ETH zürich

Turing Centre

6th International Conference on the History and Philosophy of Computing

Zurich, October 27-29, 2021

HaPoC 2021

PROGRAM AND EXTENDED ABSTRACTS

Collegium Helveticum

Swiss National Science Foundation

D GESS

6th International Conference on the History and Philosophy of Computing

Zurich, October 27-29, 2021

PROGRAM AND EXTENDED ABSTRACTS

HaPoC

Collegium Helveticum

Swiss National Science Foundation

D GESS

Contents

6th HaPoC Conference

Zurich, October 27-29, 2021

While computing appears as a technological and scientific field in constant progression, our conception and knowledge of computers are also subject to change over time. In particular, digital machines of the 20th century were inspired by the biological individual, replacing with a solipsistic mental view the cultural and social aspects attached to the image of machines in the 19th century. However, the growing cultural import of computing practices has become ever more pressing in our days in all dimensions of social life. Not only have cultural phenomena increasingly become the object of computational analysis, but computational practices have also proved inseparable from the cultural environment in which they evolve.

Therefore, it is urgent to critically address the entanglement of computing practices with the main cultural challenges of our epoch. The global and collective nature of such problems (e.g. climate change, global pandemics, systemic inequalities, resurgence of totalitarianism, to name a few) requires a comprehensive perspective on computing, where social and cultural aspects occupy a central position. For these reasons, thinking about machines asks today for an interdisciplinary approach, where art is as necessary as engineering, anthropological insights as important as psychological models, and the critical perspectives of history and philosophy as decisive as the axioms and theorems of theoretical computer science.

For more than a decade, the "History and Philosophy of Computing" Conference (HaPoC) has contributed to building such an interdisciplinary community and environment. Continuing this orientation, the HaPoC's 6th edition aims to bring together historians, philosophers, computer scientists, social scientists, designers, manufacturers, practitioners, artists, logicians, mathematicians, each with their own experience and expertise, to take part in the collective construction of a comprehensive image of computing.

Attending HaPoC-6 under Corona Conditions

We are delighted that you can take part in the 6th International Conference on History and Philosophy of Computing at ETH Zurich! As organizers, we are obliged to provide you with information on the possible risks and consequences of taking part. We kindly ask you to read through the following points and to observe the associated requierments:

Before the event:

- Anyone who has been in contact with COVID-19 patients during the last 14 days is not allowed to visit ETH Zurich, and therefore may not attend events.
- If you were diagnosed with COVID-19, you may only come back from self-isolation to ETH Zurich after the decision of the cantonal doctor or treating physician.
- We recommend that high-risk individuals only take part in the event after consulting with their primary care physician.
- If you have any symptoms of COVID-19 (such as fever or feverishness, sore throat, cough, shortness of breath, aching muscles, sudden loss of sense of smell or taste, conjunctivitis, headache, gastrointestinal symptoms, rhinitis), you may not visit ETH Zurich.
- When travelling to and from the event with public transport, please avoid rush hours and comply with the mask-wearing obligation issued by the BAG (Federal Office of Public Health). Inform yourself in good time about the Federal travel regulations. Face coverings must also be worn in the ETH Link shuttlebus.
- **For all events taking place on ETH Zurich premises, a Covid certificate is now mandatory. Please make sure you bring either the paper document or the electronic certificate in the app, along with a form of ID.**

During the event:

- Face masks must be worn inside of all ETH buildings. Please bring your own mask with you.
- A mask must be worn at all times during the event.
- Participants must follow the social distancing and hygiene rules during the event, the written instructions on site (markings, signs) and the verbal instructions of the organisers.

After the event:

- Anyone who falls ill with COVID-19 immediately after the event must inform the organiser.

In attending the event, you confirm that you have noted the information above and you meet the conditions for participation.

Venues

Due to the current pandemic situation, the Conference will take place in a hybrid format, with attendance and contributions both on-site and online.

Main Venue:

The main venue is the Werner Siemens-Auditorium, in the HIT building on the ETH Hönggerberg Campus (see site plan on the opposite page).

The Werner Siemens-Auditorium is located at the following address:

HIT E 51
Building HIT
Wolfgang-Pauli-Str. 27
8093 Zürich
Switzerland

Arrival by public transports:

Bus lines **37**, **69**, or **80** or **ETH eLink** to bus stop "Zürich, ETH Hönggerberg".

The **eLink bus** is a direct shuttle bus commuting between ETH Zentrum campus (located near the main train station of Zurich "Zürich Hauptbahnhof") and ETH Hönggerberg. This service is not on the public transportation map. At ETH Zentrum, eLink stops at the station "Zürich, ETH Polyterrasse" beneath the large terrace next to the ETH main building as well as at "Zürich, Haldenegg". Between 7 am and 7:30 am it also departs from the central train station at the station "Zürich, Bahnhofquai/HB".

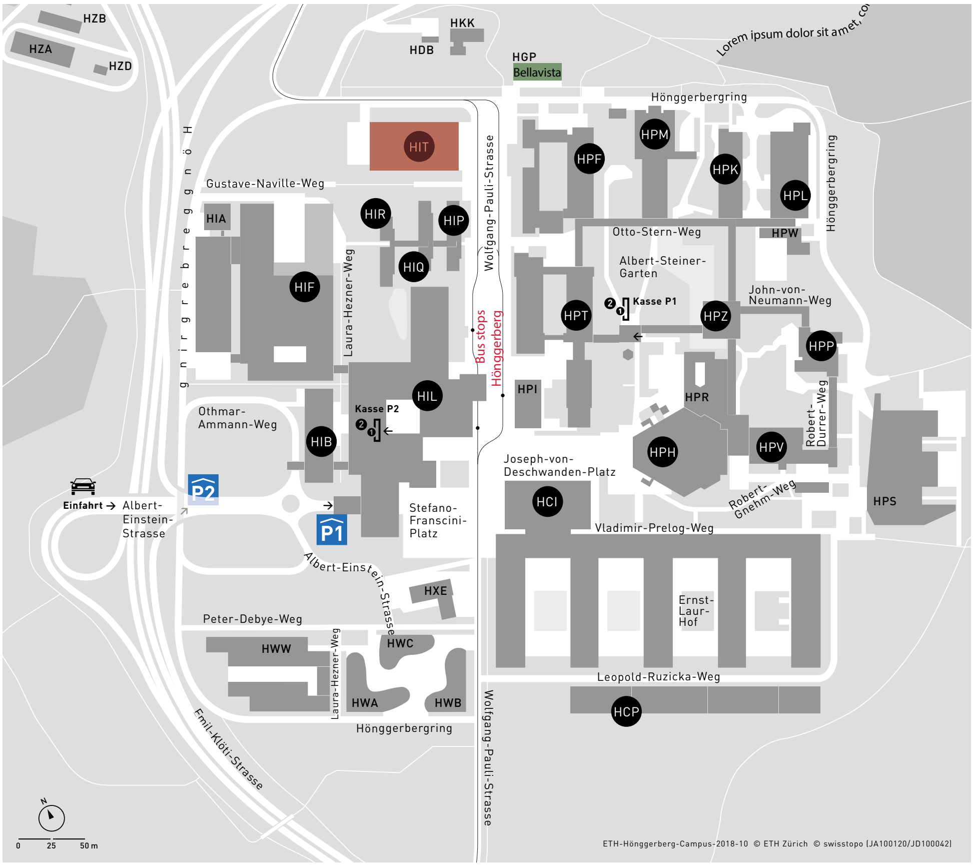
The eLink can be used for free by conference guests of the ETH.

The QR codes in the opposite page will allow you to download both the map of public transportation in Zurich and the ETH eLink timetable from the different stops.

Online Venue:

Online contributions and attendance will take place via zoom. You must be registered to receive all the necessary information.

A gather.town space will be set up during the whole Conference for online coffee breaks and spontaneous meetings between online participants.



Site plan of the ETH Hönggerberg Campus. The Werner Siemens-Auditorium is located in the HIT building (in red). The lunch venue, Bellavista Restaurant, is located a few meters away from the Conference venue (in green).

ETH HIT Building @ Google Maps

Zurich public transportation map

ETH eLink Bus Timetable

Screening Event Venue:

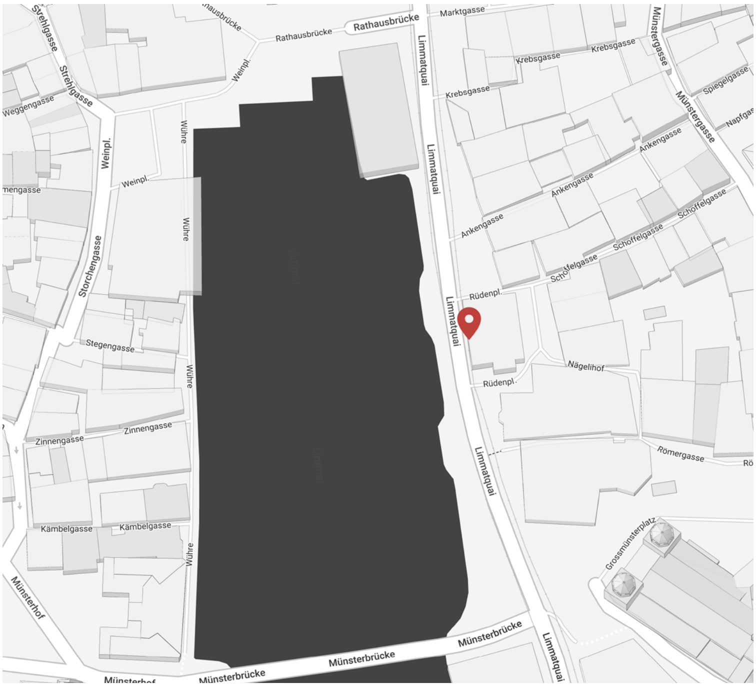
On the first evening, the program feautures a screening event at the Collegium Helveticum (ETH Campus Zentrum, Schmelzbergstrasse 25, 8092 Zürich, Building STW; see map p. 13). The event will be preceded by a reception kindly offered by the Collegium Helveticum.

Lunch Venue:

The venue for the two lunches is the Bellavista Restaurant. The restaurant is located within the Hönggerberg Campus (Hönggerbergring 47) only a few meters away from the Conference venue (see map p. 9).

Conference Dinner:

The venue for the Conference dinner is the traditional restaurant Haus zum Rüden, located at: Limmatquai 42, 8001 Zurich.



Haus zum Rüden
@ Google Maps

General Information

Registration

All on-site and online participants are required to register, filling the corresponding online form, to be found at the official HaPoC 2021 website: hapoc2021.sciencesconf.org.

Registration is mandatory and free of charge.

Instructions for speakers

Duration of contributed talks: 30 minutes (including discussion)
Duration of plenary lectures: 60 minutes (including discussion)

The conference room will be prepared to feautre full hybrid mode. For this reason, on-site presentations will only be possible using the room's computer (Mac). Contributors will be required to transfer their presentation files (in PDF) and check that they work as expected during the break before their session, at the latest. Should you have special requirements concerning your presentation (eg. file formats other than PDF), please let us know as soon as possible.

Online contributions will take place as standard zoom presentations. Please make sure to use the suitable infrastructure (microphone, earphones, stable internet connection - via ethernet cable if possible-, etc.) and to choose an appropriate environment.

Internet

In any ETH building, including the Conference room, you may connect to the *eduroam* network with your own institute's username and password. Please make sure to obtain the username and password in advance, as they may be different from your usual institutional ones.

Registration on the networks *public* and *public-5* is possible for participants who can receive text messages on their phones.

Phone and Data Roaming

Switzerland is not included in the EU roaming zone. If you're coming from the EU, please check with your service provider whether your package includes Switzerland. Local SIMs with data packages can be purchased for 20-30 CHF in all cellular company outlets (central train station, airport train station and many other places city-wide)

Electricity

Switzerland uses a unique plug (type J). You can also use a European two-prong plug (type C), but anything else would require an adapter. Be sure to get one in advance!

You can use your electric appliances in Switzerland if the standard voltage in your country is between 220 - 240 V.

Program

All times are CET (Central European Time = Zurich time)

Wednesday, October 27, 2021

08:00 – 08:50	Registration
08:50 – 09:00	Welcome
09:00 – 10:00	**Keynote: Mireille Hildebrandt** *Written and Coded 'Speech Acts'. Never the Twain Shall Meet?* Chair: Juan Luis Gastaldi
10:00 – 11:00	Session 1 – Conceptual Perspectives – Chair: Viola Schiaffonati **Javier Toscano** *Intentionalities of Code: Historical Practices and Devices. A Philosophical Account* **Edith Schmid** *Computing Systems as Social Institutions*
11:00 – 11:20	Coffee Break
11:20 – 12:50	Session 2 – Physical Aspects of Computing – Chair: Luc Pellissier **Edgar Daylight** *Church's Reception of Turing's 1936 Paper: A Philosophical Angle* **Philipp Macele** *There is no Hardware – Lynn Conway and the Mead-Conway-Revolution* **Michael Jackson** *Cyber-Physical Programming*
12:50 – 14:00	Lunch Break
14:00 – 15:30	Session 3 – Computing and the State – Chair: Liesbeth De Mol **Moritz Feichtinger** *Databasing Djungle-War: The US-Army's Data-Processing Systems During the Vietnam War, 1966-1975* **Moritz Mähr** *The Public, the Private, and the Domestication of the Information System. How Data Protection Governed the Swiss Administration in the 1970s.* **Marcelo Vianna** *"Processing the Development": Technical Groups, Profiles and Decisions on Computer Technologies in Brazil in the Late 1950s*
15:30 – 16:00	Coffee Break

16:00 – 17:30 Session 4 – Conceptual Perspectives – Chair: Giuseppe Primiero

Nick Wiggershaus
An Agential Theory of Implementation for Computer Science

David Waszek
Informational Equivalence but Computational Differences? Herbert Simon on Representations in Scientific Practice

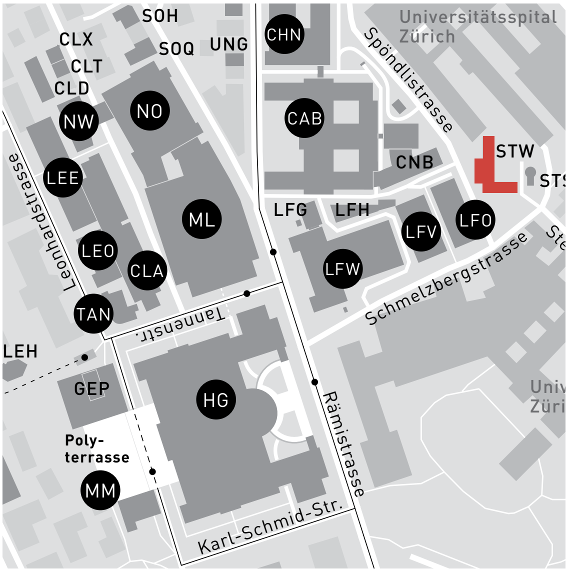
Philippos Papayannopoulos, Nir Fresco and Oron Shagrir
On Two Different Kinds of Computational Indeterminacy

17:30 – 18:30 HaPoC General Assembly – Chair: Giuseppe Primiero
All participants are kindly invited to attend

Venue change: *Collegium Helveticum, Schmelzbergstrasse 25, 8006 Zürich*

19:30 – 21:30 Screening Event: *Knit's Island*

Ekiem Barbier, Guilhem Causse and Quentin L'helgoualc'h
Excerpt of a work in progress video game documentary followed by a round table and discussion with the directors. Moderator: J. L. Gastaldi. The screening will start at 20:15, and will be preceded by an apéro. This event will take place in person only (no online participation).x



Collegium Helveticum
@ Google Maps

Site plan of the ETH Hönggerberg Campus. The Collegium Helveticum is located in the STW building (in red).

Thursday, October 28, 2021

08:30 – 09:00	Check-in
09:00 – 10:30	Session 5 – History and Philosophy – Chair: Simone Martini
	Michael Friedman *Leibniz on Stocking Frame, Computing and Weaving*
	Aggelos Biboudis, Jeremy Gibbons and Oleg Kiselyov *All Things Flow: Unfolding the History of Streams*
	Tomas Petricek and Joel Jakubovic *Complementary Science of Interactive Programming Systems*
10:30 – 10:50	Coffee Break
10:50 – 12:50	Session 6 – Social History – Chair: Roy Wagner
	Mate Szabo *The Early Days of the Hungarian Software Industry*
	Jelena Stanulovic *Influence of the Self-Management in the Development of Personal Computers in Socialist Yugoslavia During the '80s*
	Carmen Flury and Rosalía Guerrero *A National School Computer for the Emerging Digital Society: East German and Swedish Efforts to Develop a State-Mandated Educational Computer in the 1980s-1990s*
	Mirjam Mayer and Ricky Wichum *Public Data and Personal Computers. The Emergence of a Personal Computing Culture in the Swiss Federal Administration, ca. 1980*
12:50 – 14:00	Lunch Break
14:00 – 15:00	**Keynote: Thomas Haigh** *Becoming Universal: A New History of Modern Computing* Chair: Liesbeth De Mol
15:00 – 16:00	Session 7 – Artificial Intelligence – Chair: Viola Schiaffonati
	Camilla Quaresmini and Giuseppe Primiero *Data Quality Dimensions for Fair AI*
	Robin Zebrowski, John Sullins, Eric Dietrich, Bram Van Heuveln and Chris Fields *The History and Legacy of the AI Wars*
16:00 – 16:30	Coffee Break

16:30 – 17:30	Session 8 – Socioeconomic Perspectives – Chair: Juan Luis Gastaldi
	Veronica Dahl
	Doughnut Computing: Aiming at Human and Ecological Well-Being
	Chris Holland
	Slipping Through Our Fingers Even As We Tighten Our Grip on the Controller: Rentism and the Consequences for Video Games
17:30 – 18:30	**Keynote: Barbara Liskov**
	Reflections on Programming Methodology
	Chair: Simone Martini
19:30	Conference Dinner
	Venue: *Haus zum Rüden, Limmatquai 42, 8001 Zürich (see map p. 10)*

Friday, October, 29 2021

08:00 – 08:30	Check-in
08:30 – 10:00	Session 9 – Machine Learning – Chair: Viola Schiaffonati
	Jens Ulrik Hansen and Paula Quinon
	The Role of Expert Knowledge in Big Data and Machine Learning
	Maximilian Noichl
	The Epistemic Vices and Virtues of Dimensionality Reduction
	Stefan Trausan-Matu
	A Poststructuralist Perspective on Computer-Generated Literature
10:00 – 10:20	Coffee Break
10:20 – 11:50	Session 10 – History and Philosophy – Chair: Luc Pellissier
	Troy Astarte
	From Monitors to Monitors: an Early History of Concurrency Primitives
	Cliff Jones
	One Concurrent Program: Three Attempts at Its Formal Verification
	Laura Fontanella
	The Evolution of the Concept of Proof Through Realizability
11:50 – 12:50	**Keynote: Juliette Kennedy**
	On the Semantic Content of Gödel's 1931 Proof of the Incompleteness Theorem
	Chair: Giuseppe Primiero
12:50 – 14:00	Concluding Remarks and Farewell

Extended Abstracts

Keynotes

Thomas Haigh (University of Wisconsin-Milwaukee)

Becoming Universal: A New History of Modern Computing

Thursday, Oct 28, 14:00 – 15:00

Thomas Haigh will discuss the new overview history of computing he has written with Paul Ceruzzi, exploring the challenges involved in producing a coherent and comprehensive synthetic history of computing and the choices and trade-offs the authors made during the process. The biggest challenge is the remarkable flexibility of the electronic digital computer, which from 1945 to 2020 has evolved from a specialized and hugely expensive technology used for scientific computation to an inexpensive and ubiquitous technology embedded into devices of all kinds and used in almost every human activity. From this viewpoint, the computer became an (almost) universal machine only incrementally, a contrast with the theoretical perspective from which even the simplest programmable devices are often equated with Turing's Universal Machine.

Existing overview histories have not fully engaged with now-central topics such as the Internet, smartphones and mobile apps, cloud computing, video games, digital media, or automotive computing. The era saw repeated shifts in users, producers, applications and affordances which make it hard to construct a coherent narrative. The new structure addresses this by basing each chapter around a specific group of users and applications, around which "the computer" is remade with the addition of new capabilities such as interactivity or graphical communication. Given HAPOC's focus on more technical analysis, he will give particular attention to efforts made in the new book to weave discussion of computer architecture and system software into these application-led stories.

Other challenges explored during the talk include the balancing of technical detail against narrative, presentation of the computer's continuities and discontinuities with other technologies, incorporation of perspectives from social and cultural history into a technology-centered story, and engagement with an increasingly broad and diverse secondary literature. Haigh explains the decisions made by the book's authors and their relationship to its multiple intended audiences.

Mireille Hildebrandt (Vrije Universiteit Brussel)
Written and Coded 'Speech Acts'. Never the Twain Shall Meet?

Wednesday, Oct 27, 9:00 – 10:00

In this keynote I will inquire into the difference that makes a difference (Bateson) between text- and code-driven 'speech acts'. I will raise some pivotal questions regarding the relationship between counting, speech, calculation and qualification, while comparing the performative effect of written speech acts with the real-world effects of computing systems. My argument is part of a call for legal philosophers to pay keen attention to computer science and philosophy of technology, and a call to computer scientists to pay keen attention to philosophy of language, more notably speech act theory and interpretation theory. The focus will be on how the use and the affordances of natural language inform the fine line between ambiguity and shifts in meaning on the one hand and continuity and closure on the other. This will allow me to highlight where written and coded 'speech acts' differ and how their use may be complementary in a way that reinforces rather than diminishes human agency.

Barbara Liskov (MIT Computer Science & Artificial Intelligence Lab)
Reflections on Programming Methodology

Thursday, Oct 28, 17:30 – 18:30

Research in programming methodology led to the development of the principles and methods that underlie how modern software systems are designed and structured. At the center of this work are the notions of abstraction and modularity. These ideas are related: design is the process of inventing and identifying abstractions, and the implementations of the abstractions become the modules that make up the program. This talk will discuss our current understanding of abstraction and modularity and the research that got us to where we are today.

Juliette Kennedy (University of Helsinki)
On the Semantic Content of Gödel's 1931 Proof of the Incompleteness Theorem

Friday, Oct 29, 11:50 – 12:50

Gödel's 1931 proof avoids semantic notions, and yet there are residues of these in the proof. We examine the semantic aspects of the 1931 proof in light of Gödel's general views at the time and later on. Kripke argues that the negative solution of the Entscheidungsproblem follows almost immediately from Gödel [1931]. So why didn't Gödel solve the *Entscheidungsproblem?*

Screening Event

Ekiem Barbier, Guilhem Causse and Quentin L'helgoualc'h (Les Films Invisibles)
***Knit's Island* (work in progress video game documentary)**
Wednesday, Oct 27, 20:15 – 21:30 | Venue: Collegium Helveticum

"Knit's Island" is a documentary about the discovery of a neighboring world, almost fused with ours. Under the guise of avatars, a film crew enters online video games and comes into contact with players. Who are these inhabitants? Are they really playing? Through the encounters with these characters, the discovery of their fears and aspirations and the craft of their imaginary worlds, this film reveals a different perspective of the virtual and questions of our world's becoming.

Knit's Island (E. Barbier, G. Causse and Q. L'helgoulc'h, *Les Films Invisibles*)

Contributed Talks

Troy Astarte (Swansea University)

From Monitors to Monitors: an Early History of Concurrency Primitives

Session 10 | Friday, Oct 29, 10:20 – 10:50

This talk is an early output from the author's current project on the history of concurrency and sketches the early search for programming primitives. Starting by looking briefly at the management of peripherals that drove initial work, the talk covers algorithms, semaphores, conditional critical regions, and monitors.

Different parts of computers operate at different speeds. As soon as computer systems were sufficiently complex, managing these differentials efficiently became a serious concern. This was tackled as far back as 1955: Rochester wrote about 'multiprogramming', referring to the ability of a central calculator to share focus between polling I/O and processing data (Rochester 1955). Gill used the terms 'parallel programming' and 'timesharing' in the context of interference between multiple CPUs, or multiple programs sharing a CPU (Gill 1958). The basic challenge was to handle the spread of processing across space or time and it was realised these could be treated as the same problem (Conway 1963).

An early approach was to use the system controlling the computer to manage synchronisation directly (Codd 1962). This 'monitor' (as Brinch Hansen called it) could be hard to use and understand, with its workings hidden away inside assembly programs. By Codd's 1962 summary paper key concepts such as threading, shared variables, critical sections, and mutual exclusion were already identified. The prime challenge: preventing the interaction of critical resources, and unexpected outcomes from shared memory.

The early 60s was the era of algorithms and many appeared for managing concurrent processes. The core idea here is conditional progress: Dekker's algorithm (reported in (Dijkstra 1962)) arbitrates between competing processes by alternating priority in turn. The algorithm did not generalise easily to multiple processes; Lamport's Bakery algorithm was more successful but the complexity of algorithms and proofs thereof drove a search for something simpler.

