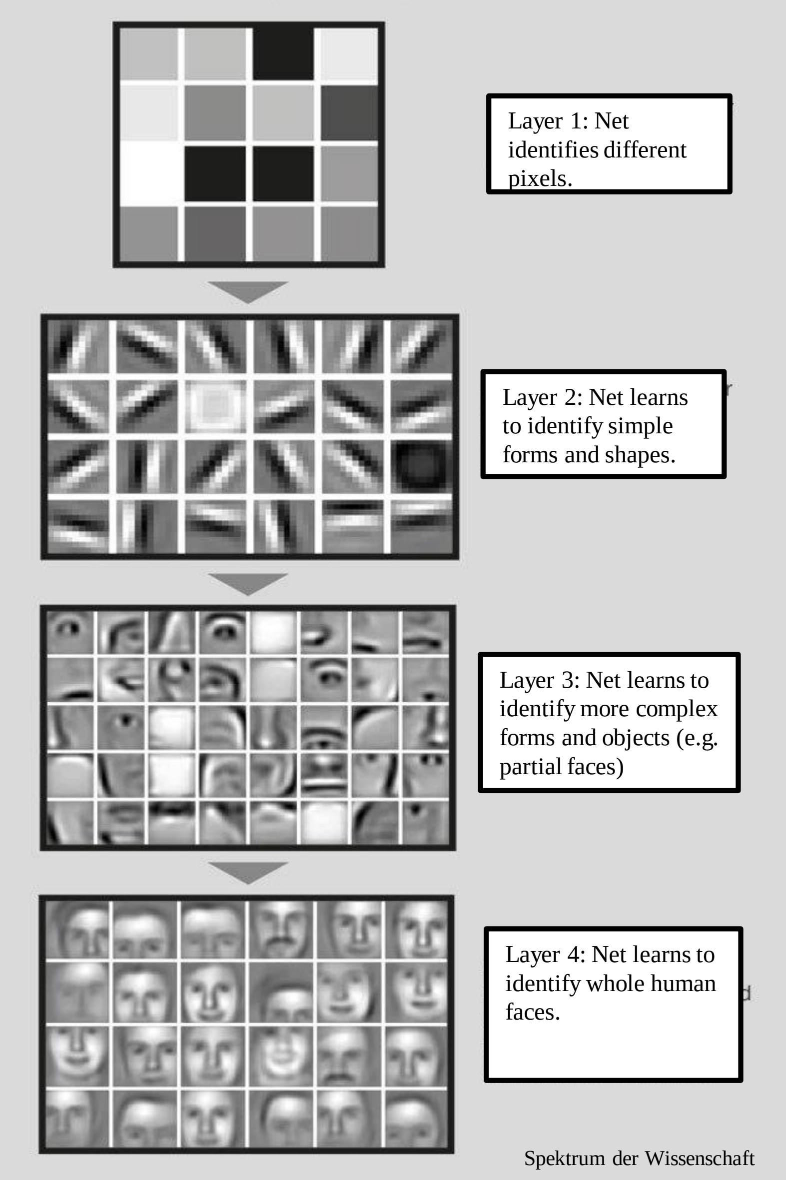
Feedback of recurrent neural network (RNN)

Learning algorithms:

- ***supervised***
- ***non-supervised***
- ***reinforcement***
- ***deep learning***





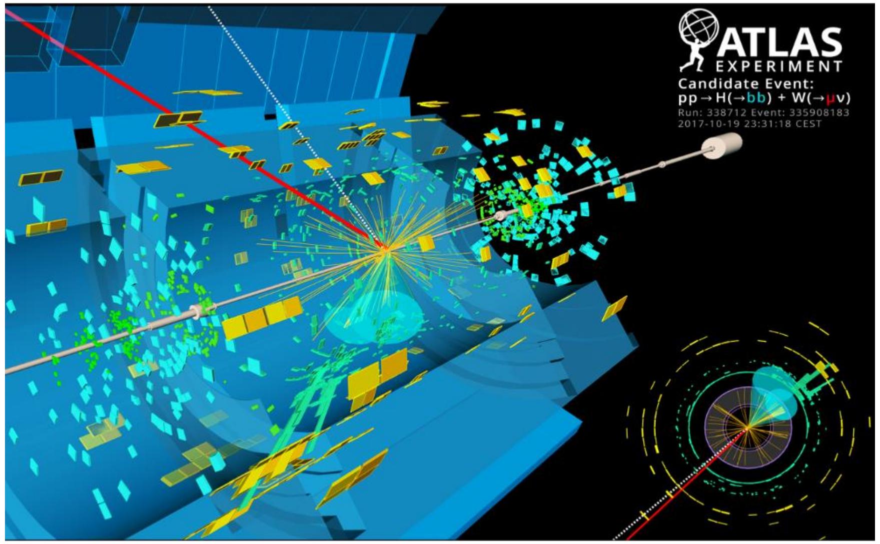


Deep Learning: How Machines learn to learn

Deep learning relates to many-layered neural nets identifying patterns and profiles with increasing complexity **(e.g. human faces). Huge mass of data can be classified into categories.**

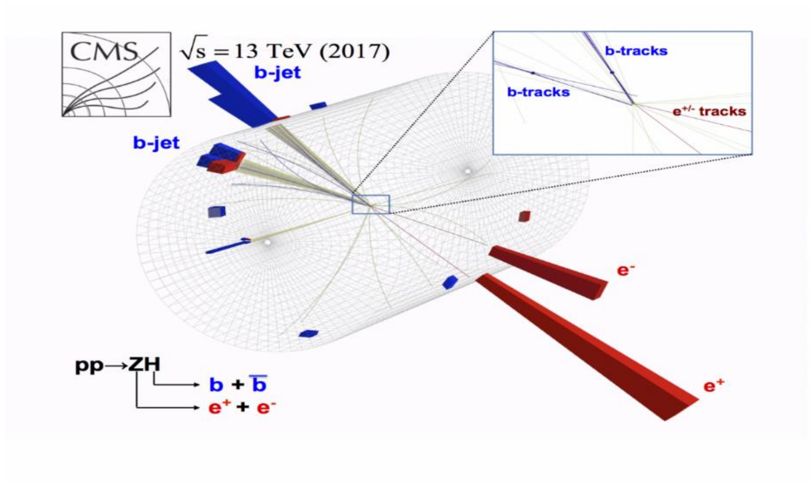
In „*Google Brain* " (Mount View CA 2014), 1 *million neurons* and 1 *billion connections* (synapses) can be simulated. *Big Data technology* enables *neuronal nets* with many (recurrent) *layers* which were only theoretically possible in 1980.

Machine Learning detects Elementary Particles



„The superb LHC (Large Hadron Collider) performance and modern machine learning techniques allowed us to identify the coupling of the Higgs boson to the heaviest fermions – explaining why there is mass in the universe.“

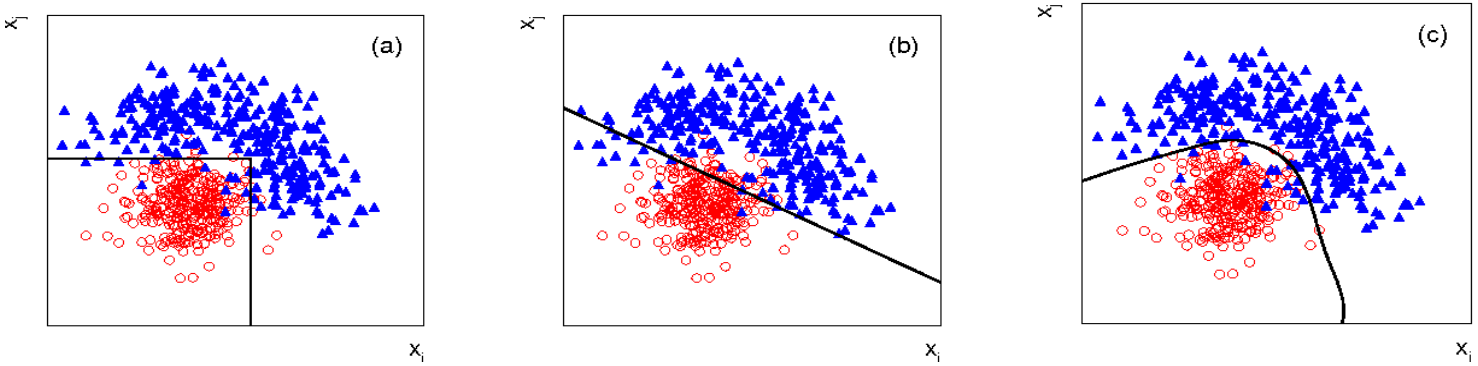
CERN 28 August 2018



The *Standard Model of particle physics predicts* that the *Higgs boson* H decays to two *bottom quarks* b, in association with a Z *boson* decaying to an *electron* e^- and an *antielectron* e^+.

This event must be identified among *billions of data* generated by proton-proton collisions (Big Data).

Pattern Recognition and Classification in Elementary Particle Physics



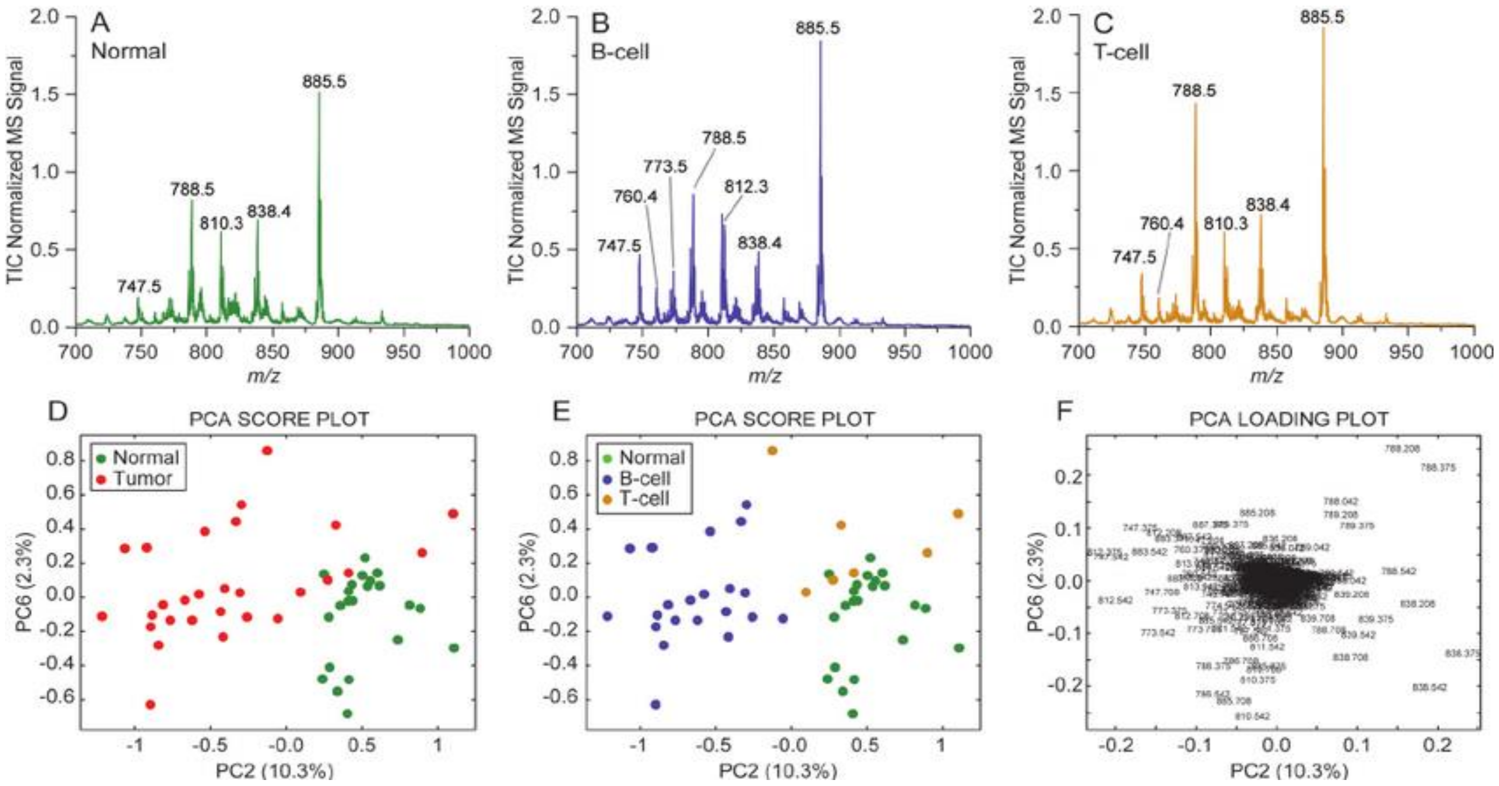
Signal* (s) *events (e.g., Higgs boson decay $H \rightarrow \tau^+\tau^-$) must be distinguished from ***background* (b) *events***.

Vector $\mathrm{x} = (x_1, \ldots, x_n)$ with n quantities of an *event* (e.g., x_1 momentum of a lepton) follows a *joint probability density function* with $f(\mathrm{x}|\mathrm{s})$ for *signal events* and $f(\mathrm{x}|\mathrm{b})$ for *background events*. (The *density* for *signal* and *background events* are indicated by the *red dots* and *blue triangles*, resp.)

***Pattern* („event") *selection* could be based, e.g., on *cuts* (a), *linear boundaries* (b), and *nonlinear boundaries* (c). An *optimal boundary* is provably obtained by using contours of *constant likelihood ratio* $\lambda(\mathrm{x}) = \frac{f(\mathrm{x}|\mathrm{s})}{f(\mathrm{x}|\mathrm{b})}$. As probability densities are in general not known, $\lambda(\mathrm{x})$ is not computable (but finite samples with *training data* by Monte Carlo methods).**

***Machine learning algorithms* should find a *function* $y(\mathrm{x})$ that *best approximates* the *likelihood ratio* $\lambda(\mathrm{x})$ for *pattern selection* of the *signal event*.**

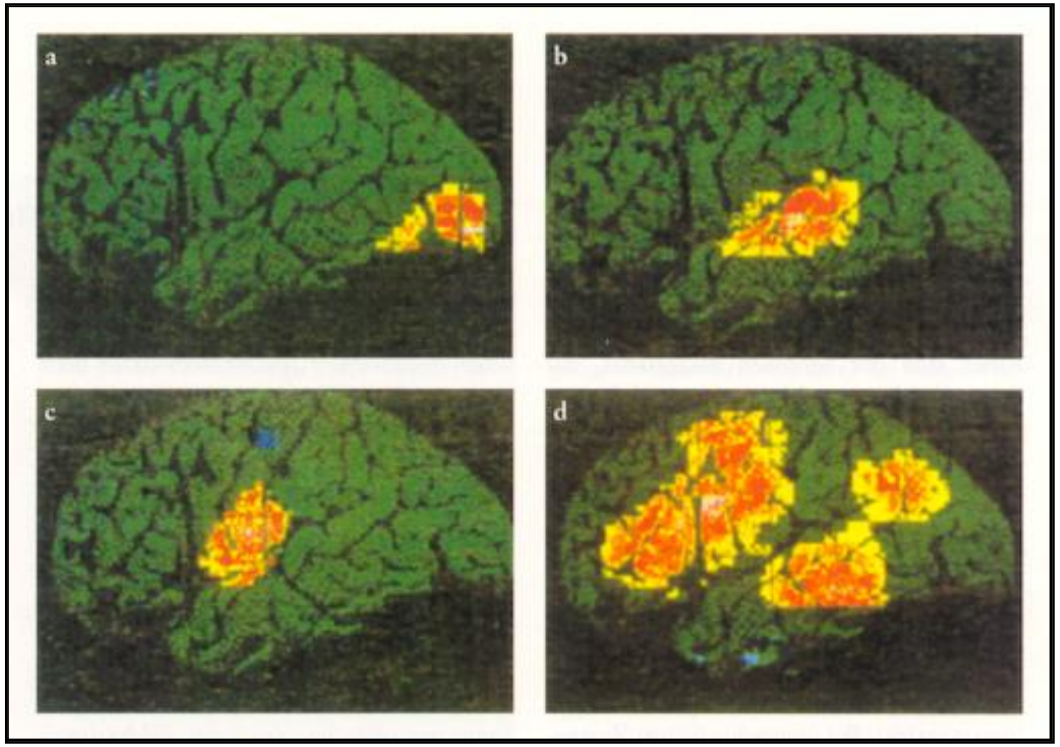
Machine Learning enables Medical Diagnosis



Machine learning (ML) supports *pattern recognition in complex data*: In tissue sections, *normal lymph nodes* are distinguished from *cancer cells* (e.g. breast cancer).

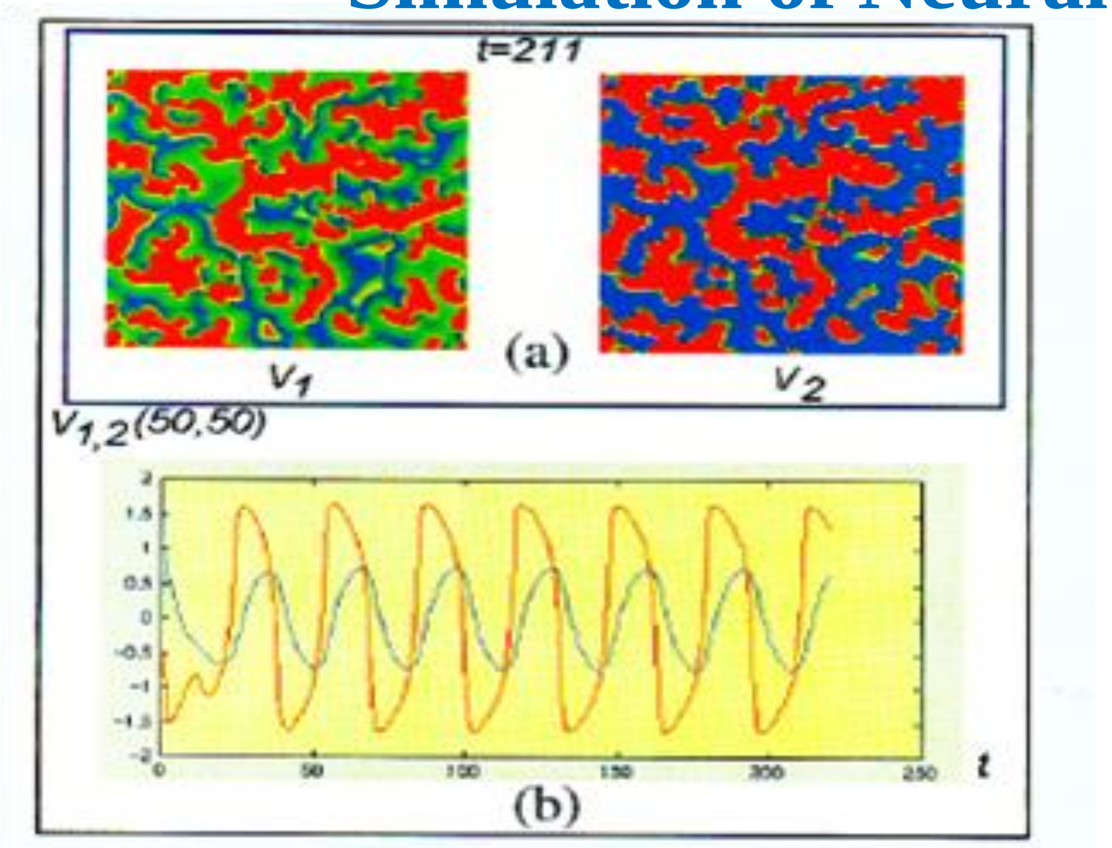
With machine learning, the *pathology in Havard* improved the accuracy from 96% to 99,5 %. IBM *Watson for Genomics* confirmed the diagnosis of physicians in 1018 cases with more than 99% and discovered additional genomic events with great significance.

Pattern Formation in the Human Brain



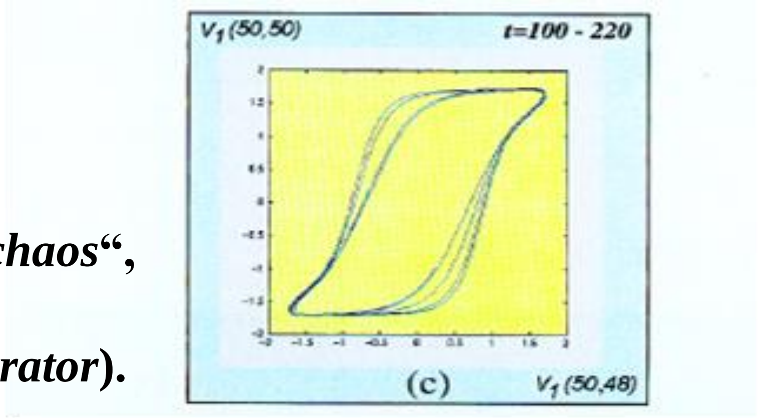
The *brain* is a *complex system* of billions of *firing neurons.* Under appropriate conditions, *neural clusters* fire *synchronously* and organize themselves in *macroscopic patterns,* corresponding to *perceptions, emotions, thoughts,* and *consciousness* (“*Brain Reading*”).

