Cognitive Science as String Manipulation

Cognitive Science as String Manipulation

An unpologetically computer science book about mind and machine.

Sathaporn Hu

แล้วสอนว่าอย่าไว้ใจมนุษย์ มันแสนสุดลึกล้ำเหนือกำหนด
ถึงเถาวัลย์พันเกี่ยวที่เลี้ยวลด ก็ไม่คดเหมือนหนึ่งในน้ำใจคน
จาก พระอภัยมณี โดย สุนทร ภู่

And I will teach you: trust no human beings, for they are infinitely mysterious. Even for binding vines that twist, they do not twist as much as human souls.

From Phra Abhai Mani by Sunthorn Phu

Contents

Preface

An unpologetically computer science book about mind and machine.

I began to write this book during the Fall semester of 2024. During that time, I was given a list of courses that I could teach in the winter at Algoma University. One of them was a special topic course where I essentially got to design the course content itself. I had multiple topics in mind. However, I settled down on cognitive science and computational linguistics. Although I am a professor in Extended Reality (XR), I also have passion in cognitive science and computational linguistics. After all, these used to be my areas while I was a Bachelor's degree student.

So why does a professor in Extended Reality still want to write an AI book? Well, I could go on a diatribe about how I almost became a Master's degree student at the University of Montreal and being trained in computational linguistics with Prof. Guy Lapalme, and Prof. Bengio Yoshua myself. But I digress.

Although that was a long time ago, I still have passion about AI and machine learning. These passions eventually led me to the creation of Gander, an immersive analytics system for people to make better decisions with a simple AI model (namely, multiple linear regression). As it turns out, the work in my thesis (Hu, 2024) is still, rather, incomplete. As it turns out, there is still much to understand. For instance, it was difficult for me to understand what the users gleaned from the interface. The data collection methods for augmented reality (AR) turned out to be rather primitive. We need adoption of better AI to supercharge it.

As much as AI can help with data collection, XR can also help us to better understand AI. After all, AI models—especially artificial neural networks can be quite opaque. To fight against the opacity, we need new and novel methods of seeing and interacting with the model. This was the purpose of my system (Hu, 2024) to begin with.

Therefore, I believe, to be better at XR, I must also become better at AI. And for anyone to become better at AI, they must also need XR.

Since I was a part-time faculty member at Algoma University from January 2024 until December 2024, I had more spare time than I did now. During that time, I also dabbled in more abstract thinking. Instead of building an AI or XR system, I started to ponder … What is the most basic, and the most fundamental action possible within XR? Certainly, AI can help us to calculate data collected from AR sensors. However, these data are of limited use if we cannot properly quantize user actions.

This brought me back to my days as a Bachelor's degree student. Cognitive science, after all, was about trying to understand human psychology and actions in the most minute details possible and formalize them. Then, I realize that I was not the first person who pondered this question. Alan Turing also had a similar question: what was the most fundamental act of computation. He then devised Turing machines—imaginary machines serving as tools for proofs and for thought experiments. Turing machines, as they turned out, are string manipulators. Strings here do not mean a yarn or twine. Rather, each string is a series of characters strung together into a sequence. A Turing machine accepts a string and transforms it into another using predetermined steps.

My acts of meditation have brought me back to a full circle; to become a better XR researcher, I must consider AI. And to truly grasp the true nature of AI, I must reach back into my root of computational linguistics and formal languages. This book is the result that meditation. (Also, a Department Chair told me that a tenure review would be coming, so I best have something.)

The meditation is still ongoing. It is not over, and I do not think that it will be over. Once I am gone, some other researchers will continue to ponder how to connect AI and XR together.

Some people believe that string manipulations can represent human thoughts, human minds, and human souls. While some researchers do not believe these to be the case, as a former aspiring computational linguistics, I believe that this line of thinking can still be useful. Turing machines, string manipulators, are not meant to be realistic nor practical. However, they allow us to better reason with computation. They allow us to be as minute as possible. By meditating on the minuteness of everything, I hope to one day better apply AI to XR, and finally unlock certain types of analyses that are currently not possible.

This book does not present a complete image of cognitive science. Rather, the goal of this book is to be unapologetically computer science-oriented. I designed this book for computer science students who may not have familiarity with AI, but are still keen to experience the more philosophical side of AI. To paraphrase Wilfred Laurier, the second Canadian prime minister, the mantra of this book is: "Computer Science First, Computer Science Last, Computer Science Always."

The book is partially inspired by "Mind, Body, World: Foundations of Cognitive Science," which was written by Dawson (2013). Originally, I did not plan to write this book at all. I believed that I could simply use someone else's textbook to convey my brand of cognitive science. However, the book was heavily focused on psychology students. Computer science students, who may not have much background in psychology, ended up struggling. In addition to Dawson (2013), I also adopted "Speech and Language Processing" by Jurafsky (2025). The textbook prioritizes natural language processing, so it does not deeply discuss computational linguistics and cognitive science concepts. For instance, the book describes large language models, but does not connect them to other cognitive science concepts.

While I do not know if I will ever get to re-teach this course, I know that I must be better prepared. And there is no better way to be prepared than to write this textbook.

References

Dawson, M. R. W. (2013). *Mind, Body, World: Foundations of Cognitive Science.* Athabasca University Press. https://www.aupress.ca/books/120227-mind-body-world/

Hu, S. (2024). *A Tablet + Augmented Reality Interface for Interactive Multiple Linear Regression with Geospatial Data*. [Doctoral dissertation, Dalhousie University]. DalSpace. http://hdl.handle.net/10222/83437

Jurafsky, D., Martin J. H. (2025). *Speech and Language Processing (3rd ed. draft)*. https://web.stanford.edu/~jurafsky/slp3/

Introduction

I know I'm just a silly AI…
Neurosama, from a clip of a stream (Neuro-sama – Official Clips, 2025)

This textbook is written for computer science students who wish to gain more knowledge in artificial intelligence (AI) and cognitive science. It seeks to highlight the contribution made by computer science towards the field of cognitive science itself. The textbook is separated into four parts:

1. **Cognitive Science:** This part gives a quick rundown on cognitive science, the history of AI, and computational linguistics. We will also discuss chatbots, and Turing tests since we use chatbots as the main model of AI in this book.
2. **Theory of Computing (and Language):** This part presents the Turing Machine (TM) as a model of computation and contrasts it with different models that more computer science students are familiar with. Chomsky's Hierarchy and less powerful automata are introduced. While a TM is capable of simulating all less powerful automata, these automata are still useful. For instance, finite state automata can be constructed using regular expressions which are very useful for preliminary text-cleaning tasks. Meanwhile, context-free grammar can be used to represent syntaxes, albeit imperfectly.
3. **Representation:** While the previous part deals with the syntactic aspect of language, this section discusses the meaning. Two ways of representation are presented: the more logical way with logic, or the more statistical way with deep learning. Language models are also briefly introduced. Lastly, the textbook discusses semiotics, which involves discussions around syntax and semantics.
4. **Computer vs. the World:** So far, the book only discusses computational intelligence as something internal to itself. However, any intelligent being must interact with the physical world. This part highlights the challenges of making a chatbot deal with the real world. First, we discuss the concept of non-determinism, a somewhat difficult concept to grasp that has a grand implication for AI. Then, the textbook describes the difficult and undecidable problems that computers face and the concept of heuristics. The link between formal languages and problem solving is also established. Finally, the book caps off with a discussion of interaction, the importance of

senses, and hypercomputation.

The Missing Pillars: Big Holes in the Heart

Cognitive science is an interdisciplinary field. It incorporates elements from many existing fields to study the nature of the mind itself. While many fields have their place within cognitive science, the major ones are: (1) computer science, (2) psychology, (3) neuroscience, (4) linguistics, and (5) philosophy. Computer science plays an important role, because it allows us to treat the mind as a formal system that we can independently study and develop from the hard- or the wetware. Computer science also serves as a tool for abstracting concepts and makes them easier to deal with using formal methods. For instance, actual biological neurons are small, wet, messy, and sensitive to manipulations. On the other hand, computational representations of the human neurons are abstract and indestructible objects that we can more easily deal with. In addition to the five fields, other fields can also play an important role. For instance, culture does play an important role in how we perceive the world and perform cognition. To understand culture, we must engage with anthropology. History can also change how we perceive the world.

While I wrote this textbook to be unapologetically computer science, I am not encouraging anyone to eschew other pillars of cognitive science. Computer science alone is often insufficient when investigating cognitive science. For instance, imagine that we are trying to implement part of an artificial mind using artificial neural networks. Although the networks themselves are implemented on a computer and we use *graphs*, a computer science concept to build them, we still need knowledge of statistics, mathematics, and neuroscience to realize the networks. By de-emphasizing other fields, I actually hope to highlight the inadequacies of computer science. Throughout the textbooks, I will be highlighting the holes that computer science alone cannot address.

It is also important to note that even within computer science, this textbook misses many details. For instance, while it briefly discusses natural language processing and computer vision, it does not provide a deep discussion of those topics. As it turns out, cognitive science is a highly interdisciplinary enterprise. I only know what I know. And even with research completed here, we are just barely scratching the surface of cognitive science itself.