Taking a metaphor from railways, Dijkstra proposed a special kind of shared integer called a 'semaphore' (Dijkstra 1962). These synchronisation primitives controlled access to critical sections using two operations, P and V (the full names he gave varied). Semaphores allowed more concise programming, and put control into the hands of programmers. However, deadlocks loomed for certain problems (for example the classic dining philosophers).

From the early 1970s, Dijkstra, Hoare, and Brinch Hansen were all searching for ways to improve programming for concurrency. The three fed on each others' work and discussions, making it difficult (and unhelpful) to apportion individual credit. This was the age of structured programming and Pascal: 'elegance' became not just an aesthetic imperative but a correctness one. Working towards a concept, Hoare proposed 'critical regions': areas of code marked off for mutual exclusion over a shared resource, sometimes with conditions for entry (Charles Antony Richard Hoare 1972). The title of this paper ('Towards a theory of parallel programming') showed Hoare's desire to generalise from a practical synchronisation primitive: part of that time period's fascination with theorising computation.

Both Brinch Hansen and Dijkstra provided ideas for improving this notion; Dijkstra's 'secretary' is worth noting for its 1970s attitude towards workplace gender roles (Dijkstra 1971). Simula-67's class concept provided the key: encapsulation. Shared resources were grouped together with the operations that could access them in a construct with mutual exclusion baked in, called a 'monitor' (C. A. R. Hoare 1974). Inbuilt queues and signalling operations controlled waiting. Monitors took the responsibility for correctly managing synchronisation away from individual processes and made a system more reliable overall— they also obeyed the hallowed principles of structured programming.

Monitors became the sole construct for managing concurrency in a few programming languages released in the 1970s; Concurrent Pascal and Modula are of particular note. These demonstrated the workability of the monitor as a realistic synchronisation primitive; unfortunately implementation exposed underlying complexity. Recursive or interfering monitor calls were difficult to get right and a proper understanding of the queueing system was essential. The monitor idea declined in popularity towards into the 80s as communication became the new focus for concurrency. However, the extensive remarks at the end of (Hansen 1996) show that monitors had a serious impact on concurrency and their principles continue to influence object-oriented programming to this day.

The search for a useful concurrency primitive demonstrates an example of collaborative ideation, with ideas bouncing between groups and individuals. The changing scope and focus also reflects changing agendas for computing: from systems and assembly programming towards high-level languages, abstraction, and the lofty ideals of structured programming. The work paved the way for increased focus on formalisation and theory-building.

Acknowledgements

This work is supported by Leverhulme Trust grant RPG-2019-020. Thanks to Cliff Jones for providing comments.

References

Codd, EF. 1962. “Multiprogramming.” In *Advances in Computers*, edited by F. Alt and M. Rubinoff, 3:77–153. Elsevier.

Conway, Melvin E. 1963. “A Multiprocessor System Design.” In *Proceedings of the November 12-14, 1963*, Fall Joint Computer Conference, 139–46.

Dijkstra, E. W. 1962. “Over de Sequetialiteit van Procesbeschrijvingen [on the Sequentiality of Process Descriptions].” Circulated privately, available in Texas Archive. https://www.cs.utexas.edu/users/EWD/translations/EWD35-English.html.

Dijkstra, E. W. 1971. “Hierarchical Ordering of Sequential Processes.” *Acta Informatica* 1: 115–38.

Gill, Stanley. 1958. “Parallel Programming.” *The Computer Journal* 1 (1): 2–10.

Hansen, Per Brinch. 1996. “Monitors and Concurrent Pascal: A Personal History.” In *History of Programming Languages—II*, edited by Thomas J. Bergin and Richard G. Gibson, 121–72. New York, NY, USA: ACM Press.

Hoare, C. A. R. 1974. “Monitors: An Operating System Structuring Concept.” *Communications of the ACM 17* (10): 549–57.

Hoare, Charles Antony Richard. 1972. “Towards a Theory of Parallel Programming.” In *Operating System Techniques*, edited by C. A. R. Hoare and R. H. Perrott, 61–71. Academic Press, New York.

Rochester, Nathaniel. 1955. “The Computer and Its Peripheral Equipment.” In *Proceedings of the Eastern Joint AIEE-IRE Computer Conference: Computers in Business and Industrial Systems*, 64–69.

Aggelos Biboudis, Jeremy Gibbons and Oleg Kiselyov (Swisscom AG, University of Oxford, Tohoku University)

All Things Flow: Unfolding the History of Streams

Session 5 | Thursday, Oct 28, 9:30 – 10:00

Facets of Streams

Over several decades programming languages have shifted away from the sequential programming model that the von Neumann architecture so vigorously imposed (Backus 1978b). Nowadays, instead of *commands* and static *storage*, programmers often have to think in terms of processes and (high-performance) transformations over streams. By streams we, in computer science, mean a sequence of elements that can be piped through a series of transformation steps. Characteristically, data is accessed in strict sequence rather than in a random access pattern. In return for limited expressiveness, we gain the opportunity to process large amount of data efficiently, in limited space.

Streams for data processing in sublinear space.

One of the most recognizable examples of streams is Unix pipes (Ritchie 1980; Ritchie and Thompson 1978), instigated by McIlroy. He recorded his vision on a yellowed sheet of paper kept pinned to his office wall (McIlroy 1964): "We should have some ways of connecting programs like garden hose – screw in another segment when it becomes necessary to massage data in another way. This is the way of IO also." This image of a 'hose', carrying, and transforming potentially infinite amount of data is evocative of all streaming libraries. A library user, like in assembling a hose, gets to declare *what* activities take place but not how the data access is scheduled. We can trace this thread back at least to Conway's design (Conway 1963) for a one-pass COBOL compiler, whose modules are like segments of a hose.

This massaging – analyzing and transforming – large amounts of data is the domain of the traditional database management (see §3), which has evolved into so-called *data analytics*: performing Extract–Transform–Load jobs (ETL) over centralized architectures called *data lakes*. ETL consists of moving data from heterogeneous sources (E) to other targets (L), after several transformation steps (T). Streams are promoted to first-class status (Shapira 2017).

Streams as event processing and correlation.

Information flow processing is the other class of stream processing applications.

It is defined as "processing continuously flowing data from geographically distributed sources at unpredictable rates to obtain timely responses to complex queries" (Cugola and Margara 2012). Its characteristic example is *complex event processing*: given incoming notifications of events observed by sources (e.g., sensor readings), “filter and combine such notifications to understand what is happening in terms of higher-level events to be notified to sinks, which act as event consumers.” Intrusion detection, environment monitoring, and online analysis of stock prices are clear examples. A less obvious example is complex user-interfaces such as modern GUI and multimedia applications.

Information flow processing thus encompasses dataflow, reactive and signal processing – often captured under the name of *stream processing systems* (Stephens 1997). It also has deep roots to Kahn process networks (Kahn 1973); networks where concurrent processes communicate only through one-way FIFO channels with unbounded capacity. Dataflow networks are usually considered a special case of Kahn networks. The first dataflow language, Lucid (Wadge and Ashcroft 1985), appeared in 1970s. The two modes of executions arise: data driven (eager evaluation) where computation depends on the availability of data; and demand driven (lazy evaluation) where the agents request data from their inputs. These terms are commonplace in programming language semantics, modern streaming libraries and more.

Streams to capture the semantics of I/O.

Landin (Landin 1965), in 1965, was the first to observe that a denotation of ALGOL-60’s for-statements with for-lists leads to a peculiar list processing where the items of an intermediately resulting list “never exist simultaneously.” Noting the similarity with Conway’s approach, he coined the term “stream”. Much later streams have been used to give the ‘pure functional’ semantics of I/O in early Haskell (Hudak et al. 2007).

The popular ‘trace semantics’ is also based on streams (Jeffrey and Rathke 2011).

We connect the dots between lists (and their constructors), streams (and their destructors), recursion over finite lists, and co-recursion over infinite streams (Rutten 2003).

Streams as iteration abstractions.

In order to understand the nature of streams as programming abstractions for *iteration*, we have to take a good look at the past. We trace the lineage of streams from FORTRAN (Backus 1978a) and IPL (Newell 1963a) to LISP (McCarthy 1978), Common LISP (Steele 1990) and Clojure (Hickey 2020). The former already include powerful mechanisms for streaming computations: sequences, streams, loops and series. We also look at the early abstractions of these mechanisms: iterators in CLU (Liskov et al. 1977), a form of generators as in IPL-V (Newell 1963b) and Alphard (Shaw, Wulf, and London 1977). Alphard in particular was aimed at the development of verifiably reliable software. Abstraction was indispensable in simplifying and modularizing Hoare-style proofs. Since iterative computations invariably loop over the elements of some collection (be it as simple as a range of integers), it made sense to separate operations on the current element (the loop body) from obtaining the next element (the loop control). The goal was to hide the details of the collection, encapsulate the state of the enumeration, and separate concerns in the proof.

These abstractions appear again and again, from modern C++, for example, offering a multitude of iterators explicitely or even implicitly through “smarter” for-loops, to C# and Python’s first-class support for generators. We revisit the historical milestones that gave shape to the semantics of streaming APIs from the perspective of programming language abstractions, and we argue that the same patterns emerge in other disciplines of computer science. We believe that a concise and focused study of the history of streams will guide the engineer in navigating the design space effectively. In the full paper we complement the conceptual discussion with an analysis of how streams took a form as a linguistic construct in several programming languages.

Motivating Example 1: Streaming APIs

Java 8 introduced lambdas (i.e., anonymous functions) with the explicit purpose of enabling streaming abstractions, which present an accessible, natural path to multicore parallelism – perhaps the highest-valued domain in current computing. Other languages, such as Scala, C#, and F#, also support lambda abstractions and streaming APIs, making streams a central theme of their approach to parallelism. Although the specifics of each API differ, there is a core of common features and near-identical best practices for users of these APIs in different languages. However, with a closer look, we can also identify key differences in the designs (Biboudis, Palladinos, and Smaragdakis 2014). For example, Java 8 introduced streams in a different way from other ecosystems, drawing inspiration from Lisp and Smalltalk. We will discuss the two topics of external versus internal iteration as the two dual dataflow representations that appear in the literature of programming languages and systems, two terms that appear frequently under different names: external vs. internal, pull vs. push and data-driven vs demand-driven.

Finally, what is a representation of streams that fully captures the generality of streaming pipelines and allows desired optimizations? To understand how the representation affects implementation and optimization choices, we review past efforts in deforestation (Wadler 1988) and stream fusion (Coutts, Leshchinskiy, and Stewart 2007).

Motivating Example 2: Database Systems

Streaming libraries and query engines in database systems share many ideas. Both areas offer a high level API, describing low-level operations over data that needs to be executed efficiently. A query expression is typically translated by a query engine into a physical plan of execution. These plans consist of (physical) operators that consume data stored in tables, in a streaming fashion using composed iterators over tuples. A database query optimizer, after cost analysis and reordering of operators, determines the most efficient physical plan. This plan, akin to a pipeline in streaming libraries, specifies how to traversing data efficiently without accumulating intermediate values. Query optimization relies on the same core idea of a dataflow representation (Pirk, Giceva, and Pietzuch 2019). For example, the dominant in the 90's Volcano model (Graefe and McKenna 1993; Graefe 1994) implements the iterator model (external iteration). In 2011, Thomas Neumann (Neumann 2011) proposes an alternative model where "instead of pulling tuples up, we push them towards the consumer operators" – which has inspired the prominent framework for big data, Apache Spark (Agarwal, Liu, and Xin 2016).

This work unfolds the history of iteration and streams, following the lineage from mathematics to other sub-disciplines of computer science. We start from Brouwer's intuitionistic analysis. We trace streams as a mathematical abstraction for the construction of infinite sequences and argue that it was Brouwer who fully grasped streams. He was handicapped however by the lack of suitable notation, which only came to light with coinduction. Tracing these foundations all the way until the present-time, we lay the ground for an interesting, holistic discussion over the principles behind such a pervasive topic.

References

Agarwal, Sameer, Davies Liu, and Reynold Xin. 2016. "Apache Spark as a Compiler: Joining a Billion Rows Per Second on a Laptop." https://databricks.com/blog/2016/05/23/apache-spark-as-a-compiler-joining-a-billion-rows-per-second-on-a-laptop.html.

Backus, John. 1978a. "The History of Fortran I, II, and III." In *History of Programming Languages*, edited by Richard L. Wexelblat, 25–74. Association for Computing Machinery. https://doi.org/10.1145/800025.1198345.

Backus, John. 1978b. "Can Programming Be Liberated from the Von Neumann Style?: A Functional Style and Its Algebra of Programs." Commun. ACM 21 (8): 613–41. https://doi.org/10.1145/359576.359579.

Biboudis, Aggelos, Nick Palladinos, and Yannis Smaragdakis. 2014. "Clash of the Lambdas." In *Proc. 9th International Workshop on Implementation, Compilation, Optimization of Object-Oriented Languages, Programs and Systems*. ICOOOLPS '14. https://arxiv.org/abs/1406.6631.

Conway, Melvin E. 1963. "Design of a Separable Transition-Diagram Compiler." *Commun. ACM 6* (7): 396–408. https://doi.org/10.1145/366663.366704.

Coutts, Duncan, Roman Leshchinskiy, and Don Stewart. 2007. "Stream Fusion: From Lists to Streams to Nothing at All." In *Proc. Of the 12th ACM SIGPLAN International Conference on Functional Programming*, 315–26. ICFP '07. Freiburg, Germany: ACM. https://doi.org/10.1145/1291151.1291199.

Cugola, Gianpaolo, and Alessandro Margara. 2012. “Processing Flows of Information: From Data Stream to Complex Event Processing.” *ACM Comput. Surv.* 44 (3). https://doi.org/10.1145/2187671.2187677.

Graefe, Goetz. 1994. “Volcano: An Extensible and Parallel Query Evaluation System.” *IEEE Trans. On Knowl. And Data Eng.* 6 (1): 120–35. https://doi.org/10.1109/69.273032.

Graefe, Goetz, and William J. McKenna. 1993. “The Volcano Optimizer Generator: Extensibility and Efficient Search.” In *Proc. Of the 9th International Conference on Data Engineering*, 209–18. ICDE ’93. IEEE Computer Society.

Hickey, Rich. 2020. “A History of Clojure.” *Proceedings of the ACM on Programming Languages* 4 (HOPL). https://doi.org/10.1145/3386321.

Hudak, Paul, John Hughes, Simon Peyton Jones, and Philip Wadler. 2007. “A History of Haskell: Being Lazy with Class.” In *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages*, 12-1-12-55. HOPL III. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/1238844.1238856.

Jeffrey, Alan, and Julian Rathke. 2011. “The Lax Braided Structure of Streaming I/O.” In *Computer Science Logic*, edited by Marc Bezem, 12:292–306. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik. https://doi.org/10.4230/LIPIcs.CSL.2011.292.

Kahn, Gilles. 1973. “A Preliminary Theory for Parallel Programs.” Research Report R0006. https://hal.inria.fr/inria-00177890.

Landin, Peter. 1965. “Correspondence Between ALGOL 60 and Church’s Lambda-Notation: Part i.” *Commun*. ACM 8 (2): 89–101. https://doi.org/10.1145/363744.363749.

Liskov, Barbara, Alan Snyder, Russell Atkinson, and Craig Schaffert. 1977. “Abstraction Mechanisms in CLU.” *Commun*. ACM 20 (8): 564–76. https://doi.org/10.1145/359763.359789.

McCarthy, John. 1978. “History of LISP.” *SIGPLAN Not*. 13 (8): 217–23. https://doi.org/10.1145/960118.808387.

McIlroy, Doug. 1964. “Advice.” https://www.bell-labs.com/usr/dmr/www/mdmpipe.html.

Neumann, Thomas. 2011. “Efficiently Compiling Efficient Query Plans for Modern Hardware.” *Proc. VLDB Endow*. 4 (9): 539–50. https://doi.org/10.14778/2002938.2002940.

Newell, Allen, ed. 1963a. *IPL-v Programmers Reference Manual*. Santa Monica, CA: RAND Corporation.

Newell, Allen, ed. 1963b. "Documentation of IPL-V." *Commun*. ACM 6 (3): 86–89. https://doi.org/10.1145/366274.366296.

Pirk, Holger, Jana Giceva, and Peter R. Pietzuch. 2019. "Thriving in the No Man's Land Between Compilers and Databases." In *9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2019/papers/p91-pirk-cidr19.pdf.

Ritchie, Dennis M. 1980. "The Evolution of the Unix Time-Sharing System." In *Language Design and Programming Methodology: Proceedings of a Symposium Held in Sydney, Australia, 10–11 September, 1979*, edited by Jeffrey M. Tobias, 25–35. Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-09745-7_2.

Ritchie, Dennis M., and Ken Thompson. 1978. "The UNIX Time-Sharing System." *The Bell System Technical Journal* 57 (6): 1905–29.

Rutten, J. J. M. M. 2003. "Behavioural Differential Equations: A Coinductive Calculus of Streams, Automata, and Power Series." *Theoretical Computer Science* 308 (1-3): 1–53. https://doi.org/10.1016/S0304-3975(02)00895-2.

Shapira, Gwen. 2017. "The Future of ETL Isn't What It Used to Be." https://www.confluent.io/blog/the-future-of-etl-isnt-what-it-used-to-be/.

Shaw, Mary, William A. Wulf, and Ralph L. London. 1977. "Abstraction and Verification in Alphard: Defining and Specifying Iteration and Generators." *Commun*. ACM 20 (8): 553–64. https://doi.org/10.1145/359763.359782.

Steele, Guy. 1990. *Common LISP: The Language*. Elsevier.

Stephens, Robert. 1997. "A Survey of Stream Processing." *Acta Informatica* 34 (7): 491–541. https://doi.org/10.1007/s002360050095.

Wadge, William W., and Edward A. Ashcroft. 1985. *LUCID, the Dataflow Programming Language*. Academic Press Professional, Inc.

Wadler, Philip. 1988. "Deforestation: Transforming Programs to Eliminate Trees." *Theor. Comput. Sci.* 73 (2): 231–48. https://doi.org/10.1016/0304-3975(90)90147-A.

Veronica Dahl (NSERC, Simon Fraser University)

Doughnut Computing: Aiming at Human and Ecological Well-Being

Session 8 | Thursday, Oct 28, 16:30 – 17:00

Introduction

Computing Sciences evolved from social organizations with unequal power distribution, based on forced hierarchies partly justified by the dualist idea, held by philosophical tradition from Plato to Descartes, that humans are separate from and superior to "nature". This idea was extended and leveraged by dominant élites to cheapen or degrade the "others" (nature, women, non-whites, resource-rich countries...) which enabled them to take from others much more than is given back.

In coherence with this mindset, and despite its many socially useful results, CS has enabled governments and corporations to use data collection, statistics and algorithms as instruments to preserve and deepen an unequal status quo which is destroying the living world and pauperizing and marginalizing women, racialized people, diversities and former colonies– in short, those whose unpaid or underpaid work and resources support the dominant groups many-fold.

The ways in which ending destructive patterns of extraction, exploitation and waste are crucial to both equity and planetary well-being, as well as ways of ending them, have been analysed elsewhere, most notably in (Hickel 2021; Raworth 2017). (Raworth 2017) offers in particular an eloquent, holistic way of measuring where we are vs. where we need to be. Here we propose a new, Doughnut-inspired branch of CS that would explicitly adopt the goal of enabling social and ecological well-being, in order to help advance this goal bottom-up, city by city, to pave the way towards top-down, internationally coordinated action.

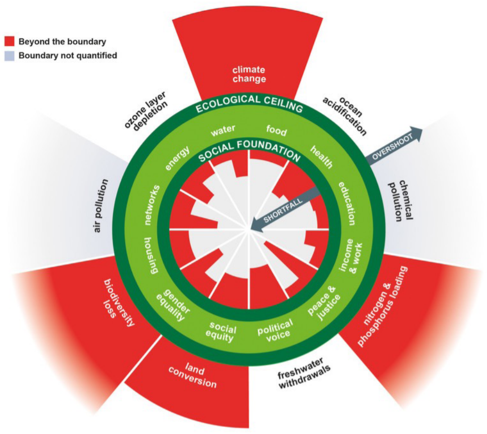
From Domination to Solidarity

Archaeological findings have uncovered a history of enduring peace and equity periods in both nomadic and agrarian ancient civilizations, suggesting that warfare, male dominance and rigid social stratification only began to appear a few millennia ago (Fry 2013). There is a priori no reason why humanity should remain stuck in domination mindsets.

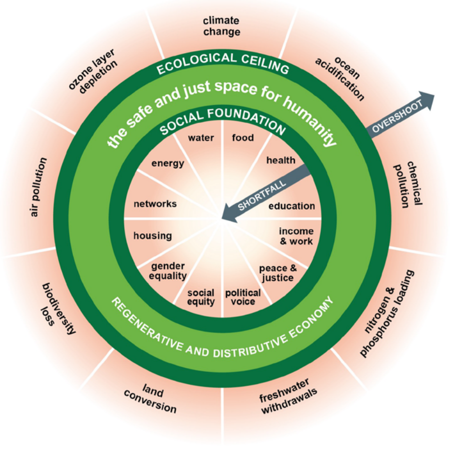
Computing Science can greatly help while we transition into cooperative mindsets and universal solidarity, through its ability to explore different possible scenarios, once given the relevant data plus spelled-out ways of combining it. While the political will to use CS thus may still be lacking, the growing evidence that growing inequity means growing universal danger might well turn the tables. In any case, the mere exploration of equity-seeking possibilities represents a major contribution to society that CS is in privileged position to make.

The Doughnut compass: Measuring the goal vs. the status-quo

Kate Raworth has translated global data on human rights indicators and planetary limits into a single visualization, shaped like a Doughnut (shown in Fig. 1, 2). Red areas inside the inner circle represent lacks in human rights. Red areas outside the Doughnut represent planetary rights transgressions. Bringing us into the safe space for humanity and the natural world (the green "dough" of the Doughnut) would show as Fig. 2. Various cities, e.g. Amsterdam, Nanaimo, Portland, have explicitly adopted the Doughnut as their transformational compass.



Where we are("DEAL," n.d.)



Our goal("DEAL," n.d.)

Doughnut Computing

Definition

Doughnut Computing is the set of computing science methodologies and applications designed to analyse and evaluate which combination of policies and actions can better lead to transforming a city's current Doughnut depiction into one that maximizes the city's positive impact on human rights and on planetary well-being. By human rights we understand those included in the UN 2015 Sustainable Development Goals; by planetary limits, those identified by the international group of earth scientists led by Johan Rockström and Will Steffen.

Computational Methodology

As the main programming methodology, we propose inferential programming because it can trace explanation, make inferences, accommodate change and consultation flexibly, and also permits exploring hypothetical scenarios (Christiansen and Dahl 2005).

Example

As proof of concept, and to illustrate the expressivity of our tools, consider the following non-deterministic Prolog grammar, whose "sentences" are all possible sets of actions capable of reducing an amount X (representing e.g. CO2 emissions in tonnes per capita, or waste per capita, etc. in a certain city or region) to zero or less. Comments are preceded "%".

```
actions(_,Status) --> {Status=<0, !}, [].
                              % goal reached
actions(N,_) --> {number_of_actions(Max), Max=N, !, fail}, [].
                              % ran out of actions
actions(N,Status) --> {N1 is N+1, action(N1,A,QA), Status1 is
                Status-QA},[action(A,QA)], actions(N1,Status1).
                              % adds Nth action
actions(N,Status) --> {N1 is N+1}, actions(N1,Status).
                              % skips Nth action
```

For instance, if we add the following 6 possible actions capable of reducing CO2 emissions in the amounts of tonnes stated in each:

```
action(1,ration_air_and_car_travel, 8).
action(2,reduce_military_spending, 7).
action(3,sequester_carbon, 6).
action(4,end_planned_obsolescence, 4).
action(5,prohibit_most_plastic, 3).
action(6,four_day_work_week, 1).
number_of_actions(6).
```

we can now call it to generate all six combinations of actions capable of reducing emissions initially amountint to 15 tonnes to zero, by setting N to 6 and the initial value of Status to 15. Here we show only the first solution:

```
[action(2,reduce_military_spending,7),
action(1,ration_air_and_car_travel,8)]
```

Now we can add a cost/benefit analysis of whatever complexity we need to define, using CHR constraints. For instance,if reducing military spending leaves 300 thousand people unemployed, we can add this consequence through declaring it as a CHR constraint and amending the second of the actions into:

```
action(2,reduce_military_spending, 7):- unemployed(2,300).
```

This being a constraint, it is thrown into the constraint bag to be picked up whenever calculations on any other part of the Doughnut leave a relevantly related consequence. E.g. if basic social services needed to ensure equity require, say, 400 thousand new employees, the following CHR rule can resolve both situations simultaneously:

```
unemployed(N,300), requires(M,400) <=> repurpose(300,N,M),
                                         requires(M,100).
```

This says that the two constraints on the left can be replaced by the new constraint that 300 thousand people must be repurposed from activity N into activity M, and that activity M now requires 100 thousand employees (the actual rule would be more general, of course, with parameters where we now have concrete values). In turn, the new constraint of repurposing employees might need further fine-grained planning, which can likewise be expressed in terms of constraints.