Simulation of Neural Cell Assemblies

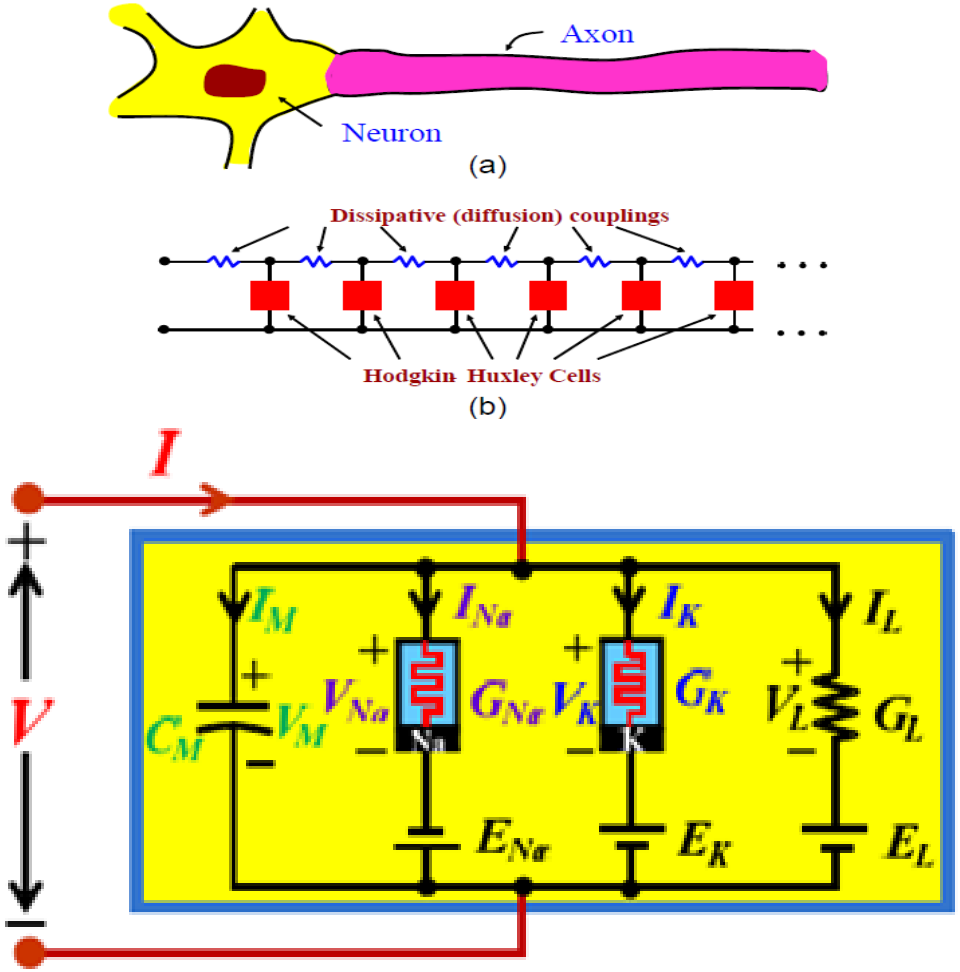


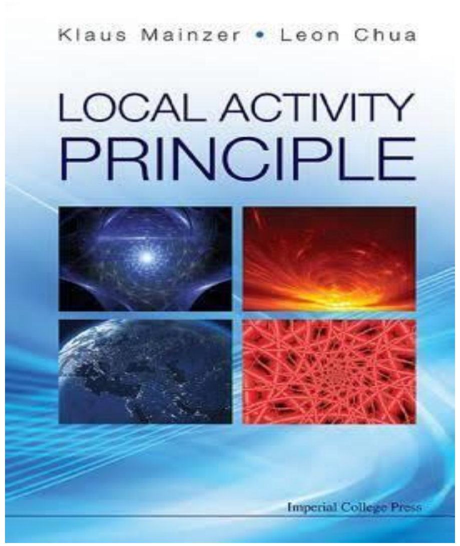
The *input* of a *neuron* can be simulated in *FitzHugh-Naguma equations* (simplification of *Hodgkin-Huxley equations*) by *electrical current*. The *degree of excitation* is denoted with *voltage variable* V_1, the *recovery* by variable V_2.

The location (*j*, *k*) = (50, 50) is situated „*at the edge of chaos*", where *local active* and *stable cells become unstable* and *chaotic* by *dissipative coupling* at *time t* = 211 (*chaos attrator*).



The Computational Brain and Neuromorphic Computers





I external axon membrane current
I_{Na} sodium ion current
I_K potassium ion current
I_L leakage current

E membrane capacitor voltage
E_{Na} sodium ion battery voltage
E_K potassium ion battery voltage
E_L leakage voltage

G_{Na} sodium ion gate (memristor)
G_K potassium ion gate (memristor)

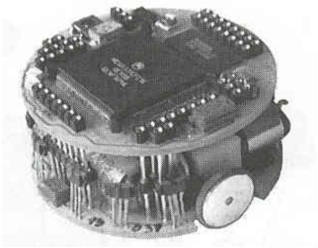
Parameter Explosion in Computational Brain Models

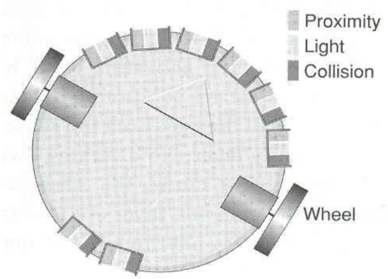
Neural networks **and** ***learning algorithms*** **are** ***mathematical causal models*** **of brain dynamics (K. Mainzer/L. Chua 2013).**

But, the ***parameter explosion*** **(10^{12} neurons with 10^{15} synapses) generates a** ***black box of Big Data*** **which needs** ***explanation*** **of** ***causal interaction*** **between brain regions (e.g., for** ***medical diagnosis*****,** ***psychotherapies*****,** ***legal and ethical questions*** **of accountability and responsibility).**

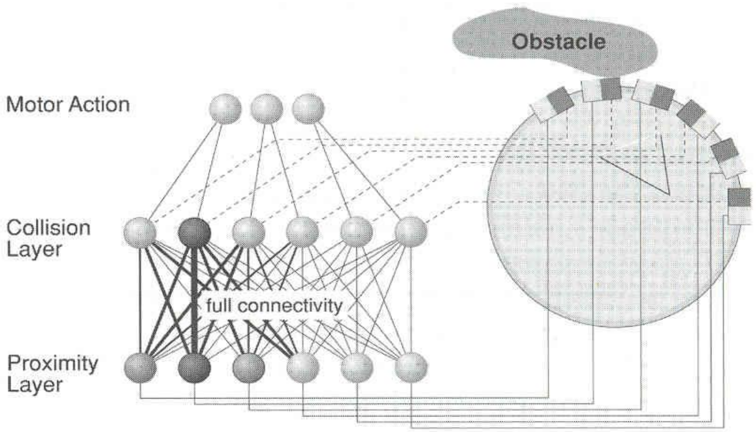
Machine Learning and Autonomous Cars

A simple *robot* with diverse *sensors* (e.g., proximity, light, collision) and *motor equipment* can generate *complex behavior* by a *self-organizing neural network*:





In the case of *collision*, the *connections* between the *active nodes* of *proximity* and *collision layer* are reinforced by *Hebbean learning*: *A behavioral pattern emerges!*



Pfeifer/Scheier 1999

Explosion of Parameters and Big Data generate a Black Box:

"Does your car have any idea why my car pulled it over?"

How many real world accidents **are required to** ***teach machine-learning based autonomous vehicles*****?**

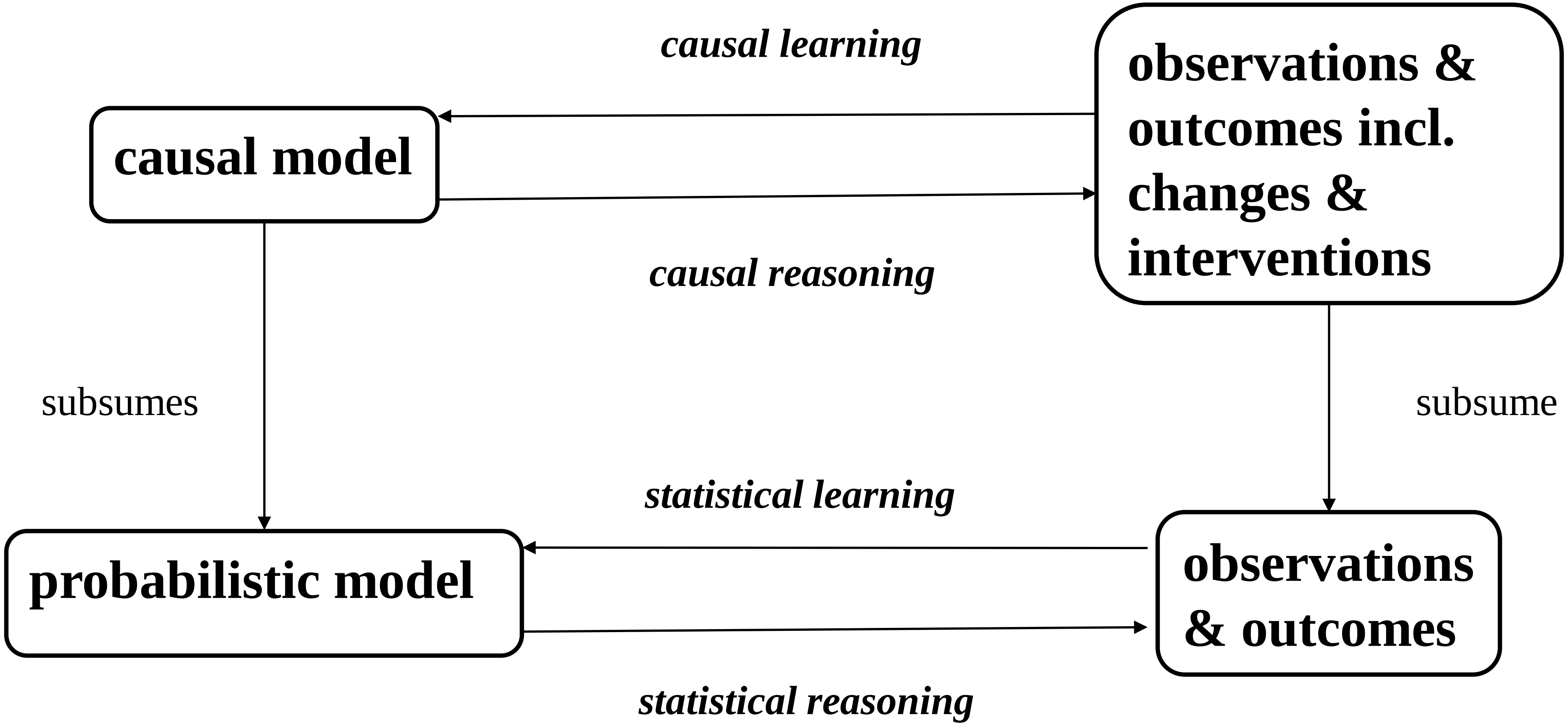
Who should be ***responsible*** **when there is an accident involving autonomous vehicles** ***(ethical and legal challenges)*****?**

We need <u>provability</u>, <u>explainability</u> and <u>accountability</u> of neural networks with <u>causal models</u> !

Blindness of Machine Learning and Big Data

***Without explanation*, big neural networks with *large statistical training data* (Big Data) are *black boxes*. *Statistical data correlations* do not replace *explanations of causes and effects*. Their *evaluation* needs *causal modeling* for answering questions of *accountability and responsibility*.**

Causal Modeling and Machine Learning



Peters et al. 2017, p. 6

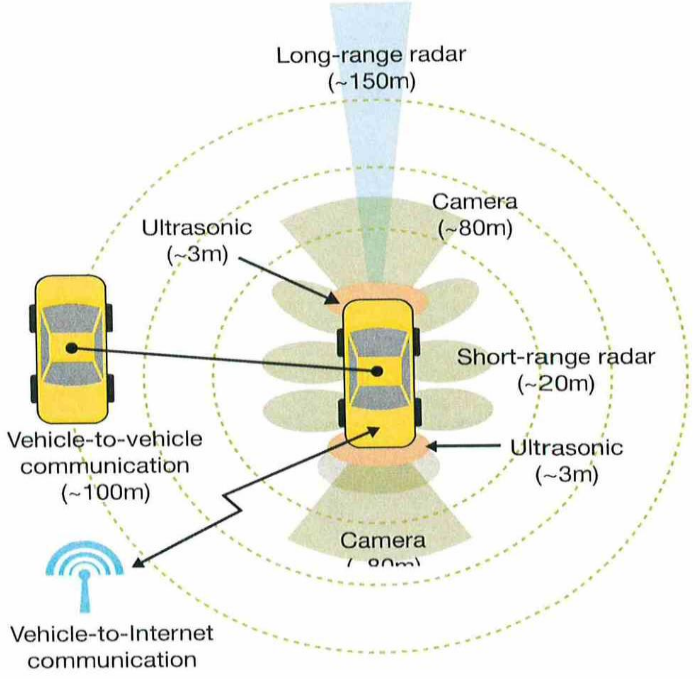
1.3 Machine Learning and Internet of Things

From the Internet to the Internet of Things

Classical Internet **is separated from *physical infrastructures*.**

Internet of Things **observes its *physical environment* by *sensors*, process their *information,* and influence their *environment* with *actuators* according to communication devices.**

Mobility as Intelligent Infrastructure



Networks of mobility with *cloud-based applications support* safe and autonomous driving.

Cars become *mobile systems with sensors* in a *global net* with *swarm intelligence*!

Smart Cities and Infrastructures

Global urbanization **is a challenge of 21st century. Smart cities become self-organizing complex sytems by** ***intelligent technologies*** **and** ***efficient infrastructures*****.**

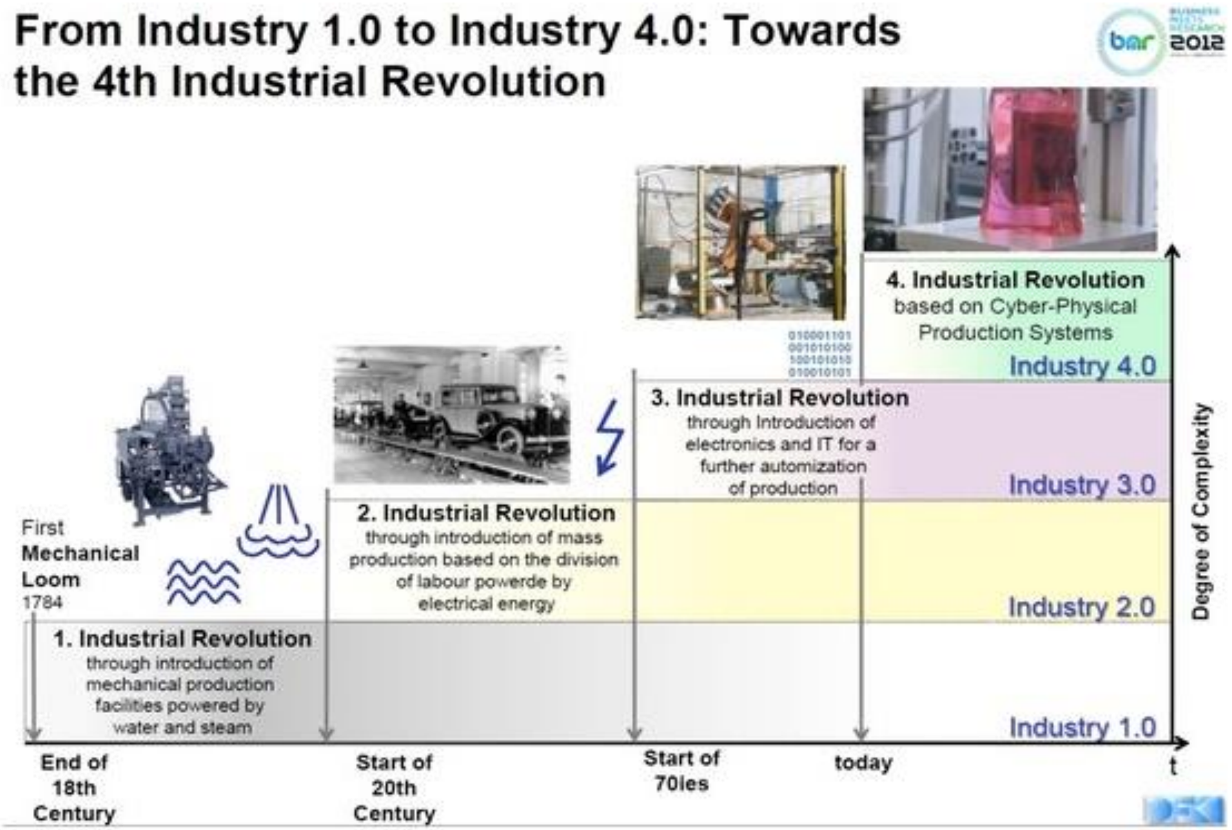
Different domains (e.g., ***civil service, mobility, energy*****- and** ***health system*****) must be integrated by** ***smart technologies*****.**

Smart Grid as Intelligent Infrastructures

Many *energy providers* of *central generators* and *decentralized renewable energy resources* lead to *power delivery networks* with *increasing complexity*.

Smart grids* mean the *integration* of the *power delivery infrastructure* with a *unified communication* and *control network.* It is a *complex information*, *supply* and *delivery system*, *minimizing losses*, *self-healing* and *self-organizing.

Intelligent Infrastructure of Industry



The *1st industrial revolution* introduced the steam engine.

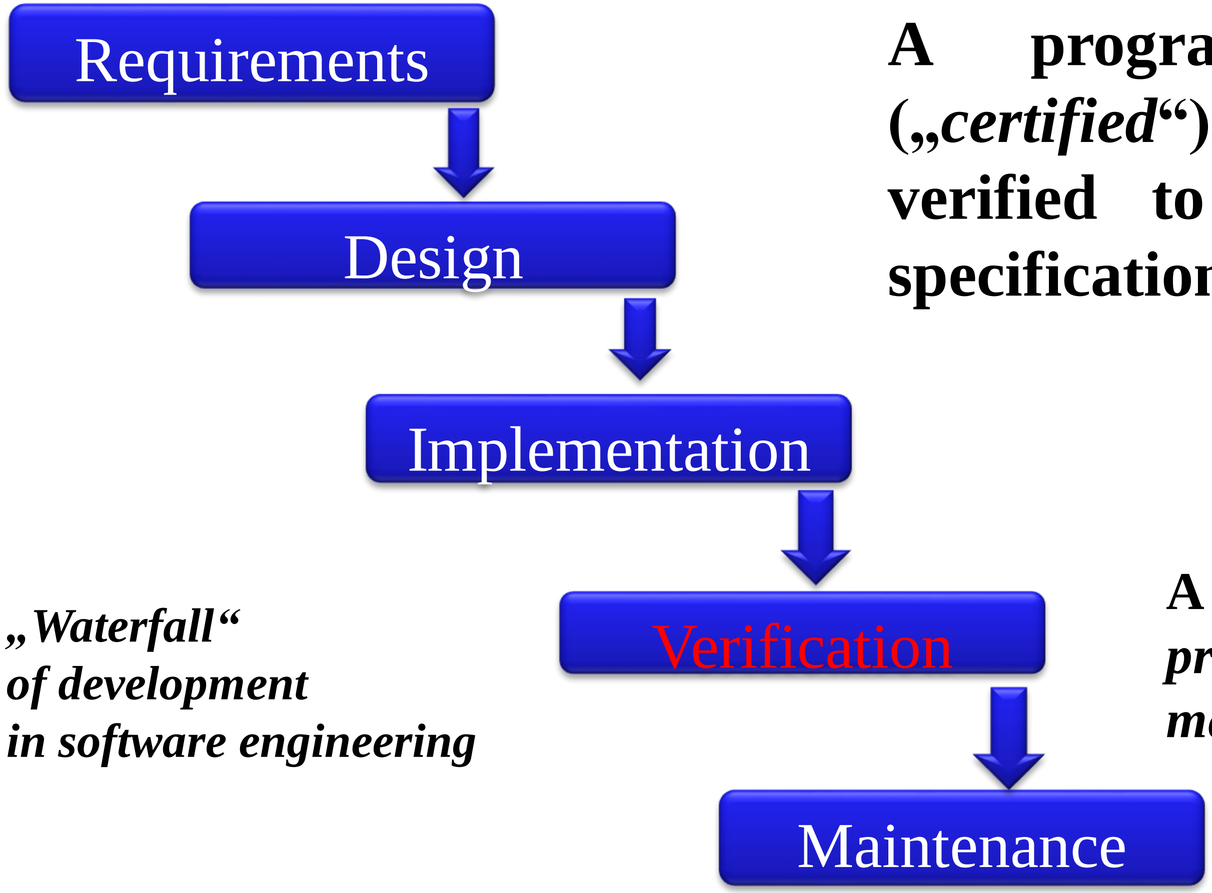
The *2nd industrial revolution* means mass production, divison of labour, and working on the assembly line.