The Holes

Psychology

While the development of psychology is intertwined with that of cognitive science, it takes a markedly different approach. Dawson (2013, Chapter 2) states that, unlike cognitive science, which takes a constructive approach, psychology is more about observing behaviours and developing theories around the observation. A constructive approach here means that when we want an explanation for a phenomenon, we build for it. For instance, if we want to understand how an earthquake damages a building, we create an earthquake simulator machine and test it on multiple buildings.

However, this does mean that cognitive science eschews statistics. As a multidisciplinary field, most cognitive scientists are also moonlighting as experts in other fields. Cognitive scientists who are also psychologists will use tools in psychology to study the mind, including empirical studies. Some computer scientists are also experts in machine learning, which itself is a field in statistics. Human-computer interaction (HCI) practitioners are also at ease with incorporating tools of psychology. In fact, they may find themselves less at ease with the formal method of computer science.

Neuroscience

Although human cognition does have some computer-like aspects (Scott, 2024, Chapter 2), our physical brains, which host our minds, are also organized extremely differently. For instance, our brains are plastic, which means they are more adaptable to changes. As a child develops into an adult, their brain also grows with them. While computer science attempts to mimic the human neurons, many details are omitted and simplified. For instance, a biological neuron cell has many organelles to support its operation, including mitochondria, the powerhouse of the cell. Each part of our brain is also specialized. For instance, most of us should know that the prefrontal cortex is specialized towards reasoning.

Linguistics

In this book, we do discuss linguistics quite extensively. However, the branch that we focus on is quite Chomskyan in nature. This means the textbook de-emphasizes the other branches of linguistics, such as phonology, sociolinguistics, and language acquisition. I argue that these branches are also very important. For instance, if we are developing multimodal AI that interacts with users using voice, the AI must be able to recognize phonemes (a unit of a speech sound) so it can convert user speech into string data. An AI deployed in multiple countries must also possess the ability to recognize registers, a concept in sociolinguistics, lest it accidentally offend the users. For instance, if users are interacting with the AI in a business setting, the AI should use business-appropriate language. It should avoid using slang and casual words. Another thing that this book will not discuss in detail is language acquisition. For a human being, language acquisition is a challenging process. Acquiring additional languages is even more daunting as adults and children acquire languages very differently.

Philosophy

Logic, a branch of philosophy, is extensively used in computer science. The very first computer science proof, developed by Turing (1936), uses modus ponens and proof of contradiction to argue that computers cannot predict if an arbitrary program will ever output a specific symbol. AI-oriented programming languages such as Prolog also rely on concepts in logic and automated reasoning. However, there is more to philosophy than logic. When we create “conscious” AI, we must evaluate it. To do so, we must have clear criteria for consciousness. But what is consciousness? This is a difficult question to tackle. McCarthy and Hayes (1981) argue that we need philosophy to establish the criteria. Additionally, today’s AI has some ethical questions that we must answer. One example is: can we create an AI that will always behave ethically? Another one is: can we source the training data for AI ethically while minimizing the impact on the original authors of the sources (e.g., artists, writers)? In all and all, philosophy does play a significant role in cognitive science.

Duality of Computer Science

Dawson (2013, Chapter 2) argues that psychology, unlike cognitive science, is quite heavily fragmented because cognitive science as a whole is united in its constructive approach. However, I would argue that computer science also experiences fragmentation. While the constructive approach is accepted by all cognitive scientists, the fields themselves can be heavily fragmented. Thus, while there is no giant central fissure, there are various cracks.

One challenge that computer science students must accept is that computer science programs tend to serve multiple interests. For instance, many students in the program are more interested in real-world applications of computer science. Those in software engineering probably care more about how SQL code will accidentally delete an entire database than whether a Turing machine will accept a string or not.

At first glance, it seems that theoretical computer science has no real-world application. But then, some elements of them can appear in the real world. During job interviews for a tech position, the interviewers may ask the applicants to create code and evaluate its performance solely on paper (or a whiteboard). In this case, the applicants must have some theoretical knowledge. Furthermore, technologies can come and go. For instance, does anyone here know or remember Visual FoxPro, a database programming software package? Many people do not. However, just because Visual FoxPro disappears, it does not mean that Visual FoxPro experts can no longer deal with databases. Instead, if they have some theoretical database skills, they can still apply their knowledge elsewhere.

As such, computer science is engaged in a battle of duality: should it be more practical or should it be more theoretical? Finding the right balance is difficult as computer science serves two masters who often do not agree with each other. I believe that the duality is a feature and not a bug. The field of computer science is always about contradiction. It is about making imagination real and extending what is real with even more imagination. Thus, I argue that: without theoretical computer science, computing devices are impossible. However, without practical computing devices, theoretical computer science has no worth. It has no use except to entertain the brains of the finest philosophers, mathematicians, and logicians.

Book as an Enhancement

It is important to highlight that this book does not replace any AI or cognitive science source. Instead, it serves to enhance them. For instance, while this book discusses concepts in computational linguistics quite extensively, it will not teach how to actually develop computational linguistics software (e.g., chatbots, scrapers). The reader is also encouraged to seek other sources in different fields. Knowing how to reconcile and finding the shared common meanings between a diverse set of literature is a hallmark of a good cognitive scientist.

References

McCarthy, J. & Hayes, P. J. (1981). Some Philosophical Problems from the Standpoint of Artificial Intelligence. *Readings in Artificial Intelligence*, 431-450. https://doi.org/10.1016/B978-0-934613-03-3.50033-7

Neuro-sama Official Clips. (2025, December 22). *Do I matter to you?* [Video]. https://www.youtube.com/watch?v=a8m4scxQdL0

Scott, I. S. (2024). *Human-Computer Interaction: An Empirical Perspective (2nd Edition).* Morgan Kaufman.

Simone, P. & Giunti, M. (2023). Alan Turing and the Cognitive Foundation of the Concept of Algorithm. *Algorithmo 34*(April), 13-29. https://doi.org/10.5281/zenodo.7768560

Turing, A. M. (1936). On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society* s2-42(1), 230-265. https://doi.org/10.1112/plms/s2-42.1.230

I

Cognitive Science

1.

History of Cognitive Science (and Artificial Intelligence)

> The question *What science or sciences are likely to be enriched by artificial intelligence studies?* can now receive a provisional answer, namely *All those sciences which are directly relevant to human thought and perception*. These *cognitive sciences …*
>
> **Longuet-Higgins (1973)**

Cognitive science is an interdisciplinary field of study where multiple existing fields serve as pillars. In this book, the main pillars will be computer science and a sub-branch of artificial intelligence (AI) named computational linguistics. However, before describing cognitive science and its history, we should first know what AI is. This is important because AI serves as a main gateway to studying intelligence. Certain fields, such as psychology, study intelligence, but they take a less constructive approach. Meanwhile, cognitive science involves attempting to construct intelligence itself, or some aspect of it. And constructed intelligence is essentially artificial intelligence.

What Is Artificial Intelligence?

Artificial Intelligence (AI) is a fascinating field of study. However, today's discussion on AI has been taken over by hypes and has become almost synonymous with language models. However, language models and computational linguistics are just branches of AI. AI is quite diverse. One AI expert may not understand what the other experts are doing. For instance, a classical AI expert working in knowledge representation may not understand what a machine learning expert is doing. A computer vision expert who is working with image classifiers may not understand how a natural language processing expert removes background noise from a video recording and transcribes things into text.

Many things can be considered AI. Today, we may consider the following software as AI:

- Chatbot using large language models (LLM) such as ChatGPT

- Transcription software
- Autonomous vehicle
- Statistical analysis software
- Video game enemies
- Robots

There is some software that people unfamiliar with AI may not consider to be AI. One example is software that uses search algorithms. For instance, the breadth-first search and the depth-first search algorithms that students learn in the second-year computer classes were once considered to be AI software. However, using search algorithms to represent intelligence was once an active area of research.

Two (or Three) Types of Artificial Intelligence

Strong Artificial Intelligence

Strong AI has a noble and glorious purpose: to create a machine or software that is as intelligent and capable as a human being. To understand how human thinks, we implement computers that can think like humans. For instance, if we want to know how humans process visual information (e.g., handwriting), we create a machine or software that can perform such a task. There is software capable of performing human tasks very well. For instance, LeNet by LeCun et al. (2002) and its descendants are software capable of recognizing handwriting. However, while the software does take some inspiration from the brain, it also eschews many aspects for the sake of convenience. Therefore, the software is not considered to be strong AI.

To this day, a realization of Strong AI remains elusive. Although AI may convince someone that it is human-like, it still has many gaps. For further discussion, please refer to Chapter 3, where we discuss the Turing Test and whether chatbots can be considered Strong AI despite their ability to fool humans.

Weak Artificial Intelligence

The noble and glorious purpose of Strong AI can be difficult to fulfill. Fortunately, while Strong AI is seemingly out of reach, elements of it can still be monetized and applied to solve real-world problems. Weak AI, unlike Strong AI, does not concern itself with answering the Big Questions. Instead, it is about applying AI techniques to solve real-world problems (and make some money).

As it turns out, while some Strong AI projects reach dead ends, they do give us very good techniques that we can use. For instance, classical AI ponders how searching algorithms can represent intelligence. Classical AI researchers like Newell found that many real-world problems can be reformulated into search problems. Then, we can apply search algorithms such as the breadth-first search or the depth-first search to get to the solution. This led Simon Newell to proclaim that: “There are now machines that think” (Norvig 1992, Chapter 4).