Thus all possible scenarios (i.e., combination of actions that together would achieve each given objective) will be output for evaluation, together with their cost/benefit consequences and their trickle-down-into-elsewhere consequences.

Conclusion

Clearly, transformation efforts will involve much more than computational support. In particular, meeting the basic rights and needs of those who lack them will require overcoming domination mindsets, starting with the most ubiquitous male domination, as well as a consciousness evolution allowing rich countries to relinquish the “right” to over-consume, over-pollute, and over-extract. In degrowing what we do not need while growing universal well-being under equity, we’ll be giving the earth a break from the increasing dangers of unbridled “growth” and wastefulness.

With this work we hope to help provide Doughnut-adopting cities the computational support they’ll need to achieve their goals.

Acknowledgement

This work was supported by the author’s NSERC Discovery grant. Thanks are due to the referees for their helpful comments, and to Juan Emilio Miralles for his help with testing and debugging the programs.

References

Christiansen, H., and V. Dahl. 2005. “HYPROLOG.” In Logic Programming.

“DEAL.” n.d. https://doughnuteconomics.org/.

Fry, D. P. 2013. *War, Peace, and Human Nature*.

Hickel, J. 2021. *Less Is More*.

Raworth, K. 2017. *Doughnut Economics*.

Edgar Daylight (Siegen University, Lille University)

Church's Reception of Turing's 1936 Paper: A Philosophical Angle

Session 2 | Wednesday, Oct 27, 11:20 – 11:50

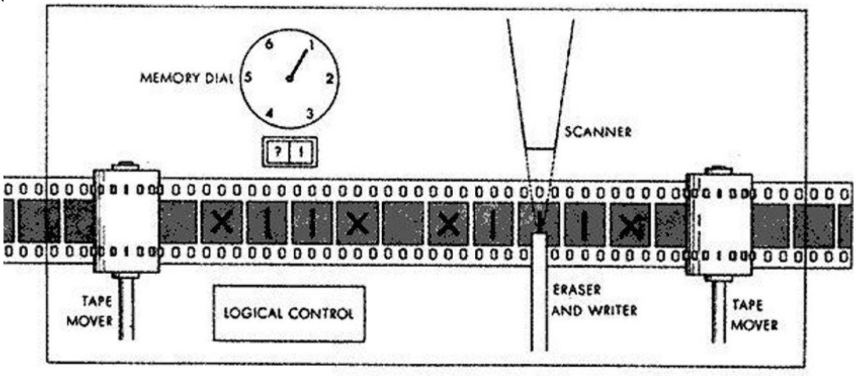
John Kemeny's 1955 article in *Scientific American*, entitled "Man Viewed as a Machine" (Kemeny 1955), wonderfully illustrates the advent of the "Turing Machine" concept in the history of science and technology. From a contemporary philosophical angle, we can say that Kemeny, a former Ph.D. student of Alonzo Church at Princeton University, blended logical and constructible (spatiotemporal) machines together, as illustrated in his Figure 1 and with the following words:

> "Turing's machines may be clumsy and slow, ..." (Kemeny 1955, 60)

> "... this slows the universal machine down considerably." (Kemeny 1955, 61)

These utterances are now common in computer science textbooks. Similar descriptions, *italicized* below, also appear in Church's 1937 comments on Turing's paper:

> "... it shall be possible to devise a computing machine, <u>occupying a finite space</u> and with working parts of finite size, which will write down the sequence to any desired number
> (



[fig:KemenyTM]Kemeny's 1955 "Turing Machine" (Kemeny 1955, 64).

Perhaps the reader is tempted, as was the present author until recently, to non-critically associate Kemeny & Church's blending (above) with that of Alan Turing (below). In Turing's 1936 words:

> "If the machine is supplied with a blank tape and set in motion [...] the subsequence of the symbols printed by it [...] will be called the 'sequence computed by the machine'." (Turing 1936, 232)
>
> "Every such operation consists of some change of the physical system consisting of the computer and his tape" (Turing 1936, 250)

Fusing logicomathematical and spatiotemporal categories can be problematic for the dualist, who wants to continually distinguish between the abstract and the physical; see, e.g., Paul Henry (Henry 1993, 121), Kjeld Schmidt (Schmidt 2011), and Daylight (Daylight 2021).

Historically, however, neither Church nor Turing were dualists in 1936-1937. They were not 'blending' or 'fusing' anything from their vantage point. I shall argue, though, that Church's philosophical position drastically differed from that of Turing. While Turing's 1936 paper was about mathematical machines according to both historical actors, the mathematics involved was instrumental and spatiotemporal for Church and not for Turing. Specifically, my forthcoming analysis will reveal that:

- Church's "Turing machines" were constructible, in line with the instrumentalism that he espoused in the 1930s. For Church, mathematical entities were "fictions" and "part of an abstract structure constructed by us to enable us to understand reality" (Anderson 1998, 137).
- Turing's machines were not physical. His ideal was the infinite, the material was finite, and it was the ideal which ruled — with time and matter being illusory; cf. (Geach 1979, 18) and (Hodges 1983, 64).

Detailing these differences between Church and Turing will provide a philosophical dimension to the historical narratives of Andrew Hodges and Leo Corry. Hodges — who, on my reading, views Turing in the main as a materialist; see (Hodges 1983, 103) and (Hodges 2000, 530) — claims, in connection with the underlined words above, that "Church's review was notably *incorrect*" (Hodges 2006, 245, original emphasis). Moreover, Hodges sees 'no obvious answer to the question why ... Church ... imputed to Turing's analysis something that was not actually there" (Hodges 2006, 245). Likewise, Corry writes that "Church used the term "computing machine" to mean any calculating machine of finite size, rather than the specific kind of "machines" introduced by Turing" in 1936. Corry concludes that Church's 1937 review "goes in just the opposite direction from Turing in his original formulation" (Corry 2017, 53).

Hodges and Corry insufficiently entertain the following historical interpretation: Church's 1937 "Turing machines" were (indeed) finite in space, but also materially extendable in time. The discrepancy between the views of Church and Turing was not mathematical but philosophical.

Metaphysical links between Turing, Arthur Eddington, and John McTaggart, which actually come from Hodges himself (Hodges 1983), will be refined in my analysis.

References

Anderson, C. A. 1998. “Alonzo Church’s Contributions to Philosophy and Intensional Logic.” *The Bulletin of Symbolic Logic* 4 (2): 129–71.

Church, A. 1937. “Review: A.M. Turing, on Computable Numbers, with an Application to the Entscheidungsproblem.” *Journal of Symbolic Logic* 2 (1): 42–43.

Corry, L. 2017. “From the Universal Turing Machine to Turing’s Analog Computer: Father of the Modern Computer?” *Communications of the ACM* 60 (8): 50–58.

Daylight, E. G. 2021. “The Halting Problem and Security’s Language-Theoretic Approach: Praise and Criticism from a Technical Historian.” *Computability* 10 (2): 141–58.

Geach, P. T. 1979. *Truth, Love and Immortality: An Introduction to McTaggart’s Philosophy*. University of California Press.

Henry, P. 1993. “Mathematical Machines.” In *The Machine as Metaphor and Tool*, edited by H. Haken, A. Karlqvist, and U. Svedin. Springer-Verlag.

Hodges, A. 1983. *Alan Turing: The Enigma*. London: Burnett Books.

Hodges, A. 2000. “Turing.” In *The Great Philosophers: From Socrates to Turing*, edited by R. Monk and F. Raphael, 493–541. Phoenix.

Hodges, A. 2006. “Did Church and Turing Have a Thesis about Machines?” In *Church’s Thesis After 70 Years*, edited by A. Olszewski, J. Woleński, and R. Janusz, ontos mathematical logic 1:242–52. ontos verlag.

Kemeny, J. 1955. “Man Viewed as a Machine.” *Scientific American* 192 (4).

Schmidt, K. 2011. “Dispelling the Mythology of Computational Artifacts.” In *Cooperative Work and Coordinative Practices*, edited by K. Schmidt, 391–413. London/New York: Springer-Verlag.

Turing, A. M. 1936. “On Computable Numbers, with an Application to the Entscheidungsproblem.” *Proceedings of the London Mathematical Society, 2nd Series* 42: 230–65.

Moritz Feichtinger (Univeristé de Lausanne)

Databasing Djungle-War: The US-Army's Data-Processing Systems During the Vietnam War, 1966-1975

Session 3 | Wednesday, Oct 27, 14:00 – 14:30

The presentation introduces the concept of databasing as a set of practices that includes epistemological, technical, social, and cultural elements. To demonstrate the analytical value of this concept, it examines a Vietnam War database system as an epistemological paradigm, a socio- technical practice, a specific material and media configuration of information technology, and a strategy for the fabrication of knowledge and suggestion of objectivity. Therefore, it discusses the practices of data collection, aggregation and analysis, as well as their representation and communication. These questions are addressed with a particular focus on the relationship between the material manifestations of the data sets and their technological management on the one hand, and the socio- technical practices and groups of people involved on the other.

At the time of the U.S. military intervention in South Vietnam in the 1960s and 1970s, an unconditional belief in the potential of systems analysis and operations research prevailed at the top of the Department of Defense. Secretary of Defense Robert S. McNamara and his closest advisors believed that even such complex political-social phenomena as guerrilla warfare could be captured, calculated and controlled with the help of information technology. Beginning in 1966, programmers, social scientists, and military analysts developed a system of several networked databases that would compile information about the military, social, and political situation in South Vietnam's rural areas in real time. The case study of Vietnam War era data processing systems opens up critical perspectives on historical and contemporary practices of databasing: the systems were intended to capture a social-cultural sphere that was alien to their developers; the technology used to do so was originally developed for other purposes, had to be adapted, and in turn reverberated back to computer and software development; information obtained through aggregation and analysis of data appeared to be more objective than information from other sources.

Science and technology studies have characterized databases to be constitutive for the information age in their attempt to make the world legible and to order it. Even in pre-digital times, the representation of phenomena and processes - especially from the sphere of the social - in databases involved significant intellectual and epistemological pre-decisions. Already the choice of the procedure for collecting and compiling data has a considerable influence on the meaningfulness of a data set. Categories and attributes need to be selected and must be rendered quantifiable and assessable by assigning them to the categories as discrete values. In the case of the Vietnam War database systems, renowned social scientists developed strategies for the collection of data and contributed to the design of aggregation and

classification procedures. Databasing is a socio-technical practice that involves the interaction between humans and between humans and machines. Understanding computing as a social practice therefore provides new perspectives on the personnel involved in digital information processing. Here it is possible to analyze how technical and sociological expertise intertwined or contradicted each other. This perspective also opens up insights into the function of "subaltern" personnel, such as technicians, female typists, research assistants, etc. An influential approach in recent studies in the history of computing places the materiality of information technology at the center of analysis. This research shows that infrastructures and media of processing and storage could have an impact on the semantic content of data sets. This relationship can be explored in detail through the case study of Vietnam War data-processing systems. Data sets went through various material manifestations and transformations, from survey sheets on cardboard to punch cards and magnetic tapes to printed and published tables and graphs. By tracing these transformations, it is possible to analyze the specific ways in which the respective materiality of media and infrastructures determined the explanatory value of the data.

The concept of databasing also includes questions about how aggregated data and their visual and textual representations served to create the impression of objectivity and evidence. The case study of the databases of the Vietnam War is instructive here, because it predates the screen age. The results of aggregating and analyzing the extensive data sets had to be made comprehensible by further abstracting and visualizing them in charts, tables, maps, etc. In addition, the example of the Vietnam War database systems is important because the practices of ordering knowledge directly impacted practices of ordering the world. Based on the system's findings and predictions, communities could be resettled, villages bombarded, and forest areas defoliated.

Hence, the case stuy and the theoretical and methodical approaches introduced in the presentation raise more general questions about the practices, epistemological foundations, and political implications of databasing in attempts to understand, model and manipulate social phenomena and processes.

Carmen Flury and Rosalía Guerrero (Zurich University of Teacher Training,)

A National School Computer for the Emerging Digital Society: East German and Swedish Efforts to Develop a State-Mandated Educational Computer in the 1980s-1990s

Session 6 | Thursday, Oct 28, 11:50 – 12:20

In the late 1970s and 1980s, several European states implemented computer literacy programs in their schools in response to the rapid development and spread of microcomputers. Some countries developed their own home-manufactured computers for the home and school markets, with suitable software to be used in educational settings (the Netherlands, the UK, Denmark, Finland) (Veraart 2014; Lean 2012; Fogh 1988; Saarikoski 2010) and some even developed purpose-built computers to be specifically used in schools in a government-backed effort to be specifically used in educational settings (e.g. the ICON in Canada, the Smaky in Switzerland, the Elwro 800 Junior in Poland) (Alberts & Oldenziel 2014; Tatnall & Leonard 2010; Mangan 1997; Chua & McCallum 1984; Sysło 2014).

Instead of equipping schools with widely available home computers by IBM, Apple or Commodore, these governments sought to develop their own hard- and software for use in classrooms. Such endeavors were motivated by the promise of economic prosperity, attempts to strengthen the domestic computer industry, as well as concerns of pedagogues and educational policymakers to adapt computer technology to meet the specific needs and demands of teachers and pupils.

In this paper, we compare two countries that followed a similar strategy in developing their own national school computer despite their stark political and economic differences. This paper aims to examine how two countries implemented a similar approach based on different rationales with different material resources and political structures to shed light on the factors that might affect the process of state-based technology development. On the one hand, East Germany, as a planned economy with an authoritarian socialist leadership, and on the other hand, Sweden, as a mixed economy with a market orientation but with a robust public sector. Both states commissioned the development of a school computer that would be used nationwide to introduce pupils to the world of micro-computing. However, the political rationales behind these efforts varied greatly. In Sweden, there was an explicit economic rationale behind the state sponsoring of this project: To boost the national computer industry while arming schools with state of the art and inexpensive computers. From 1982, computer education had become part of the national curriculum in compulsory education at the lower secondary level. Municipalities oversaw teacher training and the purchase of appropriate equipment with the help of state subventions. In 1984 started the production of the Swedish school computer COMPIS, which resulted from a technology procurement competition. Most secondary schools purchased this computer, which was expected to be user-friendly and specifically designed for educational contexts. The COMPIS introduced a completely new computer language in schools

that did not match any other used in the labor market. While producers of alternative hardware and software brought up this issue, the government considered that by allowing schools to choose their equipment, anti-competitive practices were out of the question. Schools would make their decisions based on their needs and quality assessment. However, the computer and its software did not match the expectations, and the production was discontinued after only four years. In East Germany, the computer industry‘s growing influence on schools in capitalist countries was strongly criticized. Instead, the socialist regime strived to develop a school computer in close cooperation with pedagogues to match its ideas of "socialist" education. At the same time, the already available microcomputers produced by the East German electronics manufacturer "Robotron" since 1985 were deemed inadequate for use in classrooms, as they did not match the technical requirements as set out by the newly developed ICT curricula for upper secondary schools. Thus, the "Bildungscomputer Robotron A 5105" ("BIC") was developed in 1987 and produced on a larger scale from July 1989 to April 1990. Initially, the authoritarian government in East Germany had mandated pedagogues and educational policymakers to create a list of criteria that such a school computer would need to fulfil. However, in the process of its development, the BIC‘s specifications had to be negotiated several times, as the economy of scarcity and level of technological development in East Germany did not allow for the mass production of a computer that met all the demands and requirements of educators.

Although East Germany had a population twice as large as Sweden (ca. 16 million to ca. 8 million), both countries had the necessary administrative infrastructure and control over their economy to centrally plan a project of this nature. However, both attempts failed to achieve the state‘s ambitions for several reasons. By closely examining these reasons, we seek to shed light on state initiatives‘ effects based on centralized management of technological development in other areas such as education. The two case studies draw upon historical documents by educational policymakers and government officials, such as reports, protocols, notes and correspondence, as well as newspaper articles and popular computer magazines.

References

Alberts, G. & Oldenziel R. (2014). Introduction: How European Players Captured the Computer and Created the Scenes. In: G. Alberts & R. Oldenziel (Eds.), Hacking Europe (pp. 1-22). London: Springer.

Chua T.S. & J. C. McCallum (1984) *Using Microcomputers in Computer Education*, SIGCSE Bulletin, Vol. 16 No. 4, December 1984, (pp. 25–33).

Fogh, Lise Informatics in Danish Schools (2006) in Harris Duncan (Ed.) World Yearbook of Education 1988: Education for the New Technologies (pp. 149–157). Oxon: Routledge.

Lean, Thomas (2012) *Mediating the Microcomputer: The educational character of the 1980s British* *popular Computer boom*. Public Understanding of Science 22(5).

Mangan, J. Marshall, Computer Studies as Symbolic and Ideological Action: The genealogy of the ICON in Anstead, Christopher J., et al. Subject Knowledge: Readings for the Study of School Subjects, Taylor & Francis Group, 1997. ProQuest Ebook Central, https://ebookcentral.proquest.com/lib/uu/detail.action?docID=167268.

Saarikoski P. (2010) *Computer Courses in Finnish Schools,* 1980–1995. 3rd History of Nordic Computing (HiNC), Oct 2010, (pp.150-158).

Sysło, Maciej M. (2014) The First 25 Years of Computers in Education in Poland: 1965 – 1990 in Tatnall, Arthur & Davey Bill (Eds.) Reflections on the History of Computers in Education: Early Use of Computers and Teaching about Computing in Schools (266-290). Berlin, Heidelberg: Springer.

Tatnall, A. & Leonard, R. (2010). Purpose-Built Educational Computers in the 1980s: The Australian Experience. In: A. Tatnall (Ed.), History of Computing: Learning from the Past. IFIP WG 9.7 International Conference, HC 2010, Held as Part of WCC 2010, Brisbane, Australia, September 20-23, 2010, Proceedings (pp. 101–111).

Laura Fontanella (Université Paris Est Créteil)

The Evolution of the Concept of Proof Through Realizability

Session 10 | Friday, Oct 29, 11:20 – 11:50

Mathematical constructivism was concerned with the philosophical problem of what is intended to be a proof in mathematics. In this context, Heyting formulated the so-called *BHK interpretation* which outlines the intuitionistic notion of proof. In this view, proofs correspond to *functions*: for instance a proof of $A \rightarrow B$ is a function that converts a proof of A into a proof of B. In the same period, we can observe a growing interest for the study of *computable functions*, several models were proposed to formalize this notion, eventually they were proven to be equivalent: partial recursive functions, λ-definable functions, functions which are computable by a Turing machine. This led Kleene to interpret proofs as computable functions identified with recursive functions. From his work, it originated a whole new branch of mathematical logic known as *realizability*. We discuss the evolution of the concept of mathematical proof through the development of realizability.

In modern terms, we say that the ultimate goal of realizability is to extract the computational content of mathematical theories – in fact realizability is used for programs extraction in proof assistants such as Coq. In realizability, a given theory (or a logical system) is interpreted in a model of computation by establishing a correspondence between the formulae of the theory on the one hand, and programs on the other hand, in a way that knowledge of the programs that realize a formula (i.e. that are associated with the formula) gives us information about the way the formula can be proven in the theory. For instance, an implication $A \rightarrow B$ provable in the theory would be realized by programs that take an entry of type A and return a result of type B.

Kleene's original version of realizability was an interpretation of the formulae of Heyting arithmetic as sets of indexes of recursive functions. In modern realizability, recursive functions are replaced by programs formalised in some variant of lambda calculus. For instance, Curry-Howard isomorphism can be seen as a realizability result establishing a correspondence between intuitionistic logic and simply typed lambda-calculus. In the 90's, research in this field led to pass the barrier of intuitionistic logic when Griffin [2] generalized the Curry-Howard correspondence to classical logic by using the instruction *call/cc* of the programming language Scheme. Finally, J-L. Krivine was able to develop a method (which generalizes the technique of Forcing) for realizing not only classical logic, but also Zermelo-Fraenkel set theory ZF (see [4]).

Thanks to Krivine's technique, we are now able to extract the computational content of the axioms of ZF and more. For instance, the axiom of foundation is realized by the Turing fixed point combinator, and the axiom of dependent choice can be realized (among others) by the instruction *quote* of the programming language LISP. For many years, it remained an open problem to determine whether it was possible to realize the full axiom of choice. This question was very recently answered by the affirmative by Krivine who defined a realizability model for the axiom of choice [5].

Through these results, we can see an evolution of the concept of constructive proofs and theories: if realizability originated as part of a broader research in constructive mathematics, today it allows us to give a computational meaning to the axioms of set theory including the axiom of choice, which is the paradigm of non-constructive mathematics. The idea of proofs-as-algorithms has evolved via realizability through the different models of computation proposed: from recursive functions, to lambda calculus and its variants. From Curry-Howard, we know that a proof in intuitionistic logic corresponds to a term of simply typed λ-calculus. After the work of Griffin, it was possible to interpret proofs in classical logic as terms of the λ_c-calculus (a version of λ-calculus with an instruction for *call/cc*). In Krivine's models, programs are not the only witnesses of the truth of a formula; a pair program-plus-stack (a stack is a sequence of instructions) determines a state of the so called *Krivine's abstract machine* and the whole state is involved in the computation of the proof of the formula: programs evaluate its 'degree of truth', while the stacks evaluate its 'degree of the falsity'. Moreover, in these models, programs are still formalized as λ_c-terms, but one may add customary instructions to the calculus, such as ordinals acting as inherent constants (see for instance [1]), with the only limitation that the reduction rules for processes must include the basic rules of Krivine's abstract machines ('push', 'grab', 'save' and 'restore').

We can see, then, how the notion of proof intended as en effectively computable method for witnessing the truth of a formula has changed through realizability to embrace a richer, yet still precise notion of realizer: the truth of a formula which is realized is computable, at leats in principle, in Krivine's abstract machine.

References

[1] L. Fontanella and G. Goeffroy Preserving cardinals and weak forms of Zorn's lemma in realizability models. *Mathematical Structures in Computer Science*, Volume 30 , Issue 9 , October 2020 , pp. 976 - 996

[2] Griffin T. G. (1990) A formulae-as-type notion of control. *POPL '90: Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* December 1989. Pages 47–58

[3] Kleene S. C. (1973) Realizability: A retrospective survey. In: Mathias A.R.D., Rogers H. (eds) Cambridge Summer School in Mathematical Logic. *Lecture Notes in Mathematics*, vol 337. Springer, Berlin, Heidelberg.

[4] Krivine, JL. Typed lambda-calculus in classical Zermelo-Fraenkel set theory. Arch. Math. Logic 40, 189–205 (2001).

[5] Krivine, JL. A program for the full axiom of choice. To appear in *Logical methods in computer*

science.

Michael Friedman (Humboldt University)

Leibniz on Stocking Frame, Computing and Weaving

Session 5 | Thursday, Oct 28, 9:00 – 9:30

When one looks at the history of mechanization of textile practices (spinning, weaving or knitting, for example) one of the first machines to be invented in the early modern period in Europe is the stocking frame, invented in England, which mechanized knitting in 1589. William Lee's invention of the stocking frame was somewhat astonishing, because it was an invention of a completely new machine. Indeed, at that time, when the mechanization of textile industry consisted mainly of spinning wheels and hand looms, Lee's stocking frame is not to be underestimated – indeed, it came decades before the other, more well-known textile machines of Vaucanson, or the Jacquard loom – a loom which is usually considered as precursor of the computer. But while this stocking frame was a breakthrough in the mechanization of weaving, more surprising is that Leibniz considered it as equivalent to his own calculating machine – the stepped reckoner. The aim of the talk is to inquire why Leibniz viewed these two machines as having the same complexity, and to attempt to shed a new light on how Leibniz considered weaving techniques in connection with computation.

Leibniz's references to the stocking machine appear mostly during the late 1670s. Leibniz wrote in 1676 a manuscript that was originally planned as a preface to *De quadrature arithmetica*; the title of this manuscript was "An Exoteric Dissertation. About the Use of Geometry, its Present State, its Newest Additions" [*Dissertatio exoterica de usu geometriae, et statu praesenti, ac novissimis ejus incrementis*] As Davide Crippa notes, "[s]ince 1673, Leibniz had been convinced that this realm of geometry [of the required mathematics in order to solve the problem of the quadrature of the circle] lay beyond Descartes' geometry, which was of little or no use when it came to problems of quadrature and rectifications."[2] This is also to be seen in the introduction to *Dissertatio exoterica*: Leibniz states that geometry is much broader and considers the "geometric genius":

> those "who are [endowed] with a geometric genius [talent], their discoveries are difficult and profound and put forth with much meditation. The sorts of things, which are not easily explained, and not immediately understood by any listener or viewer. From this, we have an elegant example in a weaving machine [*Machina textrice*], now frequently used here and there, with the invention of a certain Scotsman, [an invention] which occupied its inventor for nine whole years; or [another example is given] by an arithmetical instrument, which carried over all the work of the mind into wheels."[3]

The arithmetical instrument, to which Leibniz refers to, may very well be his own automatic "living" calculating machine: his "lebendige Rechenbank", which can "by itself add, subtract, multiply and divide all numbers"[4], as he notes in 1671 in a letter to Herzog Johann Friedrich from Hannover; this principle of the operation of this machine, whose model was presented in 1673 to the Royal Society in London, was based on Leibniz' invention of the stepped reckoner, now called the Leibniz wheel. If by the "arithmetical instrument" Leibniz indeed meant his own machine, what does it mean, that he compared his own calculating machine to the "weaving machine" (although associating erroneously the invention of the "weaving machine" to a "certain Scotsman")? As I aim to show, this was not the only time Leibniz referred to the stocking frame in his writings – and moreover he considered other mathematical and computational aspects of looms and weaving machines in the following years. How does Leibniz' conception of weaving machines as computational shape then conceptions of computation in the late 17th century?