The *3rd industrial revolution* additionally applied industrial robots for further automation of production.

The *4th industrial revolution* changes production on the basis of *internet of Things* (IoT). Production, marketing, and trade are transformed into a more or less *self-organizing complex system*.

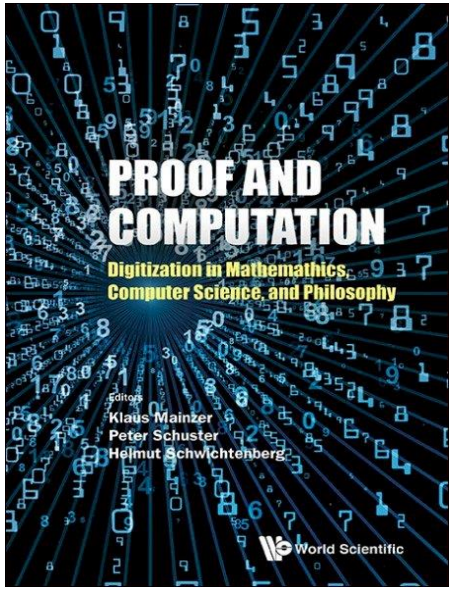
1.4 From Certification of AI-Programs to Responsible AI

Correctness of Certified Programs with Proof Assistants



„Waterfall“
of development
in software engineering

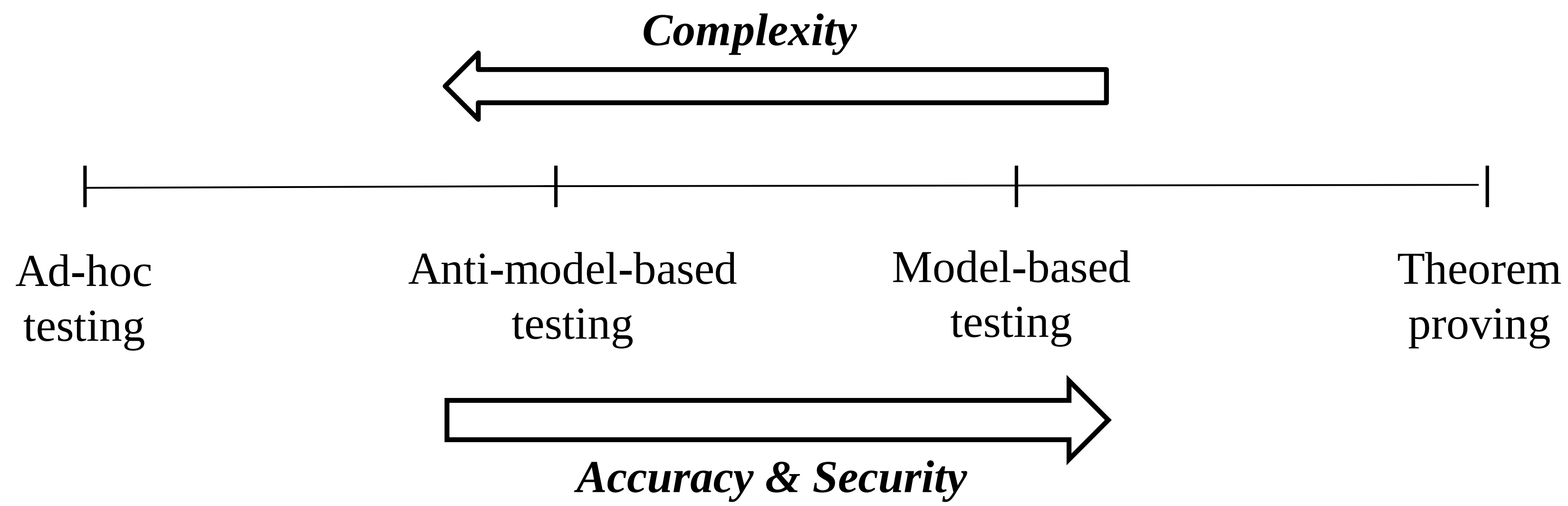
A program is <u>correct</u> („*certified*“) if it can be verified to follow a given specification.



A <u>proof assistant</u> proves the *correctness of a computer program* in a *consistent formalism* like an *<u>exact proof</u> in mathematics* (e.g., Coq, Agda, MinLog, Isabelle).

Therefore, *proof assistants* are the *<u>best formal verification</u>* of *correctness for certified programs*.

Degrees of Certification in Software Testing Research



We must aim at *increasing accuracy*, *security*, and *trust in software* in spite of *increasing complexity* of *civil* and *industrial applications*, but w.r.t. to *costs of testing* (e.g.,utility functions for trade-off time of delivery vs. market value, cost/effectiveness ratio of availability)

Certified AI-Programs and Causal Learning

Statistical machine learning works,
but we can't understand the underlying reasoning.

***Machine learning technique* is akin to *testing*,**
but it is *not enough for safety-critical systems.*

⟹ *Combination* of *causal learning* and *certified AI-programs*

2. Foundations of Constructive Proof Theory

2.1 What are Constructive Proofs ?

Constructivity – Origin and Practice of Mathematics

In *Euclidean geometry*, proofs were supported by *constructions of figures with compass and ruler* rooting in the *practice* of *geodetic* and *astronomic* measurements.

In *Cartesian geometry*, geometric forms were replaced by *coordinates*, *algebraic terms*, and *equations*.

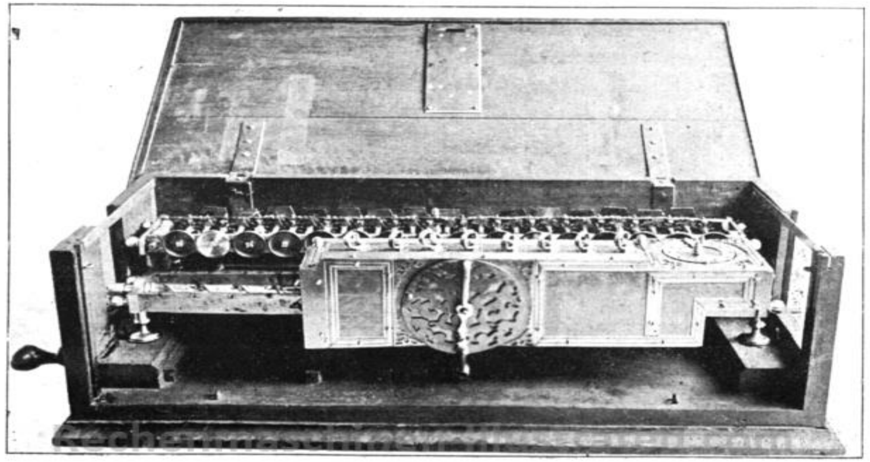
Thus, a <u>*proof of existence*</u> means *constructing* a *geometric figure* or *algebraic solution* in question.

But, what about „<u>*non-constructive*</u>“ <u>*proofs*</u>, in which one proves that something exists by assuming it does not exist, and then deriving a <u>*logical contradiction*</u>, *without* showing a way to *construct* the thing in question?

Computability – Origin and Practice of Mathematics

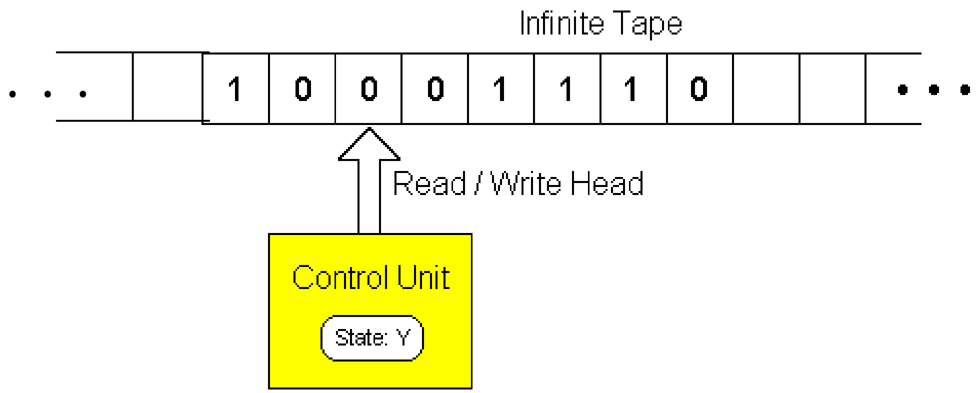
Geometric constructing and _numeric computing_ are the practical roots of mathematics. Since antiquity, _algorithms_ were supported by the _abacus_ and _calculating boards_. The intended _practitioners_ were _businessmen_ and _craftsmen_.

Since the age of mechanization, computing was supported by _calculating machines_ (e.g., Leibniz, Pascal) up to _program-controlled computers_ (e.g., Babbage) in the age of industrialization.



A _proof of existence_ means an _algorithmic solution_ realizable by a _computer_.

Turing Machine and Computing



Alan M. Turing
(1912-1954)

***Every algorithm* (*computer program*) can be simulated by a *Turing machine* (*Church's thesis*).**

Computability of Functions

***A number-theoretical function f* is *computable* (according to *Church's thesis*) if and only if (iff) *f* is *computable* by a *Turing machine* TM.**

i.e. there is a *TM-program stopping* for *numerical inputs* $x_1, \ldots, x_n$ as arguments of a function *f* (e.g., $x_1=3$, $x_2 = 5$ of the additional function $f(x_1, x_2) = x_1 + x_2$) after finitely many steps and *printing* the functional value $f(x_1, \ldots, x_n)$ (e.g. $f(3, 5) = 8$).

Computability and Decidability

For a subset *M* of natural numbers, the *characteristic function* is defined by

$$f_M(x) = \begin{cases} 1, & \text{if } x \text{ element of } M \\ 0, & \text{if } x \textit{ not} \text{ element of } M \end{cases}$$

A *numerical set M* (resp. the corresponding *property* or *predicate*) is *decidable* iff its *characteristic function* f_M is *computable.*

e.g.: The *property* that a natural number is *even* or *not* can be decided by *division* with 2.
Therefore, Leibniz' *ars iudicandi* is made precise by *Turing machines* resp. *computable functions* (according to *Church's thesis* by *μ-recursive functions).*

Computability and Enumerability

How can *solutions of problems* (Leibniz' *ars inveniendi*) be found by *machines*?

A *numerical set M* (resp. the corresponding *property* or *predicate*) is *enumerable* iff there is a *computable function f*, generating its *elements* $f(1)=x_1$, $f(2)=x_2$, …successively for all elements x_1, x_2, … of *M*.

e.g.: The *set* of all *even numbers* is *enumerable* by the *computable function* $f(n) = 2n$ with $f(1) = 2$, $f(2) = 4$, $f(3) = 6$, … for $n = 1, 2, 3, …$

Turing's Non-Computable Real Number

P_1 - . $\underline{z}_{11}$ z_{12} z_{13} z_{14} z_{15} z_{16}

P_2 - . z_{21} $\underline{z}_{22}$ z_{23} z_{24} z_{25}

P_3 - . z_{31} z_{32} $\underline{z}_{33}$ z_{34} z_{35} z_{36}

P_4

P_5 - . z_{51} z_{52} z_{53} z_{54} $\underline{z}_{55}$ z_{56}

⋮ ⋮

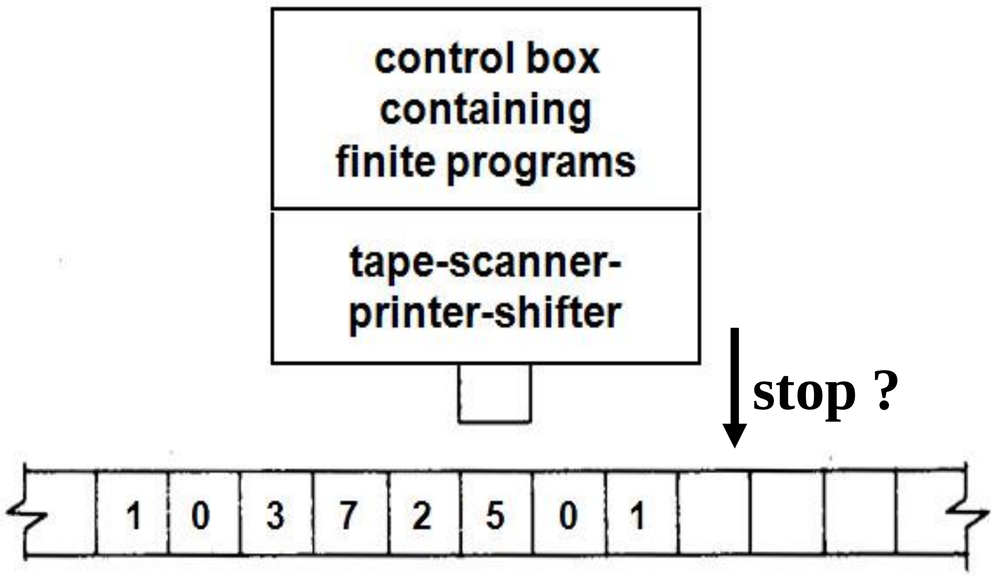
Real numbers like, e.g., π = 3,1415926 … seem to be random, but they can be computed by an *algorithm* (*Turing machine*) step by step. Every *instruction* of a *Turing machine* and the *whole program* can uniquely be *coded* by a *natural number*. We consider a list p_1, p_2, p_3, … of *machine codes* ordered along the sequence of their size.

Behind the machine codes, we note the *development of the decimal fraction* of the *real number* computed by the corresponding *machine* or the line is *empty*. We define a *new development of decimal fraction* consisting of the (underlined) *diagonal values* of the list which we changed (e.g., by addition of 1):

$$-.\, z^*_{11}\, z^*_{22}\, z^*_{33}\, z^*_{55} \ldots$$

By definition, this real number *cannot* be *found* in the *list of computable numbers*. Therefore, it is *not computable*.

Undecidability and Turing's Halting Problem



In principle, there is *no general procedure deciding* if an *arbitrary Turing machine stops* for an *arbitrary input* after finitely many steps or *not* (*halting problem of Turing machines*).

Proof: **Assumed the *halting problem* is *decidable*, then we can confirm if the *n*-th *computer program* (*n* = 1, 2, …) *computes, stops,* and *prints* a *n*-th integer behind the decimal point in *finitely many steps*. In this case, a *real number* which definitely cannot be contained in the *list of computable real numbers* must be *computable*.**

Consequence **: There is *no procedure* which can *check arbitrary computer programs* for *infinite slopes*.**

Incompleteness and Turing‘s Halting Problem

According to Turing, *incompleteness* directly follows from the *undecidability of the Halting problem*: If there is a complete formal system with formal proofs for all mathematical truths, then there is a *procedure* of *deciding* if a *computer program* will *stop* or *not*.

We run through *all possible proofs* until a proof is found that the *program stops* or a proof is found, that it *never will stop*. In that case, it could be *decided* if the computer program would *stop* after *finitely many steps* or *not* – *contrary* to the *undecidability of the Halting problem*.

Hilbert's 10th problem and Turing's Halting problem

Algebraic equations **which involve only** ***multiplication*****,** ***addition*** **and** ***exponentiation*** **of** ***whole numbers*****, are named after the third-century Greak mathematican** ***Diophantos of Alexandria*****. In 1900, David Hilbert asked for an algorithm which will decide whether a** ***diophantine equation*** **has a solution (10th problem of his famous list of 23 problems).**

In 1970, J.V. Matijasevic (V.A. Steklov Institute, St. Petersburg) proved that ***Hilbert's 10th problem*** **is equivalent to** ***Turing's Halting problem*** **and, consequently,** ***not decidable*****. (They used results of M. Davis, H. Putnam and J. Robinson 1961).**

Matijasevic‘s Proof

According to *Lagrange’s representation* of natural numbers as sum of four quadratic whole numbers, *Hilbert’s 10th problem* can be reduced to the existence of solutions in *natural numbers*.

A *predicate D* is called *Diophantine* if it is *definable* by *predicates* $x + y = z$, $x \cdot y = z$, $x^y = z$ and logical operations ∨, ∧, ∃:

$$D(x_1, \dots, x_n) \leftrightarrow \exists y_1, \dots, y_r\ P(x_1, \dots, x_n, y_1, \dots, y_r)$$ **with *P recursive***

$$\leftrightarrow \exists y_1, \dots, y_r\ f_p(x_1, \dots, x_n, y_1, \dots, y_r) = 1$$ **with *computable characteristic function* f_p as *polynom*.**

Obviously, *every Diophantine* predicate is *enumerable*. It can be proven that every *enumerable predicate* is *Diophantine*. (Matijasevic and Cudnovskij used the *Fibonacci sequence* to define an appropriate diophantine predicate.)

The *Halting problem* can be represented by an *enumerable*, but *not decidable predicate*. Therefore, the *corresponding Diophantine predicate* is also not *decidable*.

Intuitionistic Philosophy of Creative Subject

According to Brouwer, *mathematical truth* is founded by *construction of a creative subject* . Following Kant, *mathematical construction* can only be realized in a *finite process, step by step in time* like counting in arithmetic. Thus, for Brouwer, *mathematical truth* depends on *finite stages of realization in time by a creative subject* (in a definition of Kripke and Kreisel 1967) :

The creative subject has a proof of proposition *A* at stage *m* $(\sum \vdash_m A)$ iff

(CS1) For any proposition *A* , $\sum \vdash_m A$ is a *decidable* function of *A* , i.e.

$\forall x \in \mathbb{N}\ (\sum \vdash_x A \vee \neg \sum \vdash_x A)$

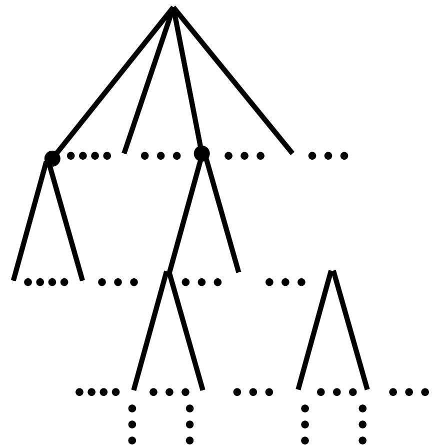
(CS2) $\forall x, y \in \mathbb{N}\ (\sum \vdash_x A \rightarrow (\sum \vdash_{x+y} A)$

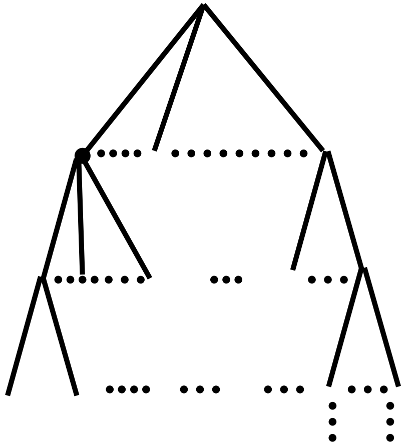
(CS3) $\exists x \in \mathbb{N}\ (\sum \vdash_x A) \leftrightarrow A$

A weaker version of CS3 is G. Kreisel's "*Axiom of Christian Charity*" (1967)

(CC) $\neg\exists x \in \mathbb{N}\ (\sum \vdash_x A) \rightarrow \neg A.$

Intuitionistic Sets of Spreads and Fans





A ***spread*** is the ***intuitionistic*** **analogue of a** ***set*****, because** ***infinite objects*** **are considered as** ***ever growing*** **and** ***never finished*****.**

Therefore, a spread is a ***countably branching tree*** **labelled with** ***natural numbers*** **or other** ***finite objects*** **and containing only** ***infinite paths*****.**

A ***fan*** **is a** ***finitely branching spread.***

A ***branch*** **is an intuitionistic** ***choice sequence*****, i.e. an** ***infinite sequence*** **of numbers (or finite objects) created** ***step by step*** **by a** ***law*** **(algorithm) or** ***without law*** **(e.g., coin). A** ***lawless sequence*** **is ever** ***unfinished*****.**

The only available information about a ***lawless sequence*** **at** ***any stage*** **is the** ***initial segment*** **of the sequence created thus far.**

Fan Principle and Fan Theorem

The *fan principle* states that for every fan *T* in which every *branch* at some point satisfies a property *A*, there is a *uniform bound* on the depth at which this property is met. Such a property is called a *bar* of *T*.