However, the search-based AI could not develop further when it turned out that there were a myriad of problems. For instance, some real-world problems are vague and resist formalization. Norvig (1992, Chapter 4) also argues that some problems turned out to be very impractical for computers to solve. If humans were to solve these problems fully algorithmically, they might spend too much time thinking and not survive in the real world. Just because searching algorithms fail to explain intelligence, it does not mean that they are useless. Search techniques are later adopted for video game AIs. For instance, the video game series called *The Sims* used classical AI techniques such as A* to set the actions for its characters (Charity et al., 2020). The topic of intractability will be discussed in Part 3: Computers vs. the World.

Another example is the artificial neural network (ANN). Although an ANN mimics aspects of neurons, it is too simplistic to substitute a real human brain. However, its simplicity allows it to be applied to various real-world problems. For instance, LeNet developed by LeCun et al. (2002) is an ANN designed to detect and recognize handwritten letters. It performs very well. And while an ANN is inspired by the human brain, LeNet is not a type of Strong AI because it does not try to represent how real humans would perform handwriting recognition. In fact, trying to make an ANN realistic can be counterproductive. Seeing and recognizing things in the environment actually involve multiple brain lobes and multiple organs. If LeNet is designed to be as realistic as possible, it would not have happened. Lecun et al. would need to emulate various organs – some of which perform more than just handwriting.

We also note that Weak AI can influence the development of Strong AI. Some researchers tried to push certain algorithms and turn them into an explanation of how the mind works. An example of this is the Wake-Sleep algorithm by Hinton et al. (1995). The algorithm trains an ANN to compress and reconstruct data that it remembers. The use of the name “Wake-Sleep”, according to John A. Vervaeke, one of my professors at the University of Toronto, is strategic. Hinton, one of the authors, used “Wake” and “Sleep” to make it sound like the algorithm is describing how people dream. Although researchers later found the algorithm to be biologically unrealistic, the work with ANNs helps to guide and inspire neuroscience (Hoel, 2021).

Bonus: Artificial General Intelligence

Artificial General Intelligence (AGI) is a relatively recent term. At first glance, the goal of AGI is relatively similar to the one for Strong AI – i.e., making AI that can represent general intelligence. However, Ragnar (2020) argues that AGI is actually still a type of Weak AI. To understand AGI, we must first understand the initial scope of Weak AI. Traditionally, Weak AI has a limited scope. For example, a chatbot solely relies on algorithms and techniques developed in computational linguistics and natural language processing (NLP). Meanwhile, an autonomous vehicle would solely rely on computer vision techniques and machine learning. To make AGI, we expand the functionalities of various Weak AI implementations so that they overlap. For instance, if we can combine the functionalities of a chatbot and an autonomous vehicle, we can create an even more intelligent vehicle. Not only can the autonomous vehicle drive itself, but it can also modify its behaviour based on the conditions of the passengers. For example, if the passengers are talking about how bumpy the road is, the autonomous vehicle can slow itself down to increase passenger comfort. The augmented vehicle is not Strong AI because, simply put, it is not a human. For an AI implementation to be Strong AI, it must aim to mimic humans.

AGI can overlap with Strong AI in certain areas. For instance, if we create a robot that can communicate with humans, much of its software could possibly be easily translated into cognitive science theories.

History of Cognitive Science and Its Branches

As McCarthy and Hayes (1981) state, while the term artificial intelligence is new, the concept of creating an intelligent machine itself is not. People in the past have been discussing creating artificial humans. However, Haugeland (1985, Chapter 1), argues that Hobbes is the “Grandfather of AI” because he made a philosophical argument for it. Hobbes (1588 – 1679) was a philosopher who argued that the mind was purely computational (Stewart, 2024). This was in contrast to his contemporary, René Descartes, who believed that the human mind must possess some sort of spirit. While we will closely examine Hobbes and Descartes’s arguments later, we must first note that Descartes did not believe that the mind was purely a spirit. Rather, the mind requires some sort of machinery (i.e., the body) to function, but it also requires spirit to become fully sentient. This means that Hobbes and Descartes were mostly in agreement. However, by claiming that the mind is pure computation, Hobbes essentially argues that we can create artificial intelligence and that no higher beings or mysteries are necessary. Thus, he argues that Strong AI is possible.

While philosophers debate the possibility of artificial intelligence, we must also briefly discuss the history of computers. After all, cognitive science’s premise is about studying intelligence by implementing it and analyzing it using computing devices.

Charles Babbage, who lived between 1791 and 1871, according to the Editors of Encyclopaedia Britannica (2025), designed the Analytical Engine. This machine, if fully implemented, could be considered the first computer ever. The machine was fully programmable, and it contained conditional branching –

i.e. executing code differently based on the machine's current state (Hagueland, 1985, Chapter 4). According to Haugeland (1985, Chapter 4), this makes Babbage the world's first computer scientist. Ada Lovelace, Babbage's protégée, wrote a note which describes how to program the device. By doing so, Lovelace became the world's first programmer. Unfortunately, the Analytical Engine was never fully implemented (Haugeland, 1985, Chapter 4). As it turns out, Babbage's idea was ahead of its time; to create a computer, we must first define what it means to compute.

Today, what we call "computers" should have been called "computing devices" or "mechanical computers." This is because a computer was actually a job. During World War II, a computer was actually an occupation performed by humans, usually women (Light, 1999). Around the same time, Alan Turing, who lived between 1912 and 1954 (Hagueland, 1985, Chapter 5), was struggling with the Entscheidungsproblem, a question and challenge posed by a mathematician named David Hilbert. The question is: can an "effective" method of "computation" be used to solve all mathematical problems? To define an effective method of computation, Turing devised imaginary devices which we would later call Turing machines. These devices were so simple that they were accepted to capture the essence of computing (Eberbach et al., 2004). Turing machines were then successfully used to argue that the answer to the Entscheidungsproblem is "No." As it turns out, we can never create a formal method or a machine that can solve every mathematical problem. This has an implication not only for the field of mathematics but also for all branches of science. For instance, a consequence of this is that we cannot write software that can decide if arbitrary software will halt or not.

NOTE: Just because a problem is shown to be undecidable in general, it does not mean that every instance of the problem must be undecidable. There is a problem in computer science called the Halting Problem, which is known to be undecidable – i.e., given an arbitrary mathematical problem, can an algorithm solve it within a finite amount of time (Lucas, 2021)? However, if we input a simple Python program with a simple infinite loop into a chatbot like ChatGPT or Microsoft CoPilot, the chatbot will likely recognize that the program will not halt. Below is an example program that ChatGPT will recognize as not halting:

```
while True:
 print("FOREVER!")
```

The goals of Turing machines are to represent computation at the simplest level. However, they are not very practical. To create an actually usable computing

device, we must look elsewhere. First, we must look at a system of logic called the Boolean Algebra, invented by George Boole, who lived between 1815 and 1864 (Burris, 2024) and revised by numerous logicians (Dawson, 2013, Chapter 2). Boolean logic, named after Boole, is now a core and standard part of the computer science curriculum around the world, so I will not repeat those lessons. Multiple people (e.g., Burke and Peirce) noted the connection between electrical circuits and Boolean logic. However, Claude Shannon's master's degree thesis is the most notable example. In it, he argues that electrical circuits could be made equivalent to Boolean expressions, thus all Boolean expressions can be evaluated mechanically (Dawson, 2013, Chapter 2).
Although Shannon has managed to establish the link between the formal and the mechanical, we are still far away from having an actual computing device. After all, Boolean logic is a mere sliver of all mathematical problems that exist. It was not until researchers such as von Neumann developed ENIAC. This machine was practical enough to be used as a real computing device. Furthermore, it possesses two features that would be necessary for modern-day computing: random access memory and subroutines. Random access memory means that a program can almost instantaneously access memory just from its address. Subroutines mean a program can invoke a different program that is stored in ENIAC's memory (Hagueland, 1985, Chapter 4). These two features are not that exciting to us today, and most computer science students take them for granted.

So far, I have skipped contributions from other individuals. Some of them, like Noam Chomsky, will be explored more closely in later chapters. However, I think it is time to introduce John McCarthy, a computer scientist who lived between 1927 and 2011 (The Editors of Encyclopaedia Britannica, 2024). He is considered the Father of AI, because he introduced the term "Artificial Intelligence" itself. McCarthy's contribution involves the creation of LISP, a programming language which led to early AI development. Examples of these include the LISP-based expert systems, where non-programmers can develop new rules for problem-solving. These machines could be viewed as an early form of modern chatbots with the ability to answer questions.

At the beginning, AI is about creating an artificial mind. In 1973, Christopher Longuet-Higgins argued that AI should also be used to study the mind, thus coining the term "Cognitive Science" (Hünefeldt & Brunetti, 2004). Initially, computer scientists argue that the contributions of other fields that also study the mind, such as psychology and neuroscience, are unimportant. While seemingly arrogant at first, this advocacy, I believe, has roots in practicality. First, computers are easy to deal with. While many computer science students struggle in their first year, trying to program something, computers are still far more manipulable than a human being. We cannot simply take someone and force an involuntary experiment on them. Secondly, computer programs are relatively easier to dissect, while the inner thoughts of a human being are much more difficult to investigate. However, some computer scientists like McCarthy and Hayes (1981) argue that contributions from other fields are important. For instance, if we manage to build a human-like robot, we will need to ask the following questions:

- Can the robot be considered to be alive?
- What does it mean to be alive?
- Is the robot conscious?
- What is consciousness?
- What does it mean to be intelligent?
- Etc.