Notes

[1] (AVII6, manuscript 491).

[2] (Crippa 2019, p. 102).

[3] Ibid., p. 487–488: "Qui Geometrico sunt ingenio eorum inventa difficilia sunt, et profunda, et multa meditatione expressa. Qualia nec facile enuntiantur, nec statim a quovis auditore aut spectatore intelliguntur, unde Exemplum elegans habemus in Machina textrice, nunc passim frequentata, Scoti cuiusdam invento, quod novennio integro occupavit autorem suum, aut in Arithmetico instrumento, quod omnem animi laborem in rotas transfert."

[4] (AI11, p. 160): "[...] eine lebendige Rechenbank nenne, dieweil dadurch zu wege gebracht wird, dass alle Zahlen sich selbst rechnen, addieren, subtrahieren, multipliciren [...]."

Bibliography

Crippa D (2019) *The Impossibility of Squaring the Circle in the 17th Century. A Debate Among Gregory*, Huygens and Leibniz. Birkhäuser, Basel.

Leibniz GW (1923–) *Sämtliche Schriften und Briefe. Berlin-Brandenburgische* Akademie der Wissenschaften/Akademie der Wissenschaften zu Göttingen, Berlin/Göttingen. Referred as A.

Jens Ulrik Hansen and Paula Quinon (Roskilde University, Warsaw University of Technology)

The Role of Expert Knowledge in Big Data and Machine Learning

Session 9 | Friday, Oct 29, 8:30 – 9:00

According to popular belief, Big Data and Machine Learning provide a *brand-new* approach to science that has the potential to revolutionize scientific progress (Hey et al., 2009; Kitchin, 2014). The extreme version of this belief is illustrated by Anderson's claim that Big Data and Machine Learning in science will lead to the "end of theory" (Anderson, 2008). The idea behind this extreme version of the belief is that advanced Big Data and Machine Learning algorithms enable us to mine vast amounts of data related to a given problem without prior knowledge and we do not need to worry about causality, as correlation is all that is needed.

The extreme version of this belief is not seriously held by many philosophers of science, but there are several serious attempts to determine the extent to which Big Data and Machine Learning imply a resurgence of inductive methods (Pietsch, 2021) or agnostic science (Napoletani et al., 2021). Without claiming that "the theory came to its end", these approaches advocate new scientific methods that can be applied to various fields in a similar way, without the need for domain-specific knowledge.

Two questions arise in connection with these views: Where did these ideas come from? and to what extent are they justified? The first question could be addressed by following the hype around Big Data and Machine Learning in industry and how easily conversations about new innovations and disruptions translate into conversations about paradigm shifts in science. The leakage of the style of argumentation and the attraction to hype, from industry to science, should be carefully watched and frequently questioned. In these times of pressure to bridge the gap between business and science, it can be difficult to distinguish valuable insights and ideas from superficial buzz-talk. The case of Big Data and Machine Learning is one of the areas, which shows how important it is to clearly map the flow of knowledge between business and science. We do not, however, make any scientific progress towards this question, we only sketch our observations here.

Regarding our second question, we argue that using methods from Big Data and Machine Learning is not a passive mode where you feed raw data into a Big Data or Machine Learning algorithm and wait for the algorithm to detect correlations between features of a massive dataset. We argue that there is always work in manipulating data, cleaning data, etc. that requires significant domain knowledge of scientific applications.

We agree that expert domain knowledge used in Big Data and Machine Learning is of a different kind from that required by traditional methods. The question we ask is about the required amount of expert knowledge needed for Big Data and Machine Learning methods to function in an efficient manner. By carefully assessing examples of Big Data and Machine Learning applications in science, such as skin cancer detection (Esteva et al., 2017), protein folding (Senior et al., 2020) and language generation (Brown et al., 2020), we assess which knowledge plays a role in each of these cases. We observe that significant domain knowledge is needed, not so much in theory and model building, but in data preparation and validation of Machine Learning models. Data needs to be labelled appropriately, decisions need to be made about which data to include, and new features might need to be created. Furthermore, we need to know what constitutes a well-functioning Machine Learning algorithm for a given problem, what we should measure and what is a good enough value to constitute a solution to the problem.

In addition to expert knowledge about data samples, specific knowledge of Machine Learning is also often needed, as algorithms cannot be applied blindly in practice. A promising model architecture needs to be selected, appropriate data augmentation techniques need to be applied to increase the performance of the algorithm, and the algorithm needs to be tuned and adjusted to get good enough Machine Learning performance.

We do not intent to dismiss new scientific methods or new lines of research, but to disclose what work and knowledge the new methods require, and thus show in more detail what is new and what is business as usual. Big Data and Machine Learning methods may be used in a more agnostic way, but they do not lead to completely agnostic science. It is not a question of changing or revolutionizing science, but of expanding the methodological toolkit. This will certainly not necessarily revolutionize all science, but it could lead to changes in subfields and provoke the emergence of new fields or endeavors, such as digital humanities or computational social science.

In our talk, we will discuss concrete case studies (skin cancer detection, protein folding, language generation), where we present methods that are used and we highlight those moments where expert knowledge is involved. We will attempt to classify various aspects of expert knowledge involved in the application of Big Data and Machine Learning methods, for instance, the expert knowledge necessary at the training data sample preparation or the expert knowledge necessary for choosing algorithms. We will also suggest that, depending on the field, the range of traditional methods vs agnostic methods varies, leading us to believe that the process of "agnosticization" is different from field to field, and that the possibility to reach the 'no-theory" stage varies depending on domain. Consequently, we observe that the way in which Big Data and Machine Learning methods enter scientific methodology involves continuous small conceptual shifts rather than a rigid paradigm shift in Kuhn's sense.

References

Anderson, C. (2008). The End of Theory: The Data Deluge Makes the ScientificMethod Obsolete. *Wired Magazine*, 16(7). https://www.wired.com/2008/06/pb-theory/

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., … Amodei, D. (2020). Language Models are Few-Shot Learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Advances in Neural Information Processing Systems* (Vol. 33, pp. 1877–1901). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., & Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639), 115–118. https://doi.org/10.1038/nature21056

Hey, T., Tansley, S., & Tolle, K. (Eds.). (2009). *The fourth paradigm: data-intensive scientific discovery*. Microsoft Research. file:///C:/Users/Edson.MARLEY/Desktop/4th_paradigm_book_complete_lr.pdf

Kitchin, R. (2014). Big Data, new epistemologies and paradigm shifts. *Big Data & Society*, 1(1), 2053951714528481. https://doi.org/10.1177/2053951714528481

Napoletani, D., Panza, M., & Struppa, D. (2021). *The Agnostic Structure of Data Science Methods.* https://arxiv.org/pdf/2101.12150.pdf

Pietsch, W. (2021). *Big Data*. Cambridge University Press.

Senior, A. W., Evans, R., Jumper, J., Kirkpatrick, J., Sifre, L., Green, T., Qin, C., Žídek, A., Nelson, A. W. R., Bridgland, A., Penedones, H., Petersen, S., Simonyan, K., Crossan, S., Kohli, P., Jones, D. T., Silver, D., Kavukcuoglu, K., & Hassabis, D. (2020). Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792), 706–710. https://doi.org/10.1038/s41586-019-1923-7

Chris Holland (Bakersfield College)

Slipping Through Our Fingers Even As We Tighten Our Grip on the Controller: Rentism and the Consequences for Video Games

Session 8 | Thursday, Oct 28, 17:00 – 17:30

Pizza parlors were changed forever in the mid-1970s with the arrival of the first video game, *Pong*. Atari introduced the Atari 2600 in 1977, enabling gamers to play their favorite video games at home thanks to Jerry Lawson's invention of the interchangeable video game cartridge (Hardawar, 2015). While Kittler (Kittler, 1995), and Kaldrack and Leeker (Kaldrack & Leeker, 2015) offer arguments for software not existing as machine-independent faculties and there being no such thing as "stand-alone hardware," Manovich (Manovich, 2013) responds software is able to take on different forms and can be very independent of machines via media hybridization; a reality in the video game industry. Frase (Frase, 2016) contends with increasing controls regarding intellectual property, the debate between Kittler, Manovich, and Kaldrach and Leeker misses the increasing encroachment of rentism into the video game world. Nuebert emphasizes this by stating that these developments create "the illusion of a new product to justify the repeated resale of what is fundamentally the same good" (Neuburt, 2015, p. 268) for gamers. Between shrinking video game cartridges and the dawn of the internet and cloud technology the rentism involved with video games is more apparent and making the concept of "property" malleable. Lison emphasizes this by pointing out that with subscription services taking the lead, "shrink-wrapped software indicated the final moments of a political economy fundamentally predicated upon the commodity form…..[and] are masked through the circulation of material goods" (Lison, 2015, p. 59).

"Who owns the robots owns the world," was stated by Harvard University labor economist Richard Freeman and quoted by Frase in his discussion of Rentism (Frase, 2016, p. 70). Stark words considering the encroachment of rentism on video games. Gamers that awaited new released of games to play at home seemingly got their playability added onto when developers like Nintendo and Sega introduced gaming systems that made it possible for friends to play games in the privacy of their own homes everywhere on the planet. Marking a shift from shrink-wrapped software to *Software as a Service* (SaaS) within the video game industry. A shift that has become even more complex when video games like Nintendo's revamped for the Nintendo Switch, *The Legend of Zelda: Link's Awakening* that follow the shrink-wrapped sense of ownership as a stand-alone game find themselves overshadowed by *Fortnite*, a game that is available initially as a free download – yet still requires gamers to pay a subscription fee if they want to play the game with or against other gamers. Further blending things are games like *Animal Crossing: New Horizons* that on the one hand make it still possible to largely play the game without a subscription service while on the other making it enticingly apparent that other elements cannot be enjoyed without a subscription. With this blurring of boundaries between "contractually-independent drivers, part-time landlords (or sublessors), and contingent workers with paying customers, software becomes the means for the supposed 'disintermediation' of buyers from sellers in an immaterial labor market more accurately defined in terms of service than 'sharing'" (Lison, 2015, p. 68). With the Intellectual Property that are deeply woven into video games, this ties into Frase's contention that "those who control

the most copyrights and patents become the new ruling class" (Frase, 2016, p. 71). The key element being the struggle shifting from ownership that's centered on private property that can be touched to one that revolves over the control of copied patterns that are intangible (Frase, 2016).

A central and necessary element for this conversation is the overarching human relationship with play. Wright offers that for centuries play has been defined as being "a noncoercive relationship, an imaginative method of engaging everyday life in a specific space and time that we may or may not consider an accomplishment" but rather is "accompanied by a sense of well being" (Wright, 2018, p. 4). It is even seen by modern scholars as means for developing social connectivity of various types (McGonigal, 2011). With new forms of community that can be formed and pursued in places that are both public and private in the video game world, it is becoming increasingly difficult for gamers who aren't liking how things are going with the terms of agreement video games come with to simply take their money and go elsewhere (Taylor, 2006). This is demonstrative of Marx's contention that the individuals involved in the exchange of commodities places the buyer and the seller into a relationship with one another (Marx, 2010). However, this implies that when money is exchanged for a desired commodity the value given in the exchange will be gotten back in "an equivalent amount of value embodied in the commodity" (Wayne, 2012, p. 19). It raises the question: are gamers truly getting an equal value in exchange for games they have purchased and/or pay a monthly fee to stream? Jagoda contends that video games have expanded the field of questioning and "invite an active and uncertain experimentation that demands great care" (Jagoda, 2020, p. 33). Particularly when play is now being redefined as "the mediated relationship between containment and expression woven into the objectification of labor in the world . . .in so far as it speaks to noncoercive relationships in the organization of our species being' (Wright, 2018, pp. 9–10)

While all of this sounds like the set-up for a video game with its setting in a dystopian world, it is all in fact quite real. Adding to this are the countless articles from entities like *PCmag.com*, *gamedesigning.com*, and *digitaltrends.com* that actively encourage this shift by offering articles that center on increasing the appeal of streaming games as well as how to's for putting together a streaming game station. With full encouragement, and very little in the way of questioning the shift. Which ties into rentism because as "partisans of capitalism, it is often convenient to pretend that property is some naturally occurring fact, but it is a really a social construction that must be delineated and enforced by the power of the state" (Frase, 2016, p. 75). It will be the purpose of this study to explore the shift from ownership to rentism in the phenomenon that is subscription-based gaming and ask: what does the implied ownership mean for gamers who simply want to play?

References

Frase, P. (2016). *Four Futures: Life After Capitalism*. Verso.

Hardawar, D. (2015, February 20). *Jerry Lawson, a self-taught engineer, gave us video game* *cartridges*. Engadget. https://www.engadget.com/2015-02-20-jerry-lawson-game-pioneer.html

Jagoda, P. (2020). *Experimental Games: Critique, Play, and Design in the Age of Gamification*. University of Chicago Press.

Kaldrack, I., & Leeker, M. (2015). There is no Software, there are just Services: Introduction. In I. Kaldrack & M. Leeker (Eds.), *There is No Software, There Are Just Services* (pp. 9–19). Meson Press.

Kittler, F. (1995). There is No Software. *C-Theory: Theory, Technology, Culture*, 32. http://www.ctheory.com/article/a032.html.

Lison, A. (2015). From Shrink Wrap to Services: The Universal Machine and Universal Exchange. In I. Kaldrack & M. Leeker (Eds.), *There is no Software, There are just Services* (pp. 57–71). Meson Press.

Manovich, L. (2013). *Software Takes Command.* Bloomsbury Academic.

Marx, K. (2010). *Das Kapital*. Pacific Publishing Studio.

McGonigal, J. (2011). *Reality is Broken*. The Penguin Press.

Neuburt, C. (2015). "The Tail on the Hardware Dog": Historical Articulations of Computing Machinery, Software, and Services. In I. Kaldrack & M. Leeker (Eds.), *There Is No Software, There Are Just Services* (pp. 21–37). Meson Press.

Taylor, T. (2006). *Play Between Worlds*. The MIT Press.

Wayne, M. (2012). *Marx's Das Kapital For Beginners*. For Beginners.

Wright, J. T. (2018). Liberating Human Expression: Work and Play or Work Versus Play. *American Journal of Play*, 11(1), 3–25.

Michael Jackson (Independent Consultant)

Cyber-Physical Programming

Session 2 | Wednesday, Oct 27, 12:20 – 12:50

A cyber-physical system is bipartite: a machine and a governed world. The machine is conventional computing equipment; the governed world is a heterogeneous physical ensemble of natural phenomena, engineered devices, built and natural infrastructure, human participants, and much else. The machine must ensure a desired behaviour in the governed world. Undesired behaviour is system failure.

Designing a desired behaviour to satisfy the requirements is the Requirements Problem. By contrast, this talk focuses on the Programming Problem for critical systems. How can developers, developing software, achieve adequate confidence that the machine executing the software will reliably ensure the desired behaviour? For a critical system they must aim at the highest degree of reliability that is reasonably possible.

The Programming Problem springs from two inescapable characteristics of cyber-physical systems. First, the Restricted Interface. Desired behaviour is naturally expressed in terms of large physical actions and process structures at many levels, and of phenomena remote from the machine-world interface of physical sensors and actuators. But the interface allows the machine and the governed world to communicate only in terms of elementary events and states. Second, the Formalism Restriction. The governed world is everywhere non- formal: everything is in flux; nothing is atomic; everything is connected; nothing is compositional; everything is contingent; there are no theorems.

These restrictions distinguish cyber-physical programming from traditional formal development in the most fundamental way. Solving the problem “Calculate C, the GCD of integers A>0 and B>0,” the programmer knows the specification of GCD and the appropriate number theory; the interface communicates the arguments; the programming language offers the integer type and its operations. The cyber-physical programmer has none of these assets and benefits.

To address the Formalism Restriction demands an adequately (but inevitably imperfectly) faithful model of the governed world. The Restricted Interface highlights a specific demand on the model: to describe how the low-level interface phenomena are related to other phenomena of the governed world remote from the interface and only indirectly accessible to the machine.

This additional demand is only a special case of the crucial need for causality in modelling the governed world. State and event phenomena in the governed world are related by causal links effectuated by physical domains. By the nature of the governed world, effectuation is always contingent both on the state of the effectuating domain and on other factors.

The laws of physics play a part in forming the model, but they are very far from sufficient. Polanyi [2] identifies two conceptual levels to explain and understand the properties and behaviours of the governed world. At the lower, more basic, level, behaviour is constrained by physical laws—from interactions of elementary particles, to chemical interactions, to conservation laws. At the upper level, behaviour—always subject to physical laws—is further restricted by the physical shaping of natural and artificial domains and other things, and by their juxtapositions.

It is chiefly at Polanyi's upper level that the possibility of a reliable formal model is challenged. The idiosyncratic shapes of physical domains, both engineered and natural, their subjection to multiple simultaneous physical effects, their direct and indirect interactions by design and by chance, and the perpetual subjection of their properties to change and degradation over time, limit every formal model to a crude approximation to the reality.

Restriction to crude approximation should not discourage us. Niels Bohr [3] wrote of physicists' models: "In our description of nature the purpose is not to disclose the real essence of the phenomena but only to track down, as far as possible, relations between the manifold aspects of our experience." What is true for scientists is even more compelling for engineers. The governed world model need only suffice for the desired governed world behaviour, for the purpose, conditions and context of its enactment. Model development and behaviour development must proceed hand in hand.

Critical systems have complex behaviours and a rich and complex governed worlds. What structure is appropriate for the governed world model? At first sight it may seem that the governed world model should be structured to reflect the governed world: that is, the structure of interacting physical domains which together provide the substrate for enacting the governed behaviour. In this approach, each domain might be modelled as an object. The disadvantage is that each domain's model must represent the combination of all the properties relevant to all aspects of the behaviour enactments in which the domain participates.

A better structure of the governed world model reflects the structure of the system behaviour. This is especially effective for a system whose behaviour has a clear structure. In an aircraft flight management system the governed world models for cruising and landing are clearly distinct. In automotive software, many behavioural features are enacted in response to driver commands or to conditions or circumstances encountered during system operation. An advantage of this structuring principle is that the partial models can be disjoint and can even be contradictory when their corresponding behaviours are mutually exclusive. The model of a car's steering appropriate to highway driving and the model appropriate to automatic parallel parking can be unrelated.

Modelling in the large must be distinguished from modelling in the small. Modelling in the large designs the large structure of the model, as briefly mentioned above. Modelling in the small builds one partial model. Within this one model, a clear distinction can, and should, be made between an axiomatic model and a behavioural model. In the axiomatic model, only governed world phenomena appear: causal links, imputed to their effectuating domains, are regarded as axiomatic for the model. In the behavioural model the interaction with the machine is added, producing a representation of the resulting behaviour to which both machine and governed world itself contribute.

The cyber-physical programmer's product is the behaviour of the bipartite system—the interacting machine and governed world. Knowledge of the machine's programming language is obviously essential. Knowledge of the governed world is harder to obtain, but no less necessary.

[1] Albert Einstein; Geometry and Experience, 1921.

[2] Michael Polanyi; The Tacit Dimension, pp39-40, U Chicago Press, 2009.

[3] Niels Bohr; Atomic Physics & Human Knowledge; Chapman & Hall, 1958.

Cliff Jones (Newcastle University)

One Concurrent Program: Three Attempts at Its Formal Verification

Session 10 | Friday, Oct 29, 10:50 – 11:20

The talk traces a relatively linear sequence of early research ideas on ways to verify concurrent programs formally; but it does so forwards and then backwards in time. After briefly outlining the context, the key insights from three distinct approaches from the 1970s are identified (Ashcroft/Manna, Ashcroft (solo) and Owicki). The main part of the talk focuses on a specific program taken from the last published of the three pieces of research (Susan Owicki's): her verification of her Findpos example is outlined followed by attempts at verifying the same example using the earlier approaches. Reconsidering the prior approaches on the basis of Owicki's useful example illuminates similarities and differences between the proposals. Along the way, observations about interactions between researchers (and some "blind spots") are noted.

Prior work on sequential programs

The reference points for formal verification of *sequential* (non-concurrent) programs are taken here as [Flo67] and [Hoa69]. Around 1970, the technical challenges of verifying *concurrent* software came into focus; furthermore, as pointed out in [Ash75], debugging concurrent programs is so challenging as to offer a greater payback for formality.

Ashcroft/Manna

Zohar Manna wrote his PhD at CMU supervised by Floyd and Perlis. Manna moved to Stanford in 1969. Before he began the collaboration outlined below, he had written about non-deterministic programs in [Man69a]. Ed Ashcroft did his PhD in London University supervised by John Florentin.

The publication of interest here is [AM71] (pre-printed as a 1970 Stanford report [AM70]). The key insight is that concurrent programs can be reasoned about by expanding them into equivalent non-deterministic programs. There is a crucial caveat that the atomic steps of the concurrent threads must be identified. Providing this is done properly, it is not difficult to see that producing sequential threads that represent every possible merging of the original concurrent threads is possible. The expansion factor is however exponential and the programs tackled in [AM71] are rather limited: there are no specifications given for the examples whose tests and state changes are shown as uninterpreted symbols. More tellingly, one of two concurrent threads in their Program 3 contains exactly one (atomic) statement: finding all of the places into which this has to be merged in the other thread limits the cardinality of the expanded non-deterministic threads to (roughly) the number of atomic statements in the longer thread of the concurrent original.

The verification of each of the non-deterministic threads is done in a style that is derived from that of Manna's ex-supervisor Floyd. More precisely, the details actually reflect a proposal that Manna had published as [Man69b].

Ashcroft

Ashcroft moved to Waterloo in January 1971. The starting point for his next step was a recognition of the unacceptable size of the expanded non-deterministic program from the previous joint work. In [Ash75] he formalises an approach whose insight is to use control states that can be understood by thinking of keeping fingers on all of the statements that could be executed next in the concurrent threads. There are still potentially many transitions between control states but it is much clearer than in the previous work how they can be merged. Ashcroft tackles a significant example: a simplified airline reservation system. Both this and the previous joint approach cover blocking techniques that reduce the number of merge points.

In both approaches considered so far, a significant "blind spot" is effectively ignoring Hoare's 1969 paper. Furthermore, they offer only "post-facto" verification in the sense that they start with (a flowchart representation of) an existing program.

Owicki

(Some of this material benifited from a recent interview with Susan Owicki.)

Owicki's Cornell thesis [Owi75] is easily approached via a joint paper with her supervisor David Gries [OG76] — her approach is commonly referred to as the "Owicki-Gries method".

The insight here is that a separate Hoare-style sequential proof of one concurrent thread can be reviewed to see if statements in other threads interfere with steps of the sequential proof. Owicki's approach is clearly based on Hoare's 1969 paper and it is interesting to look at both stimuli from IFIP's WG2.3 and that group's role in amplifying the influence of the research.

Owicki's key example (*Findpos*) uses two concurrent threads to search for the lowest array index that points to a positive element. The threads interfere with each other so that either can cease searching beyond a satisfying index found by the other thread. Owicki's approach is based on [Hoa69] and comes closer to compositional development in the "posit-and-prove" style of [Hoa71].

Attempting Owicki's example with the earlier approaches

As the talk will illustrate, attempting to verify Owicki's *Findpos* example using the earlier approaches is revealing.

Working backwards, careful application of Ashcroft's solo approach can group his control states to much the same effect as Owicki's *interference freedom* check of proofs does. The distinction that remains is more revealing: whereas Owicki's method is based on Hoare's way of annotating programs, Ashcroft separates out the assertions about the program in a way that adumbrates proofs using temporal logics (s research area to which Manna made significant contributions).

Moving back further, the structure of *Findpos* is such as to render the Ashcroft-Manna approach impractical. The reason is that both branches have loop and conditional statements and the required expansion into non-deterministic threads explodes.

Acknowledgements

This research is supported by Leverhulme grant RPG-2019-020.

References

AM70. E. A. Ashcroft and Z. Manna. Formalization of properties of parallel programs. Technical Report AIM–110, Stanford Artificial Intelligence Project, 2 1970. Published as [AM71].

AM71. E. A. Ashcroft and Z. Manna. Formalization of properties of parallel programs. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, 6, volume 6, pages 17–41. Edinburgh University Press, 1971.

Ash75. Edward A Ashcroft. Proving assertions about parallel programs. *Journal of Computer and System Sciences*, 10(1):110–135, 1975.

Flo67. R. W. Floyd. Assigning meanings to programs. In J.T. Schwartz, editor, *Mathematical Aspects of Computer Science*, volume 19 of *Proc. of Symposia in Applied Mathematics*, pages 19–32. American Mathematical Society, 1967.