FAN Principle:

$\forall\alpha \in T\ \exists x\ A(\overline{\alpha}(x)) \rightarrow \exists z\ \forall\alpha \in T\ \exists y \leq z\ A(\overline{\alpha}(y))$

with α *choice sequences* and $\overline{\alpha}(x)$ the *initial segment* of α with the first x elements.

FAN Theorem:

Every *continuous real function* on a *closed interval* is *uniformly continuous*.

Proof: ***Fan Principle***

Brouwer-Heyting-Kolmogorov (BHK) Proof Interpretation of the Intuitionistic Logical Constants

BHK interpretation **explains the** ***meaning of logical constants*** **in terms of** ***proof constructions*** **: (Heyting 1934; Kolmogorov 1932; Kohlenbach 2008)**

i. There is *no proof* for $\perp$.

ii. A *proof* of $A \wedge B$ is a pair (*q,r*) of proofs, where *q* is a proof of *A* and *r* is a *proof* of *B*.

iii. A *proof* of $A \vee B$ is a pair of (*n,q*) consisting of an *integer n* and a *proof q* which proves *A* if $n = 0$ and resp. *B* if $n \neq 0$.

iv. A *proof* *p* of $A \rightarrow B$ is a *construction* which *transforms* any *hypothetical proof q* of *A* into a *proof* *p*(*q*) of *B*.

v. A *proof* *p* of $\forall x A(x)$ is a *construction* which *produces* for *every construction* c_d of an *element d* of the *domain* a proof $p(c_d)$ of *A*(*d*).

vi. A *proof* of $\exists x A(x)$ is a pair (c_d, q), where c_d is the *construction* of an *element* *d* of the *domain* and *q* is a *proof* of *A*(*d*).

Computable Functionals and Constructive Proofs

The *disadvantage* of the *BHK-interpretation* is the *unexplained notion* of *construction* resp. *constructive proof*. K. Gödel wanted that *constructive proofs* of *existential theorems* provide *explicit realizers*. Therefore, he replaced the notion of *constructive proof* by the more definite and less abstract concept of *computable functionals* of *finite type*.

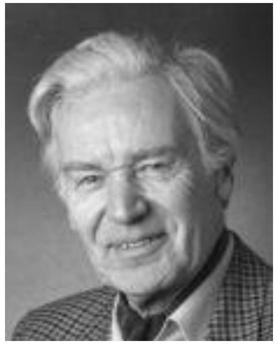
But *Gödel's proof interpretation* is largely *independent* of a *precise definition* of *computable functionals* : One only needs certain basic functionals as computable (e.g., primitive recursion in finite types) and their closure under composition.

Following Gödel, every *formula A* is assigned with the *existential formula* $\exists x A_1(x)$ with $A_1(x)$ $\exists$-free. Then, a realizing term *r* with $A_1(r)$ must be *extracted* from a *derivation of A* (‚*Dialectica-Interpretation*' 1958)

2.2 Basics of Constructive, Classical, and Intuitionistic Mathematics

Constructive Mathematics with Classical Logic

H. Weyl (1885-1955)



P. Lorenzen (1915-1994)

In "*Differential and Integral*" (1964), Lorenzen used Weyl's technique in "*Das Kontinuum*" (1918) to develop a *predicative analysis*, which can *reconstruct classical analysis* with the *principle of excluded middle* as far as analysis is *constructively* founded.

The *set of natural numbers* is given by *inductive construction* of *terms* /, //, … .

Constructive sets and *functions* are *abstractions* of *inductively defined terms* (e.g. variables $s, t, \dots, s+t, s \cdot t$) resp. *formulas* (e.g., $s^2 > 1, \exists r\, r < s$):

A *set M* is *inductively defined* by the equivalences

$$1, x_1, \dots, x_m \in M \leftrightarrow A(x_1, \dots, x_m)$$
$$n+1, x_1, \dots, x_m \in M \leftrightarrow B_M(n+1, x_1, \dots, x_m)$$

if the formula $A(x_1, \dots, x_m)$ does not contain the symbol M and the formula $B_M(n+1, x_1, \dots, x_m)$ may contain partial formulas $s, t_1, \dots, t_m \in M$ (with terms $s, t_1, \dots, t_m$), but only such that $s < n+1$.

Induction Principle of Predicative Analysis

The *induction definition* can be contracted in a *comprehension scheme*:

$$n, x_1, \ldots, x_m \in M \leftrightarrow A_M(n, x_1, \ldots, x_m)$$

with formula $A_M(n, x_1, \ldots, x_m)$ which only contains symbol M in partial formulas $s, t_1, \ldots, t_m \in M$ (with terms $s, t_1, \ldots, t_m$), but only such that $s < n$.

Starting with the *construction of natural numbers, further constructive objects* are generated by *inductive construction of terms and formulas* about *already constructed objects*:

Example: *Real numbers*

Definition (*Equivalence of Cauchy sequences*): $(r_n) \sim (s_n) \equiv (r_n) - (s_n)$ null sequence

If $A((t_n))$ is an *invariant formula* about (t_n) with $(r_n) \sim (s_n) \wedge A((r_n)) \rightarrow A((s_n))$, then write $A(\lim_{n\to\infty} t_n)$ with *term* $\lim_{n\to\infty} t_n$ *of real numbers*.

Constructive Mathematics with Intuitionistic Logic

E. Bishop (1928-1983)

In *Foundations of Constructive Analysis* (1967), Bishop could prove most of the important theorems of *real analysis* with *constructive methods without contradicting classical mathematics* as Brouwer's *intuitionistic mathematics* did.

***Natural numbers* are given as *fundamental construction* of the *human mind* (Kant, Kronecker, Brouwer).**

- **A *constructive set* M is defined by a *rule to construct an element of* M in *finite steps*, by a method to *prove* that two elements of M are *equal*, and a *proof* that this equality $=_M$ is an *equivalence relation*.**
- **A *constructive function* $f: M \to N$ is a *rule* which associates *an element* $b \equiv f(a)$ of a *set* N to *each element* a of a *set* M, in such a way that b can be found by a *finite routine* when a is given. *Equal elements* of N must be associated to *equal elements* of M.**

The Real Number System of Constructive Analysis

In Bishop‘s constructive analysis, *rationals* are given as expressions p/q with integers p,q and $q \neq 0$. A *sequence of rational numbers* is a *rule* which associates to each positive integer n a rational number r_n.

A *sequence* (r_n) *of rational numbers* is *regular* iff

$|r_m - r_n| \leq m^{-1} + n^{-1}$ for all positive integers m, n.

A *real number* is a *regular sequence of rational numbers*.

Two *real numbers* $x \equiv (r_n)$ and y $\equiv (s_n)$ are *equal* iff $|r_n - s_n| \leq 2n^{-1}$ for all positive integers n.

Notice that Bishop‘s *constructive real numbers* are *no equivalence classes*, but identified with *regular sequences of rational numbers*.

Bishop‘s Influence on Proof Systems

In 1985, Robert Constable acknowledged the *influence of Bishop* on the design of NuPrl designed to „*execute constructive proofs*“ by extracting programs from proofs:

„Shortly after we had executed our first constructive proof, I wrote to Bishop informing him of what I took to be an historic event. I told him how much his writings and his encouragement had meant to us on the long road to this accomplishment. I was crushed to receive my letter back unopened, marked „recipient deceased“.“

2.3 Basics of Reverse Mathematics

Reverse Mathematics in Antiquity

Since Euclid (Mid-4th century – Mid 3rd century BC), *axiomatic mathematics* has started with *axioms* to *deduce* a *theorem*. But the "*forward*" *procedure* from *axioms* to *theorems* is not always obvious. How can we *find appropriate axioms* for a proof starting with a *given theorem* in a „*backward*" (*reverse*) *procedure*?

Pappos of Alexandria (290-350 AC) called the "*forward*" *procedure* as "*synthesis*" with respect to Euclid's *logical deductions* from *axioms* of geometry and *geometric constructions* (Greek: "*synthesis*") of corresponding figures. The *reverse search procedure of axioms* for a given theorem was called "*analysis*" with respect to *decomposing* a *theorem* in its *necessary* and *sufficient conditions* and the *decomposition* of the *corresponding figure* in its *building blocks*.

Classical Reverse Mathematics

Reverse mathematics **is a modern** ***research program*** **to determine the** ***minimal axiomatic system*** **required to** ***prove theorems*****. In general, it is** ***not possible*** **to start from a** ***theorem*** τ **to prove a** ***whole axiomatic subsystem*** T_1**. A** ***weak base theory*** T_2 **is required to** ***supplement*** τ**:**

If $T_2 + \tau$ **can prove** T_1**, this** ***proof*** **is called a** ***reversal*****.**
If T_1 ***proves*** τ **and** $T_2 + \tau$ **is a** ***reversal*****, then** T_1 **and** τ **are said to be** ***equivalent over*** T_2**.**

Reverse mathematics **allows to determine the** ***proof-theoretic strength*** **resp.** ***complexity*** **of** ***theorems*** **by** ***classifying*** **them with respect to** ***equivalent theorems*** **and** ***proofs*****. Many** ***theorems*** **of** ***classical mathematics*** **can be** ***classified*** **by** ***subsystems*** **of** ***second-order arithmetic*** $\mathbb{Z}_2$ **with** ***variables*** **of** ***natural numbers*** **and** ***variables*** **of** ***sets of natural numbers*****.**

The Subsystems of Second-Order Arithmetics $\mathcal{Z}_2$

Arithmetical formulas **can be *classified* according to the *arithmetical hierarchy* $\sum_n^0, \prod_n^0$, and Δ_n^0. We can distinguish $\sum_n^0, \prod_n^0$, and Δ_n^0- schemas of *induction* and *comprehension*. That is also possible for the *analytical hierarchy* $\sum_n^1, \prod_n^1$, and Δ_n^1**

A *structure* of an (*arithmetical*) *set M* defines its *variables* and *non-logical symbols* (constants, operations) satisfying relations between *variables* : e.g., $\mathbb{Q} = (M, +_{\mathbb{Q}}, -_{\mathbb{Q}}, \cdot_{\mathbb{Q}}, 0_{\mathbb{Q}}, I_{\mathbb{Q}}, <_{\mathbb{Q}}, =_{\mathbb{Q}})$ *structure* of *rational numbers*.

A *model* of a *set* of (*arithmetical*) *formulas* is a *structure* with the *same non-logical symbols* and all *formulas* in the set are in the *model* as well.

The *arithmetical* and *analytical hierarchies* yield *classifications* of *axiomatic subsystems* of $\mathcal{Z}_2$ with *increasing proof-theoretic power* and corresponding *structures* of $\mathcal{Z}_2$-models.

$\mathcal{Z}_2$- Subsystems and Philosophical Research Programs

The *five* most commonly used $\mathcal{Z}_2$ - *subsystems* in *reverse mathematics* correspond to *philosophical programs* in *foundations of mathematics* with *increasing proof-theoretic power* starting with the *weakest* RCA_0-*subsystem* .

RCA_0: *Turing's computability*
WKL_0: *Hilbert's finitistic reductionism*
ACA_0: *Weyl's & Lorenzen's predicativity*
ATR_0: *Friedman's & Simpson's predicative reductionism*
$\prod_1^1 - CA_0$: *impredicativity*

$\Delta_1^1 - CA_0$ yields *systems* of *hyperarithmetic analysis* (Feferman et al.) with Δ_1^1-*predicativism*:

T is a *theory* of *hyperarithmetic analysis* iff

i. its ω-*models* are *closed* under *joins* and *hyperarithmetic reducibility*

ii. it *holds* in $\mathrm{HYP}(x)$ for all *x*

Constructive Reverse Mathematics

Classical reverse mathematics **(Friedmann/Simpson) uses *classical logic* and *classification of proof-theoretic strength* with RCA_0 (Δ_1^0*-recursive comprehension*) as *weak subsystem*.**

Constructive reverse mathematics **(Ishihara et al.) uses *intuitionistic logic* and *Bishop's constructive mathematics* (BISH) as *weak subsystem* of a *constructive classification* :**

BISH = $\mathcal{Z}_2$ + *Intuitionistic Logic* + *Axioms of Countable*, *Dependent* and Unique Choice

Intuitionistic Mathematics (Brouwer, Heyting et al.):

INT = BISH + *Axiom of Continuous Choice* + *Fan Theorem*

Constructive Recursive Mathematics (Markov et al.):

RUSS = BISH + *Markov's Principle* + *Church's Thesis*

Classical Mathematics (Hilbert et al.):

CLASS = BISH + *Principle of Excluded Middle* + *Full Axiom of Choice*

Bishop's Constructive Mathematics BISH

BISH is an *informal mathematics* with *intuitionistic logic* and *function existence axioms* :

Axiom of Countable Choice:

$$\forall n \in \mathbb{N}\ \exists x \in X\ A(n,x) \rightarrow \exists f \in X^{\mathbb{N}} \forall n \in \mathbb{N}\ A(n, f(n))$$

Axiom of Dependent Choice:

$$\forall x \in X\ \exists y \in X\ A(x,y) \rightarrow \forall x \in X\ \exists f \in X^{\mathbb{N}}(f(0) = x \wedge \forall n \in \mathbb{N}\ A(f(n), f(n+1)))$$

Axiom of Unique Choice:

$$\forall x \in X\ \exists!\, y \in Y\ A(x,y) \rightarrow \exists f \in Y^{X}\ \forall x \in X\ A(x, f(x))$$

Bishop's *constructive* (forward) *mathematics* (BISH) intends to find a *constructive substitute A'* for a *classical theorem A* such that

BISH $\vdash A'$ and CLASS $\vdash A \leftrightarrow A'$

When *A* and *A'* are *not equivalent* in BISH, *A* can sometimes be shown to do *not admit a constructive proof* by giving a "*Brouwerian counterexample P*" to *A* such that

BISH $\vdash A \rightarrow P$ and BISH $\nvdash P$

Markov's Constructive Recursive Mathematics (RUSS)

RUSS is *Bishop's constructive mathematics* (BISH) with *Markov's principle* and *Church's thesis* :

The *following* are *equivalent* in BISH:

(1) Markov's principle (MP):
$\forall \alpha \in \mathbb{N}^{\mathbb{N}}(\neg\neg\exists n(\alpha(n) \neq 0) \rightarrow \exists n(\alpha(n) \neq 0)$

(2) $\forall x \in \mathbb{R}(\neg\neg(0 < x) \rightarrow 0 < x)$

Remark: MP is an instance of the *double negation elemination* $\neg\neg P \rightarrow P$ which is rejected in INT, but accepted in RUSS.

MP is *weaker* than LPO. The *following are equivalent* with the weak Markov principle:

(1) Weak Markov's principle (WMP):
$\forall \alpha \in \mathbb{N}^{\mathbb{N}}(\forall \beta \in \mathbb{N}^{\mathbb{N}}(\neg\neg\exists n\beta(n) \neq 0 \vee \neg\neg\exists n(\alpha(n) \neq 0 \wedge \beta(n) \neq 0) \rightarrow \exists n\, \alpha(n) \neq 0$

(2) $\forall x \in \mathbb{R}(\forall y \mathbb{R}\ \neg\neg(0 < y) \vee \neg\neg(y < x) \rightarrow 0 < x)$

2.4 Basics of Intuitionistic Type Theory

Curry-Howard Correspondence

In 1969, the logician W.A. Howard observed that Gentzen's *proof system of natural deduction* can be directly interpreted in its *intuitionistic version* as a *typed variant* of the mode of *computation* known as *lambda calculus*.

According to Church, $\lambda a.b$ means a *function* mapping an element a onto the function value b with $\lambda a.b[a] = b$. In the following, *proofs* are represented by terms $a, b, c, \dots$; *propositions* are represented by $A, B, C, \dots$.

Examples:

$$\dfrac{\lambda a(\lambda b.a) \quad \begin{matrix}[A]\\ \vdots\\ B \rightarrow A\end{matrix}}{(\rightarrow \mathrm{I}) \quad A \rightarrow (B \rightarrow A)}$$

$$\dfrac{\lambda a.b \quad \begin{matrix}[A]\\ \vdots\\ B\end{matrix}}{(\rightarrow \mathrm{I}) \quad A \rightarrow B}$$

A proof is a program, and the formula it proves is the type for the program.

Gentzen‘s Sequent Calulus and Lambda Calculus

Intuitionistic sequent calculus	Lambda calculus type assignment rules
$\dfrac{}{\Gamma_1,\alpha,\Gamma_2\vdash\alpha}$ Ax	$\dfrac{}{\Gamma_1, x{:}\,\alpha,\ \Gamma_2 \vdash x{:}\,\alpha}$
$\dfrac{\Gamma,\alpha\vdash\beta}{\Gamma\vdash\alpha\to\beta} \to I$	$\dfrac{\Gamma, x{:}\,\alpha \vdash t{:}\,\beta}{\Gamma \vdash \lambda x.\,t{:}\,\alpha \to \beta}$
$\dfrac{\Gamma\vdash\alpha\to\beta \quad \Gamma\vdash\alpha}{\Gamma\vdash\beta} \to E$	$\dfrac{\Gamma \vdash t{:}\,\alpha \to \beta \quad \Gamma \vdash u{:}\,\alpha}{\Gamma \vdash tu{:}\,\beta}$

Proving $\Gamma \vdash \alpha$ means having a *program* that, given values with the *types* listed in Γ, manufactures an *object of type α*. An *axiom* corresponds to the *introduction of a new variable* with a new, unconstrained *type*, the → *I* rule corresponds to *function abstraction* and the → *E* rule corresponds to *function application*.

$t{:}\,\alpha$ means „*t proves α*“ as well as „*t is of type α*“.

Propositions as Types in Intuitionistic Type Theory

$$\bot = \emptyset$$
$$\top = 1$$
$$A \vee B = A + B$$
$$A \wedge B = A \times B$$
$$A \supset B = A \rightarrow B$$
$$\exists x{:}A.B = \Sigma x{:}A.B$$
$$\forall x{:}A.B = \Pi x{:}A.B$$

According to the *Curry-Howard interpretation* of *propositions as types*, $\Sigma x{:}A.B$ is the *disjoint sum* of the A-indexed family of types B and $\Pi x{:}A.B$ is its *cartesian product*.

The canonical elements of $\Sigma x{:}A.\,B$ are *pairs* (a,b) such that $a{:}\,A$ and $b{:}\,B[x \coloneqq a]$ (the type obtained by substituting all free occurrences of x in B by a). The elements of $\Pi x{:}A.\,B$ are (*computable*) *functions* f such that $fa{:}\,B[x \coloneqq a]$, whenever $a{:}\,A$.

Theorem on Prime Numbers under Curry-Howard Interpretation

The theorem expresses that there are arbitritrarily large primes:

$$\forall m{:}\,\mathrm{N}.\,\exists n{:}\,\mathrm{N}.\,m < n \wedge Prime(n)$$

Under the *Curry-Howard interpretation* this becomes the *type of functions* which map a *number m* to a *triple* (*n*,(*p*, *q*)), where *n* is a *number*, *p* is a *proof* that *m* < *n* and *q* is a *proof* that *n* is *prime*:

$$\Pi m{:}\,N.\,\Sigma n{:}\,N.\,m < n \times Prime(n)$$

This is the *proofs as programs principle*: a *constructive proof* that there are arbitrarily large primes becomes a *program* which *given any number* produces a *larger prime* together with *proofs* that it indeed is *larger* and indeed is *prime*.

Martin-Löf‘s Intuitionistic Type Theory

In addition to the *type formers* of the *Curry-Howard interpretation*, the logician and philosopher P. Martin-Löf extended the *basic intuitionistic type theory* (containing *Heyting‘s arithmetic of higher types* HA$^{\omega}$ and *Gödel‘s system* T *of primitive recursive functions of higher type*) with *primitive identity types*, *well founded tree types*, *universe hierarchies* and general notions of *inductice* and *inductive–recursive definitions*.