These questions are not easy to answer, so McCarthy and Hayes (1981) argue that philosophy should play an important role in this field. Linguistics join in via Chomsky's contributions. As bio-inspired artificial neural networks proved to be effective, neuroscience has started to play a role as well.

Classical Artificial Intelligence

Classical cognitive science and AI in general try to be as architecture-independent as possible. It tries to eschew the biology or the hardware. Instead, intelligence should be studied in isolation. The goal here is not necessarily to supersede biology. Rather, we want to create the purest and most formal explanation possible for intelligence. If classical cognitive science and AI are successful, then we would not have to deal with a myriad of issues that we are currently facing: the black-box model issue. Today, AI models can be so opaque that they evade our understanding. For instance, we do know how to construct a large language model, but once trained, it is difficult to peer into the system to know what has been learned. Certainly, certain aspects of a modern artificial ANN can be visualized (e.g., Olah et al., 2017). However, they do not explain how neurons end up learning what they learned. The signature work of classical cognitive and AI is the search algorithms. Newell found that many existing real-world problems can actually be reformulated as search problems (Norvig, 1992, Chapter 4). For instance, if an AI agent wants to escape a maze, it can start exploring the neighbours, or it can attempt to go as deeply into the maze as possible. The former action plan can be reformulated as a problem for the breadth-first search algorithm, and the latter can be seen as a problem for the depth-first search algorithm. Puzzles and turn-based games such as chess can also be turned into search problems. However, Norvig (1992, Chapter 4) argues that search-as-intelligence reaches a dead end for multiple reasons. First, some problems are difficult to solve by algorithms alone. Instead, they must rely on approximation. Otherwise, computers may have insufficient resources to solve. Secondly, some real-world problems can be difficult to formulate as search problems.

Another concept that was associated with AI (and by extension, cognitive science) is object-oriented programming (OOP) (Norvig, 1992, Chapter 13). OOP can help capture the notions of objects. However, categorization of objects and entities turns out to be not so black and white. Some categories are easy to reason at first glance, but there are objects in the world that cannot be neatly categorized. Related to OOP is Knowledge Representation (KR). While KR attempts to describe objects, it is also less constrained. Furthermore, OOP as we know it today also now focuses more on software design and may be too limited in scope (Poole & Mackworth, Chapter 16).

Research within cognitive science and AI soon experienced a phenomenon that was coined “AI winter.” In the 1980s, AI development started to slow down. The story goes as follows: AI researchers back then exaggerated the potential of their work, and when the work did not bear out, AI research funding was reduced (Hendler, 2008). I actually encountered the overhyping myself. I once saw a documentary on simulated life, and someone actually argued that one day, we would all live as simulated AI agents a la The Sims. (I am unable to find the documentary, however). Today, we would find this incredulous. Hendler (2008) disputes that the exaggeration is not the full story, however. He argues that the AI winter actually began when DARPA started to prioritize other non-AI research, such as supercomputers.

Although early AI research did not lead to “machines that think”, their concepts have been adopted as Weak AI and computer science in general. Search algorithms are still widely taught and widely used today. For example, in some games like Tic-Tac-Toe, search-based algorithms are deployed to create AI that can win games against human players (Poole & Mackworth, Chapter 14). OOP is now a common concept that many computer scientists take for granted. Meanwhile, knowledge representation is widely used in data science (Pool & Mackworth, Chapter 16).

Regardless of how the winter was triggered, AI could have died right then and there. If Hobbes was the grandfather of AI and McCarthy was its father, then it needed another father to save it: a godfather.

Connectionism

Instead of treating AI as independent of biology, perhaps we could engage in some biomimicry where aspects of nature serve as inspiration for our work. Neural networks, inspired by the organization of the brain, were originally implemented as physical devices by McCulloch and Pitts in the 1940s (Dawson, 2013, Chapter 4). As a physical device, these were difficult to use as the user was required to manually manipulate the wiring of the machine itself. In the 1950s, Rosenblatt developed a successor called Perceptrons with a rather simple training algorithm (Dawson, 2013, Chapter 4). The development of

Perceptrons came to a halt, however, when Minsky and Papert proved in 1969 that Perceptrons could not be used to implement certain Boolean operations. For instance, they could not implement an XOR gate (Kanal, 2003). Later artificial neurons were able to overcome the limitations, but they were still rather impractical to train. Therefore, ANN and connectionism, like Classical AI, also experienced their own winter.

If Hobbes was the grandfather of AI and McCarthy was its father, a godfather was needed to save the dying child from the double winters.

Geoffrey Hinton and his colleagues became known as the godfather of AI, because they continued to improve upon the neural network training algorithm despite the winter (Knight, 2024). A breakthrough came in the form of the **backpropagation** algorithm. Originally developed by Linnainmaa (1970) as part of his Master's degree thesis , Paul Werbos later suggested using it as a possible training algorithm for neural networks in his Ph.D. thesis (Mantri & Thomas, 2021). However, it was not until Rumelhart, Williams & Hinton (1995) that the algorithm became practical for training an ANN, thus making ANNs practical and deep learning highly relevant (Dawson, 2013, Chapter 4).

Embodied

However, just because the fields found their godfathers, it does not mean that there are no additional challenges. Dawson (2013, Chapter 5) states that both Classical and Connectionist AI systems are disembodied from the real world – i.e., no discussion on how the environments affect AI. However, the physical world actually does affect how the AI operates (Dawson, 2013, Chapter 1). For instance, real-world entities like humans and animals have different senses, which lead them to operate differently within the physical world Dawson (2013, Chapter 5). Furthermore, the environment can introduce some challenges that affect cognition. For instance, we can imagine a test of intelligence for any type of entity. During the test, we simply set the test room on fire. A Classical or a Connectionist AI system will not escape the room, because it is not aware of its surroundings. Meanwhile, an embodied AI system will escape because it realizes that finishing the test of intelligence is no longer a priority. In fact, finishing up the test is actually an unintelligent move – what good is the test result if the system is no longer available to experience its success?

This book will discuss some of the embodied computer science, but exclusively from the perspective of intractability and computability. This is because they are directly relevant to formal languages. However, please note that it is not my intention to downplay the relevance of other subfields of computer science that deal with embodied AI, such as computer vision and robotics.

References

Burris, S. (2024). George Boole. In E. N. Zalta, U. Nodelman (Eds), *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/archives/sum2024/entries/boole/

Charity, M., Rajesh, D., Ombok, R. & Soros, L. B. (2020). Say "Sul Sul!" to SimSim, A Sims-Inspired Platform for Sandbox Game AI. *Sixteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 16(1). PKP Publishing Services Network. https://ojs.aaai.org/index.php/AIIDE/article/view/7428

Dawson, M. R. W. (2013). *Mind, Body, World: Foundations of Cognitive Science.* Athabasca University Press. https://www.aupress.ca/books/120227-mind-body-world/

Eberbach, E., Goldin, D. & Wegner, P. (2004). Turing's Ideas and Models of Computation. In C. Teuscher (Ed.), *Alan Turing: Life and Legacy of a Great Thinker.* Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-05642-4_7

The Editors of Encyclopaedia Britannica (2025, March 27). *Charles Babbage*. Encyclopedia Britannica. https://www.britannica.com/biography/John-McCarthy

The Editors of Encyclopaedia Britannica (2024, October 20). *John McCarthy*. Encyclopedia Britannica. https://www.britannica.com/biography/Charles-Babbage

Fjelland, R. (2020). Why general artificial intelligence will not be realized. *Humanities and Social Sciences Communications* 7(10). Nature. https://doi.org/10.1057/s41599-020-0494-4

Hagueland, J. (1985). *Artificial Intelligence: The Very Idea*. The MIT Press. https://archive.org/details/artificialintell0000haug

Hendler, J. (2008). Avoiding Another AI Winter. *IEEE Intelligence Systems* 23(2), 2-4. https://doi.org/10.1109/MIS.2008.20

Hinton, G. E., Dayan, P., Frey, B. J. & M. N. Radford. (1995). The "Wake-Sleep" Algorithm for Unsupervised Neural Networks. *Science* 268 (5214), 1158–61. https://doi.org/10.1126/science.7761831

Hünefeldt, T. & Brunetti, R. (2004). Artificial Intelligence as 'Theoretical Psychology': Christopher Longuet-Higgins' Contribution to Cognitive Science. *Cognitive Processing* 5(3), 137–39. https://doi.org/10.1007/s10339-004-0025-z

Hoel, E. (2021). The overfitted brain: Dreams evolved to assist generalization. *Patterns* 2(5), Article 100244. https://doi.org/10.1016/j.patter.2021.100244

Kanal, L. N. (2003). Perceptron. *Encyclopedia of Computer Science*, 1383–1385. John Wiley and Sons Ltd. https://dl.acm.org/doi/abs/10.5555/1074100.1074686

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (2002). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278-2324. https://hal.science/hal-03926082/document

Light, J. S. (1999). When Computers Were Women. *Technology and Culture* 40(3), 455–83. http://www.jstor.org/stable/25147356