Hoa69. C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

Hoa71. C. A. R. Hoare. Proof of a program: FIND. *Communications of the ACM*, 14(1):39–45, January 1971.

Man69a. Z. Manna. The correctness of non-deterministic programs. Memo AI-95, Department of Computer Science, Stanford University, 8 1969.

Man69b. Z. Manna. The correctness of programs. *Journal of Computer and System Sciences*, 3(2):119–127, 5 1969.

OG76. S. S. Owicki and D. Gries. An axiomatic proof technique for parallel programs I. *Acta Informatica*, 6:319–340, 1976.

Owi75. S. S. Owicki. *Axiomatic Proof Techniques for Parallel Programs.* PhD thesis, Department of Computer Science, Cornell University, 1975. Published as technical report 75-251.

Philipp Macele (Leuphana University)

There is no Hardware – Lynn Conway and the Mead-Conway-Revolution

Session 2 | Wednesday, Oct 27, 11:50 – 12:20

While the first fully electronic computing systems did not differentiate between hardware and software, today Babylonian towers of programming languages are layering multiple levels of code on top of integrated circuits. These symbolic abstractions appear in the shape of interfaces, libraries, compilers and interpreters. Contrary to the claim of the German media theorist Friedrich Kittler that “there ist no software”, hardware then seems to play a relatively small role - an impression that is also reflected in scientific discourse through the existence of research fields such as software studies. Not least because on the other side of the hardware/software-dualism, the process of development has long since removed itself from the level of materiality. Chip design, as Kittler, writes in his paper "There is no software" on chip development based on the Intel 4004 microchip, is no longer done manually on drawing paper, but with the help of Computer Aided Design and Hardware Description Languages [1]. Yet in fact, no significant change in electronic circuit design did occur until the end of the 1970s with the emergence of the first VLSI chips. VLSI, short for Very-Large-Scale-Integration, describes a degree of integration of chips that contain 20,000 to 1,000,000 transistors. The degree of integration refers to the absolute number of transistors in an integrated circuit and results from the integration density, the number of transistors per unit area, and the chip size. LSI chips, so Large-Scale Integration chips, contain between 500 and 20,000 transistors; the Intel 4004 mentioned by Kittler has approx. 2300 transistors and therefore belongs into this category.

During my research on this topological change in the structure of chips, I came across the so-called Mead-Conway design methodology and after some research, this seems to be a decisive change in chip development and, among other things, the starting point for the development of computer aided design systems for the design of microchips. These programs enable the computer aided, partly automated design of chips. The systems allow a symbolic formal conception, which is automatically translated into a graphical mask. Kittler speaks of exactly this help in the development of chips when he writes of Computer Aided Design. But these elaborate computer-aided design techniques and methods did not appear suddenly, with the development of the Intel 4004, but emerged with and after the so-called Mead-Conway revolution.

So while Kittler dates the end of the human writing act to 1971 and the development of the Intel 4004, it has to be noted that even after the Intel 4004 chip, microchips continued to be designed with the help of drawing paper and pens, using the same methods and media as before. This changed with the spread of the Mead-Conway Design Methodology.

My talk will focus on this methodological change in chip design in the late 1970s and early 1980s which I would like to suggest made chip development a process somewhat independent from materiality. Starting from the 1976 DARPA report "Basic Limitations in Microcircuit Fabrication Technology" [2], that predicted the end of Moore‘s Law unless new design methods and system were developed which were better suited to the monolithic microchip fabrication process and the properties of silicon. I will describe how this report caused the brothers Ivan and Bert Sutherland to launch a DARPA-funded research project based at Xerox PARC and Caltech University that transformed microchip design from a material to a symbolic process. During their work, the research group pursued an approach of developing a completely new method based on the properties of silicon, but abstracted from the materiality and the manufacturing process in the design phase. I will outline how this departure from the field of electronics and the accompanying simplification became possible and could be methodically implemented. Finally, I will show how this method has spread and was institutionalized.

I will focus on both, the human actors such as Lynn Conway as well as the Sutherland brothers and non-human actors, like the ARPANET or the Xerox Alto workstations, and institutions, for example Xerox PARC or the M.I.T., that caused or enabled this change. The goal is to show that the VLSI design method of Lynn Conway and Carver Mead is a paradigm shift, that has completely changed chip development in the late 1970s and early 1980s.

Notes

[1] See: Kittler, Friedrich: Draculas Vermächtnis. Technische Schriften. Leipzig, Reclam Verlag 1993. S. 226 f.

[2] Ivan E. Sutherland, Carver A. Mead, Thomas E. Everhart: Basic Limitations in Microcircuit Fabrication Technology, o. O. 1976, online: https://www.rand.org/pubs/reports/R1956.html

Moritz Mähr (ETH Zurich)

The Public, the Private, and the Domestication of the Information System. How Data Protection Governed the Swiss Administration in the 1970s

Session 3 | Wednesday, Oct 27, 14:30 – 15:00

Drawing on a historical case study, this article examines the question of how society and its institutions in Switzerland adopted new technologies in the 1970s. It focuses on the Swiss Federal Administration and its electronic tools, and on the public debate about the laws and norms regulating the use of computers and personal data in the administration. One exemplary electronic tool was the Central Aliens Register [1], which had been in use as a statistical tool by the Federal Aliens Police since 1971. The connection of computer terminals to the IBM 360/65 mainframe computer transformed the statistical register into an interactive information system in the mid-1970s. This enhancement massively expanded the agency's scope of action, and the CAR became one of its most important tools for policing tasks. When a scandal involving an illegal collection of paper files became public in 1976, reminiscences of the Watergate scandal were awakened among the Swiss public and politicians. However, the paper files quickly faded into the background and fear of modern surveillance technologies dominated the discussion. An expert commission on data protection law then investigated the extent to which and how the use of computers in the administration needed to be regulated. The administration, for its part, feared that it would no longer be able to perform its duties adequately without electronic tools. The ensuing discussion in the public and the parliament negotiated not only the limits of the use of computers, but also the relationship between a computerized administration and its subjects.

For this study, press articles and tv broadcasts from various Swiss media as well as reports, minutes, correspondence, and other official sources from the Swiss Federal Archives are evaluated. To analyze this debate, the concept of the trading zone by Peter Galison is used. With trading zones he describes the simultaneous change of technical conditions, scientific knowledge and social patterns of interpretation. Trading zones illustrate how different thought collectives working with computers define common problems and develop a common language. Trading zones mark the formal as well as informal event spaces where problems, interactions and structural connections between digital processes and administrative routines can be reconstructed. The concept of trading zones thus enables a source-based analysis of the formal as well as informal connections between the public, the state executive and computer technology [2].

Thus, this research contribution fits into a history of computing and administration influenced by Science and Technology Studies. The studies of Agar, Johansson, and Danyel highlight that a history of computer-based administrations should be written as a history of reciprocal negotiation processes between the computer and the administration [3]. Studies at the intersection with Surveillance Studies are particularly interested in how personal registers were used to control a growing population and how they changed police agencies and their investigation methods. In this context, Linhardt, Berlinghoff, Bergien, Mangold, and Frohman address in their studies of computer use in the Federal Criminal Police Office of the FRG, the Ministry of State Security of the GDR, and in France the police access to police and non-police databases from the 1960s to the 1980s and productively relate them to the data protection debate [4]. Although computerized personal records and investigation methods came into public focus in the late 1970s in France and in the 1980s in Germany, the development of data protection should not be understood as a reaction to the public debate about data protection. As the above-mentioned studies show, the administration recognized data security and data protection at an early stage as areas of governmental and administrative action in need of regulation and created scope for design and action by playing an active role in legislation.

Notes

[1] Koller, "The Central Register of Foreigners. A Short History of Early Digitisation in the Swiss Federal Administration"; Espahangizi and Mähr, "The Making of a Swiss Migration Regime."

[2] See Galison, "Computer Simulations and the Trading Zone"; On the historicization of the concept and the question of its political implications, see Engemann and Schrickel, "Trading Zones of Climate Change"; On the "trading zone" as a method for historical knowledge production, see Fickers and Heijden, "Inside the Trading Zone".

[3] Agar, *The Government Machine; Johansson, Smart, Fast and Beautiful. On Rhetoric of Technology and Computing Discourse in Sweden 1955-1995*; Danyel, "Zeitgeschichte der Informationsgesellschaft".

[4] Linhardt, "Elemente einer politischen Soziologie der Polizei- und Bevölkerungsregister in Deutschland und Frankreich (1970er und 1980er Jahre)"; Berlinghoff, "»Totalerfassung« Im »Computerstaat« - Computer Und Privatheit in Den 1970er Und 1980er Jahren"; Bergien, "»Big Data« als Vision. Computereinführung und Organisationswandel in BKA und Staatssicherheit (1967–1989)"; Mangold, Fahndung nach dem Raster. *Informationsverarbeitung bei der bundesdeutschen Kriminalpolizei,* 1965-1984; Bergien, "Südfrüchte Im Stahlnetz."; Frohman, *The Politics of Personal Information*, chap. 3.

Bibliography

Agar, Jon. *The Government Machine: A Revolutionary History of the Computer*. History of Computing. Cambridge, Mass: MIT Press, 2003.

Bergien, Rüdiger. "»Big Data« als Vision. Computereinführung und Organisationswandel in BKA und Staatssicherheit (1967–1989)." *Zeithistorische Forschungen/Studies in Contemporary History* 14, no. 2 (August 15, 2017): 258–85. https://doi.org/10.14765/zzf.dok.4.969.

Bergien, Rüdiger. "Südfrüchte Im Stahlnetz. Der Polizeiliche Zugriff Auf Nicht-Polizeiliche Datenspeicher in Der Bundesrepublik, 1967-1989." In Wege in Die Digitale Gesellschaft: Computernutzung in Der Bundesrepublik, 1955-1990, edited by Frank Bösch and Martin Sabrow, 39–63. Geschichte Der Gegenwart, Band 20. Göttingen: Wallstein Verlag, 2018.

Berlinghoff, Marcel. “»Totalerfassung« Im »Computerstaat« - Computer Und Privatheit in Den 1970er Und 1980er Jahren.” In Im Sog Des Internets: *Öffentlichkeit Und Privatheit Im Digitalen Zeitalter*, edited by Ulrike Ackermann, 93–110. Frankfurt: Humanities Online, 2013.

Danyel, Jürgen. “Zeitgeschichte der Informationsgesellschaft.” Application/pdf. *Zeithistorische Forschungen/Studies in Contemporary* History 9, no. 2 (2012): 161–67. https://doi.org/10.14765/ZZF.DOK-1598.

Engemann, Christoph, and Isabell Schrickel. “Trading Zones of Climate Change: Introduction.” *Berichte Zur Wissenschaftsgeschichte* 40, no. 2 (2017): 111–19. https://doi.org/10.1002/bewi.201701847.

Espahangizi, Kijan, and Moritz Mähr. “The Making of a Swiss Migration Regime. Electronic Data Infrastructures and Statistics in the Federal Administration, 1960s–1990s.” *Journal of Migration History* 6, no. 3 (October 8, 2020): 379–404. https://doi.org/10.1163/23519924-00603005.

Fickers, Andreas, and Tim van der Heijden. “Inside the Trading Zone: Thinkering in a Digital History Lab.” *Digital Humanities Quarterly* 014, no. 3 (September 1, 2020).

Frohman, Larry. *The Politics of Personal Information: Surveillance, Privacy, and Power in West Germany*. New York: Berghahn, 2021.

Galison, Peter. “Computer Simulations and the Trading Zone.” In *From Science to Computational Sciences. Studies in the History of Computing and Its Influence on Today’s Sciences*, edited by Gabriele Gramelsberger, 97–130. Zürich: Diaphenes, 2011.

Johansson, Magnus. *Smart, Fast and Beautiful. On Rhetoric of Technology and Computing Discourse in Sweden 1955-1995*. Linköping: Linköping University Electronic Press, 1997.

Koller, Guido. “The Central Register of Foreigners. A Short History of Early Digitisation in the Swiss Federal Administration.” *Media in Action*, no. 1 (June 6, 2017): 81–92. https://www001.zimt.uni-siegen.de/ojs/index.php/mia/article/view/6.

Linhardt, Dominique. “Elemente einer politischen Soziologie der Polizei- und Bevölkerungsregister in Deutschland und Frankreich (1970er und 1980er Jahre).” In *Daten*, edited by David Gugerli, Michael Hagner, Michael Hampe, Barbara Orland, Philipp Sarasin, and Jakob Tanner, 99–116. Nach Feierabend 3. Berlin: Diaphanes, 2007.

Mangold, Hannes. *Fahndung nach dem Raster. Informationsverarbeitung bei der bundesdeutschen Kriminalpolizei*, 1965-1984. Vol. 23. Zürich: Chronos, 2017.

Mirjam Mayer and Ricky Wichum (ETH Zurich)

Public Data and Personal Computers. The Emergence of a Personal Computing Culture in the Swiss Federal Administration, ca. 1980

Session 6 | Thursday, Oct 28, 12:20 – 12:50

In her book *The Second Self. Computers and the Human Spirit* (1984), Sociologist Sherry Turkle described the emergence of a computing culture built around microcomputers. These computers, Turkle claimed, play a vital role as a catalyst of culture formation. What makes the first owners of American microcomputers an interesting sociological subject of inquiry was that they used the computer as a medium of reflection for personal, societal, political und educational matters. The new computing culture went hand in hand with a shift in meaning of the technical object: "Computers, long a symbol of depersonalization, were recast as ‚tools for conviviality‘ and ‚dream machines.‘ Computers, long a symbol of the power of the ‚big‘ – big corporations, big institutions, big money – began to acquire an image as instruments for decentralization, community, and personal autonomy." (Turkle 1984: 161)

Elsewhere in postmodern society, too, the personal computer provoked reflections on the existing. Drawing back on sources of the Swiss Federal Archive (BAR), our paper examines the emergence of a personal computing culture of the 1980s which included the use of personal computers as administration tools, information systems and New Public Management policies in the Swiss Federal Administration. In contrast to the private space, public administrations are legally highly formalized organizations. Thus, it was by no surprise that experimental fields were already set up there in the 1950s to test the new connections between administrative routines and digital processes in mainframe computers (Edwards 1996; Akera 2000; Agar 2003). But how did a highly structured organization deal with the new autonomy of users inside and outside, the strategies of decentralization, values of transparency, and even the sense of community that unfolded, especially in the private sphere?

Computing culture is not a peaceful space of ignorant users and charitable technology companies. Rather, computing cultures are spaces of sociotechnical change. Computing cultures are shaped by conflictual reconfigurations of organizational power, alliances, forms of communication, conceptual work, user interests, and technological change (Stark 2009; Rankin 2019; Gugerli/Wichum 2021). To bring these contested trading zones between the computer and its environments into focus, we study two interfaces between computers and administration.

First, we examine an information system called “STATINF”, which was introduced in 1986 after more than ten years of conceptual work. Users now could not only access governmental databases from the comfort of their homes and study public data conveniently on their private screens. They could also retrieve data from various databases according to their individual tastes, have them combined, and then edit them visually to their liking by graphical software. The arcanum of state knowledge has become arbitrarily individualizable and reproducible via private printers and floppy disks. For this interface "EDP training" was not required (Bundesamt für Statistik 1986).

Second, we investigate the interface between the civil servant and the personal computer that was built by the use of the PC and its various peripherals in the Swiss Federal Administration in the 1980s. The Federal Office of Organization (Bundesamt für Organisation), in a sense the ultimate supervisor of bureaucratic computer use, had to decide how much personality of its users could be moved and be allowed into the PC. The federal personnel had to be taken into account for any efforts in office automation projects as autonomous and potentially unreliable user (Schwendener 1983).

In the last part of our paper, we conclude the two case studies from the Swiss Federal Administration in the 1980s. We discuss consequences and effects that resulted from this personal computing culture for the knowledge of the state, the autonomy of citizens and civil servants, and the distinction between private and public.

References

J. Agar, *The Government Machine. A Revolutionary History of the Computer.* Cambridge, Massachusetts; London: The MIT Press, 2003.

A. Akera, "Engineers or Managers? The Systems Analysis of Electronic Data Processing in the Federal Bureaucracy," in *Systems, Experts, and Computers*, A. C. Hughes and T. Hughes, Eds. Cambridge: The MIT Press, 2000, pp. 191–220.

Bundesamt für Statistik, *Was ist STATINF?*, Bern 1986.

P. N. Edwards, *The Closed World. Computers and the Politics of Discourse in Cold War America*. Cambridge, Massachusetts: MIT Press, 1996.

D. Gugerli and R. Wichum, *Simulation for All. The Politics of Supercomputing in Stuttgart*. Zürich: Chronos, 2021.

J. L. Rankin, *A People's History of Computing in the United States*. Cambridge, MA.; London: Harvard University Press, 2018.

F. Schwendener, *Persönliche Computer für Verwaltungsaufgaben? Eine Studie mit Informationen, Empfehlungen und Einsatzkonzept für die Bundesverwaltung*, Bern, 1983.

D. Stark, *The Sense of Dissonance. Accounts of Worth in Economic Life*. Princeton; Oxford: Princeton University Press, 2009.

Maximilian Noichl (University of Vienna)

The Epistemic Vices and Virtues of Dimensionality Reduction

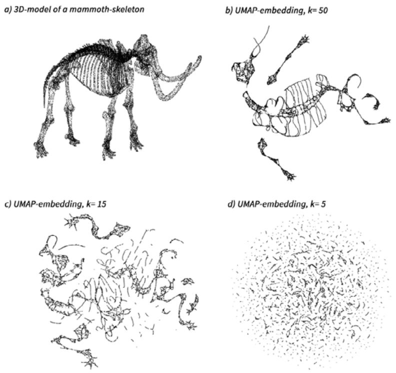
Session 9 | Friday, Oct 29, 9:00 – 9:30

Despite their ubiquity in scientific and commercial data analysis, techniques of *dimensionality-reduction* have only attracted little philosophical interest [1]. Dimensionality-reduction refers to the process, by which the variables of a dataset are reduced in number in such a way, that the resulting derivative dataset retains the most important properties of the original one. This can be useful both as a preprocessing step for further analysis, e. g. clustering, as well as visualization in two or three dimensions.

Linear methods for dimensionality reduction like *Principal Component Analysis* have been in use for nearly a century and have found application in many different areas of science, in chemistry, as well as linguistics [2], genetics [3] and psychology [4].

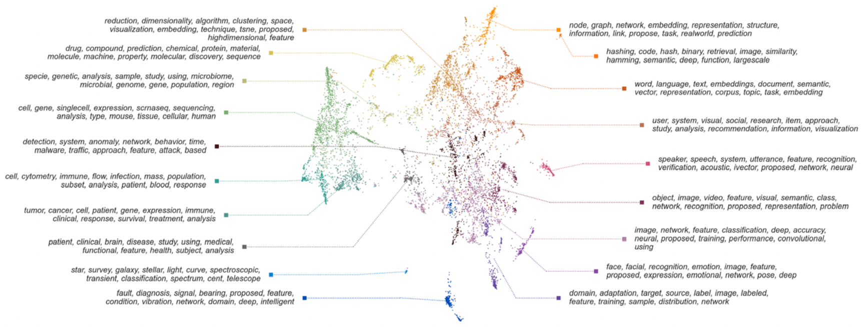
While there has been philosophical discussion of one method of linear dimensionality reduction, philosophers seem to have paid no attention at all to methods of non-linear dimensionality reduction, like t-SNE [5], UMAP [6] and Tmap [7] which have been rapidly adopted in domains as different as genomic databases, recordings of the songs of whales [8] and birds [9], to visualize the prime factors of a million numbers [10], and to classify images or texts.

All these techniques allow for the analysis and especially the visualization of far more complex structures than linear methods, in a wide variety of domains. But as Fig. 1 illustrates, the structures they produce can be highly sensitive to the choice of hyperparameters as well as random initialization.



A demonstration of the influence of the nearest-neighbors parameter k on the UMAP-embedding of a 3-dimensional model (a) of a mammoth skeleton. (Courtesy: Smithsonian Institute). All values for k in the embeddings (b)-(d) are within the commonly recommended range. While (b) gives a good two-dimensional representation of the original topology, (c) is very disjoint, and (d) completely unreadable. Note that in common applications of UMAP the datasets will span not three, but many thousand dimensions, making visual "sanity checks" impossible.

To trace the methods' proliferation across various disciplines we collect a sample of articles with citations to the original technical publications which introduced these techniques. An example of such an analysis that shows the broad range of applications is given in Fig. 2. We link the results from this study to an investigation of the formal and informal information sources which surround these techniques, including documentations, comments on open-source code, tutorials, and message-board discussions.



A UMAP-plot of 9162 articles, arranged by thematic similarity, that make use t-SNE, UMAP, Tmap or TriMAP. The wide range of applications is clearly visible.

Based on this analysis, we argue that the adoption of nonlinear dimensionality reduction brought with itself a shift in the salience of several epistemic virtues, as practitioners took it up to stress values like accessibility, interactivity, explorability and transparency of the mutability of results, over immediate comprehensibility and traceability. Under these conditions, non-epistemic values of software-implementation in contemporary science, like speed and ease of use interweave with epistemic ones.

Literature

Aston, J. A. D., Chiou, J.-M., & Evans, J. P. (2010). Linguistic pitch analysis using functional principal component mixed effect models. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 59(2), 297–317. doi: 10.1111/j.1467-9876.2009.00689.x

Baird, D. (1987). Exploratory Factor Analysis, Instruments and the Logic of Discovery. *The British Journal for the Philosophy of Science*, 38(3), 319–337.

Lenz, M., Müller, F.-J., Zenke, M., & Schuppert, A. (2016). Principal components analysis and the reported low intrinsic dimensionality of gene expression microarray data. *Scientific Reports*, 6(1), 25696. doi: 10.1038/srep25696

Maaten, L. van der, & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(Nov), 2579–2605.

McDonald, K. (2019). Data of the Humpback Whale. https://medium.com/@kcimc/data-of-the-humpback-whale-9ef09c5920cd.

McInnes, L., Healy, J., & Melville, J. (2018). UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. arXiv:1802.03426 [Cs, Stat]. Retrieved from http://arxiv.org/abs/1802.03426

Mulaik, S. A. (1991). Factor Analysis, Information-Transforming Instruments, and Objectivity: A Reply and Discussion. The British Journal for the Philosophy of Science, 42(1), 87–100.

Probst, D., & Reymond, J.-L. (2020). Visualization of very large high-dimensional data sets as minimum spanning trees. *Journal of Cheminformatics*, 12(1), 12. doi: 10.1186/s13321-020-0416-x

Sainburg, T., Thielk, M., & Gentner, T. Q. (2019). *Latent space visualization, characterization, and generation of diverse vocal communication signals* [Preprint]. Animal Behavior and Cognition. doi: 10.1101/870311

Spearman, C. (1904). „General Intelligence," Objectively Determined and Measured. *The American Journal of Psychology*, 15(2), 201–292. doi: 10.2307/1412107

Williamson, J. (2018). *What do numbers look like?* https://johnhw.github.io/umap_primes/index.md.html.

Wills, M. (2017). Are Clusters Races? A Discussion of the Rhetorical Appropriation of Rosenberg Et Al.'S "Genetic Structure of Human Populations". *Philosophy, Theory, and Practice in Biology*, 9(12). doi: http://dx.doi.org/10.3998/ptb.6959004.0009.012

Philippos Papayannopoulos, Nir Fresco and Oron Shagrir (Hebrew University of Jerusalem, Ben Gurion University of the Negev)

On Two Different Kinds of Computational Indeterminacy

Session 4 | Wednesday, Oct 27, 17:00 – 17:30

A variety of fields, from physics and biology to the cognitive and brain sciences, have begun to import the language of computation and information-processing to describe their studied systems. Interpreting certain physical systems as carrying out specific computations can provide additional insights into a system's behavior and organization. However, characterizing a system computationally is a challenging task, even when the system's physical behavior is sufficiently known. This is because there may be more than one computational profile consistent with the same, underlying physical behavior. In other words, some systems exhibit what is often described as *computational indeterminacy*: a detailed description of the physical dynamics of a (closed) system cannot pin down a unique computational identity. To determine such an identity, typically some external knowledge and hypotheses about the system are invoked as well; perhaps parameters about its environment, its computational goals, the semantic content of its computational states or others. This is a well-known phenomenon, of course, which has been extensively discussed in the literature, under various different guises and names.

The aim of this talk is to show that what has typically been considered and referred to as the phenomenon of computational indeterminacy, in fact subsumes two separate phenomena that ought to be distinguished. We present the two phenomena as two different kinds of indeterminacy and attempt to show that their varied nature is owing to different inter-level relations pertaining to the process of computationally individuating the system. One kind of indeterminacy arises at the level of formally/functionally characterizing the system, and another kind arises at the level of interpreting what the computation is about.