His extension increases the proof-theoretic power of the theory and its application to programming as well as to formalization of mathematics.

Intuitionistic Type Predicate Logic

Besides the given rules for Π, there are *analogous rules* for *other type formers* corresponding to the *logical constants* of *typed predicate logic*:

Π-formation	Π-introduction	Π-elimination
$\dfrac{\Gamma \vdash A \quad \Gamma, x: A \vdash B}{\Gamma \vdash \Pi x: A.B}$	$\dfrac{\Gamma, x: A \vdash b: \beta}{\Gamma \vdash \lambda x.b: \Pi x: A.B}$	$\dfrac{\Gamma \vdash f: \Pi x: A.B \quad \Gamma \vdash a: A}{\Gamma \vdash fa: B[x := a]}$

Π-*equality* is introduced by *β-conversion* and *η-conversion*:

β-conversion	*η*-conversion
$\dfrac{\Gamma, x: A \vdash b: B \quad \Gamma \vdash a: A}{\Gamma \vdash (\lambda x.b)a = b[x := a]: B[x := a}$	$\dfrac{\Gamma \vdash f: \Pi x: A.B}{\Gamma \vdash \lambda x.fx = f: \Pi x: A.B}$

***Conguence rules* preserve *equality*:**

congruence rule
$\dfrac{\Gamma \vdash A = A' \quad \Gamma, x: A \vdash B = B'}{\Gamma \vdash \Pi x: A.B = \Pi x: A'.B'}$

Intuitionistic Type Arithmetic

As in Peano arithmetic, the *natural numbers* are generated by 0 and the *successor operation* s:

N-formation	N-introduction
$\Gamma \vdash \mathrm{N}$	$\Gamma \vdash 0{:}\mathrm{N}$ $\quad \dfrac{\Gamma \vdash 0{:}\mathrm{N}}{\Gamma \vdash s(a){:}N}$

The *elimination rule* states that these are the *only* ways to generate a natural number. The function $f(c) = R(c, d, xy.e)$ is defined by *primitive recursion* on the natural number c with base d and *step function* $xy.e$ (or $\lambda xy.e$) which maps the value y for the previous number $x{:}\,\mathrm{N}$ to the value for $s(x)$:

N-elimination
$\dfrac{\Gamma, x{:}\,\mathrm{N} \vdash C \quad \Gamma \vdash c{:}\mathrm{N} \quad \Gamma \vdash d{:}C[x := 0\,] \quad \Gamma, y{:}\,\mathrm{N}, z{:}\,C[x := y] \vdash e{:}C[x := s(y)\,]}{\Gamma \vdash R(c, d, yz.e){:}\,C[x := c]}$

N-equality (under appropriate premisses)
$R(0, d, yz.e) = d{:}\,C[x := 0]$
$R(s(a), d, yz.e) = e[:= a, z := R(a, d, yz.e)]{:}\,C[x := s(a)]$

The Universe of Small Types

To overcome the *impredicativity* of the „*type of all types*", Martin-Löf introduced a *universe* U *of small types closed under all type formers of the theory*, *except* that it does not *contain itself*:

U-formation
$\Gamma \vdash \mathrm{U}$

U-introduction
$\Gamma \vdash \emptyset : \mathrm{U} \qquad \Gamma \vdash 1 : \mathrm{U}$
$\dfrac{\Gamma \vdash A : \mathrm{U} \quad \Gamma \vdash B : \mathrm{U}}{\Gamma \vdash A + B : \mathrm{U}} \qquad \dfrac{\Gamma \vdash A : \mathrm{U} \quad \Gamma \vdash B : \mathrm{U}}{\Gamma \vdash A \times B : \mathrm{U}}$
$\dfrac{\Gamma \vdash A : \mathrm{U} \quad \Gamma \vdash B : \mathrm{U}}{\Gamma \vdash A \to B : \mathrm{U}}$
$\dfrac{\Gamma \vdash A : \mathrm{U} \quad \Gamma, x : A \vdash B : \mathrm{U}}{\Gamma \vdash \Sigma x : A.\ B : \mathrm{U}} \qquad \dfrac{\Gamma \vdash A : \mathrm{U} \quad \Gamma, x : A \vdash B : \mathrm{U}}{\Gamma \vdash \Pi x : A.\ B : \mathrm{U}}$
$\Gamma \vdash \mathrm{N} : \mathrm{U}$

U-elimination
$\dfrac{\Gamma \vdash A : \mathrm{U}}{\Gamma \vdash A}$

Type-Theoretic Universe U and the Grothendieck Universe

The *type-theoretic universe* U is analogous to a *Grothendieck universe* in *set theory* which is a *set of sets closed* under all the ways sets can be *constructed* in *Zermelo-Fraenkel set theory*:

1. $x \in U, y \in x \Rightarrow y \in U$ (transitivity)
2. $x, y \in U \Rightarrow \{x, y\} \in U$
3. $x \in U \Rightarrow \mathcal{P}(x) \in U$ (power set)
4. $\{x_\alpha\}_{\alpha \in I}$ family of elements of U, $I \in U \Rightarrow \bigcup_{\alpha \in I} x_\alpha \in U$

Alexander Grothendieck (1928-2014) used his universe as a way of *avoiding proper classes* in *algebraic geometry*. Its *existence* goes *beyond the usual axioms* of *Zermelo–Fraenkel set theory* and implies the *existence of strongly inaccessible cardinals*.

***Tarski–Grothendieck set theory* is an axiomatic treatment of set theory, used in some *automatic proof systems*, in which every set belongs to a Grothendieck universe. The concept of a *Grothendieck universe* can also be defined in a *topos* (*category theory*).**

The Axiom of Choice is a Theorem in Intuitionistic Type Theory

In *intuitionistic type theory*, the *axiom of choice* is an *immediate consequence* of the *BHK-interpretation* of the *intuitionistic quantifiers*:

Theorem:

$$(\Pi x{:}A.\,\Sigma y{:}B.\,C) \rightarrow \Sigma f{:}(\Pi x{:}A.\,B).\,C[y := f\,x]$$

Proof:

- $\Pi x{:}A.\,\Sigma y{:}B.\,C$ is the *type of functions* which map elements $x{:}A$ to pairs (y, z) with $y{:}B$ and $z{:}C$.
- The *choice function* f is obtained by *returning the first component* $y{:}B$ of this pair.

In *set theory*, the *axiom of choice* is *in general not constructive*. (Types are not in general appropriate constructive approximations of sets in the classical sense.)

General Identity Type Former

The rules for I express that the *identity relation* is *inductively generated* by the *proof of reflexivity* (*constant* r):

I-formation	I-introduction
$\dfrac{\Gamma \vdash A \quad \Gamma \vdash a : A \quad \Gamma \vdash a' : A}{\Gamma \vdash \mathrm{I}(A, a, a')}$	$\dfrac{\Gamma \vdash A \quad \Gamma \vdash a : A}{\Gamma \vdash \mathrm{r} : \mathrm{I}(A, a, a)}$

The elimination rule for the identity type is a generalization of identity elimination in predicate logic (*elimination constant* J):

I-elimination
$\dfrac{\Gamma, x : A, y : \mathrm{I}(A, a, x) \vdash C \quad \Gamma \vdash b : A \quad \Gamma \vdash \mathrm{c} : \mathrm{I}(A, a, b) \quad \Gamma \vdash d : C\,[x \coloneqq a, y \coloneqq r]}{\Gamma \vdash \mathrm{J}(c, d) : C\,[x \coloneqq b, y \coloneqq c]}$

J-equality (under appropriate assumptions)
$\mathrm{J}(r,d) = d$

Inductive Types in Intuitionistic Type Theory

An *inductive type* is *freely generated* by a certain number of *constructors*.

Examples: a) Type $\mathbb{N}$ of natural numbers with *constructors*

- $\mathbf{0}: \mathbb{N}$
- $\mathbf{succ}: \mathbb{N} \to \mathbb{N}$

b) Type List(A) of finite lists of elements of type A with *constructors*

- $\mathbf{nil}: \mathrm{List}(A)$ (empty list)
- $\mathbf{cons}: A \to \mathrm{List}(A) \to \mathrm{List}(A)$ (add an element to the front of the list)
- $\mathbf{app}: \mathrm{List}(A) \to \mathrm{List}(A) \to \mathrm{List}(A)$ (concatenate two lists)

An *induction principle* proves a *statement* for a *type freely generated by its constructors*.

Example: W-*type* $\mathrm{W}_{(a:A)}B(a)$ of *well-founded trees* with *nodes* labeled by elements $a : A$ and $B(a)$-many *branches*. We *prove* a statement $E: \mathrm{W}_{(a:A)}B(a) \to \mathcal{U}$ about *all elements of the type* $\mathrm{W}_{(a:A)}B(a)$ by proving it for its *constructor(s)*.

3. From Proof Theory to Proof Assistants

2.1 Intuitionistic Type Theory and Proof Assistants
2.2 Verification of Circuits in Proof Assistants: Basics
2.3 Verification of Circuits in Proof Assistants: Applications

3.1 Intuitionistic Type Theory and Proof Assistant

Terms of the Calculus of Constructions (CoC)

CoC is a *type theory* of Thierry Coquand et al. which can serve as ***typed programming language*** as well as ***constructive foundation of mathematics***. It extends the *Curry-Howard isomorphism* to *proofs* in the *full intuitionistic predicate calculus*. Coc has very few *rules of construction* for terms:

- T is a *term* (*Type*).
- P is a *term* (*Prop*).
- *Variables* $(x, y, z, \ldots)$ are *terms*.
- If A and B are *terms*, then (AB) is a *term*.
- If A and B are *terms* and x is a *variable*, then $\lambda x : A. B$ and $\forall x : A. B$ are *terms*.

The *objects* of CoC are ***proofs*** (terms with propositions as types), ***propositions*** (small types), ***predicates*** (functions that return propositions), ***large types*** (types of predicates, e.g., P), T (type of large types).

Inference Rules of CoC

Γ is a sequence of type assignments $x_1 : A_1, x_2 : A_2, \ldots$; K is either T or P :

$$\frac{}{\Gamma \vdash P : T} \qquad \frac{\Gamma \vdash A : K}{\Gamma, x : A \vdash x : A}$$

$$\frac{\Gamma, x : A \vdash B : K \qquad \Gamma, x : A \vdash N : B}{\Gamma \vdash (\lambda x : A.N) : (\forall x : A.B) : K}$$

$$\frac{\Gamma \vdash M : (\forall x : A.B) \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B[x := N]}$$

$$\frac{\Gamma \vdash M : A \qquad A =_\beta B \qquad B : K}{\Gamma \vdash M : B}$$

Logical Operators and Data Types in CoC

Coc has very few basic operators. The *only logical operator* for forming *propositions* is ∀ :

logical operators:

$$A \Rightarrow B \quad \equiv \forall x{:}A.B \quad (x \notin B)$$
$$A \wedge B \quad \equiv \forall C{:}P.(A \Rightarrow B \Rightarrow C) \Rightarrow C$$
$$A \vee B \quad \equiv \forall C{:}P.(A \Rightarrow C) \Rightarrow (B \Rightarrow C) \Rightarrow C$$
$$\neg A \quad \equiv \forall C{:}P.(A \Rightarrow C)$$
$$\exists x{:}A.B \equiv \forall C{:}P.(\forall x{:}A(B \Rightarrow C)) \Rightarrow C$$

data types:

booleans: $\forall A{:}P.A \Rightarrow A \Rightarrow A$

naturals: $\forall A{:}P.(A \Rightarrow A) \Rightarrow (A \Rightarrow A)$

product $A \times B$: $A \wedge B$

disjoint union $A + B$: $A \vee B$

Calculus of Inductive Constructions (CiC)

CiC is based on CoC enriched with *inductive* and *co-inductive definitions* with the following *rules for constructing terms*:

- ***identifiers*** **refer to** ***constants*** **or** ***variables*****.**
- (AB) ***application*** **of a** ***functional object*** A **to** B
- $[x{:}\,A]B$ ***abstraction*** **of variable** x **of type** A **in term** B **to construct a** ***functional object*** $\lambda x \in A.B$
- $(x{:}\,A)B$ ***term*** **of type Set corresponds to** $\prod_{x\in A} B$ **product of sets.**
  $(x{:}\,A)B$ ***term*** **of type Prop corresponds to** $\forall x \in A\ B$**.**

If x **does** ***not*** **occur in** B, $A \rightarrow B$ **is an abbreviation which corresponds to**

- ***set of all functions*** **from** A **to** B
- ***logical implication***

Inductive Types in CiC*

An *inductive type* is *freely generated* by a certain number of *constructors*.

Examples: a) Type $\mathbb{N}$ of natural numbers with *constructors*

- $0 : \mathbb{N}$
- $\text{succ} : \mathbb{N} \rightarrow \mathbb{N}$

b) Type $\text{List}(A)$ of finite lists of elements of type A with *constructors*

- $\text{nil} : \text{List}(A)$
- $\text{cons} : A \rightarrow \text{List}(A) \rightarrow \text{List}(A)$
- $\text{app} : \text{List}(A) \rightarrow \text{List}(A) \rightarrow \text{List}(A)$ (concatenate two lists)

Inductive proofs make it possible to prove statements for *infinite collections* of objects (e.g., integers, lists, binary trees), because all these *objects* are constructed in a *finite number of steps*.

An *induction principle* of an *inductive type* proves a *statement* for a *type freely generated by its constructors*.

* C. Paulin-Mohring (1993), Inductive Definition in the System Coq: Rules and Properties (Research Report 92-49, LIP-ENS Lyon)

Co-Inductive Types in CiC*

Besides *inductive types*, there are *co-inductive types* concerning *infinite objects* (e.g., potentially infinite lists, potentially infinite trees with infinite branches).

***Terms* are still be obtained by *repeated uses of constructors* such as in *inductive types*. However, there is *no induction principle* and the *branches* may be *infinite*.**

In *practical domains* such as *telecommunication*, *energy*, or *transportation*, *streams* are examples with *infinite execution* which are defined by constructor `Cons`:

```
CoInductive Stream (A : Set) : Set :=
Cons : A → Stream → Stream
```

Contrary to the *inductive type* of a `list`, there is *no constructor* of the empty list. Thus, *finite lists cannot* be constructed.

* E. Giménez (1996), Un calcul de constructions infinies et son application à la vérification de systèmes communicants (PhD thesis Lyon)

Equivalence of Streams in CiC

Accessors of a *stream* `l` are defined by functions on the structure of the stream with *head* `hd` and *tail* `tl`:

```
Definition Head: Stream → A := [l] Cases l of (Cons hd _ ) ⇒ hd end.
Definition Tail: Stream → Stream := [l] Cases l of (Cons  _ tl) ⇒ tl
end.
```

Two *streams* `l` and `l‘` are *equivalent* iff their *heads* are *equal* and their *tails* are *equivalent*. In CiC, *equivalence of streams* is represented by a *co-inductive definition*:

```
CoInductive EqS : Stream → Stream → Prop := eqs : (l , l‘ : Stream)
                                                  (Head l) = (Head l‘) →
                                                  (EqS (Tail l)( Tail l‘)) →
                                                  (EqS l l‘).
```

Production of Streams in CiC

The *mapping* of a given function f on *two streams* l and l' is *co-recursively defined* in CiC:

```
CoFixpoint Map2 : (A, B, C : Set)
                  (A → B → C) → (Stream A) → (Stream B) →(Stream C) :=
   [A, B, f, l, l‘]
   (Cons (f (Head l)(Head l‘))(Map2 f (Tail l)(Tail l‘)))
```

The function $Prod$ builds the *stream of the pairs*, element by element, *of two streams* of type $(Stream\ A)$ and $(Stream\ B)$ respectively. $Prod$ is the result of the *application* $Map2$ to the function $(pair\ A\ B)$, where $pair$ is the *constructor* of the *cartesian product* $A * B$. In CiC, $Prod$ is represented by:

```
Definition Prod := [A, B : Set] (Map2 (pair A B ))
```

The Coq Proof Assistant*

Coq implements a *program specification* which is based on the *Calculus of Inductive Constructions* (CiC) combining both a *higher-order logic* and a *richly-typed functional language*.

The commands of Coq allow

- **to *define functions* or *predicates* (that can be evaluated efficiently)**
- **to *state mathematical theorems* and *software specifications***
- **to *interactively develop formal proofs* of these *theorems***
- **to *machine-check* these *proofs* by a relatively small certification (kernel)**
- **to *extract certified programs* to languages (e.g., Objective Caml, Haskell, Scheme)**

Coq provides *interactive proof methods*, *decision* and *semi-decision algorithms*. Connections with *external theorem provers* are available.

Coq is a platform for the verification of mathematical proofs as well as the verification of computer programs in CiC.

* Y. Bertot, P. Castéran (2004), Interactive Theorem Proving and Program Development: Coq'Art: CiC (Springer)

3.2 Verification of Circuits in Proof Assistants: Basics

Verification of Circuits with Co-Induction in Coq

A *hardware* or *software program* is correct („*certified by Coq*") if it can be *verified* to follow a given *specification* in CIC.

Example: Verification of circuits*

The *structure* and *behaviour of circuits* can mathematically be described by *interconnected finite automata* (e.g., Mealy machines). In circuits, one has to cope *with infinitely long temporal sequences* of data (*streams*).

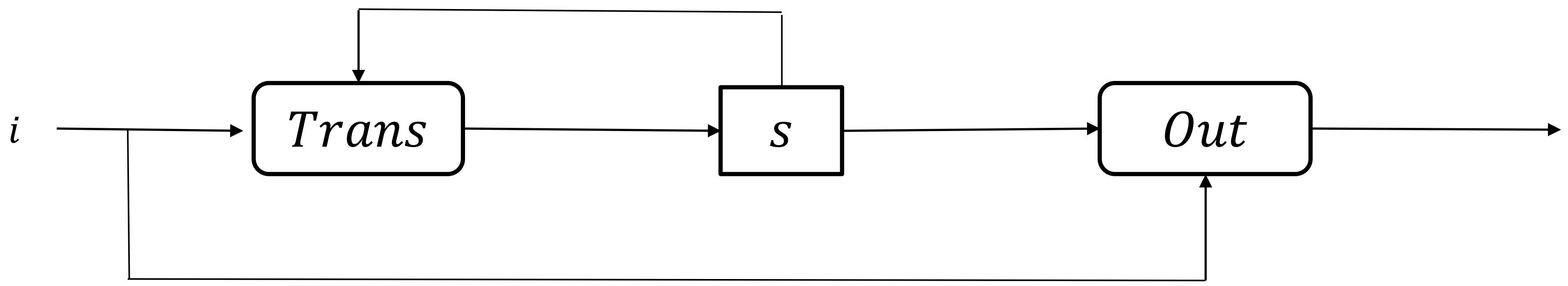
A *circuit* is correct iff, under certain conditions, the *output stream* of the *structural automaton* is *equivalent* to that of the *behavioural automaton*.

Therefore, *automata theory* must be *implemented* into CiC with the *co-inductive type of streams*.

* S. Coupet-Grimal, L. Jakubiec (1996): Coq and Hardware Verification: a Case Study (TPHOLs ‚96, LCNS 1125, 125-139)

Specification of Mealy Automata

A Mealy automaton is a 5-tuple $(I, O, S\ Trans, Out)$ with *input set* I, *output set* O, *state set* S, *transition function* $Trans: I \times S \rightarrow S$, and *output function* $Out: I \times S \rightarrow O$.



Given an *initial state s* , the *Mealy machine* computes an *infinite output sequence* („*stream*“) in response to an *infinite input sequence* („*stream*“).