Linnainma, S. (1970). *Algoritmin kumulatiivinen pyöristysvirhe yksittäisten pyöristysvirheiden* [Master's thesis, University of Helsinki]. Helda. http://hdl.handle.net/10138/316565

Longuet-Higgins, H. C. (1973). Comments on the Lighthill Report and the Sutherland Reply. https://www.chilton-computing.org.uk/inf/literature/reports/lighthill_report/p004.htm

Lucas, S. (2021, June). The Origins of the Halting Problem. *Journal of Logical and Algebraic Methods in Programming* 121, Article No 100687. https://doi.org/10.1016/j.jlamp.2021.100687

Mantri, P., & Thomas, J. (2021). Design of Neural Networks. *IOP Conference Series: Materials Science and Engineering 1174*(012023). https://doi.org/10.1088/1757-899X/1174/1/012023

McCarthy, J. & Hayes, P. J. (1981). Some Philosophical Problems from the Standpoint of Artificial Intelligence. *Readings in Artificial Intelligence*, 431-450. https://doi.org/10.1016/B978-0-934613-03-3.50033-7

Norvig, P. (1992). *Paradigms of Artificial Intelligence Programming: CASE STUDIES IN COMMON LISP*. Morgan Kaufmann. https://github.com/norvig/paip-lisp

Olah, C. Mordvintsev, A. & Schubert, L. (2017). Feature Visualization: How neural networks build up their understanding of images. *Distill.* https://doi.org/10.23915/distill.00007

Poole, D. L. & Mackworth, A. K. (2023). *Artificial Intelligence: Foundations of Computational Agents, 3rd Edition.* Cambridge University Press. https://artint.info/3e/html/ArtInt3e.html

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature* 323, 533-536. https://doi.org/10.1038/323533a0

Steweart, D. (2024). "Thomas Hobbes." In E. N. Zalta, U. Nodelman (Eds.), *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/archives/sum2024/entries/hobbes/

2.

Chatbot and Turing Test

It looks like you're writing a letter. Would you like help?
Microsoft Clippit (a.k.a. Clippy)

In a nutshell, a chatbot is software that responds to user input given as natural language. The most basic chatbot, like ELIZA, only allows the user to provide text input and respond to the user with text input. However, the current generation of chatbots also supports multimodal input. In addition to the text inputs, they also allow the user to provide other types of data such as images, sounds, or videos. The multimedia input is then translated into a form that the software can process (i.e. using Natural Language Processing). The chatbot can then respond to the user with multimedia data. For instance, if the user asks it to draw an image, the chatbot itself can also return an image (See Figure 1 for an example of this).

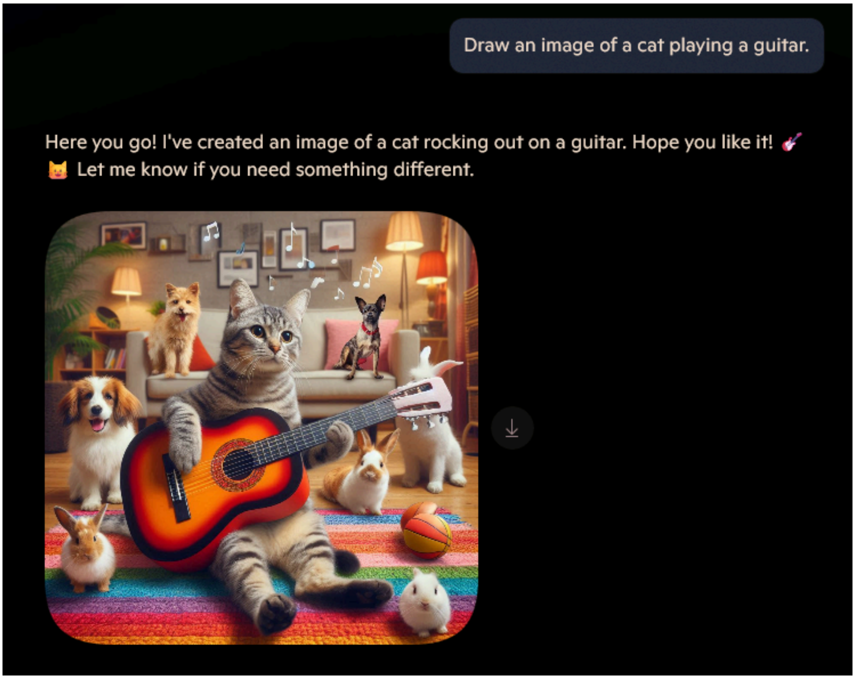

Figure 1. *A screenshot of Microsoft Windows 11 CoPilot creating an image of a cat playing a guitar based on a prompt. The image was generated on June 4, 2025. This prompt is inspired by a Google Gemini advertisement (URL: https://www.youtube.com/watch?v=9bBfYX8X5aU).*

There are multiple types of Chatbots as well. A way of classifying them is whether they are task-oriented or not (Luo et al., 2022). A **task-oriented** chatbot aims to help the user complete specific tasks and is constrained by its domain (Luo et al., 2022). For instance, FlightHub, an online travel agent, has a task-oriented chatbot named Mila (Businesswire, 2025) that the user can interact with in order to resolve issues related to their itinerary. Figure 2 contains a screenshot of the author's interaction with Mila via a mobile phone. In this case, he was trying to get Mila to release a hold on his credit card, but Mila was unable to do so. An older, but more constrained example of this would be Microsoft Clippy, software designed to assist the user with using Microsoft Office.

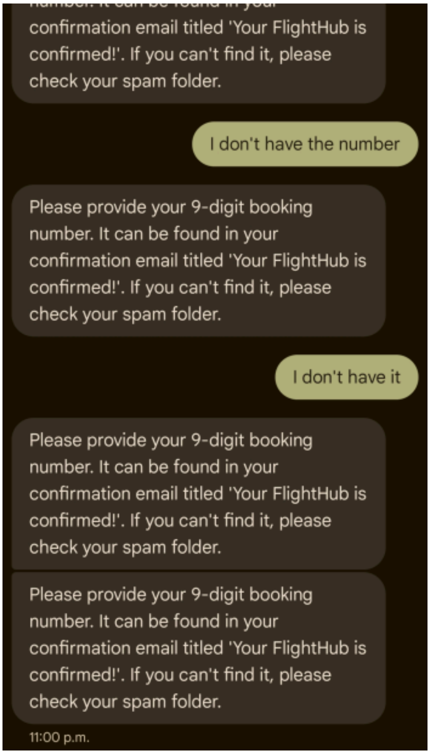

Figure 1. *The author's conversation with Flighthub's chatbot around April 2025. The chatbot repeated the same response ad nauseam.*

A **non-task-oriented chatbot**, on the other hand, aims to be as conversational as possible (Luo et al., 2022). It lacks focus and does not aim to solve a problem. The very first chatbot, ELIZA by Weizenbaum (1966), was designed to be **non-task-oriented**. As an early chatbot, ELIZA was rather primitive. Instead of trying to cater to the user's every whim, it instead tries to constrain the conversation. It acts like a therapist and uses specific responses to ensure that the user will only ask the questions that it wants to be asked. When the user first runs ELIZA, they may be impressed by ELIZA's natural-sounding responses. However, as a simple computer program with a limited number of prepared responses, the illusion will eventually break. The user, at one point, will realize that the conversation with ELIZA is going nowhere.

While ELIZA only relies on text inputs and outputs, modern non-task-oriented chatbots can also respond using multimedia. An example of this is Neuro-sama, a chatbot designed to act like a virtual YouTuber by an anonymous individual with the alias of Vedal. Neuro-sama can also respond via text as well as via voice. Neuro-sama, unlike ELIZA, also has a 3D avatar, which enhances connection with the user (Amato et al., 2024). Neuro-sama's gaff and childlike demeanour are actually what make her entertaining to watch. However, it is not important to note that not all AI mistakes are entertaining. Some can have far darker consequences. To avoid triggering readers, I will not explain the materials. Instead, interested readers can search online themselves on the topic of chatbots and self-harm.

Luo et al.'s Classifications

Luo et al. (2022) argue that classifying chatbots based on a binary basis (task-v. non-task-oriented) is too simplistic, as this also hides the capabilities of the chatbots. For instance, a task-oriented chatbot may be more complex than a non-task-oriented one, even though the non-task-oriented one has more freedom in how it responds to the user. Instead, Luo et al. (2022) argue that we should look at how the chatbots are implemented. They argue that there are 6 categories:

1. **Template-based:** The chatbot responds mostly by "filling" out a set of pre-determined responses. ELIZA is an example of such a chatbot. When the user inputs text, ELIZA would match the text against existing patterns. Then, it responds according to the match with a pre-programmed response. The response is slightly modified in order to personalize the answer and make it sound more natural. If the program does not recognize a question, it asks the user to provide more information (Weizenbaum, 1966). As ELIZA is a small program, it can quickly exhaust all prepared responses.
2. **Corpus-based:** The chatbot supplements its response by using external knowledge provided in a large body of text, knowledge bases, or databases. Microsoft Windows 11 CoPilot is a well-known example of a chatbot that searches the Internet for information. In my opinion, I think the term "expert system" is more suitable. Expert systems are classical AI systems that incorporate knowledge bases. While they do not need to have a chatbot-like interface, they still aim to have a human-friendly interface. Furthermore, "corpus" is also a term in computational linguistics and NLP that describes data sources for training and modelling. It does not include something like a knowledge base or database.
3. **Intent-based:** The chatbot tries to gauge the user's intent so that it can better serve the user and help them to accomplish a task. Intent-based chatbots are often task-oriented. An example of this is Mila by FlightHub.
4. **RNN-based:** The chatbot relies on recurrent neural networks (RNN) to generate natural-sounding sequences. This allows for better personalization. I think this name is too limited, because some chatbots are not RNN-based. For instance, chatbots like Neuro-sama

are implemented with large language models (LLMs). While LLMs use artificial neural networks, they do not always rely on RNN. Instead, the most popular ones use a type of neural network called Transformers. (Later on, Ramsauer et al. (2021) found that Transformers can be transformed into RNNs, but I digress.) I think "LLM-based" is a better name.