More precisely, suppose that we are trying to develop a computational specification for a given physical system, such as a neuronal network or an artifact, with known physical behavior. Formulating a computational description involves more than just describing the physical dynamics of the system in some mathematical formalism. It requires also some abstract formalism, couched in computational terms (e.g., states of some finite-state automaton, Boolean or other mathematical functions), a correspondence between physical states and states of the abstract formalism, and some interpretation of what the physical states or the terms in the abstract formalism represent. The abstract formalism may or may not be distinct from the mathematical formalism that describes the physics of the system. For example, a system whose physical behavior is, at some level of analysis, described by the equation $Vo = V_1+V_2$ may or may not be said to realize a computation of the sum of two input voltages, V_1, V_2, according to the goals and the context of the system's study (or its intended application, if it's a designed system). Furthermore, as we intend to show, this particular choice may be related to computational indeterminacy as well.

An abstract formal structure and some interpretation is what the scientist tries to determine, in computationally individuating a physical system (e.g., in computational neuroscience). The first challenge, then, is how to carve up the physical state space in order to correspond physical states to abstract/functional ones, thereby giving rise to some relevant abstract formalism or some functional organization of the system. In other words, the first question concerns which physical states to group together into parts of the same abstract/functional state. As we explain in the talk, there are systems susceptible to varied ways of grouping physical states together, such that there is no inherent reason for preferring one way to another; consequently, some external restrictions may be required to functionally characterize the system (e.g., contextual, mechanistic, or semantic). This is the first kind of encountered indeterminacy, which we call "*functional indeterminacy*".

Assuming the abstract/functional organization specified, there exists a second kind of indeterminacy that pertains to computational individuation. This concerns specifying what is computed by the system; that is, interpreting the content of the abstract/functional states or interpreting what the physical states represent [1]. Sometimes a given functional profile can allow of more than one semantic interpretation of it. For example, consider a physical system whose functional profile is consistent with implementing some logic gate (say, an AND). For certain gates, by swapping the logic values that we take a certain voltage range to represent, we obtain a different (but related) Boolean function, which is called the dual of the first gate (in our case, an OR). The choice between the two, then, is too indeterminate, unless some external restrictions are provided. We call this second kind of indeterminacy "*interpretative indeterminacy*".

Decoupling the two kinds of indeterminacy has two significant merits. First, pointing out that what has traditionally been regarded as a unified problem is actually two separate phenomena, each of a different underlying nature, offers new insight into the problem of computational individuation. The various accounts of computational individuation in the literature purport to deal with the challenge of computational indeterminacy differently; thus, in assessing these accounts it is instructive to see which kind of indeterminacy exactly each of them faces and whether the other kind is still a remaining challenge or not. Second, we suggest that the two kinds of indeterminacy, when taken at their limits, can be seen as underlying two important, unrelated thus far, problems in philosophy of computation: triviality arguments and deviant encodings. The former can be seen as a limiting case of functional indeterminacy, whereas the latter can be seen as a limiting case of interpretative indeterminacy.

Finally, computational individuation intertwines with computational implementation, in a way that makes it difficult to give a nuanced account of the former without taking into consideration accounts of the latter. Although we do not defend any specific views of “implementation” and “computation” in this talk, we do examine how our proposed accounts of the two kinds of indeterminacy relate to some main accounts of “implementation” and “computational individuation” in the literature.

Notes

[1] The disjunction in this clause is owing to the fact that determining what the scope of the relevant interpretation is (the intermediate abstract/functional states or directly the physical states) depends on one’s precise accounts of “computation” and of “computational individuation”.

Tomas Petricek and Joel Jakubovic (University of Kent)

Complementary Science of Interactive Programming Systems

Session 5 | Thursday, Oct 28, 10:00 – 10:30

Is it worth looking at the history of programming, not just for the sake of history, but in order to discover lost ideas that could be utilized by present-day computer scientists to advance the state of the art of programming? We answer the question in the affirmative, propose a metho¬dology for doing so and present an early experiment that recovers a number of interesting programming ideas from, against all odds, Commodore 64 BASIC.

Programming systems and languages

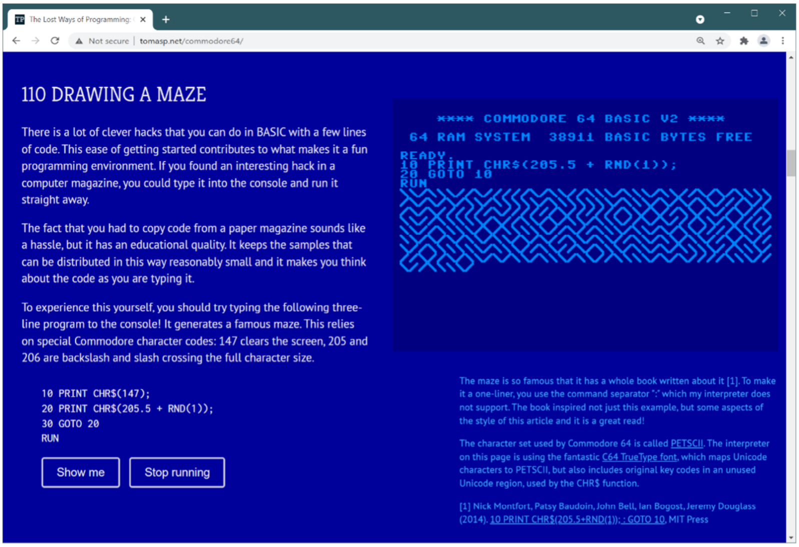
The history of programming may seem too short to comprise multiple incommensurable scientific paradigms, but there have certainly been interesting shifts over time. I focus on the shift from *programming systems to programming languages* discussed by Gabriel (2012) and hinted by Priestley (2011). Work on programming systems considers interactive environments like Smalltalk or Interlisp. It focuses on what is happening in the system; a study of programming system considers source code, state of the running system as well as user interactions. Now dominant work on programming languages focuses on meaning of a program and considers solely the code. This shift enabled much contemporary programming language research and led to the development of useful program analysis tools, including e.g., type systems. But what ideas have been victim to Kuhn loss associated with the paradigm shift? What programming ideas are lost if we only think in terms of programming languages?

Past work on programming systems explored, for example, ways of modifying programs while running (Bobrow et al., 1988) and interesting ways of constructing programs by interacting with the programming environment (Smith, 1977). Such research cannot be easily expressed in terms of the *programming language* research paradigm and many of the systems that implemented it are no longer available in an executable format. So, how can we recover such ideas and make them available to present-day computer scientists?

Complementary science

To be effective, specialist science needs to accept certain assumptions about what it studies and computer science is no difference. However, this means that there are interesting questions outside of its core focus. The methodology of *complementary science* proposed by Chang (2012) aims to study such questions. It proposes looking at the history of science, and use the historical perspective to recover forgotten knowledge, use it for critical assessment of contemporary scientific research and attempt to further develop such forgotten ideas. As Chang suggests, such extension of past research is worthwhile in cases where scientists abandoned (some aspects of) a scientific research paradigm too early.

We argue that the move from programming systems to programming languages is one such case where computer scientists abandoned a research paradigm too early. In the case of programming, the methodology may be even more valuable. First, programming research does not aim for truth, but aims for adequacy (Cross, 2006). As such, it may abandon ideas not because of any experimental failure, but for more social reasons. Second, ideas on programming also do not come solely from the world of (computer) science as many innovative systems are built by practitioners and "hackers". Complementary science provides a method that can reintegrate such ideas into the realm of academic research.


The Lost Ways of Programming:
Not secure | tomasp.net/commodore64/

110 DRAWING A MAZE

There is a lot of clever hacks that you can do in BASIC with a few lines of code. This ease of getting started contributes to what makes it a fun programming environment. If you found an interesting hack in a computer magazine, you could type it into the console and run it straight away.

The fact that you had to copy code from a paper magazine sounds like a hassle, but it has an educational quality. It keeps the samples that can be distributed in this way reasonably small and it makes you think about the code as you are typing it.

To experience this yourself, you should try typing the following three-line program to the console! It generates a famous maze. This relies on special Commodore character codes: 147 clears the screen, 205 and 206 are backslash and slash crossing the full character size.

```
10 PRINT CHR$(147);
20 PRINT CHR$(205.5 + RND(1));
30 GOTO 20
RUN
```

```
    **** COMMODORE 64 BASIC V2 ****
 64 RAM SYSTEM  38911 BASIC BYTES FREE
READY.
10 PRINT CHR$(205.5 + RND(1));
20 GOTO 10
RUN
```

The maze is so famous that it has a whole book written about it [1]. To make it a one-liner, you use the command separator ":" which my interpreter does not support. The book inspired not just this example, but some aspects of the style of this article and it is a great read!

The character set used by Commodore 64 is called PETSCII. The interpreter on this page is using the fantastic C64 TrueType font, which maps Unicode characters to PETSCII, but also includes original key codes in an unused Unicode region, used by the CHR$ function.

[1] Nick Montfort, Patsy Baudoin, John Bell, Ian Bogost, Jeremy Douglass (2014). 10 PRINT CHR$(205.5+RND(1)); : GOTO 10, MIT Press

Figure 1. Reconstruction of a programming environment for Commodore64 BASIC, running a well-known one-liner for generating a maze (Montfort et al, 2012). Available at http://tomasp.net/commodore64

Case study: Reconstructing Commodore64 BASIC

In our initial work on studying programming systems using the method of complementary science, we explore the interactive programming environment of Commodore64 BASIC (Figure 1). This is a perfect example of a programming system outside of the mainstream computer science – the BASIC language "mutilates the mind beyond recovery" (Dijkstra, 1984) and the Commodore64 programming environment seem like a toy for hobbyists. Yet, the programming environment has a number of interesting characteristics. For example:

- *Interaction uniformity*. The system uses a single interaction for both editing programs and running programs. If a line starts with a number, it is inserted into the program, otherwise it is executed directly.

- *Simple & flexible workspace*. The use of line numbers makes it easy to write and test parts of a program independently, as well as to debug programs (without a dedicated debugger) by setting variable values and jumping to a desired line.

Studying programming systems

Interactive programming systems, both past and present, present an interesting challenge for their rigorous study. In particular, many of their interesting features are only apparent when following a realistic interaction with the system. This is difficult to reconstruct for systems that are no longer available or rely on specialized hardware. Edwards et al. (2019) review a number of possible methods that have been used for presentations of interactive programming systems in recent years, including video recordings and interactive web-based essays. We believe those approaches can be fruitfully combined with the complementary science methodology.

In particular, the Commodore64 BASIC reconstruction (Figure 1), developed by the first author, is an interactive web-based essay that combines a tutorial walkthrough that illustrates the development of a simple game with an interactive console where the game can be run. The console only supports a limited subset of the system, but is representative enough to illustrate the interesting aspects of the programming environment. The reader can type and run code on their own, but also use the “Show me” button to replay the interaction.

Conclusions

We believe that the history of programming contains numerous interesting paths not taken that are worth recovering and that could provide inspiration for contemporary research. Doing so is, however, not simply a matter of reading old academic papers. In order to gain a fundamental understanding, we need a historical perspective that understands systems through the perspective of the paradigm from which they originate. The case of *programming systems* vs. *programming languages* illustrates this point well. If we look merely at the language used by a programming system, we can easily miss the most important characteristic of the system, that is how the programmer interacts with it. To realize this, we need a historically-informed reading of the system. Finally, reconstructing ideas about interactive programming systems and making them accessible to contemporary computer scientist (a goal we adopt from *complementary science*) also requires the use of suitable media that allow interaction – such as interactive web-based essays.

References

Gabriel, R.P. (2012). *The structure of a programming language revolution*. In Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software (pp. 195-214).

Priestley, M. (2011). *A science of operations: machines, logic and the invention of programming*. Springer Science & Business Media.

Bobrow, D.G., DeMichiel, L.G., Gabriel, R.P., Keene, S.E., Kiczales, G. and Moon, D.A. (1988). *Common lisp object system specification*. ACM Sigplan Notices, 23(SI), pp.1-142.

Smith, D.C. (1977). *A computer program to model and stimulate creative thought*.
Basel: Birkhauser.

Chang, H. (2012). *Is water H2O? Evidence, realism and pluralism* (Vol. 293). Springer Science & Business Media.

Cross, N. (2006). *Designerly Ways of Knowing*. ISBN 978-1-84628-300-0. Springer.

Montfort, N., Baudoin, P., Bell, J., Douglass, J. and Bogost, I. (2014). 10 PRINT CHR$(205.5 + RND(1));: GOTO 10. MIT Press

Dijkstra, E. (1984). *The threats to computing science* (EWD898). Delivered at ACM 1984 South Central Regional Conference, November 16–18, Austin, Texas.

Edwards, J., Kell, S., Petricek, T. and Church, L. (2019). *Evaluating programming systems design*. In proceedings of PPIG 2019.

Camilla Quaresmini and Giuseppe Primiero (University of Milan)

Data Quality Dimensions for Fair AI

Session 7 | Thursday, Oct 28, 15:00 – 15:30

AI systems are not intrinsically neutral. Biases trickle into smart objects in the urban space, in any type of technological tool, from filter apps on social media to electronic credit loans. The gap between rich and poor, the confusion of biological sex and gender, the deficit due to prejudices related to the social identity of a subject are becoming more constitutive because they are increasingly mechanized (O'Neil 2016). In particular when dealing with people, AI algorithms reflect technical errors originating with the problem of mislabeling data. Recurrently, systems mistakenly identify patterns in data, making wrong classifications, as illustrated in (Crawford and Paglen 2019b) and (Crawford and Paglen 2019a). Especially with mass biometric surveillance, such biases are becoming mainstream, as those systems feed wrong and discriminatory classifications (Buolamwini and Gebru 2018), perpetuating structural racism and systemic marginalisation (Leslie 2020). Automatic decision making systems work with gate-keepers algorithms using data to make decisions that affect people's lives. Detailed examples are given since (Friedman and Nissenbaum 1996) and more recently in (O'Neil 2016). These systems are not systematically guarded against bias. Their fairness can be formulated in two phases: first, identifying and correcting problems in datasets, as a model trained with mislabeled datasets will provide biased outputs; second, correcting the algorithms (Hooker 2021), as even in the design of algorithms biases can emerge. In this paper we focus in particular on the first problem.

Our contribution relies on an analysis of the CleanLab toolkit (Northcutt, Jiang, and Chuang 2021) whose aim is to mitigate biases in the sets of data attempting to implement diversity. Fairness is a multifaceted social construct, depending on context and culture, hence different definitions of fairness are possible. We show how tools like CleanLab define fairness in relation to a reference group, focusing mainly on the accuracy and completeness dimensions of data. Similar cases can be made for other tools like AIF360 (Bellamy et al. 2018) and Aequitas (Saleiro et al. 2019). These tools focus on predictive accuracy based on a metric of group/individual fairness. But other dimensions of data quality as studied in the literature (Batini and Scannapieco 2006; Illari and Floridi 2014; Redman 1996; Wang and Strong 1996) appear necessary and – we claim – quality cannot be improperly reduced to mere accuracy.

We provide an analysis of data quality dimensions which can be used to extend the reach and usefulness of fairness assessment tools. We start considering correctness, to identify cases in which a datapoint can move from a privileged to an unprivileged group, or viceversa. Then relevance allows us to identify the appropriate attributes on which to build privileged vs. unprivileged groups, and to characterize the limits of such identification. Completeness is considered when deciding how to work with missing data to categorize groups as privileged or unprivileged. Finally, as the very categorization in privileged vs unprivileged groups must also be disjoint by default, the consistency dimension is called upon, given the possibility of datapoints which can be categorized in both partitions is in principle possible, but technically excluded. In fact, the existence of a non-categorical datapoint, e.g. non-binary on a gender predicate, is absolutely crucial to fairness problems.

To validate our approach, we propose a case study of the FERET database (n.d.), a collection of 14.126 facial images gathered between 1993 and 1996 to test and evaluate face recognition algorithms. This was created as a part of a program with the aim of developing technologies to be used by law enforcement personnel to increase security. By way of example we focus on the attribute Gender, considered noisy because it is no longer representative of real world settings, as the label set only includes "male" and "female" and several other possible categories are ignored. For each datapoint, the algorithm calculates the projected probability that an image is assigned to the respective label. We construct a study case by considering noisy examples passed to the algorithm. A first noisy case with respect to such a classifier would be represented by the inclusion of images of individuals which might identify as non-binary, and as such the corresponding data point would be wrongly classified in either label; a second case, would be images of transgender individuals, and as such the corresponding data point would be correctly classified in both label. In the former case, the label set becomes incomplete with respect to the dataset; in the second case, the dataset is inconsistent with respect to the label set. In either case, the identification of a privileged category will force a biased choice on the labeling dataset with examples of non-binary classifications. A simple binary category, e.g. the gender one, becomes noisy with respect to a dataset including datapoints that lack a category, so the label becomes incomplete, or for datapoints that are identifiable under both label, and therefore the labels are not disjoint. These two examples correspond in CleanLab to the output of a classifier as a matrix with a mislabeling error rate of 100%, and 50% respectively.

On the basis of this example, we provide a theoretical characterization of dimensions useful for fair AI, with the aim of laying the ground for an extension of associated assessment tools. In particular, we try to understand how we can improve the classification algorithm in the case of noisy examples by an incomplete or inconsistent label set.

n.d. https://www.nist.gov/itl/products-and-services/color-feret-database.

Batini, Carlo, and Monica Scannapieco. 2006. *Data Quality: Concepts, Methodologies and Techniques. Data-Centric Systems and Applications*. Springer.

Bellamy, Rachel K. E., Kuntal Dey, Michael Hind, Samuel C. Hoffman, Stephanie Houde, Kalapriya Kannan, Pranay Lohia, et al. 2018. "AI Fairness 360: An Extensible Toolkit for Detecting, Understanding, and Mitigating Unwanted Algorithmic Bias." https://arxiv.org/abs/1810.01943.

Buolamwini, Joy, and Timnit Gebru. 2018. "Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification." In *Conference on Fairness, Accountability and Transparency, FAT 2018, 23-24 February 2018, New York, NY, USA*, 81:77–91. Proceedings of Machine Learning Research. PMLR. http://proceedings.mlr.press/v81/buolamwini18a.html.

Crawford, Kate, and Trevor Paglen. 2019a. “Excavating AI: The Politics of Training Sets for Machine Learning.” The AI Now Institute, NYU. 2019. https://excavating.ai.

Crawford, Kate, and Trevor Paglen. 2019b. *Training Humans*. Quaderno, # 26. Milan: Fondazione Prada.

Friedman, Batya, and Helen Nissenbaum. 1996. “Bias in Computer Systems.” *ACM Transactions on Information Systems (TOIS)* 14 (3): 330–47. https://doi.org/10.1145/230538.230561.

Hooker, Sara. 2021. “Moving Beyond “Algorithmic Bias Is a Data Problem“.” *Patterns* 2 (4). https://www.cell.com/action/showPdf?pii=S2666-3899%2821%2900061-1.

Illari, Phyllis, and Luciano Floridi. 2014. *The Philosophy of Information Quality*. Springer International Publishing.

Leslie, David. 2020. “Understanding Bias in Facial Recognition Technologies.” *CoRR* abs/2010.07023. https://arxiv.org/abs/2010.07023.

Northcutt, Curtis G., Lu Jiang, and Isaac L. Chuang. 2021. “Confident Learning: Estimating Uncertainty in Dataset Labels.” *Journal of Artificial Intelligence Research* (JAIR) 70: 1373–1411. https://arxiv.org/abs/1911.00068.

O’Neil, Cathy. 2016. *Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy*. New York: Crown.

Redman, Thomas C. 1996. *Data Quality for the Information Age*. Massachusetts: Artech House.

Saleiro, Pedro, Benedict Kuester, Loren Hinkson, Jesse London, Abby Stevens, Ari Anisfeld, Kit T. Rodolfa, and Rayid Ghani. 2019. “Aequitas: A Bias and Fairness Audit Toolkit.” https://arxiv.org/abs/1811.05577.

Wang, Richard Y., and Diane M. Strong. 1996. “Beyond Accuracy: What Data Quality Means to Data Consumers.” *J. Manag. Inf. Syst*. 12 (4): 5–33. http://www.jmis-web.org/articles/1002.

Edith Schmid (ETH Zurich)

Computing Systems as Social Institutions

Session 1 | Wednesday, Oct 27, 10:30 – 11:00

In the second half of the 20th century, we became witness to the huge transformational power IT had on businesses and corporate culture. With the turn of the century, due the extensive usage of the internet and digital services, strongly facilitated by smart phones, IT has also started entering our private circles. Computing systems have by now become an integral part of our cultural habitats. My contribution aims to facilitate the dialogue between the disciplines concerned with the effects of this IT-enabled, new industrial revolution on our cultural habitats.

A prerequisite for such a dialogue is a mutual and shared understanding of the concepts ‚Computing System' and ‚Culture'. What do the participants in the dialogue mean when they use these terms?

In Social Sciences, we tend to conceptualize a computing system as a kind of machine, a technical system, or a tool. While this is not wrong, it's a characterization of a computing system that is way too abstract in the eyes of a Computer Scientist. A computing system is an artefact that has been built to process information, is linked to information sources, and feeds information into other systems. While computing disciplines have sophisticated language to describe such systems, this language is too abstract for Social Sciences. On the other hand, it does not suffice to understand today's computing systems simply as distributed systems, consisting of hardware like chips up to complex cloud systems and supercomputers.

Vice versa, Computer Scientists mainly perceive humans as users or customers of computing systems, or as data generating resources. Naturally, they deal with humans in their cultural habitats, since the usability of a computing system is key for its success, and they apply methods borrowed from Social Sciences, particularly in the phase of system designs, such as e.g., Ethnomethodology. Still, the notion of ‚Culture' in IT is seen from a very particular perspective, as a kind of ground on which technical artefacts must be built, and less from a historical or philosophical perspective of how cultures change, i.e., how they are established, how they evolve, or how they fade away.

Social Institution as common ground

I would like to propose *the concept of a social institution as common ground*. (I say *social* institution to emphasize the involvement of human communities):

- I invite Social Scientists to look at *computing systems as social institutions*, and to challenge computing systems in the same way as they did it when institutions were transformed by previous industrial revolutions, e.g., by the steam engine or the assembly line.
- Vice versa, I invite Computer Scientists to look at social *institutions as programs controlling human agents*.

My *thesis* is: *computing systems*, designed and produced by the IT industry as the engines of the digital industrial revolution, and social institutions, as constitutive entities of our societies, *share a common structural core*. This holds, I will argue, not just metaphorically, but structurally in a strong sense. This common structural core will then allow for shared concepts in Computer Sciences and Social Sciences to reach a mutual understanding of computational practices in our cultural environment: I propose a structural model and a terminology that is translatable for both Social Sciences and Computer Sciences in their respective conceptual worlds.

What are social institutions?

In Social Sciences and Philosophy, institutions are considered a constitutive, *structuring element of societies or cultures*. However, there is no consensus on the definition of the term ‚institution'. E.g., in economics, emphasis is put on transactions and the coordinating function of institutions, whereas in law it's about the nature of their legal norms, and conflict resolution.

Given that there exist different definitions of institutions, I am not trying to propose a full *theory of institutions*. Rather, I will describe a *structural core* that I believe is shared by *all* kinds of institutions. As such, my approach is neutral to various theoretical approaches. Moreover, I will show that the proposed structural elements are also present in computing systems, if we consider them dynamically, over their full lifecycle.

Structurally speaking, the *function* of institutions in human societies is to govern the behavior of social agents, ranging from individuals to groups, such as families or organizations. A given institution constrains the set of possible actions an agent may perform in a given situation by forbidding some actions and allowing others. Think about marriage, the traffic system, universities, religions, meetings, marketplaces, or the way we greet each other.

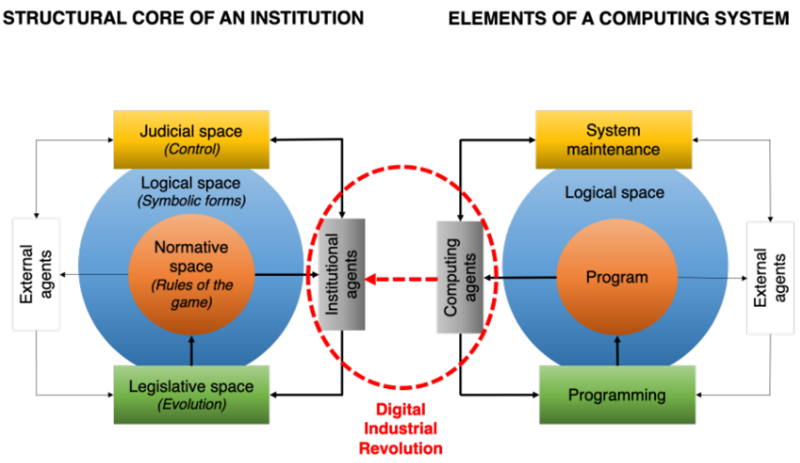
Metaphorically speaking, institutions act in communities of human agents the way a program does in a computing system relative to its machines. It is here that we can find a common ground where Social Science and Computer Science can conceptualize social institutions and computing systems in the same manner: namely as a kind of program controlling societies of agents – human agents in social institutions, artificial agents (or machines) in computing systems, respectively.