Implementation of Mealy Automata in CiC

```
Variables I, O, S : Set .
Variable Trans : I → S → S.
Variable Out : I → S → O.

CoFixpoint Mealy : (Stream I) → S → (Stream O) := [inp, s]
    (Cons (Out (Head inp) s) (Mealy (Tail inp)(Trans (Head inp) s)).
```

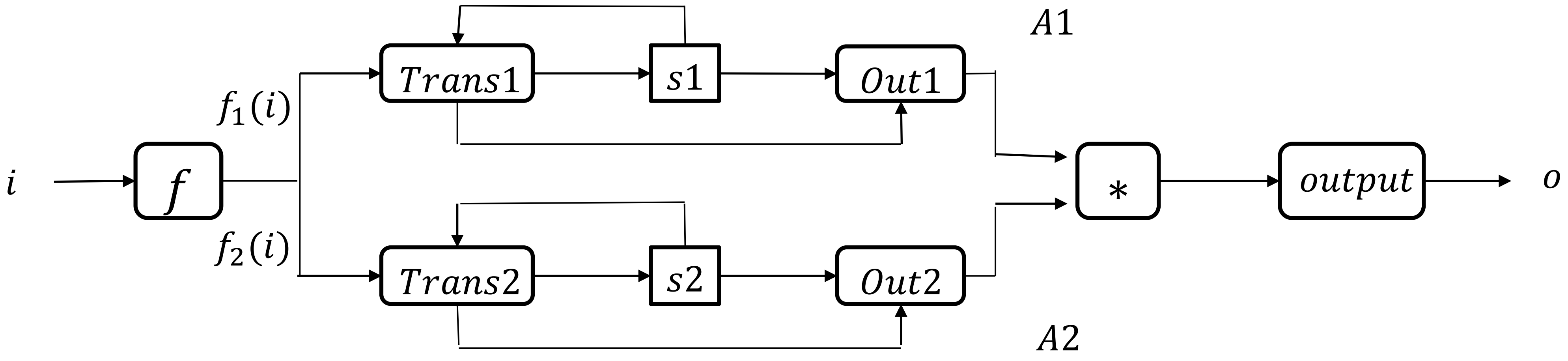
The first element of the *output stream* is the result of the *application* of the *output function Out* to the first input (the *head* of the *input* stream *inp*) and to the *initial state s*. The *tail* of the *output stream* is then computed by a *recursive call* to *Mealy* on the *tail* of the *input stream* and the *new state*. This new state is given by the function *Trans*, applied to the *first input* and the *initial state*.

The *streams* of *all the successive* states from the *initial one s* is obtained similarily:

```
CoFixpoint States : (Stream I) → S → (Stream S) := [inp, s]
    (Cons s (States (Tail inp)(Trans (Head inp) s))).
```

Network of Automata

In a network, *automata* are *inter-connected* by *parallel composition, sequential composition,* and *feedback composition of synchronous sequential devices*.



In the *parallel composition* of two *Mealy automata* $A1$ and $A2$, $f = (f_1, f_2)$ builds from the current input i the *pair of inputs* $(f_1(i), f_2(i))$ for $A1$ and $A2$, $output$ computes the *global outputs* of $A1$ and $A2$.

Implementation of Parallel Automata in CiC

```
Variables I1, I2, O1, O2, S1, S2, I, O : Set
Variable Trans1 : I1 → S1 → S1. Variable Trans2 : I2 → S2 → S2.
Variable Out1 : I1 → S1 → O1.   Variable Out2 : I2 → S2 → O2.
Variable f : I → I1*I2.         Variable f : O → O1*O2.
Local A1 := (Mealy Trans1 Out1). Local A2 := (Mealy Trans2 Out2).

Definition parallel : (Stream I) → S1 → S2 := [inp, s1, s2]
    (Map output (Prod (A1 (Map Fst (Map f inp)) s1)
                      (A2 (Map Snd (Map f inp)) s2))).
```

The *initial states* of automata $A1$ and $A2$ are $s1$ and $s2$. The *input* of $A1$ is obtained by mapping the first projection Fst on the stream resulting from the mapping of the function f on the *global stream* inp. Then $(A1(Map\ Fst\ (Map\ f\ inp))s1)$ is the *output stream* $A1$. That of $A2$ is defined similarly. Finally, the *parallel composition* is obtained by mapping the function $output$ on the *product* of the *output streams* of $A1$ and $A2$.

Invariant Relations of Mealy Automata*

The *equivalence* of *structure* and *behaviour of circuits* can be proved by certain *invariant relations* of *states* and *streams* in the corresponding Mealy automata.

Consider two *Mealy automata* $A1 = (I, O, S_1, Trans1, Out1)$ and $A2 = (I, O, S_2, Trans2, Out2)$ with the *same input set* and the *same output set*. Given p streams, a *relation* which holds for all p-tuples of elements at the same rank is called an *invariant* of these p streams.

In CiC, an *invariant relation P* with respect to *input set I* and the *state sets* S_1 and S_2 can be definied by co-induction:

```
CoInductive Inv [P : I → S1 → S2 → Prop] :
                (Stream I) → (Stream S1) → (Stream S2) → Prop :=
  C_Inv : (inp : (Stream I))(st1 : (Stream S1))(st2 : (Stream S2))
          (P (Head inp) (Head st1) (Head st2))→
          (Inv P (Tail inp) (Tail st1) (Tail st2))→
          (Inv P inp st1 st2).
```

*S. Coupet-Grimal, L. Jakubier, Hardware Verification using co-induction in Coq (Laboratoire d‘Informatique de Marseille, URA CNRS 1787)

Invariant State Relation of Mealy Automata in CiC

Let R be a relation on the state space $S_1 \times S_2$ and P a relation on $I \times S_1 \times S_2$.

***R* is *invariant* under *P* for the *automata* *A*1 and *A*2 iff**

$$\forall i \in I\ \forall s_1 \in S_1\ \forall s_2 \in S_2$$
$$(P(i, s_1, s_2) \wedge R(s_1, s_2)) \Rightarrow R(\,Trans1\ (i, s_1), Trans2\ (i, s_2)).$$

The *invariance* of relation *R* can be implemented into CIC :

```
Definition Inv_under := [P : I→ S1 → S2 → Prop][R : S1 → S2 → Prop]
    (i : I)(s1 : S1)(s2 : S2)
    (P i s1 s2) → (R s1 s2) →(R (Trans 1 i s1)(Trans2 i s2)).
```

An *output relation* is strong enough to induce the *equality of the outputs* of two automata:

```
Definition Output_rel := [R : S1 → S2 → Prop]
                         (i : I)(s1 : S1) (s2 : S2)
                         (R s1 s2) →(Out1 i s1)=(Out2 i s2).
```

Proof Scheme for Circuit Correctness.

The correctness of a circuit **is proved by the** ***equivalence*** **of its** ***structure*** **and** ***behaviour*** **which are represented by two** ***composed Mealy automata*****. The** ***equivalence of composed Mealy automata*** **can be proved by the** ***equivalence lemma of invariant relations*** **(which is also represented in CiC) :**

If *R* is an *output relation invariant* under *P* that holds for the *initial states*, if *P* is an *invariant* for the *common input stream* and the *state streams* of each automata, then the *two output streams* are *equivalent*.

```
Lemma Equiv_2_Mealy :
(P : I → S1 → S2 → Prop)(R : S1 → S2 → Prop)
(Output_rel R) → (Inv_under P R) → (R s1 s2) →
(inp : (Stream I)) (s1 : S1) (s2 : S2)
(Inv P inp (States Trans1 Out1 inp s1)(States Trans2 Out2 inp s2)) →
(EqS (A1 inp s1) (A2 inp s2)).
```

Proof by co-induction

3.3 Verification of Circuits in Proof Assistants: Application

Certification of a 4 by 4 Switch Fabric

A *switch fabric* is a *network topology* in which *nodes* interconnect via one or more *switches*. The *switching element* performs *switching of data* from 4 *input ports* to 4 *output ports* and *arbitrating data clashes* according to the *output port requests* made by the input ports.*

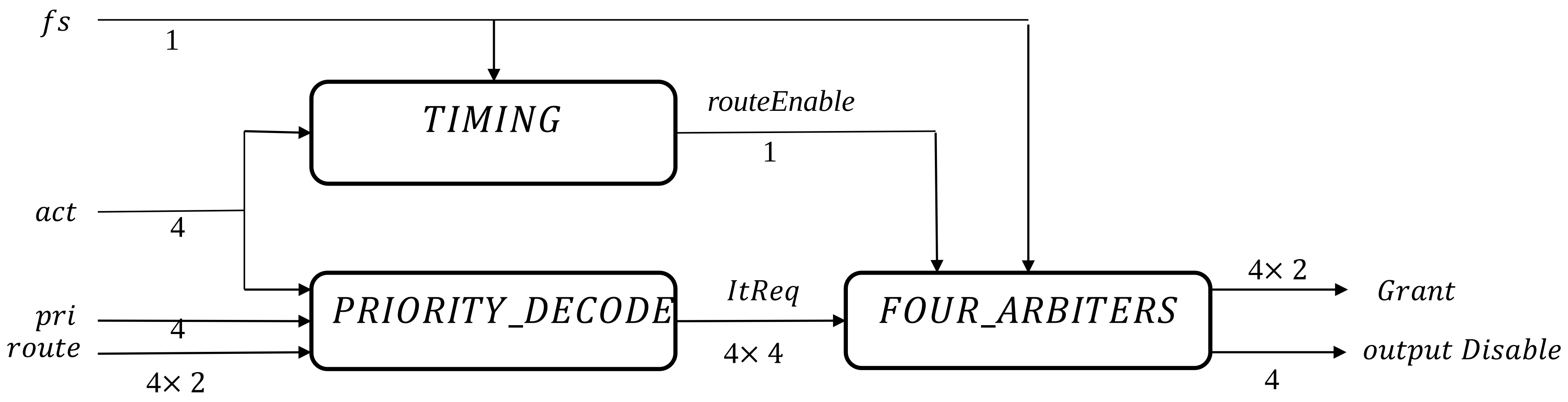
The most significant part for *verification* is the Arbitration Unit. It decodes *requests* from *input ports* and *priorities* between data to be sent, and then performs *arbitration*.

* Local area network based on ATM (Systems Research Group, Cambridge University)

Structure of the Arbitration Unit

The arbiration unit is the interconnection of three modules:

- ***FOUR_ARBITERS*** **performs the** ***arbitration*** **for** ***all output ports*** **(following Round Robin algorithm)**
- ***TIMING determines when*** **the** ***arbitration*** **process can be triggered.**
- ***PRIORITY_DECODE*** **decodes the** ***requests*** **and filters them according to their** ***priority***



Outline of the Proof of Correctness*

	The *correctness* of a switch fabric requires an *equivalence proof* of its *structural automaton* and *behavioural automaton*. It follows from the *verification* of its *modules* that compose the *Arbitration* unit.
(1)	Proof that the *behavioural automata* for *TIMING*, *FOUR_ARBITERS*, and *PRIORITY_DECODE* are *equivalent* to the three corresponding *structural automata*.
(2)	Construction of the *global structural automaton structure_ARBITRATION* by *interconnecting* the *structural automata* of the three *modules TIMING*, *FOUR_ARBITERS*, and *PRIORITY_DECODE* .
(3)	Construction of the *global behavioural automaton Composed_Behaviours* by *interconnecting* the *behavioural automata* of the the three *modules TIMING*, *FOUR_ARBITERS*, and *PRIORITY_DECODE* .
(4)	Proof that *Composed_Behaviours* and *structure_ARBITRATION* are *equivalent* (which follows from (1) and by applying the *lemmas* stating that the *equivalence of automata* is a *congruence* for the *composition rules*).
(5)	Proof that *Composed_Behaviours* is *equivalent* to the *expected behaviour Behaviour_ARBITRATION*. (*Composed_Behaviours* is more abstract than *structure_ARBITRATION* .)
(6)	The equivalence of *Behaviour_ARBITRATION* and *structure_ARBITRATION* is obtained from (4) and (5) by using the *transitivity* of of the *equivalence* on the *streams*.

* S. Coupet-Grimal, L. Jakubier, Hardware Verification using co-induction in Coq (Laboratoire d'Informatique de Marseille, URA CNRS 1787)

Advantages of the Coq Proof Assistent for Verification of Software/Hardware

- In Coq, a *verification of a computer program* is as *strong* and *save* as a *mathematical proof in a constructive formalism*.
- The use of Coq *dependent types* provide *precise* and *reliable specifications*.
- The use of Coq *co-inductive types* provide a *clear modelling* of *streams* in *circuits* (without introducing any temporal parameter).
- The use of Coq *co-induction* allows to capture the *temporal aspects* of the *proof processes* in one *lemma*.
- The *hierarchical* and *modular approach* allows *correctness results* in a *complex verification process* related to *pre-proven components*.

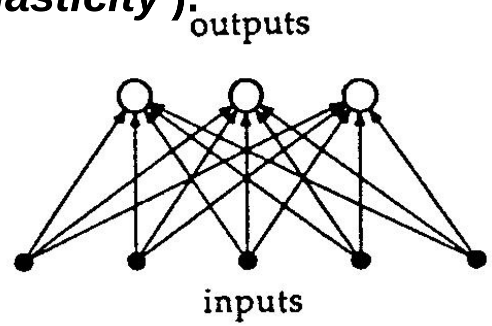
4. Verification in Machine Learning

3.1 Basics of Machine Learning
3.2 Causal and Statistical Learning
3.3 Testing, Verification, and Certification of Programs
3.4 Perspectives of Responsible AI

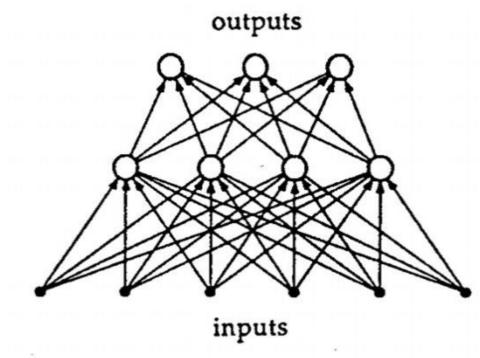
4.1 Basics of Machine Learning

Neural Networks and Learning Algorithms

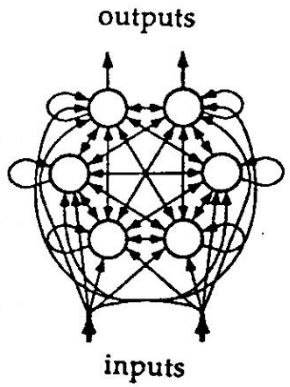
Neural networks **are *complex systems* of *firing* and *non-firing neurons* with *topologies* like brains. There is no central processor (‚mother cell'), but a *self-organizing information flow* in cell-assemblies according to rules of synaptic interaction (‚*synaptic plasticity*').**



Feedforward with one synaptic layer



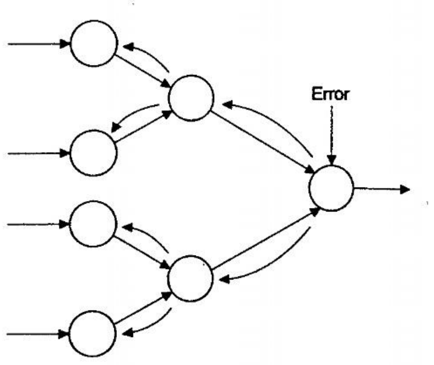
Feedforward with two synaptic layers (Hidden Units)

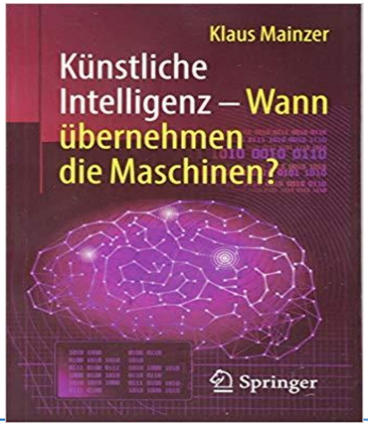


Feedback of recurrent neural network (RNN)

Learning algorithms:

- ***supervised***
- ***non-supervised***
- ***reinforcement***
- ***deep learning***





Definition of a (Finite Size Recurrent) Neural Network

A (recurrent) *neural network* $\mathcal{N}$ is presented by a *directed graph* of *nodes* called neurons.

Each neuron *updates its activation* value by applying a composition of a *one-variable function* with a *linear combination* of the *activations of all neurons* x_j $(j = 1, \ldots, N)$, the *external inputs* u_k $(k = 1, \ldots, M)$, and *synaptic weights* of rational coefficients a_{ij}, b_{ij}, c_i.

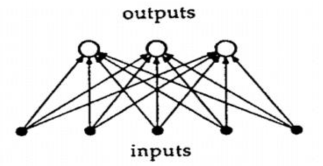
Each *processor's (cellular) state* is updated by

$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij} x_j(t) + \sum_{j=1}^{M} b_{ij} u_j(t) + c_i\right)$$

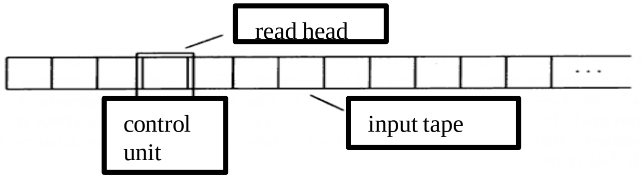
with x_i *states of activation*, u_j *inputs at the previous instants*, *synaptic weights* a_{ij}, b_{ij}, c_i, and sigmoid (e.g., saturated-linear) function σ:

$$\sigma(x) := \begin{cases} 0, if\ x < 0 \\ x, if\ 0 \le x \le 1 \\ 1, if\ x > 1 \end{cases}$$

Equivalence of Neural Networks, Automata, and Machines

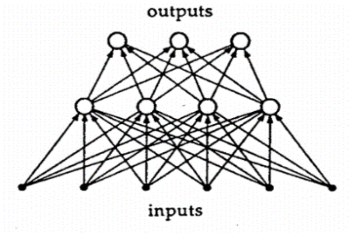


digital McCulloch-Pitts net
with *integer weights*

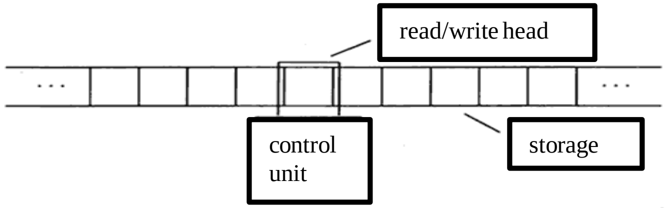


finite automaton

recognition of *computable („regular“) languages* *

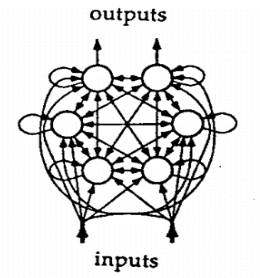


digital net
with *rational weights*

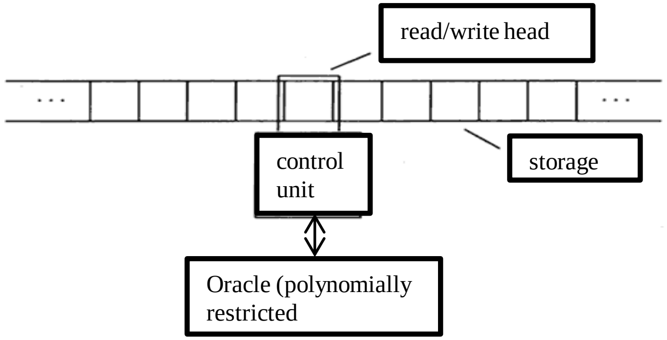


Turing machine

recognition of *computable („recursive“) languages* (Chomsky grammar) **

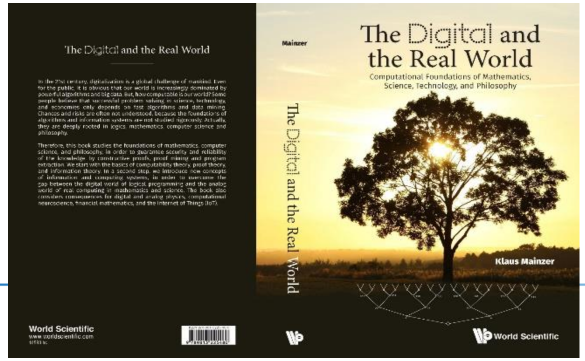


analog recurrent net
with *real weights*



Turing oracle machine

recognition of *(non-recursive) languages* (beyond Chomsky) ***



* S.C. Kleene (1956); **, *** H.T. Siegelmann, E.D. Sontag (1995), (1994); K. Mainzer (2018)

Acceptance and Recognition of Languages

A <u>*language*</u> $L \subseteq \{0,1\}^+$ is <u>*accepted*</u> by a *formal net* $\mathcal{N}$ if, for every *word* $\omega \in L$, ω is *accepted* by $\mathcal{N}$, and for every word $\omega \notin L$, ω is *rejected* or *not classified* by $\mathcal{N}$.

L is <u>*recognized*</u> or <u>*decided*</u> by *net* $\mathcal{N}$ if L is *accepted* by $\mathcal{N}$ and its *complement* is *rejected* by $\mathcal{N}$.

Let $T: \mathbb{N} \rightarrow \mathbb{N}$ be a *total function* on natural numbers.