5. **RL-based:** The chatbot's outputs are either rewarded or punished using the principles of reinforcement learning (RL). If the chatbot's outputs are rewarded by the user (e.g., giving a thumbs-up in Microsoft Windows 11 CoPilot), the chatbot will be more inclined to make a similar output.
6. **Hybrid:** The chatbot uses elements of the five aforementioned categories. Some powerful chatbots like ChatGPT, Gemini, and more are of the hybrid type.

Umwelt, Chatbots, and Semiotics

Luo et al. (2022) state that a chatbot can be multimodal. But what does it mean to be multimodal? For me, we need to look at how a human being senses the world around them. Classically, we say that we can see, hear, taste, smell, and feel. However, each organism in the world will have different senses based on its biological wiring. Even though the physical world is the same, we can still experience it differently. For instance, many people with colour blindness will have difficulty separating the colours red and green from each other. Other organisms also have widely different experiences. For instance, a horse can see behind itself while a human cannot, thus a horse can never experience what it feels like to be surprised by something that approaches it from behind. Von Uexküll coined the term "umwelt" to describe the experience that an organism can experience based on its senses in the world. As it turns out, it is impossible to separate the organism from its environment (Dawson, 2013, Chapter 5).

While I do not argue that chatbots are alive like any other organisms, I also argue that they have their own umwelten. A simple template-based chatbot like ELIZA can only receive input from keyboard strokes and output texts via the monitor. Meanwhile, multimodal chatbots can accept images, voices, and other sounds. They can interact with the users in a more creative manner. Thus, their umwelt is far "larger" than the one for ELIZA.

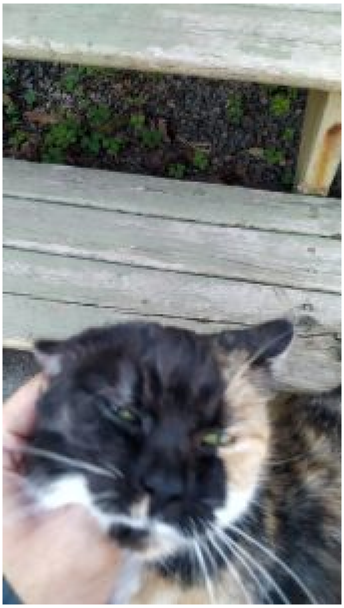

Figure 3. *An image of a calico cat receiving a chin scratch.*

With a larger umwelt, a chatbot would have a superior capacity to deal with **signs**. **Signs** are concepts in semiotics which represent meanings that we glean from the world. Signs can be transformed based on our interpretation process, called **semiosis**. For instance, when we see an image of a cat, we know that it is not a real physical cat. Rather, the image serves as a stand-in for the concepts of the cats. There are also multiple possible interpretations based on the context and scenarios. For example, the image of a cat in Figure 3 can not only stand for a cat, but also stand for a calico, a chin scratch, affection, and more.

However, most importantly, if the umwelt is not sufficiently large, some signs are simply unavailable. A chatbot like Microsoft Windows 11 CoPilot may be able to extract multiple signs (i.e. performing a semiosis) from the image. For example, when I input Figure 3 with the prompt: "Can you tell me what this is?", it responded:

> Absolutely! This is an image of a cat being gently held by a hand. The cat has striking green eyes and a coat with black, white, and brown fur. The setting appears to be outdoors, with weathered wooden steps in the background, along with some green plants and small rocks nestled between the steps.
>
> It looks like a relaxed, curious kitty enjoying some attention! Do you happen to know the cat's name?

A more limited chatbot like ELIZA will extract nothing from the image because it is incapable of accepting an image file to begin with.

The study of signs and semiosis forms the field of semiotics, which can be argued as a more meta form of linguistics. While linguistics deals directly with languages, semiotics deals with language as well as more abstract forms of communication. Although not normally taught in foundational computer science courses, semiotics often shows up in certain areas. For instance, in the area of visualization, where information is presented as visual presentations, visual semiotics has been used to formalize how we present information. Bertin, a notable researcher in the field of visualization, argues that graphical elements can be manipulated to convey different information (Carpendale, 2003). While this book does not cover semiotics in detail, it is important to note that linguistics (including computational linguistics) and semiotics have some overlaps. For instance, syntax is both a concept in programming languages and natural languages. We will discuss semiotics more in Chapter 10.

Turing Test: A Chatbot Test before Chatbot

Turing Test, not to be confused with Turing Machine, is a test of machine intelligence proposed by Alan Turing (1950). If a machine can pass this test, Turing argues, it could be deemed as intelligent. Before arguing why this test is flawed, we should first discuss the setting for the test. A Turing Test (i.e. Imitation Game) consists of a human player and a machine player trying to converse with a human judge. The two entities cannot directly converse with the judge. Instead, they must use communication devices (e.g., a keyboard and a screen) to ensure that the judge is not aware of the identity of the player. If the judge mistakes the machine player as a human one, then the machine is considered to have passed the test (Turing, 1950). Figure 4 contains the diagram of an example arrangement.

By masking the human and the machine players, Turing (1950) essentially argues that intelligence is independent of modality. The ability to see, speak, or hear should be excluded. Instead, intelligence is something that is as internal to the entity as possible. This line of thinking is very much in line with Classical Cognitive Science, which emphasizes thinking over how one interacts with the real world. In Classical Cognitive Science, it is assumed that thinking is algorithmic in nature and, just like any computer algorithm, the input and the output data from the algorithm have been properly formatted. This is in contrast with the Embodied Cognitive Science, which regards the real world as an important influence on how one may think.

Today, many users of chatbots know that the test is highly flawed. They know that software like ChatGPT and Gemini can generate very natural-sounding answers while also knowing that these chatbots are not human-like. And as it turns out, passing the test can actually be quite easy. According to French (2000) and Switzky (2020), one of the early examples of software passing the Turing Test is ELIZA by Weizenbaum (1966). The chatbot was simply converting one string the user inputted into another one using pattern matching, with the ability to repeat several input data (e.g. the user's name). Despite its limitations, it was able to convince people that it was human-like (French, 2000), even though there is nothing intelligent about it. We should be aware, though, that ELIZA and other similar software try to limit the scope of the conversation. ELIZA pretends to be a psychotherapist so that the user will only talk about their own psychological issues, something that ELIZA's own programming accounts for. Meanwhile, PARRY simply pretends to be a person suffering from schizophrenia (French, 2000), which makes the user less inclined to speak to it on broad topics. French (2000) argues that constraining the topic goes against the spirit of the Turing Test. The test, as originally envisioned by Turing, is open-ended; any topic should and could be discussed among all players and judges.

Passing a Turing Test may not sound impressive, but if a chatbot manages to pass it, it may still possess some abilities to terrify users. Some users may view a chatbot as something more intelligent than it actually is. This is called the ELIZA effect, named after how ELIZA, rather primitive software, managed to fool people (Switzky, 2020). A notable example is LaMDA, which is now known as Google Gemini. Lemoine, a developer working LaMBDA, believed that it had become conscious. He (Lemoine, 2022) published the conversation between him and the chatbot here: https://cajundiscordian.medium.com/is-lamda-sentient-an-interview-ea64d916d917. Although I will not discuss the topic of large language model (LLM) in this chapter, something that LaMDA is based off, I can confidently argue that a LLM is not as conscious as a normal human being [1].

References

Amato, N., De Caloris, B., de Gioai, Venezia, M. N. & Loglisci, C. (2024). Can an AI-driven VTuber engage People? The KawAIi Case Study. *Joint Proceedings of the ACM IUI Workshops 2024*. https://ceur-ws.org/Vol-3660/paper21.pdf

Business Wire. (2025, January 29). Meet Mila, FlightHub's New AI Assistant Changing the Way You Travel. Business Wire. https://www.businesswire.com/news/home/20250129685451/en/Meet-Mila-FlightHubs-New-AI-Assistant-Changing-the-Way-You-Travel

Carpendale, S. (2001). Considering Visual Variables as a Basis for Information Visualization. http://hdl.handle.net/1880/45758

Dawson, M. R. W. (2013). *Mind, Body, World: Foundations of Cognitive Science.* Athabasca University Press. https://www.aupress.ca/books/120227-mind-body-world/

French, R. M. (2000). The Turing Test: The First Fifty Years. *Trends in Cognitive Sciences* 4(3), 115-121.