Such a common understanding is pressing, since human agents, who live in any given cultural habitat and follow the rules of its institutions, are increasingly confronted with artificial agents as fellow actors. My proposed model provides a common ground on which we can discuss and design the new *hybrid communities* of human and artificial agents to better control the transformational impact the latter have on our cultural habitats.

The structural core of an institution

I propose that the structural core of social institutions on the one hand consists of two static elements, which I call the logical, and the normative space, respectively. They are static in the sense that they must remain unchanged while we play the institutional ‚game', namely the symbolic forms we use and the rules we apply. On the other hand, the structural core of institutions must also model possible changes of an institution. For this it contains two dynamical elements which guide the changes of the static elements and monitor the agent's actions: I call them the institution's legislative and judicial space, respectively. Here lie the processes that control the ‚game', on the object level of the agent's actions as well as on the metalevel of the structure of the institution, which is why they are called dynamic.

In a computing system, the corresponding elements would be the following: for the logical space, we can use the same terminology. With regards to the normative space, this is what corresponds to the program of a computing system. As for the judicial space, it's what can be considered system maintenance, whereas the legislative space is the act of programming.


STRUCTURAL CORE OF AN INSTITUTION
ELEMENTS OF A COMPUTING SYSTEM
Judicial space (Control)
Logical space (Symbolic forms)
Normative space (Rules of the game)
Legislative space (Evolution)
External agents
Institutional agents
Computing agents
System maintenance
Logical space
Program
Programming
External agents
Digital Industrial Revolution

We experience the cultural effects of IT where artificial agents of computing systems come to interact with the human agents of the social institutions, as e.g., if self-driving cars enter the road traffic ‚game'. The red circle in the figure above illustrates the emerging hybrid communities that are of interest in my research.

Summary and outlook

With the proposed structural model for institutions, if we see computing systems also dynamically, including its specification, programming, and management, it should be quite natural for Philosophers and Social Scientist to conceive them as institutions. In particular, it should become clear that the digital industrial revolution not only creates hybrid communities where we merge with artificial agents, but these artificial agents immigrate in our social institutions with their respective institutional background, rooted in their owner's domains. This means that the hybrid communities of human and artificial agents belong to different institutions, i.e., they obey the judiciary space of their owners and are subject to the legislative space of their programmers, as long as they are not demanded to integrate into and adapt to their host institution. Social Scientists should therefore consider them not primarily as service-providing agents, but as bearers of alien institutions that need to be adapted to their host habitat.

The proposed model allows for the conceptualization of the effects of the digital industrialization in more familiar ways. It further allows Computer Scientists to draw from a wealth of knowledge from Social Sciences and Philosophy, allowing for a targeted discussion of highly topical issues that enables them to add value to the habitat of the customers – and not to harm them.

Jelena Stanulovic (University of Belgrade)

Influence of the Self-Management in the Development of Personal Computers in Socialist Yugoslavia During the ‘80s

Session 6 | Thursday, Oct 28, 11:20 – 11:50

Departing from the Soviet socialist model, Yugoslavia experimented with self-management (self-governance) until 1989 when self-management was abolished due to the economic downfall of the country. The ideology of this economic system in Yugoslavia grounds on the belief ''where there is no self-government, there is no socialism''. The collective work was ideally defined to be autonomous to be able to make all decisions independently, without outside intervention, such as state intervention. However, the state had a monopoly, and numerous government interventions had a strong political orientation (Horvat, 1989).

''Factories to the workers'' was not just a slogan in Yugoslavia, but an economic policy that enabled workers to undergo a decision-making role in the manufacturing and management process in the industry. In Yugoslavia, companies were state-owned where the state assigned their administration to the workers of those companies, and that is how Yugoslav workers were given the central role in the system. The workers in self-management socialism were not considered as hired workers but members of the work collective. The exploitation of the human workforce is seen in systems such as capitalism and state governed economy (such as one under Stalin's Russia), so the companies in Yugoslavia were also named to be collective or social enterprises instead of state ones.

Yugoslav self-management companies competed locally and internationally. On the outside, they were very similar to the companies of the capitalist system: companies did everything possible to increase their income. However, the income was shared among the workers, and the company authorities were the work councils that decided upon the fair share.

In the 80s, while struggling with an economic crisis, the federal government enforced measures on import customs and taxes, which harmed technology deployment. Unlike in Western countries, home-use computers were very rare and were not practically distributed to end-users, due to the authorized import price that was exceedingly lower than the PC market price, and due to the high custom duty costs.

All the ''collective'' companies were more focused on the development of the industry-related computers (called terminals at the time) instead of making home computers for the consumer market (Uzelac, 1981).

In the early 1980s, the goals of the Yugoslav economy were transcribed from the political economy of European countries sharing the idea of the growth of SMEs and innovation culture.

> ''We should not underestimate the fact that a small economy is a real economic and social precondition for the organized return of our workers temporarily employed abroad to rationally invest their savings and thus create conditions for their productive employment and social security for themselves and their families.“ (Rožić, 1983).

The innovation we see in science and technology was brought to the public and lasted for few months thanks to a group of enthusiasts and hobbyists that have cooperated with the most important magazine for the popularization of science and science fiction in Yugoslavia. The magazine is Galaksija, and it was published once a month until the 1990s. The editorial context of the magazine was pro-science and technology. This magazine started sort of a revolution breaking the system without breaking the law. The magazine published how to build the PC from the imported hardware components that were to be found on the market. The home laboratory computer gained enormous popularity through an action led by that magazine in a special edition „Computers in your home“, and reached the homes of people, who dared to make it themselves.

That was in September 1983, when the first article on how to build a homemade computer was published. In the following months, computer hardware and software were being improved, and „Galaksija“ tried to find the most suitable way to organize the purchase of self-construction kits, relying on the „small economy“ that was just emerging in those years. Those were small companies, still state-owned, but more agile. The magazine received more than 10.000 orders for Galaxy computers to be constructed by them and several agreements with the small economy were signed for both hardware and software commercial exploitation.

The free software component was also included since in 1984 the broadcasting of the first programs, almost gaming ones, for computers via radio signal begun. Voja Antonic is the overall creator of the PC Galaksija, and with help of the other software developers and broadcasters, considered to be a part of hacker culture in Yugoslavia (Jakić, 2014). Yugoslav computer hobbyists and enthusiasts culture somewhat resemble a hacker, free-spirited culture based on the shared pursuit of science, especially movement for open software.

This work will tend to analyze the influence of self-management in Yugoslav socialism on hacker culture. Although the self-management companies were profit-oriented, all the profit was shared among the workers to the benefit of all. Being a social enterprise, the company was owned by all the people, like the open software is intended to be owned by its users.

On the other hand, government restrictions and major changes in technology development conveyed the use of science and technology in households. This brings us to an entrepreneurial culture that had to suffer failure until 1989 when self-management was abolished. The research also indicates the appearance of the gray economy that comes up as a category and as a relationship present in the process of appearance of new inventions and discoveries.

Interviews (new and already published ones) with the ''Galaksija'' developers and magazine publishers will be analyzed along with their work before and after ''Galaksija'' using the biographical research method. Socio-cultural paradigm change towards entrepreneurial culture and Western influence on the cultural elite in Yugoslavia played an important role and this will lead the author's future research.

References:

Castells, Manuel: *The Internet Galaxy, Reflections on the Internet, Business and Society*, 2001.

Horvat, Branko: *ABC of Yugoslav socialicm (ABC jugoslavenskog socijalizma)*, Globus, Zagreb, 1989.

Horvat, Branko: Economy and its policy in Yugoslavia (Privredni sistem i ekonomska politika Jugoslavije), Instutut ekonomskih nauka, Beograd, 1970.

Jakić B.: *Galaxy and the New Wave: Yugoslav Computer Culture in the 1980s*. In: Alberts G., Oldenziel R. (eds) Hacking Europe. History of Computing. Springer, London, 2014. https://doi.org/10.1007/978-1-4471-5493-8_5

Pilić Rakić, Vera: *Grey economy (Siva ekonomija)* , Udruženje nauka i društvo Srbije, Beograd, 1997.

Status and the development of the small economy (Položaj i razvoj male privrede), Group of Authors, Beograd: IRO ''Ekonomika'', 1983.

Uzelac, Branislav: *Influence of External Trade on Computers Production and Appliance in Yugoslavia (Spoljnotrgovinske implikacije proizvodnje i primene elektronski računara u Jugoslaviji)*, Ekonomski Fakultet, Beograd,1981.

Woodward, Susan L.: *Orthodoxy and Solidarity: Competing Claims and International Adjustment in Yugoslavia*. International Organization 40, no. 2 (1986): 505–45. http://www.jstor.org/stable/2706845.

Mate Szabo (University of Oxford)

The Early Days of the Hungarian Software Industry

Session 6 | Thursday, Oct 28, 10:50 – 11:20

The aim of this talk is to discuss the beginnings and early developments of the Hungarian software industry from the end of the 1960s into the 1980s and to contribute to the scant literature on the history of the software industry in the Eastern Bloc. The state of Hungarian computing by the late 1960s will be explained within the larger landscape of the Eastern Bloc, while later developments will be linked to the country's interaction with Western European countries as well.

The software industry as we think of it today developed due to the appearance of the software computable computer families in the second half of the 1960s (first and foremost IBM's S/360). Together with the proliferation of computers, this transformed how software was created. Before the mid-1960s, most software was custom-made by the company's own in-house programmers. Soon, independent so-called software houses were established, offering software packages that could be easily tailored to customers' needs. These software houses also operated independently from computer manufacturers and promised computer users a significant decrease in their software costs. This type of software distribution was in full bloom by 1970 (Campbell-Kelly 2003).

It is well known that by the end of the 1960s, the Eastern Bloc was lagging several years behind the West in computer hardware and was severely "undercomputerized." However, due to multiple factors, the situation with regards to software was even worse than the hardware. Indeed, a Soviet national report in 1969 assessed the software industry to be on the American level of 1960 (Przhijalkovskiy 2014). This was due to multiple factors: from the Stalin Era's known hostility towards cybernetics and computerization, to a lack of appreciation for non-scientific computing, and to consequences of the characteristics of the planned economy. In addition, a certain feature of the hardware industry prevented the development of the software industry as well: namely, the (relatively) high number of different, incompatible computer makes, with very few of them reaching considerable numbers in serial production. This not only meant that there was no software portability, but that large software projects, like systems software, could not be "cost-efficient." This kept the software industry in the era of custom-made software, where only the most crucial and indispensable applications were developed.

Many of the same issues hampered the development of the Hungarian software industry even more. The country did not have a domestic computer hardware industry and had less than 100 computers altogether by the end of the 1960s. As the purchasing of these computers was haphazard, it meant that there were so many different brands and makes of computer that no one type had more than ten installations. These circumstances kept Hungary in the era of custom-made software. Besides producing compilers for the most widely used languages, the two software developing “companies” (INFELOR (Havass 2011) and SZKI (Kovács 2011)) made one-of-a-kind programs for the ministries and the largest actors in the country’s industry.

However, somewhat surprisingly, the Hungarian software industry established several cooperations with Western European computer companies through “software export.” This was due to the country’s strategic approach to the purchase of Western European computers and licenses. In order to facilitate real knowledge transfer beyond mere technology transfer from these purchases, it was requested from these companies that a portion of the price be paid through the services of Hungarian computing enterprises, usually by contribution to software development. Since Western European companies were willing to “buy market shares” in Eastern Europe to counteract the global dominance of IBM, this strategy proved to be successful. For example, SZKI was working closely with the German Siemens company from 1970 until the early 1990s. These cooperations proved to be mutually beneficial: while the Western European companies gained access to a relatively cheap labor force, the Hungarian programmers acquired Western “work culture” and earned higher wages.

Another factor impacting the Hungarian software “industry” from the second half of the 1970s was the relatively large amount of hardware built in the Eastern Bloc that was based on Western, mostly IBM and PDP, makes. Since these clones were able to run original Western software (with small modifications), software adaptation became an important service and a significant source of income for the Hungarian software enterprises.

In my talk, I aim to discuss these tendencies that impacted the beginnings and early development of the Hungarian software industry and its main actors in more detail.

References

Campbell-Kelly, Martin. 2003. *From Airline Reservations to Sonic the Hedgehog. A History of the Software Industry*. Cambridge: MIT Press.

Havass, Miklós. (ed) 2011. *A SZÁMALK és elődei.* Budapest.

Kovács, Győző. (ed) 2011. *Volt egyszer egy Szki…* Budapest: Pannónia Print Nyomda

Przhijalkovskiy, Victor. 2014. “Historic Review on the ES Computers Family.” translated by Alexander Nitussov. Online accessed on the 6th of November, 2019 at: http://www.computer-museum.ru/english/es_comp_family.php?sphrase_id=432639

Javier Toscano (APRA Foundation)

Intentionalities of Code: Historical Practices and Devices. A Philosophical Account

Session 1 | Wednesday, Oct 27, 10:00 – 10:30

Computing does not only imply an interaction with machines, but also –maybe more poignantly– a way of thinking. As historians of technology acknowledge, computing meant in the past so much as counting, or even reasoning (e.g. Leibniz 1890). But in this sense, the history of computing has a much earlier beginning than what is popularly thought. The first machines that we can recognize as abstract computers were imagined by Charles Babbage in the 19th Century, but the first codes were assembled centuries before, to be performed by social machineries. Of course, in order for this account to unfold, we need to precise what coding and programming are (or can be), and how they articulate together a social structure to produce cultural meaning, a certain dynamic and a given output. Drawing on early definitions of programming by mathematicians von Neumann and Goldstine (1947), as well as logicians Newell, Simon and Shaw (1958), but especially on a recent human-evolutionary hypothesis by cognitive scientist Michael Tomasello (2014), which maintains that humankind relies on a cooperative mechanism that ultimately resulted in a particular modality of social thinking, this article explores the deep historical foundations of computing and coding from a reinterpretation of specific cultural practices.

Articulating a notion of code, Newell et. al. wrote that “the appropriate way to describe a piece of problem solving behavior is in terms of a program [...]. Computers come into the picture only because they can, by appropriate programming, be induced to execute the same sequences of information processes that humans execute when they are solving problems.” (1958). In this sense, the machine is contingent, and only the processes of defining, framing and solving a problem become relevant. This allows to shift our attention from the object to the interrelated and normative cultural practices that sustain a computing infrastructure. On that track, this research implies a reassessment of historico-cultural materials following Alfred Schutz’s (1979) and Harold Garfunkel’s (2008) empirical phenomenology, but it also follows main leads from Foucault’s critical genealogy (1963). Nevertheless, it does not attempt to structure a full-fledge genealogical account, since the practices and devices that it aims to highlight have responded to distinctive historical moments and values, which might not be simply added up under a consistent category or discursive socio-historical entity. Instead, the aim is to order the underlying social structures through what Tomasello terms *collective intentionalities*, signaling thus different moments of epistemological, political, spiritual or existential intensity. Under this guise, this project inspects ancient coding technologies in religious, magical, legal and proto-scientific domains. The research will work through a variety of examples –obscure but also well-known references (Mumford 1934, Eliade 1958, White 1974, DeVries and Smith 1992, Suchman 2006, Truitt 2016)– realigning them through shifting categories in order to explore

socio-historical functions of command and control, calculation and ordering, knowledge and transfer, protection and safeguarding, contemplation, synthesis and regeneration. The aim of the article is not only to think about these practices as the forerunners of today's programming codes, but also to observe the wide variety of material and symbolic articulations with which humanity has invested cultural devices and artifacts in order to impact its milieu and produce a human world.

References

De Vries, K. and Smith, D. (1992). *Medieval Military Technology*. Toronto: Toronto University Press.

Eliade, M. (1958). *Patterns in Comparative Religion*, tr. Rosemary Sheed. New York: Sheed & Ward.

Foucault, M. (1963). *Naissance de la Clinique*. Paris: PUF.

Garfinkel, H. (2008). *Towards a Sociological Theory of Information*. London and Boulder: Paradigm.

Goldstine H.H. and von Neumann, J. (1947). Planning and Coding of Problems for an Electronic Computing Instrument. *Report on the Mathematical and Logical Aspects of an Electronic Computing Instrument II* (I). Princeton: Institute for Advanced Study.

Leibniz, G. W. (1668/1890). "Scientia Generalis. Characteristica." In *Die Philosophische Schriften, Gerhardt*, K. I. (ed.), Berlín: Weidmansche Buchhandlung.

Mumford, L. (1934). *Technique and Civilization*. New York: Harcourt, Brace and Company.

Newell, A., Shaw, J. C., & Simon, H. A. (1958). Elements of a theory of human problem solving. *Psychological Review* 65(3), 151–166

Schütz, A. and Luckmann, Th. (1979). *Strukturen der Lebenswelt I and II*. Frankfurt a.M.: Suhrkamp.

Suchman, L. (2006). *Human-Machine Reconfigurations*. Cambridge: Cambridge University Press.

Tomasello, M. (2014). *A Natural History of Human Thinking*. Cambridge and London: Harvard University Press.

Truitt, E. R. (2015). *Medieval Robots*. Philadelphia: University of Pennsilvania Press.

White, L. (1974). *Medieval Technology and Social Change*. Londres y Oxford: Oxford University Press.

Stefan Trausan-Matu (University Politehnica of Bucharest)

A Poststructuralist Perspective on Computer-Generated Literature

Session 9 | Friday, Oct 29, 9:30 – 10:00

Natural Language Generation (NLG) is one of the most challenging applications of Natural Language Processing (NLP) in Artificial Intelligence (AI), especially for the case of long sequences of texts, such as stories and novels. However, there already are successful NLG programs that write news, for example, about weather forecasts (Gatt and Krahmer, 2018) or sport reports (Tanaka-Ishii et al., 1998).

There is also a long history of computer programs that generate short stories, starting even from 1973 (Klein's Novel Writer) as Gervás (2009) reported, including Tale-Spin (Meehan, 1977), MINSTREL (Turner, 1993), MEXICA system (Pérez, 1999), BRUTUS (Bringsjord and Ferrucci 1999), Virtual Storyteller (Theune et al., 2003), Fabulist (Riedl and Young, 2006), etc. These story generation systems were classified by several researchers in almost similar ways, as being based on character simulation (for example, in Tale-Spin and Virtual Storyteller), those attempting to model the author's thinking during the writing process (in MEXICA and MINSTREL), story-based systems (Fabulist), and those world-based, in which story is developing as a result of how the characters are trying to achieve their individual goals in a world governed by rules (in Tale-Spin and Story Generator).

The theories used for the development of the above-mentioned systems are structuralist, being based, for example, on grammars of stories, which have as precursor Propp's morphology of folk stories (Propp, 1928, 1968). From another perspective, these NLG systems may be considered as following a rationalistic approach, starting from grammars, rules, and knowledge bases. However, as in other AI domains, an empirical approach gained popularity in recent years: deep artificial neural networks that generate text after they are trained from large corpora (Iqbal and Qureshi, 2020). There are now even whole novels generated by NLG programs, which may be bought from Amazon (Cope, 2020). Deep Learning in NLP is based on language models inspired from the idea that „The meaning of a word is its use in the language" of Wittgenstein (1953). There is a debate whether deep learning is a structuralist discipline (Bruchansky, 2019)

Both rationalistic and empirical NLG approaches mentioned above may be appealing at a first sight for literature, but for generating a novel, which contains imaginary life stories, the results are disappointing, readers immediately may realize their mechanistic character, the fact that the 'author' is not human, it does not have the experience of life, that "there is no one to grow intimate with, no empathetic flow between two living, breathing beings."[1] In fact, the lack of empathy was emphasized as a main problem of AI, a hermeneutic phenomenological perspective being recommended, in the ideas of Heidegger and Gadamer (Winograd, 1987), which may be operationalized by the so-called hermenophore tools (Trausan-Matu, 2017).

An alternative to the previously mentioned approaches (founded on structuralism, on Wittgenstein, Heidegger and Gadamer) is based on the dialogism theory of Mikhail Bakhtin (1981, 1984). He says that the "word is born in a dialogue as a living rejoinder within it; the word is shaped in dialogic interaction with an alien word that is already in the object. A word forms a concept of its own object in a dialogic way" (Bakhtin, 1981). Bakhtin was a critic of the structuralism ideas of de Saussure about the words (Volosinov, 1973) and some considered him a poststructuralist "avant la lettre" (Volosinov's book was first published in Russian in 1929). Nevertheless, Bakhtin is a very particular case, hard to classify. Some included him in the group of Russian formalists, others considered him as a structuralist. Anyway, Julia Kristeva appreciated him and exploited some of his ideas, for example, intertextuality.

Bakhtin considered that the life experience is characterized by a space-time complex, which is represented in novels by chronotopes (Bakhtin, 1981), which are "potentially a unique mix of individual, cultural, and institutional calibrations" (Kent, 2009, p. 78). Moreover, a realistic life experience should have a complex, dialogical, multithreaded, polyphonic weaving (Bakhtin, 1984), like the musical counterpoint: "Everything in life is counterpoint, that is, opposition" (Bakhtin, 1984), as in celebrated Dostoevsky's novels (Bakhtin, 1984).

The polyphonic model of discourse is now used also in developing computer programs for text analysis (Trausan-Matu, 2013) and music generation from conversations (Trausan-Matu and Calinescu, 2015). In the light of Bakhtin's ideas, we consider that the computer generation of longer texts (for example, of novels) should try to aim at inducing chronotopes and constructing a polyphonic weaving, introduced through the co-occurrence of several voices in an extended sense (for example, repetitions of words (Tannen, 2007), threads of discussion or ideas (Trausan-Matu, 2013)). Moreover, voices should enter, similarly to musical counterpoint, in a game of inter-animation that is characterized by pairs of divergences-convergences (Bakhtin, 1981; Trausan-Matu, 2013). A special attention should be paid also to the existence and influence of authoritarian voices (Bakhtin, 1968).

A question raises naturally if we consider the philosophy of computing: Can computer programs generate a novel that can pass a kind of a Turing test, a novel that can be appreciated by many people, not only due to its originality in character by some of them. Moreover, would be possible to be appreciated more than a few such generated novels? One answer could be to refute the structuralist approaches that were used until now for NLG and to take a poststructuralist position, which may be provided by the dialogism of Bakhtin. However, if his ideas were implemented in programs for analyzing discourse, as mentioned herein, it is questionable if a novel generated by a computer program according to his ideas could be a solution.

References

Bakhtin, M.M. (1968) *Rabelais and His World*, MIT Press

Bakhtin, M.M. (1981) *The Dialogic Imagination: Four Essays*, University of Texas Press.

Bakhtin, M.M. (1984) *Problems of Dostoevsky's Poetics*, University of Minnesota Press.

Bringsjord, S. and Ferrucci, D.A. (1999). *Artificial Intelligence and Literary Creativity: Inside the Mind of BRUTUS, a Storytelling Machine*. Routlege.

Bruchansky, C. (2019) Machine learning: A structuralist discipline?, *AI & Society*, Springer, 34, pp. 931–938

Cope, D. (2020) *Back Story: A Computer Generated Novel*, Epoc Books

Gatt, A. and Krahmer, E. J. (2018). "Survey of the State of the Art in Natural language Generation: Core tasks, applications and evaluation." *Journal of Artificial Intelligence Research*, 61(1), pp. 65-170, AI Access Foundation.

Gervás, P. (2009). "Computational approaches to storytelling and creativity", *AI Magazine* 30.3, pp. 49–62.

Iqbal, T., Qureshi, S. (2020) The Survey: Text Generation Models in Deep Learning, *Journal of King Saud University - Computer and Information Sciences*, in press

Kent, S.J. (2009) Novelizing Simultaneous Interpretation, in *Procs of 2nd Int. Conf. Perspectives and Limits of Dialogism in Mikhail Bakhtin*, pp. 74-86, https://www.academia.edu/9650284/Novelizing_Simultaneous_Interpretation, last accessed on 31 July 2019.

Meehan, J.R. (1977a) Tale-Spin, an interactive program that writes stories, *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pp. 91–98, Kaufmann.

Pérez y Pérez, R. (1999). *MEXICA: A Computer Model of Creativity in Writing*. PhD Dissertation, The University of Sussex.

Propp, V. ([1928] 1968). *Morphology of the Folktale*. 2nd ed., University of Texas Press.

Riedl, M.O. and Young, R.M. (2006). "Story Planning as Exploratory Creativity: Techniques for Expanding the Narrative Search Space", *New Generation Computing*, 24 (3), pp. 303–323, Springer.

Tanaka-Ishii, K., Hasida, K. and Noda, I. (1998) "Reactive content selection in the generation of real-time soccer commentary", In *COLING-ACL*, pp. 1282–1288.

Tannen, D. (2007). *Talking voices: Repetition, dialogue, and imagery in conversational discourse* (2nd ed.), Cambridge University Press.

Theune, M., Faas, S., Nijholt, A. and Heylen, D. (2003): "The Virtual Storyteller: Story Creation by Intelligent Agents", In S. Göbel et al. (eds.): Proceedings TIDSE 2003: *Technologies for Interactive Digital Storytelling and Entertainment*, pp.204-215, Fraunhofer IRB Verlag.