The *language* L is *recognized* or <u>*decided in time*</u> T by the $\mathcal{N}$ if any *word* $\omega \in \{0,1\}^+$ is *correctly classified* in *time* not greater than $T(|\omega|)$.

Verification of Neural Networks and Learning Algorithms

Digital neural networks are equivalent to appropriate *automata* (with respect to certain cognitive tasks).

The *structure* and *behaviour of automata* can be implemented into the *Calculus of inductive Constructions* (CiC).

Thus, in principle, their *conformance* could *verify* the *correctnesss of circuits of automata* and, therefore, the *correctness of neural networks in Coq*.

Even *analog neural networks* (with real weights) could be implemented into CiC extended by *higher inductively defined structures* in HoTT to *verify* their *correctness in Coq*.

4.2 Causal and Statistical Learning

What does Probabilistic Reasoning and Pobabilistic Learning mean?

Probability theory is based on a *model* of a *random experiment* or *probability space* $(\Omega, \mathcal{F}, P)$ with Ω set of all *outcomes (data)*, $\mathcal{F}$ collection of *events* $A \subseteq \Omega$, and P *measure* assigning a *probability* to each event.

Probabilistic reasoning tries *to infer properties* of the *outcomes (data)* of random experiments from a *given mathematical structure* $(\Omega, \mathcal{F}, P)$.

Probabilistic learning tries to infer *properties* of the *underlying statistical model* from the *outcomes of experiments*.

Example of Probabilistic Learning

Example **:**

Given $(x_1, y_1), \ldots, (x_n, y_n)$ *observed data* with $x_i \in \mathcal{X}$ *inputs* and $y_i \in \mathcal{Y}$ *outputs* $(1 \leq i \leq n)$.
***Metric spaces* $\mathcal{X}$ and $\mathcal{Y}$ are equipped with the *Borel σ-algebra*.**

Assume that each (x_i, y_i) is independently generated by the same *unknown random experiment*, i.e. realizations of *random variables* $(X_1, Y_1), \ldots, (X_n, Y_n)$ i.i.d. (independent and identically distributed) with *joint distribution* $P_{X,Y}$ and *measurable function* $X: \Omega \rightarrow \mathcal{X}$ as random variable.

Try to infer properties of joint distribution $P_{X,Y}$ such as:*

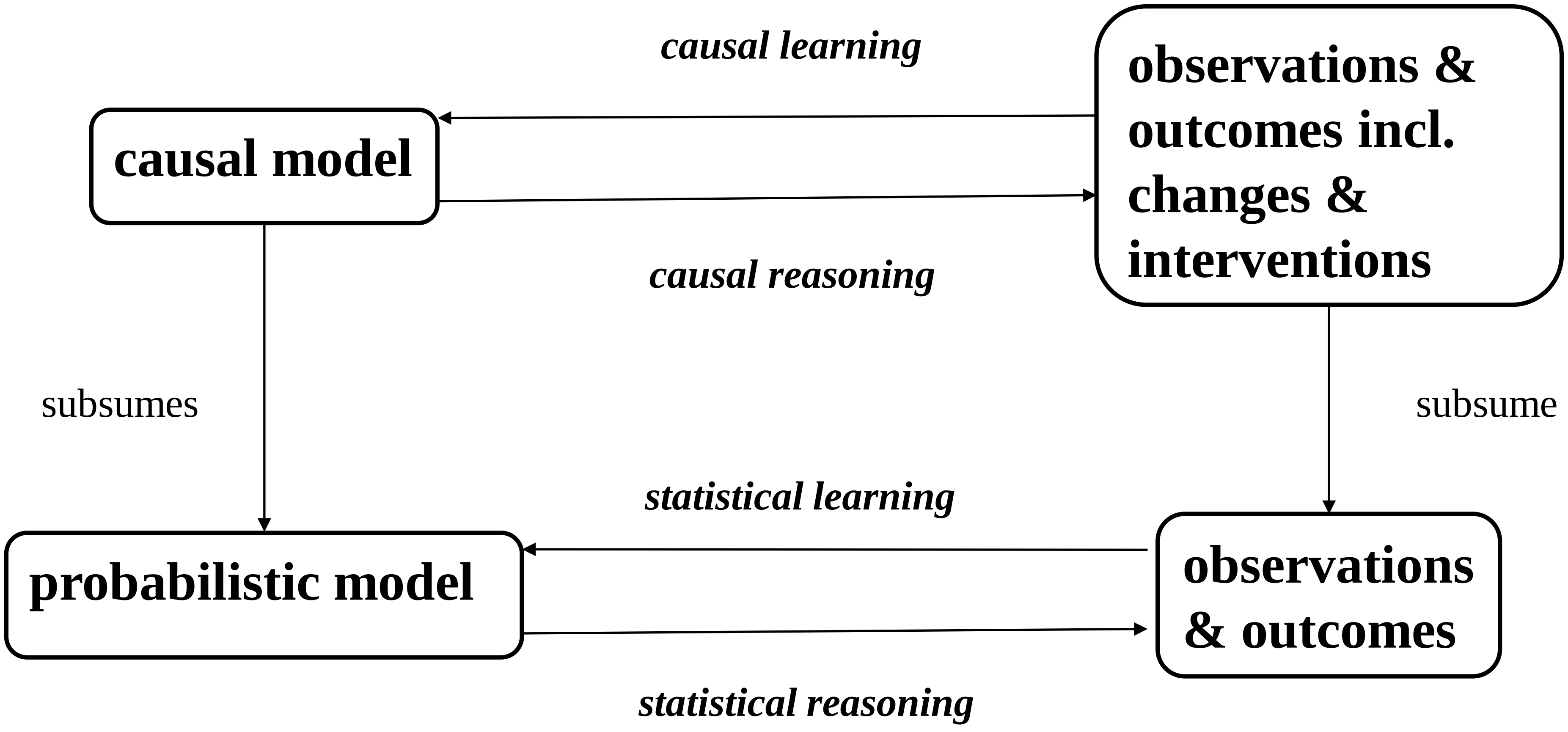
(i) the *expectation of the output* $f(x) = \mathbb{E}[Y|X = x]$ given the input (*regression*)

(ii) a *binary classifier* assigning each x to the class that is more likely:
$f(x) = \text{argmax}_{y \in \mathcal{Y}} P(Y = y|X = x)$ with $\mathcal{Y} = \{\pm 1\}$

(iii) the *density* $p_{X,Y}$ of $P_{X,Y}$ (assuming it exists)

* Vapnik 1998

Causal Modeling and Machine Learning



Peters et al. 2017, p. 6

Definition of Structural Causal Models

A *structural causal model* (SCM) $\mathfrak{C} = (\mathbf{S}, P_{\mathrm{N}})$ consists of a collection $\mathbf{S}$ of d *structural assignments* $X_j := f_j(\mathrm{PA}_j, N_j)(j = 1, \dots, d)$ with $\mathrm{PA}_j \subseteq \{X_1, \dots, X_d\} \setminus \{X_j\}$ *parents* of X_j and a *joint distribution* P_{N} over the (*jointly independent*) *noise variables* $\mathrm{N} = N_1, \dots, N_d$ (i.e. P_{N} *product distribution*).

The *graph* $\mathcal{G}$ of SCM is generated by one *vertex* (node) for each X_j and *directed edges* from each parent in PA_j to X_j.

X_j is called *direct effect* of the elements of PA_j as *direct causes* of X_j.

Proposition on Entailed Distributions

An SCM $\mathfrak{C}$ defines a *unique distribution* $P_{\mathrm{X}}^{\mathfrak{C}}$ over the variables $\mathrm{X} = (X_1, \dots, X_d)$ such that $X_j := f_j(\mathrm{PA}_j, N_j)(j = 1, \dots, d)$.

Proofs of Causal Structures

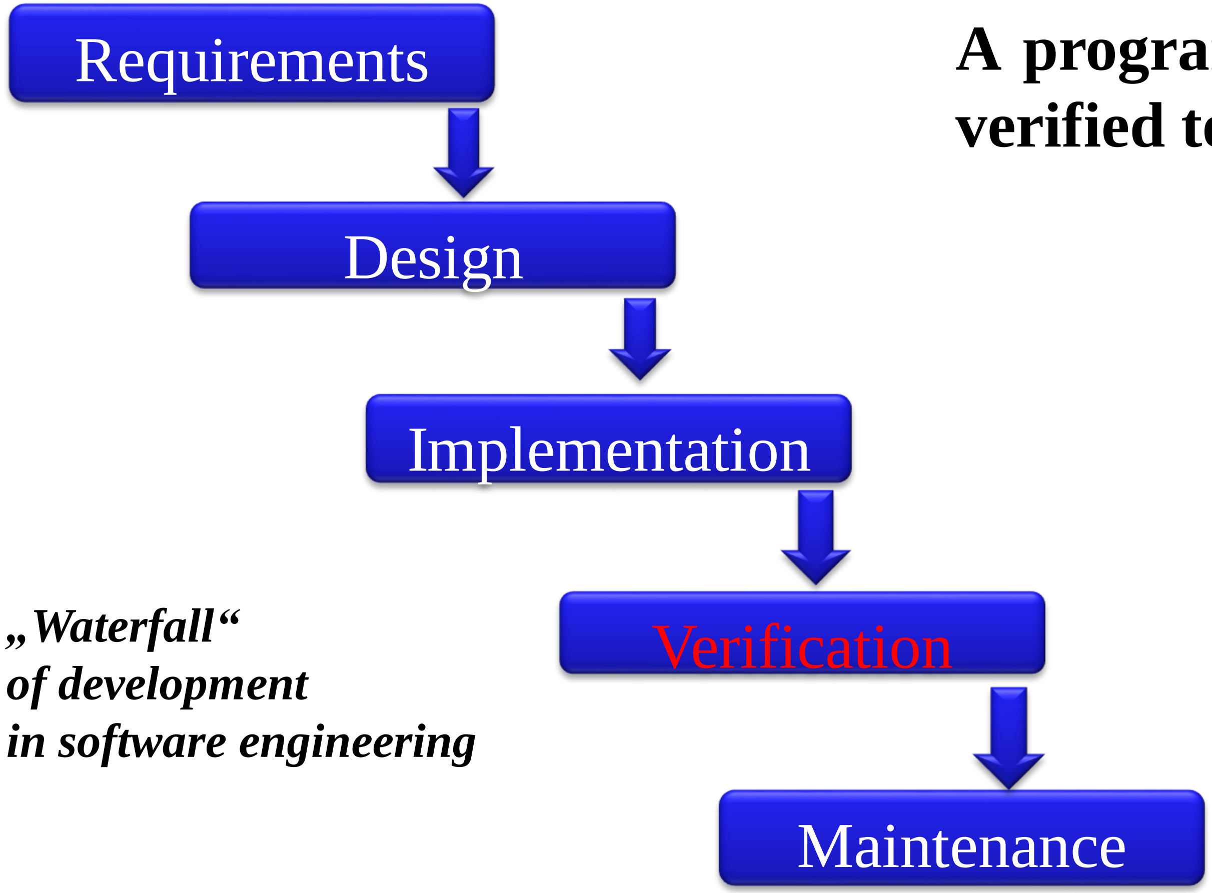
Under the *assumption of different types of structural models* $\mathfrak{C}$ (i.e. theories of mathematical laws) with Gaussian noise, *causal graph structures* $\mathcal{G}$ can be *provable identified* from the *joint distribution of data*. (Results for *non-Gaussian noise* are also available.)

Types of structural models	Types of equations	Condition on functions	Proofs of uniquely identifiable causal graphs
Structural Causal Models SCM (general)	$X_j := f_j(X_{\mathbf{PA}_j}, N_j)$	–	no
Additive Noise Models ANM	$X_j := f_j\left(X_{\mathbf{PA}_j}\right) + N_j$	nonlinear	yes
Causal Additive Models CAM	$X_j := \sum_{k \in \mathbf{PA}_j} f_{jk}(X_k) + N_j$	nonlinear	yes
Linear Gaussian	$X_j := \sum_{k \in \mathbf{PA}_j} \beta_{jk}(X_k) + N_j$	linear	no
Linear Gaussian with equal error invariance	$X_j := \sum_{k \in \mathbf{PA}_j} \beta_{jk}(X_k) + N_j$	linear	yes

Peters et al. 2017, p. 138

4.3 Testing, Verification, and Certification of Programs

Correctness of Certified Programs with Proof Assistants



„Waterfall“
of development
in software engineering

A program is <u>correct</u> („*certified*“) if it can be verified to follow a given specification.

A <u>proof assistant</u> proves the *correctness of a computer program* in a *consistent formalism* like a *constructive proof in mathematics* (e.g., Coq, Agda, MinLog).

Therefore, *proof assistants* are the <u>*best formal verification*</u> of *correctness for certified programs*.

Ad-Hoc and Empirical Testing versus Model-Based Testing

Empirical testing lays *directly* on the *analysis of program executions*. It collects information from executing the program either after *actively* soliciting some executions, or *passively* during operation and try to *abstract* from these some *relevant properties of data* or of *behavior*.

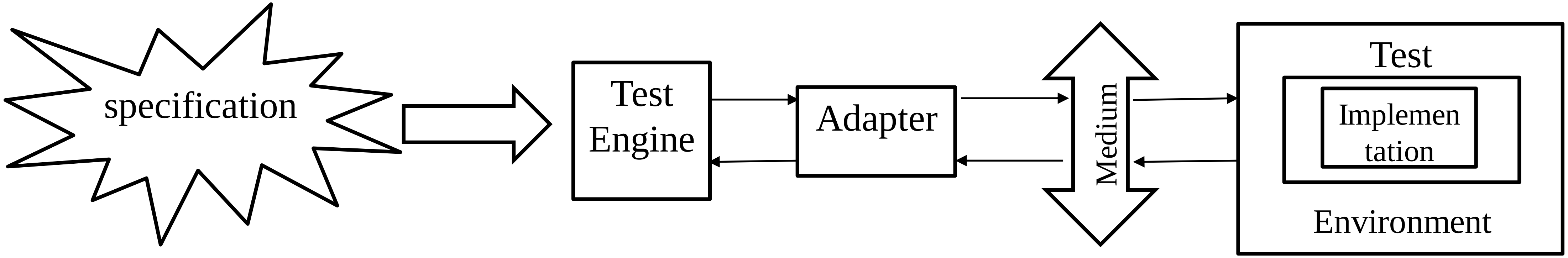
On this basis, it is decided whether the system *conforms* to the *expected behavior*.

Model-based testing uses a *model of the system* that is based on the *design*. From this model, *test input* is automatically generated and *executed* by a *test tool*.

The *output of the system* is *automatically compared* to the *output specified by the model* of the system (conformance of implementation with specification).

If the *system passes all the generated tests*, then the system is considered to be *correct*.

Test Tool Architecture of Model-Based Testing

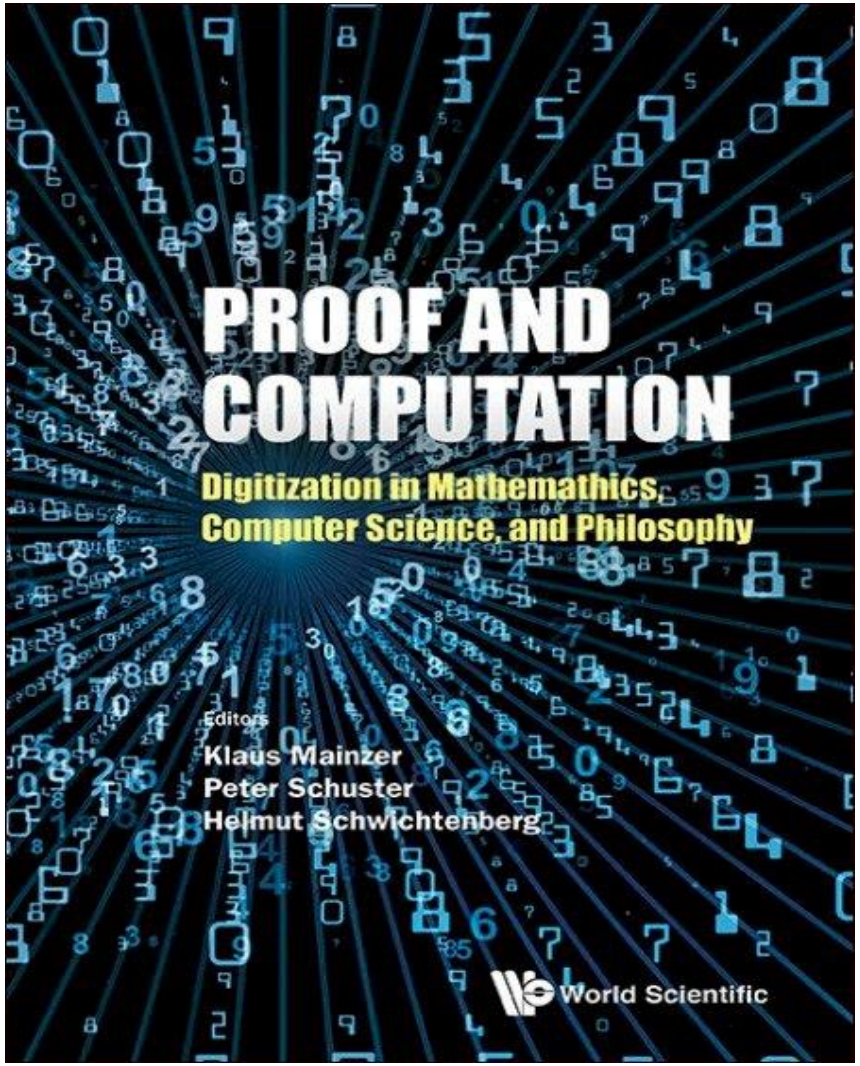


The <u>test engine</u> implements the *test generation procedure*:

It steps through the *specification of the model* and computes the sets of ***<u>allowed input and output actions</u>***.

If an ***<u>output action</u>*** is observed, then the *test engine evaluates* whether this output is ***<u>allowed</u>*** by the ***<u>specification of the model</u>*** (<u>conformance</u> of <u>implementation/specification</u>).

If some output is observed that is ***<u>not allowed</u>*** according to the specification, then the test is ***<u>terminated</u>*** with the ***<u>verdict fail</u>***. As long as the verdict fail is not given, the test terminates with the ***<u>verdict pass</u>***.



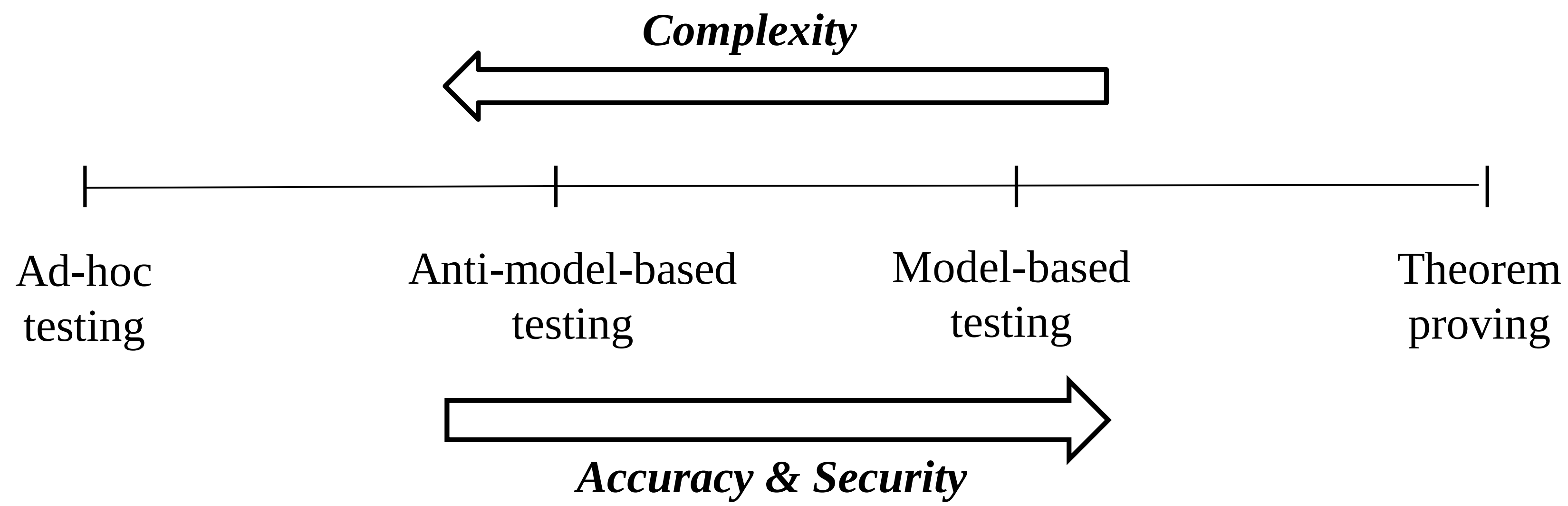
Proof Assistants

A <u>proof assistant</u> proves the *correctness of a computer program* in a *consistent formalism* like a *constructive proof in mathematics* (e.g., Coq, Agda, MinLog, Isabelle).