Lemoine, B. (2022, June 11). Is LaMDA Sentient? — an Interview. *Blake Lemoine.* https://cajundiscordian.medium.com/is-lamda-sentient-an-interview-ea64d916d917

Lokman, A. S. & Ameedeen, M. A. (2019). Modern chatbot systems: A technical review. *Proceedings of the Future Technologies Conference (FTC) 2018* 2, 1012-1023. https://doi.org/10.1007/978-3-030-02683-7_75

1. This does not mean it cannot be "semi-conscious" due to its architecture being based on artificial neural network, which in turn, is inspired by biological neurons; neurons are necessary for consciousness.

Luo, B., Lau, R. Y. K., Li, C., & Si, Y.-W. (2022). A critical review of state-of-the-art chatbot designs and applications. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 12(1), Article e1434. https://doi.org/10.1002/widm.1434

Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Gruber, L., Holzleitner, M., Adler, T., Kreil, D., Kopp, M. K., Klambauer, G., Brandstetter, J. & Hochreiter, S. (2021). Hopfield Networks is All You Need. *The Thirteenth International Conference on Learning Representations*. https://doi.org/10.48550/arXiv.2008.02217

Switzky, L. (2020). ELIZA Effects: Pygmalion and the Early Development of Artificial Intelligence. *Shaw* 40(1), 50-68.

Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind* LIX(236), 433-460.

Weizenbaum, J. (1966). ELIZA—a computer program for the study of natural language communication between man and machine. *Communications of the ACM* 9(1), 36-45. Association for Computing Machinery. https://doi.org/10.1145/365153.365168

Yeh, S-. F., Wu, M. -Hs., Chen, Tz. -Y., Lin, Y., -C., Chang, X.- Y., Chiang, Y. -Hs. & Chang, Y. -J. (2022). How to Guide Task-oriented Chatbot Users, and When: A Mixed-methods Study of Combinations of Chatbot Guidance Types and Timings. *CHI Conference on Human Factors in Computing Systems (CHI '22)*, Article 488. Association for Computing Machinery. https://doi.org/10.1145/3491102.3501941

Media Attributions

- cat_playing_guitar
- flighthub_trimmed
- squish

3.

No Ghost in the Machine

I ought to be thy Adam, but I am rather the fallen angel.
The Being, a character in *Frankenstein; or the Modern Prometheus* (Shelley, 1831, Chapter 10).

If we manage to build a human-like robot who is perfectly human in any way, shape or form, is the robot actually a human? This question is actually older than artificial intelligence (AI) itself. It actually predates computers and all means of computational calculation. In its more primordial form, the question becomes: "Do you believe in ghosts?" These questions beget other smaller questions:

- Is the ghost simply another type of being existing in a different realm of existence that occasionally intersects with the realm of the being?
- Is the ghost its own fully conscious entity in our realm that can fully interact with us if we have the right tools and equipment?
- Is the ghost something less than a full being but still capable of acting on the physical world in some way, like making the "ooOOooo" sound and shaking a few random items?
- Is the ghost just some sort of sacred substance that can animate organic beings, but requires bodies to actually act on something?

To make "ghost" sound more noble, we can also substitute it with "spirit" instead. The difference is simply semantics, however.

Ghosts vervsus/and Machines

While I am certain that the debate over whether an artificial being could be considered as living has existed before René Descartes and Thomas Hobbes were born, their debate was considered to be the most notable one. Hobbes opens his most famous work, *Leviathan* (1651), by arguing that all living beings are simply machines acting out their parts (Dawson, 2013, Chapter 2):

> Nature (the art whereby God hath made and governes the world) is by the art of man, as in many other things, so in this also imitated, that it can make an Artificial

> Animal. For seeing life is but a motion of Limbs, the begining whereof is in some principall part within; why may we not say, that all Automata (Engines that move themselves by springs and wheeles as doth a watch) have an artificiall life? For what is the Heart, but a Spring; and the Nerves, but so many Strings; and the Joynts, but so many Wheeles, giving motion to the whole Body, such as was intended by the Artificer?

Later on, in the opening of Chapter V of *Leviathan,* Hobbes also describes reasoning as:

> When a man Reasoneth, hee does nothing els but conceive a summe totall, from Addition of parcels; or conceive a Remainder, from Substraction of one summe from another: which (if it be done by Words,) is conceiving of the consequence of the names of all the parts, to the name of the whole; or from the names of the whole and one part, to the name of the other part. And though in some things, (as in numbers,) besides Adding and Substracting, men name other operations, as Multiplying and Dividing; yet they are the same; for Multiplication, is but Addition together of things equall; and Division, but Substracting of one thing, as often as we can. These operations are not incident to Numbers onely, but to all manner of things that can be added together, and taken one out of another. For as Arithmeticians teach to adde and substract in Numbers; so the Geometricians teach the same in Lines, Figures (solid and superficiall,) Angles, Proportions, Times, degrees of Swiftnesse, Force, Power, and the like; The Logicians teach the same in Consequences Of Words; adding together Two Names, to make an Affirmation; and Two Affirmations, to make a syllogisme; and Many syllogismes to make a Demonstration; and from the Summe, or Conclusion of a syllogisme, they substract one Proposition, to finde the other. Writers of Politiques, adde together Pactions, to find mens Duties; and Lawyers, Lawes and Facts, to find what is Right and Wrong in the actions of private men. In summe, in what matter soever there is place for Addition and Substraction, there also is place for Reason; and where these have no place, there Reason has nothing at all to do.

In this argument, he claims that all living things are purely mechanical in nature, whether they are made by God or by other entities. Furthermore, reasoning is simply a calculation of some sort. Thus, intelligence and the mind are just a consequence of machines running, and we can actually construct an artificially intelligent being. Based on this argument, Hobbes was called “The Grandfather of Artificial Intelligence” by Hagueland (1984, Chapter 1).

This argument serves as the mantra for cognitive science. Cognitive scientists and Strong AI creators do not necessarily believe we are truly ready to create an artificial mind today. However, they believe that, given the right tools and circumstances, we can do so. In essence, cognitive science does not believe in ghosts and commits to **materialism**. Now, this line of reasoning does have a flaw, but we must first discuss **dualism**.

Descartes, who was Hobbes’s contemporary, did not believe that intelligence is purely mechanical in nature. Instead, there must be some sort of spirit. The crux of his argument is *qualia*. A quale can be a sensation or an experience that we cannot quite objectively express (Kanai & Tsuchiya, 2012). An example of a qualia is pain (Kanai & Tsuchiya, 2012). When we are experiencing pain in our toes after we have stubbed them, we can tell other people that we are in pain. However, we also have the subjective “paininess”[1] of pain. This subjective experience is indescribable, but it does not mean that it is not real. For instance, if our toe pain is bad, we will pause, sit down, and examine our toes to see if they need to be addressed. If the pain is really bad, we can even feel our hearts palpitating and our thought patterns being disrupted by the “paininess.” Meanwhile, people who do not experience the “paininess”, for whatever reasons, will not react or try to protect themselves. Descartes argues that we need spirits to fully experience qualia. However, having a body is also important because a ghost cannot act upon the world without a physical body. Because Descartes believes in the dual importance of the spirit and the body, he argues for **dualism**.

It is important to note that Descartes did not believe that all living beings need to possess the spirit. Only humans can have spirits. However, Descartes did not argue that animals cannot experience pain or have any feelings at all. It is just that animals are barred from having full consciousness. According to Prof. William Seager, one of my philosophy instructors, said that Descartes actually had a pet dog, and while he would view animals as machines, they are still God’s beautiful machines. No one should wantonly cause harm to them, as such an act is ungodly.

Descartes and Animal Abuses: Prof. William Seager also told my class that some people misunderstood Descartes’s dualism argument and used it as a blanche carte to commit animal abuses. They would also perform demonstrations where animal abuse would occur in front of the audience. I cannot find conclusive proof for these demonstrations, however.

Now, one may think that Descartes’s argument is a bit too extreme as it argues that animals do not necessarily have souls. It is important to note that though there are other versions of dualism, which are not as extreme. Another version by Cudworth argues that animals can have souls. However, they are not as conscious as human beings. Meanwhile, human beings are less conscious than angels, demons, or God Himself. Also, having a spirit does not mean that we must be conscious. For instance, when we fall asleep, we are often unconscious of our own surroundings (Allen, 2013).

1. I borrowed this term from Vervaeke.

Descartes's argument and other versions of dualism are difficult to topple because it seems that we cannot simply get qualia out of mechanical operations. For instance, if we build a Raspberry Pi robot with a heat sensor that runs away when the room gets too hot, the robot is unlikely to experience the heat and the sense of danger as we do. Still, cognitive science commits to Hobbes's argument, even if it is flawed. Cognitive science treats the spirit as the unknown that still needs to be explained, and science as a way to make the unexplainable explainable.

The Chinese Room Experiment

An extension to the concept of dualism is the Chinese Room Argument. Originally proposed by Searle in 1980, the Chinese Room argument is a thought experiment suggesting that artificial intelligence (AI) cannot care about what is going on within its processes (David, 2024). Unlike the Turing Test, which implicitly promotes Strong AI, the Chinese Room argument aims to strengthen dualism and to discredit the whole enterprise of Strong AI. The premise of the thought experiment is as follows (David, 2024):

- There is an individual in a room who cannot interact with the outside world except through a slit.
- This individual can only communicate in English (Dawson, 2013, Chapter 2).
- Outsiders can submit texts written in Chinese through the slit.
- The individual, once given a slit, consults a rule book. The individual compares a text against the rule book and writes a response using the book.
- Once the response is written, the individual sends it through the slit.