Trausan-Matu, S. (2013) "A Polyphonic Model, Analysis Method and Computer Support Tools for the Analysis of Socially-Built Discourse", *Romanian Journal of Information Science and Technology* 16(2-3), pp. 144-154.

Trausan-Matu, S. (2017) Hermenophore tools, a new perspective on text analysis. *Revista Romana de Interactiune Om-Calculator* 10(1), 75-88.

Trausan-Matu, S. (2019) Computer-based Story Generation. An Analysis from a Phenomenological Standpoint. *International Journal of User-System Interaction* 12(1), 39-53.

Trausan-Matu, S., Calinescu, A. (2015) "Compunerea de muzică prin sonificarea conversațiilor chat conform modelului polifonic", *Revista Romana de Interactiune Om-Calculator*, 8(1), pp. 33-44, Ed. MatrixRom.

Turner, S.R. (1993). *Minstrel: a computer model of creativity and storytelling*. PhD Dissertation, University of California at Los Angeles.

Volosinov, V.N. (1973) *Marxism and the Philosophy of Language*. Cambridge, Massachusetts: Harvard University Press.

Winograd, T. (1987) Thinking machines: Can there be? Are we?, *Report No. STAN-CS-87-1161*, Stanford, 1987.

Wittgenstein, L. (1953). *The philosophical investigations*. Oxford: Blackwell.

Notes

[1]https://www.universityofcalifornia.edu/news/will-ai-write-next-great-american-novel, downloaded on 3rd May 2021

Marcelo Vianna (Federal Institute of Education, Sc. and Tech. of Rio Grande do Sul)

"Processing the Development": Technical Groups, Profiles and Decisions on Computer Technologies in Brazil in the Late 1950s

Session 3 | Wednesday, Oct 27, 15:00 – 15:30

The first computers arrived in Brazil in the late 1950s, stimulated by the rise of the national developmentalist government of Juscelino Kubitschek (JK). Through the Plano de Metas (Plan of Goals, 1956-1961), an ambitious project for structural transformation in the Brazilian economy aimed at the import substitution process underway since the 1930s, the government sought the development of a Brazilian consumer goods industry, with diversification of the secondary sector, accompanied by massive investments in strategic sectors considered bottlenecks for development such as Energy, Transport, Steel and Oil. In order to achieve the objectives of the plan, there was a growing incorporation of specialists in technical areas in the state structure, with debates on the use of new technologies to make it feasible. Out of these technocratic spaces, the best known was the Automotive Industry Executive Group (Shapiro, 1994), which defined the foundations of the automobile industry in Brazil.

Among the technologies used, electronic computers aroused the interest of the executors of the *Plano de Metas*, from the first demonstrations carried out by Sperry Rand in the country in 1957. The government conceived two technical groups – GTAC (Working Group on the Application of Computers) in August 1958 and GEACE (Executive Group for the Application of Electronic Computers) in April 1959 – with the task of mediating computational technologies, since their implantation until their operation. While GTAC would be tasked with exploring the applications of "electronic computers" and defining which types of equipment – among those existing on the market – would be more technically and economically suitable for "Brazilian problems", GEACE sought to encourage the installation of Data Centers and approve the acquisition of computers by the State and the private sector. In summary, the members of these groups should apply their expertise to determine the appropriate procedures to install computer technology and to prepare users, through intensive training, to use computers, expanding the precarious technical knowledge base in Brazil.

Our research will discuss the composition and decisions made by these two technical groups. In general, its members must be understood as modernizing agents capable of managing the options and uncertainties that a technopolitical approach imposed on those involved (Hecht, 2009), that is, how much the specialists' decisions, based on their knowledge, could subsidize State policies. The profile of the members of GTAC/GEACE is important to understand the choices made by the State, which encouraged the acquisition of foreign technologies in order to provide rapid industrialization. This perception, typical of the developmental thinking of the period (Evans, 1979), understood that underdeveloped countries could take advantage of the "high stock of scientific and technological resources" available in the United States and Western Europe, without compromising their scarce resources (Erber, 1981).

The presence of the military in the GTAC/GEACE was highlighted (Langer, 1987) and seemed to be justified by the strategy of the JK government in bringing the Armed Forces closer to the *Plano de Metas*, offering strategic positions in the administrative machine and beckoning with technological modernization, in order to control their most radical sectors (Vianna, 2016). Most of the military members of GTAC/GEACE were specialists who had deepened their studies abroad in strategic areas such as Nuclear Energy and Electronics. An example was Corvette Captain Geraldo Nunes Maia, who had contact with computer technologies at MIT while pursuing his doctorate in Electronics in the 1950s, becoming one of the first supporters of computer technologies in Brazil.

In turn, civilian members at GTAC/GEACE, such as engineer César Cantanhede (representative of the National Confederation of Industry), brought modernizing experiences from the fields of Economics and Administration, seeking to establish the use of digital technologies aimed at commercial automation. The presence of Helmut Schreyer, one of the foreign representatives of GTAC/GEACE, should also be highlighted: a pioneer of Computing, his experience in computer projects provided technical support to the work of GTAC/GEACE, having built one of the first prototypes in the country as a professor at the Army Technical School.

Although there was a convergence of the members for the defense of the import of computer technologies, GTAC/GEACE was marked by internal conflicts and controversial decisions that would cost them their existence in the long run. The disputes between members who advocated a scientific approach to Computing and those more focused on commercial applications generated disagreements in their actions, the approval of requests for computer imports and the establishment of training courses for operators. One of the controversies involved the proposal to create a Government DPC, from the import of a UNIVAC 1105 computer to carry out the 1960 Census. The political pressure to install it, without several technical and economic points being duly clarified, resulted in equipment that was inadequate and unable to operate properly, which could not process the census data. The failure of the initiative, coupled with the lack of political support and a network of relations with the technical-scientific community, would contribute to the GTAC/GEACE's inability to offer alternatives to the growing domain of IBM (Vianna, 2016), and the latter eventually assumed the role of agent of technological modernization in Brazil in the early 1960s.

References

Erber, F. (1981). Science and Technology Policy: A Review of the Literature. Latin American Research Review, 16 (1): 3-56.

Evans, P. (1979). Dependent Development: the alliance of multinational, State and Local Capital in Brazil. Princeton: Princeton University Press.

Hecht, G. (2009). The Radiance of France – Nuclear Power and National Identity after World War II. Cambridge: MIT.

Langer, E. (1989). Generations of scientists and engineers – origins of the computer industry in Brazil. Latin American Research Review, 24 (2): 95-111.

Shapiro, H. (1994). Engines of Growth: The State and Transnational Auto Companies in Brazil. New York: Cambridge University Press.

Vianna, M. Between Bureaucrats and Specialists – The Formation and Control of Informatics Field in Brazil (1958-1979). (In Portuguese).

David Waszek (CNRS, Archives Henri-Poincaré)

Informational Equivalence but Computational Differences? Herbert Simon on Representations in Scientific Practice

Session 4 | Wednesday, Oct 27, 16:30 – 17:00

It has become commonplace in contemporary philosophy of science to emphasize that, in scientific practice, "representations matter“ (Humphreys 2004, 98) —drawing a well-chosen diagram or introducing a suitable symbolic notation, for instance, can be of considerable help in problem-solving. To make sense of this, Herbert Simon suggested thinking of representations on the model of data storage in computers (Simon 1978; Larkin and Simon 1987): just as the same data can be encoded in different ways, making certain processing of it more or less efficient, two representations can be "informationally“ equivalent but differ "computationally.“ In our days of pervasive computing, such a picture of representational differences seems quite natural and has proven popular, with or without explicit reference to Simon,[1] but it is rarely spelled out and its difficulties are not usually noticed. By reconstructing and contextualizing Simon‘s research program on representations, this paper aims to clarify what it does and does not achieve.

As we shall see, the original context of Simon‘s work on representations was cognitive psychology, more precisely a debate on whether there were image-like mental representations (Section 1); to this debate, Simon brought a very specific methodology, based on the study of human problem-solving through computer simulations (Section 2). This twofold context sheds light on Simon‘s 1978 attempt to clarify what it means to represent something in a certain form (e.g., "as an image“): he needed an account of representational forms that could be operationalized in psychological experiments, but would also be broadly applicable to the (not necessarily mental) representations used in problem-solving; to get there, he naturally turned to the ways data is stored in computers.

Simon‘s idea, which is introduced informally in Section 3, is roughly that (1) two ways of storing data can be called informationally equivalent if they are algorithmically intertranslatable, and (2) that such informationally equivalent ways to store data can differ by their computational properties, that is, by the kinds of processing they facilitate. Further, he thought that on this basis, one could fully characterize representational forms *abstractly* by their computational properties, and that this would permit a full empirical solution to the imagery debate.

This, I shall argue, is where problems arose. To buttress his computational account of representational differences, Simon relied on concepts that were then being developed in computer science, those of *data type* and of *data structure*, which he defined in idiosyncratic ways. But, as Section 4 argues, it all amounts to nothing more than characterizing representations by the fast operations on them. This, however, cannot do what Simon needs. Instead of fully reducing representational differences to computational differences, the effect of such a computational approach is rather to shift the topic: it only allows comparing representations *relative to a particular way of using them*. This shift becomes clear when examining Larkin and Simon's work on scientific representations in light of the preceding discussion, which I do in Section 5.

In effect, Simon's talk of computational differences between "representations" involves shifting the word to a new meaning, which for clarity I shall call representations-as-computational-roles. This specific conception of representations is different from, and indeed in conflict with, the one usual in the philosophy of science (roughly, representations as meaningful artifacts—like diagrams on paper—or possibly *types* of such): depending on its use, the same representation in the usual sense can correspond to different representations-as-computational-roles; conversely, two very different-looking representations, say a diagram and a table, can be used in computationally equivalent ways. This means that, despite what Simon initially meant to do (and despite what his slogan might still suggest), representational differences cannot be accounted for in a purely computational way. Rather, one should countenance not two, but three dimensions of variation among representations: informational differences, differences in representational form, and finally—depending on how these representational forms are used—computational differences. Furthermore, while the computer model suggests that there is an intrinsic link between the way data is encoded (corresponding to the representational form) and the sorts of processing it allows performing efficiently, the case of human beings manipulating external representations is not so simple, because training plays a crucial role. In the end, only limited aspects of representational differences are helpfully called "computational."

Notes

[1] Larkin and Simon (1987) remains one the most cited papers ever published by the journal Cognitive Science. For examples of references to their analysis in the recent philosophy of science, see Vorms (Vorms 2011, 2012) or Kulvicki (2010, 300). Humphreys (2004, 95--100) does not mention Simon explicitly, but his discussion of representations is very much in Simon's spirit: in fact, although Humphreys seems to have learnt of it indirectly, the most striking example he uses---a game that an appropriate representation shows to be a variant of tic-tac-toe---was initially designed by Simon (1969, 76), who called it 'number scrabble" (see also the latest edition: Simon 1996, 131).

References

Humphreys, Paul. 2004. *Extending Ourselves: Computational Science, Empiricism, and Scientific Method*. Oxford; New York: Oxford University Press.

Kulvicki, John. 2010. „Knowing with Images: Medium and Message.“ *Philosophy of Science* 77 (2): 295--313.

Larkin, Jill H., and Herbert A. Simon. 1987. „Why a Diagram Is (Sometimes) Worth Ten Thousand Words.“ *Cognitive Science* 11 (1): 65--100.

Simon, Herbert A. 1969. *The Sciences of the Artificial*. Cambridge, Mass.; London: MIT Press.

Simon, Herbert A. 1978. „On the Forms of Mental Representation.“ In *Perception and Cognition: Issues in the Foundations of Psychology*, edited by C. Wade Savage, 3--18. Minnesota Studies in the Philosophy of Science, IX. Minneapolis: University of Minnesota Press.

Simon, Herbert A. 1996. *The Sciences of the Artificial*. 3rd ed. Cambridge, Mass.; London: MIT Press.

Vorms, Marion. 2011. „Representing with Imaginary Models: Formats Matter.“ *Studies in History and Philosophy of Science* Part A 42 (2): 287--95.

Simon, Herbert A. 2012. „Formats of Representation in Scientific Theorizing.“ In *Models, Simulations, and Representations*, edited by Paul Humphreys and Cyrille Imbert, 250--69. Routledge Studies in the Philosophy of Science 9. New York; Abingdon: Routledge.

Nick Wiggershaus (Lille University)

An Agential Theory of Implementation for Computer Science

Session 4 | Wednesday, Oct 27, 16:00 – 16:30

Computer programs are multifaceted entities and their metaphysical nature is hard to come by. First characterizations focused on the *dual nature* of programs, according to Moor (1978) for instance, they can be understood on a physical level as well as on a symbolic level; Colburn (1999) conceives them as "concrete abstractions" that face problems similar to the mind-body problem of bridging distinct ontological categories. Yet, both approaches fail to provide detailed answers how the formal/symbolic level is linked to the concrete device. Recent approaches offer a finer grained ontology of the "symbolic levels" instead, characterizing programs as computational artifacts (Turner 2018) or as ontologically stratified (Primiero 2019). Here, the different levels of abstraction are linked by symbolic implementation (SI): algorithms can be "written" in a high-level programming language like C or Python, which in turn can be "translated" into machine code, etc. In this setting, implementation (SI) is a *prescriptive* notion, a "top-down" process.

Nevertheless, ultimately the original question 'How and when is an abstract sequence of computations implemented in a physical substrate?' remains. This is called the problem of physical implementation (PI) and has been studied extensively in the philosophy of mind. Note, that in contrast to (SI), the role of implementation and computation here is rather *descriptive* – the ongoings and properties of natural systems like the brain are described and explained in terms of physical computation. The perspective of engineered artifacts that execute actual programs (developed by humans) play a rather minor role and so the results from the brain-science discourse can't be immediately applied to computer science.

The aim of this talk is to provide a notion of (PI) that takes into account the insights of the previous discourse in philosophy of mind and transpose them to the demands of computer science. For doing so, I present an *agential theory of implementation* – the novelty of which is that human induced implementation/computation is to be explained with recent concepts of (a) the unconventional computing community and (b) the "scientific representation" literature. The latter will be the conceptual tool for bridging the abstract-concrete gap when implementing programs in a physical device. Subsequently, the paper unfolds in three steps:

(1) First, I review existing accounts of physical computation. Virtually all of these accounts emerged from philosophy of mind/cognitive science, trying to solve (PI). Despite their notable differences – causal/dispositional/counterfactual-accounts, semantic accounts, and the functional-mechanistic account (Piccinini 2017) – they largely share the conviction that the

causal topology of a system mirrors or maps onto a model of computation. This mapping obtains independently of humans. Yet, the aforementioned accounts of physical computation/ implementation do not spell out the details when it comes to the implementation of actual human designed programs on the physical level. From the computer science perspective, important concepts, where chief among these is the notion of correctness, are neglected.

(2) Secondly, I show that these problems can be overcome by an array of hitherto underappreciated accounts of physical computation. Most notably among them is the so-called Abstraction/Representation Theory (ART) (Horseman et al. 2018), which emerged in the unconventional computing community. In a nutshell, the idea is that all physical computers require (scientific) representation in order to act as predicting devices for the results of abstract computations. However, since most serious notions of scientific representation (at least partly) depend on human beliefs, desires and intentions, (Fletcher2018) suggests to spell out ART in agential terms. I argue that this modified agential notion of physical implementation is a promising candidate to meet the implementation-desiderata of computing practice. The account is reciprocal and can thus account for the best of both worlds ((SI) and (PI)) requiring (i) a suitable underlying dynamic X of the computational system *plus* (ii) an agent or epistemic community that use(s) the computational system, under interpretation I, for their purposes.

(3) However, so far (agential) ART is formulated in a rather theoretical manner only. For further clarifying the role of scientific representation and its use with respect to actual devices, a link to the recently developed DEKI account in the literature of scientific representation is established (Frigg&Nguyen 2018). Subsequently, I demonstrate that DEKI is promising in connection with an agential theory of implementation for two reasons: First, because Frigg and Nguyen conveniently make use of an actual computational system as their prime example for material models – the MONIAC. Secondly, and more important, the reciprocal features (i) and (ii) (see previous paragraph) are met by DEKI, which shows striking parallels. Accordingly, a concrete object X is turned into a model, by an interpretation I. But only if X exhibits interesting behavior, e.g., from a selected internal structure and its dynamics, scientists may learn something from this structure by appropriate interpretation I. By thus combining the agential version of ART with the DEKI account, a robust notion of an agential theory of implementation for computer science emerges.

References

(Colburn, 1999) Colburn, T. R. (1999). "Software, abstraction, and ontology". *The Monist*, 82(1):3–19.

(Fletcher, 2018) Fletcher, S. C. (2018). "Computers in abstraction/representation theory". *Minds and Machines*, 28(3):445–463.

(Frigg and Nguyen, 2018) Frigg, R. and Nguyen, J. (2018). "The turn of the valve: representing with material models". *European Journal for Philosophy of Science*, 8(2):205–224.

(Horsman et al., 2018) Horsman, D., Kendon, V., and Stepney, S. (2018). "Abstraction/ Representation Theory and the Natural Science of Computation". In Cuffaro, M. E. and Fletcher, S. C. (eds.), *Physical Perspectives on Computation, Computational Perspectives on Physics*, 127–150. Cambridge University Press.

(Moor, 1978) Moor, J. H. (1978). "Three myths of computer science". *The British Journal for the Philosophy of Science*, 29(3):213–222.

(Piccinini, 2017) Piccinini, G. (2017). "Computation in physical systems". In Zalta, E. N. (ed.), *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2017 edition.

(Primiero, 2019) Primiero, G. (2019). *On the Foundations of Computing*. Oxford University Press.

(Turner, 2018) Turner, R. (2018). *Computational Artifacts*. Springer.

Robin Zebrowski, John Sullins, Eric Dietrich, Bram Van Heuveln and Chris Fields (Beloit College, Sonoma State University, Rensselaer Polytechnic Institute)

The History and Legacy of the AI Wars

Session 7 | Thursday, Oct 28, 15:30 – 16:00

In the late 1970s, Aaron Sloman confidently predicted that “within a few years, if there remain any philosophers who are not familiar with some of the main developments in artificial intelligence, it will be fair to accuse them of professional incompetence” (1978). And while it may be true that many, although surely not most, philosophers are familiar with some shallow aspects of the philosophical debates about AI (such as the Chinese Room thought experiment), it definitely did not come to pass that lacking knowledge about AI has been understood as a failure of philosophers generally. We hope to provide a solid primer for anyone wanting to familiarize themselves with where we in AI are now, and how we got here. In this panel, we will trace four distinct arguments that dominated the philosophical interactions with AI for its first 50 years, four separate AI Wars. Then, we discuss the ongoing debates and the future of AI, focused on consciousness, ethics, and embodiment. In this presentation, we proceed from arguments in our recently-published book, *Great Philosophical Objections to Artificial Intelligence: The History and Legacy of the AI Wars* (Bloomsbury, 2021) and engage with the ways philosophy as a field moved on from many of those debates without ever resolving them, setting up the foundations for the contemporary debates and sometimes even ignoring those earlier lessons.

What we’re calling The First War focused on the logical cogency of AI as a project in general. This will be familiar as the argument around Gödel’s Incompleteness theorem, especially as it’s wrapped up with the halting problem. We trace the way both of these claims framed an argument that emerged in multiple places, showing that the first big AI War concerned itself with whether humans and machines were even possibly the same sorts of systems. Humans could, apparently, solve the Halting Problem, stepping outside of our first-order language into a meta level, and we had several mathematical formulations that appeared to prove our machines could never do the same. We’ll trace this development and show how theorists seemed to be arguing from within different levels of analysis, ultimately reducing the problem down to one of alleged human cognitive specialness.

Although these wars are not strictly chronological and overlap one another historically, The Second War can be seen as growing from the debates about logical cogency into questions concerning what kinds of computational architectures might possibly enable the kinds of cognition humans seem to have (without necessarily examining whether human sorts of intelligence are the goal we should be aiming for). This war saw GOFAI pitted against connectionist architectures, in a battle that played out in classrooms, books, and funding schemes. This was a battle of symbolic versus subsymbolic representations, and it also spilled over into early robotics, where the hope was to cast off as many representations as possible and let the world be its own model.

Importantly, as is often the case in philosophy, the primary debates shifted over time, despite there never being broad consensus congealing around one side or another. This is particularly clear in light of the Third War, which focused on meaning and semantics, and how language is understood to relate to both mind and world. The AI-specific version of this is most famous via Searle's (1980) Chinese Room, although Dreyfus's phenomenological critiques (1972) certainly preceded it. Here we find the Symbol Grounding Problem, as the details of language and brains complicated our attempts to build systems with actual minds, capable of thought. This debate is as old as philosophy, but it's history with AI offers new insights.

The Fourth War, while appearing to be a set of debates around the nature of rationality, the possibility of creativity, and the limits of human cognition, can more appropriately be understood as the setting for The Frame Problem, and all of its ancillary questions. How is it that when I want to make a peanut butter and jelly sandwich, I don't have to evaluate whether or not the paint on the walls is going to spontaneously change colors, or that the knives won't explode when I open the drawer? The Frame Problem might be one of the greatest problems that the AI Wars have given us, and it shows itself in myriad surprising places throughout philosophy and the cognate disciplines dealing with debates around AI.

Taken together, we show how these four overarching debates structured philosophy's interactions with AI, in part leading to the AI winter in the 1990s. The current apparent-successes of algorithms trained on big data sets have revived some of these wars in new guises, but the current and future AI wars seem to be focused on a few threads that have been present, though slightly hidden, throughout the history of AI, namely ethics, consciousness, and embodiment. Here, we'll lay out the history of these philosophical movements, and offer predictions based on this history about where the current debates will lead us.

Works Cited

Dietrich, E., Fields, C., Sullins, J., Van Heuveln, B., and Zebrowski, R. (2021) *Great Philosophical Objections to Artificial Intelligence: The History and Legacy of the AI Wars*. London, Bloomsbury.

Dreyfus, H. (1972) *What Computers Can't Do: The Limits of Artificial Intelligence*. New York, Harper and Row.

Searle, J. (1980) "Minds, Brains, and Programs." *Behavioral and Brain Sciences 3*: 417-457

Sloman, A. (1978) *The Computer Revolution in Philosophy: Philosophy, Science, and Models of the Mind*. Harvester.

Program Committee

Janet Abbate (Virginia Tech)

Nicola Angius (University of Sassari)

Maarten Bullynck (Université Paris 8)

Felice Cardone (Università di Torino)

Leo Corry (Tel Aviv University)

Liesbeth De Mol (CNRS - Université de Lille 3)

Walter Dean (The University of Warwick)

Gilles Dowek (INRIA and ENS Paris-Saclay)

Helena Durnova (Masaryk University)

Juan Manuel Durán (Delft University of Technology)

Juan Luis Gastaldi (ETH Zurich)

Jean-Baptiste Joinet (Université Jean Moulin Lyon 3)

Benedikt Loewe (University of Amsterdam)

Andrea Magnorsky

Simone Martini (Università di Bologna)

Pierre-Eric Mounier-Kuhn (CNRS)

Baptiste Mélès (Université de Lorraine)

Maël Pégny (Université de Paris-1)

Luc Pellissier (Université Paris-Est Créteil)

Warren Sack (UC Santa Cruz)

Viola Schiaffonati (Politecnico di Milano)

Gisele Secco (Universidade Federal de Santa Maria)

Sonja Smets (University of Amsterdam)

Franck Varenne (University of Rouen - ERIAC & IHPST Labs)

Mario Verdicchio (Università degli Studi di Bergamo)

Organization

HaPoC-6 is organized by:

Turing Centre Zurich (ETH Zurich)

Chairs:

Juan Luis Gastaldi (ETH Zurich, Turing Centre Zurich)
Luc Pellissier (Université de Paris-Est Créteil)

Organizing Team:

Juan Luis Gastaldi (ETH Zurich, Turing Centre Zurich)
Luc Pellissier (Université de Paris-Est Créteil)
Victoria Laszlo (ETH Zurich)
Sara Booz (ETH Zurich)
Niklaus Müller (ETH Zurich)

With the support of:

Swiss National Foundation (Scientific Exchanges Grant)
DHST/DLMPST Commission for the History and Philosophy of Computing (HaPoC)

In collaboration with:

Chair of History and Philosophy of Mathematical Sciences (D-GESS, ETH Zurich)
Chair of History of Technology (D-GESS, ETH Zurich)
Chair for Philosophy (D-GESS, ETH Zurich)
Collegium Helveticum (ETH-UZH-ZHdK)

Under the auspices of:

DHST/DLMPST Commission for the History and Philosophy of Computing (HaPoC)

Contact:

Turing Centre Zurich, ETH Zurich
Clausiusstrasse 49
8092 Zurich
Tel. +41 44 632 31 32
turingcenter@ethz.ch

Notes

ETH Zürich
Turing Centre
Clausiusstrasse 49
8092 Zürich

www.turing.ethz.ch

Herausgeber: Turing Centre Zurich, ETH Zurich
Redaktion: Victoria Laszlo & Juan Luis Gastaldi
Gestaltung: Victoria Laszlo & Juan Luis Gastaldi