Therefore, *proof assistants* are the *<u>best formal verification</u>* of *correctness for certified programs*.

There are *restricted practical applications* (e.g., Metro line in Paris with Coq), but not for *increasing complexity in industry*.

Degrees of Certification in Software Testing Research



We must aim at *increasing accuracy*, *security*, and *trust in software* in spite of *increasing complexity* of *civil* and *industrial applications*, but w.r.t. to *costs of testing* (e.g.,utility functions for trade-off time of delivery vs. market value, cost/effectiveness ratio of availability)

4.4 Perspectives of Responsible Artificial Intelligence

Certified AI-Programs

Statistical machine learning works,
but we can't understand the underlying reasoning.

Machine learning technique **is akin to** ***testing,***
but it is ***not enough for safety-critical systems.***

$\Longrightarrow$ ***Combination*** **of** ***<u>causal learning</u>***
with ***<u>certified programs</u> of model-based testing,***
satisfaction techniques, and theorem proving

„Trust & Verification by Type-checking "

Proof Theory
(e.g., type theory)

Mathematics
(e.g., analysis, homotopy theory)

Computer Science
(e.g., theorem prover)

Science and Society
(*e.g.*, AI, Big Data)

References:

Bertot, Y.; Castéran, P. (2004): *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Springer: New York.

Bishop, E.; Bridges, D. (1985): *Constructive Analysis*. Springer: New York.

Howard, W. A. (1969): The formulae-as-types notion of construction. In: Seldin, J. P.; Hindley, J. R. (eds.), *To H.B. Curry: Essays on Combinatory Logic*, *Lambda Calculus and Formalism*, Academic Press: Boston, MA, 479–490.

Kohlenbach, U. (2008): *Applied Proof Theory: Proof Interpretations and Their Use in Mathematics*. Springer: Berlin.

Lorenzen, P. (1965): *Differential und Integral*. *Eine konstruktive Einführung in die klassische Analysis*. Akademische Verlagsgesellschaft. Frankfurt.

Mainzer, K. (2018): *The Digital and the Real World. Computational Foundations of Mathematics, Science, Technology, and Society*. World Scientific Singapore.

Mainzer, K; Schuster, P.; Schwichtenberg, H. (Eds.) (2018): *Proof and Computation. Digitization in Mathematics, computer Science, and Philosophy*. World Scientific Singapore.

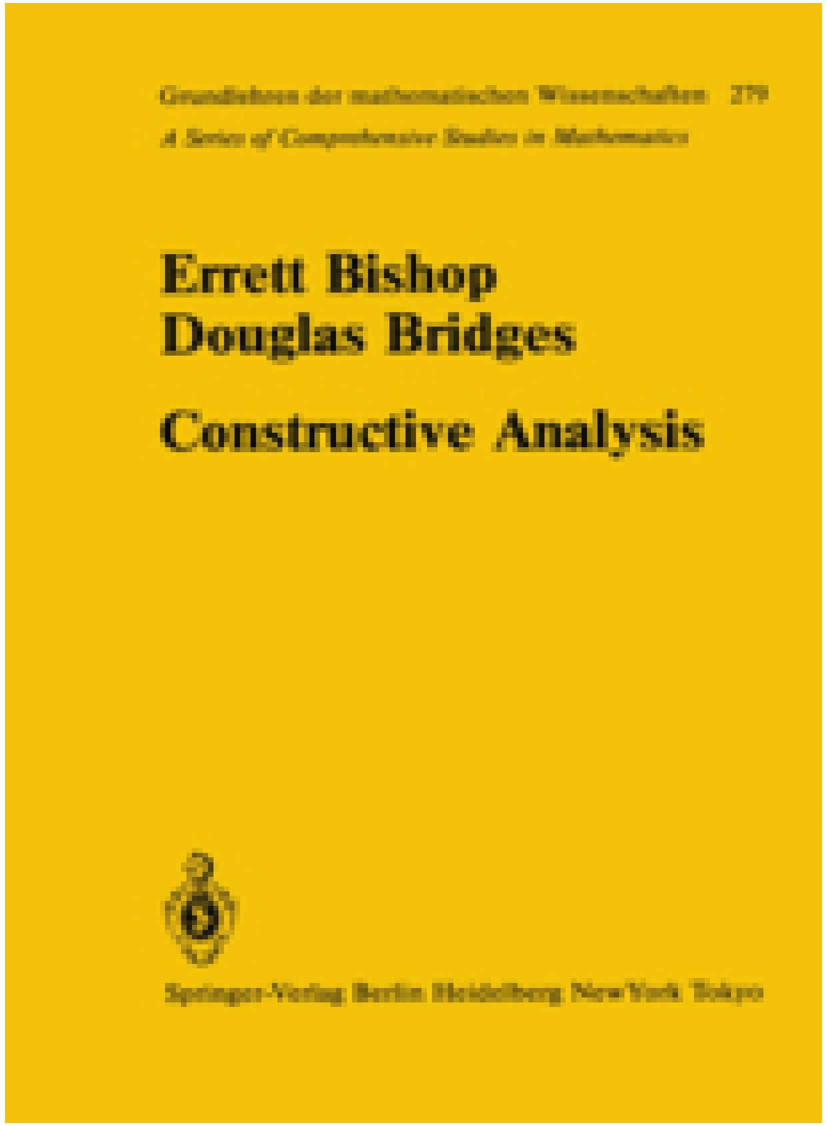
Mainzer, K. (2019): *Artificial Intelligence. When do Machines take over*? Springer (Translation of 2nd German edition 2019)

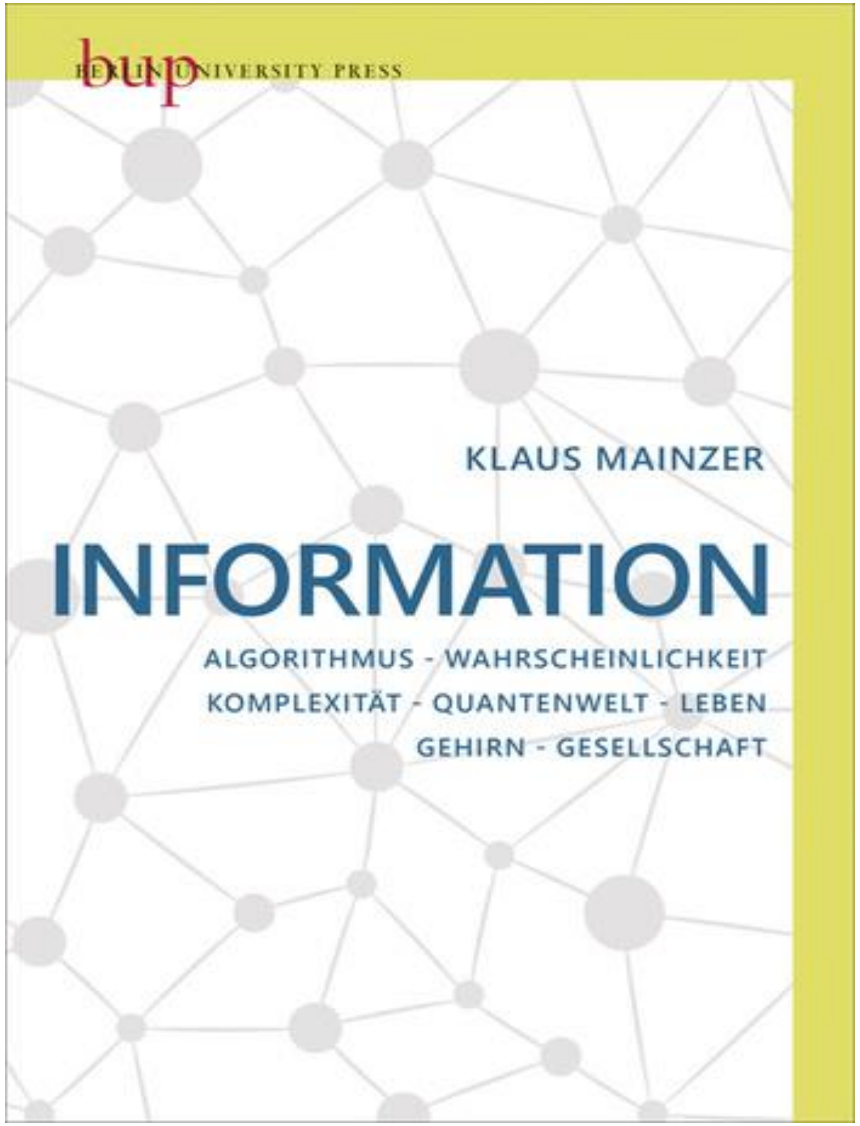
Martin-Löf, P. (1998): An intuitionistic theory of types. Twenty-five years of constructive type theory (Venice, 1995). In: *Oxford Logic Guides* 36, Oxford University Press: New York, 127-172.

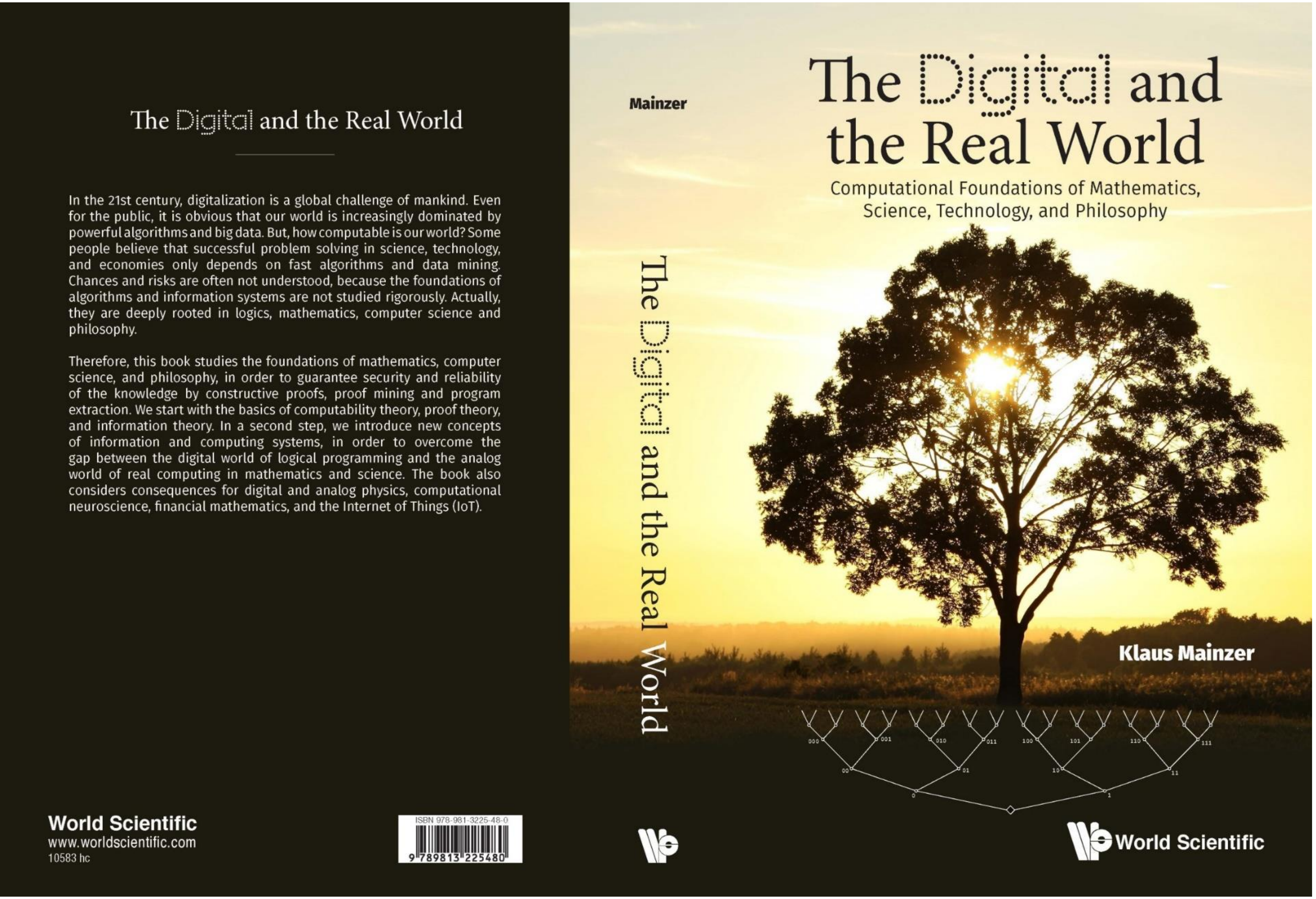
Palmgren, E. (1998): On universes in type theory. In: G. Sambin, J. M. Smith (eds), *Twenty-five years of constructive type theory*, Clarendon Press: Oxford, 191-204.

Peters, J.; Janzing, D.; Schölkopf, B. (2017): *Elements of Causal Inference. Foundations of Learning Algorithms*. MIT Press; Cambridge Mass.

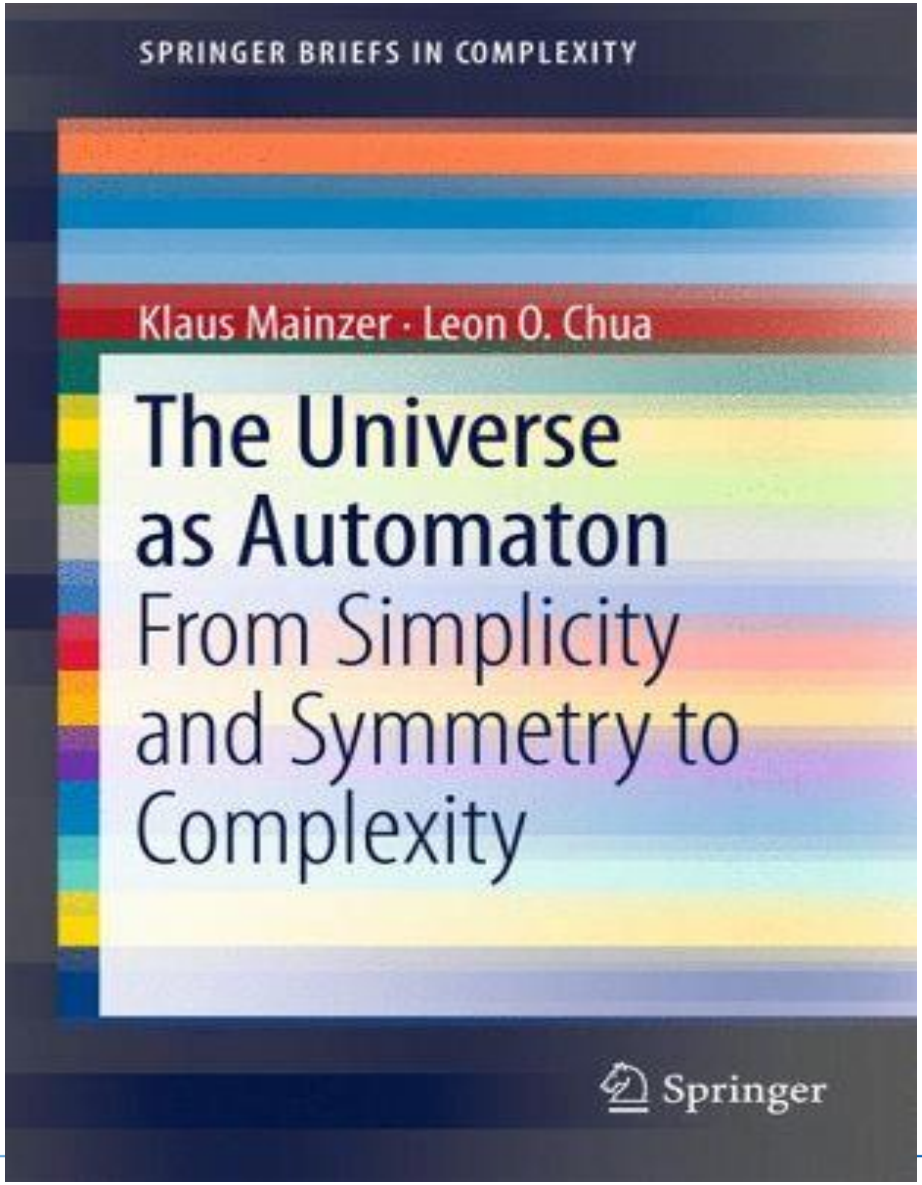
Weyl, H. (1918): *Das Kontinuum. Kritische Untersuchungen über die Grundlagen der Analysis.* De Gruyter: Leipzig.


Grundlehren der mathematischen Wissenschaften 279
A Series of Comprehensive Studies in Mathematics
Errett Bishop
Douglas Bridges
Constructive Analysis
Springer-Verlag Berlin Heidelberg New York Tokyo


bup
BERLIN UNIVERSITY PRESS
KLAUS MAINZER
INFORMATION
ALGORITHMUS - WAHRSCHEINLICHKEIT
KOMPLEXITÄT - QUANTENWELT - LEBEN
GEHIRN - GESELLSCHAFT


The Digital and the Real World
In the 21st century, digitalization is a global challenge of mankind. Even for the public, it is obvious that our world is increasingly dominated by powerful algorithms and big data. But, how computable is our world? Some people believe that successful problem solving in science, technology, and economies only depends on fast algorithms and data mining. Chances and risks are often not understood, because the foundations of algorithms and information systems are not studied rigorously. Actually, they are deeply rooted in logics, mathematics, computer science and philosophy.
Therefore, this book studies the foundations of mathematics, computer science, and philosophy, in order to guarantee security and reliability of the knowledge by constructive proofs, proof mining and program extraction. We start with the basics of computability theory, proof theory, and information theory. In a second step, we introduce new concepts of information and computing systems, in order to overcome the gap between the digital world of logical programming and the analog world of real computing in mathematics and science. The book also considers consequences for digital and analog physics, computational neuroscience, financial mathematics, and the Internet of Things (IoT).
World Scientific
www.worldscientific.com
10583 hc
ISBN 978-981-3225-48-0
Mainzer
The Digital and the Real World
The Digital and the Real World
Computational Foundations of Mathematics, Science, Technology, and Philosophy
Klaus Mainzer
World Scientific


}essentials{
Klaus Mainzer
Wie berechenbar ist unsere Welt
Herausforderungen für Mathematik, Informatik und Philosophie im Zeitalter der Digitalisierung
Springer VS


SPRINGER BRIEFS IN COMPLEXITY
Klaus Mainzer · Leon O. Chua
The Universe as Automaton
From Simplicity and Symmetry to Complexity
Springer


Klaus Mainzer
Die Berechnung der Welt
Von der Weltformel zu Big Data
C.H.Beck

Künstliche Intelligenz – Wann übernehmen die Maschinen?

Jeder kennt sie. Smartphones, die mit uns sprechen, Armbanduhren, die unsere Gesundheitsdaten aufzeichnen, Arbeitsabläufe, die sich automatisch organisieren, Autos, Flugzeuge und Drohnen, die sich selber steuern, Verkehrs- und Energiesysteme mit autonomer Logistik oder Roboter, die ferne Planeten erkunden, sind technische Beispiele einer vernetzten Welt intelligenter Systeme. Sie zeigen uns, dass unser Alltag bereits von KI-Funktionen bestimmt ist.

Auch biologische Organismen sind Beispiele von intelligenten Systemen, die in der Evolution entstanden und mehr oder weniger selbstständig Probleme effizient lösen können. Gelegentlich ist die Natur Vorbild für technische Entwicklungen. Häufig finden Informatik und Ingenieurwissenschaften jedoch Lösungen, die sogar besser und effizienter sind als in der Natur.

Seit ihrer Entstehung ist die KI-Forschung mit großen Visionen über die Zukunft der Menschheit verbunden. Löst die „künstliche Intelligenz" also den Menschen ab? Dieses Buch ist ein Plädoyer für Technikgestaltung: KI muss sich als Dienstleistung in der Gesellschaft bewähren.

ISBN 978-3-662-48452-4

▸ springer.com

Mainzer

Künstliche Intelligenz – Wann übernehmen die Maschinen?

Klaus Mainzer

Künstliche Intelligenz – Wann übernehmen die Maschinen?

Springer