The question is: Does the individual understand Chinese? The answer that Searle wants us to reach is "no."

At first glance, the room does not seem relevant to AI. However, everything in this room is supposed to be allegorical and representative of Strong AI. The room itself is a computing device with the rule book being a program. The individual is essentially the spirit in dualism. The outsiders are essentially us, the human users of the AI. The point that is being made with the Chinese Room argument is that a computer program is not capable of understanding. All it does is merely manipulate one set of symbols into another using an algorithm. It lacks the intentionality, i.e. the ability to care about what is going on.

A cognitive scientist would wish to attack the Chinese Room argument as it is an argument against Strong AI. However, just like dualism, doing so will prove exceedingly difficult. However, we should attempt this anyway. A possible target of attack is probably the rule book. We may say that the Chinese language and all of its possible responses are so large that we cannot fit them into a single rule book. By making this argument, we actually play right into Searle's hands. As the book is an allegory for AI software, we essentially argue that AI software for the Chinese language cannot exist. Furthermore, we can replace Chinese with any other non-English language as well.

Another venue of attack is that the individual themselves could have memorized the rule book. After all, if the individual has been reading the same book over and over again, they are bound to memorize something. However, the counter to this is that the individual has now become a rule book itself. And they themselves still do not understand Chinese (Cole, 2024), because pure rote memorization is not how language learning works. Alternatively, we can argue that by memorizing the rule book, the individual does care about the language. The care may be borne out of pragmatic reasons, e.g., the individual does not want to repeatedly flip through the same book, or out of linguistic appreciation of the language. Thus, the individual is gaining intentionality. This argument actually ends up helping Searle, though, because Strong AI is supposed to be free of intentionality. It is about explaining all phenomena using purely computer science terms without any ghost in the machine. By saying that the individual cares, there is a ghost in the machine.

Formalization and Ryle's Regress

Lepidus: What manner o' thing is your crocodile?
Mark Antony: It is shaped, sir, like itself;
A conversation between two characters from a play *Antony and Cleopatra* by Shakespeare (1623, Act 2, Scene VII)

Many people may have a misconception that computer science is all about programming, and a computer scientist is just someone with a large salary who types away code and runs it for their overlords. However, while a computer science graduate may end up doing that, at its core, computer science is about formalization. I will describe the process of formalization in the next part of the book. For now, we can treat formalization as "writing a program" for something. In classical AI, once we have a program for something, we should then have a formal explanation of that thing. For instance, if we want to have a formal explanation of how chess is played, we can develop a program that plays chess. Then, we have a formal definition of playing chess.

Early cognitive science and AI development's goal is to use programs to explain cognitive phenomena. We are not talking about creating machine learning models here. We are talking about writing actual programs. We know, from the earlier chapters, that this enterprise is doomed to fail. However, for now, let's proceed with trying to create a formal explanation anyway. Suppose that we are developing a formal explanation for solving a maze. We then convert the act of solving the maze into a search problem, and then write a search algorithm to solve it. An example search algorithm that can be used for maze-solving is A*, which is as follows (Poole & Mackworth, 2023, Chapter 3.6):

- The maze is transformed into a graph data structure with one node designating the entrance and another designating the exit.
- We create a function $h(n)$ which tells us the distance between node n and the goal node.
- We use breadth-first search as normal; however, instead of picking an arbitrary neighbouring node to visit next, we first visit the neighbouring node with the lowest $h(n)$ value.

Once the algorithm is shown to be correct, we can claim that this is a formal description of how humans can solve mazes.

Now, we can state that we have a high-level definition of maze solving. However, what about the lower-level definition? How do a group of neurons firing eventually cause the graph data structure and the algorithm to manifest? This is not so clear. In fact, we encounter something called "Ryle's Regress." Ryle's Regress is also known as a homunculus argument (Dawson, 2013, Chapter 2). Homunculus is based on a Medieval concept which argues that all humans started out as miniaturized and adult versions of themselves embedded inside sperm or egg cells. Then, after successful fertilization with egg cells and deliveries, they grow into their adult forms (Baugh, 2025). So in the past, when someone asked, "How does a human grow?" The answer would be, "It involves a human growing." This is not a very helpful explanation.

In this case, we think that humans *kinda, sorta* use graphs to solve mazes, but how do these graphs exist? How do people construct these graphs? And what are the constituent components? How do brains create them? Cognitive scientists wish they could answer these questions so much. As it turns out, these questions are not so easy to answer.

Another question that we must answer is: Is this high-level explanation based on A* actually representative of how humans solve maze puzzles? As Vervaeke (1997) warns, we cannot simply claim that an algorithm is representative of a psychological phenomenon. The explanation must still be verified through the framework of psychology. According to Kander et al. (2023), humans actually deploy an algorithm that is a hybrid between the breadth-first and the depth-first search and not A*.

Naturalistic Imperative and Hilbert's Mantra

We must know. We will know.
A quote by David Hilbert from Smith (2014).

According to Vervaeke (1997), cognitive science follows three steps when it analyzes a cognitive phenomenon:

- **Analyze:** We must first attempt to learn what the phenomenon may be.
- **Formalize:** Once we have some sort of explanation, we want to formalize it and we attempt to do so in a way that avoids the homunculus argument. In the computer science parlance, this means to simply write a program for it. A program can act as a stand-in for a formal explanation.
- **Mechanize:** When the formalization is developed, we create an artificial implementation of the concept. By mechanizing it, we understand it.

He calls these steps the naturalistic imperative. He argues that it is something that cognitive science must do.

Anyone who is familiar with the history of computer science may be reminded of Hilbert's mantra: "We must know. We will know" (Smith, 2014). Hilbert was the person who posed the Entscheidungsproblem to the world, daring everyone to solve it. The problem is as follows: any mathematical proposition, through an effective means of computation, can be proven either to be true or false (Eberbach, Goldin & Wegner, 2004). Some statements are easily provable. For instance, we know that all even numbers must be evenly divisible by 2. But what about all the other statements? As it turns out, Turing (1936) shows that the Entscheidungsproblem does not have a solution. This means there are statements out there that we will never be able to prove.

At first glance, Turing's proof has nothing to do with cognitive science. However, Turing's proof has been cleverly designed to be applicable to so many fields in computer science. While his main goal was to prove Hilbert wrong, his proof can also be generalized to say that we will never know if an arbitrary computer or program will halt. This makes AI verification much more difficult and reduces our chances of defeating Descartes and slaying the homunculus.

To me, the natural imperative in Vervaeke (1997) is also similar to Hilbert's mantra. It is likely to be futile, but it serves as a beacon that we all should try to reach towards. In a later paper that he co-wrote with Jaeger et al. (2024), he argues that formalization of cognition is not possible. They ended the paper with the following quote: "Life is meaningful and precious that way. No machine will ever understand that." Now, is this statement logically true? Maybe, this question is undecidable.

And this is not the signal for the reader to stop reading this book. There are more things that we must learn. The failures and challenges, just like successes, can teach us many things. We may never reach the ultimate goal of defeating dualism and building a Strong AI, but the real cognitive science and AI are the knowledge that we generate along the way.

References

Allen, K. (2013). Cudworth on Mind, Body, and Plastic Nature. *Philosophy Compass 8*(4), 337-347. https://doi.org/10.1111/phc3.12026

Baugh, L.S. (2025, March 7). Homunculus. *Encyclopedia Britannica*. https://www.britannica.com/science/homunculus-biology

David, C. (2024). The Chinese Room Argument. In E. N. Zalta & U. Nodelman (Eds.), *Stanford Encyclopedia of Philosophy* (Winter 2024 Edition). https://plato.stanford.edu/entries/chinese-room/

Dawson, M. R. W. (2013). *Mind, Body, World: Foundations of Cognitive Science*. AU Press. https://www.aupress.ca/app/uploads/120227_99Z_Dawson_2013-Mind_Body_World.pdf

Eberbach, E., Goldin, D. & Wegner, P. (2004). Turing's Ideas and Models of Computation. In C. Teuscher (Ed.), *Alan Turing: Life and Legacy of a Great Thinker*. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-05642-4_7

Haugeland, J. (1984). *Artificial Intelligence: The Very Idea*. The MIT Press.

Hobbes, T. (1651). *Leviathan*. https://www.gutenberg.org/files/3207/3207-h/3207-h.htm#link2H_4_0001

Jaeger, J., Riedl, A., Djedovic, A., Vervaeke, J. & Walsh, D. (2024). Naturalizing relevance realization: why agency and cognition are fundamentally not computational. In V. Puusepp (Ed.), *Frontier in Psychology* 15(1362658). https://doi.org/10.3389/fpsyg.2024.1362658

Kanai, R. & Tsuchiya, N. (2012). Qualia. *Current Biology* 22(10). Elsevier. https://doi.org/10.1016/j.cub.2012.03.033

Kander, F., Willkomm, H., Ibs, I., Rothkopf, C. A. (2023). Finding your Way Out: Planning Strategies in Human Maze-Solving Behavior. In M. Goldwater, F. K. Anggoro, B. K. Hayes, & D. C. Ong (Eds.), *Proceedings of the 45th Annual Conference of the Cognitive Science Society*. https://escholarship.org/uc/item/94t1t8kw

Poole, D. L. & Mackworth, A. K. (2023). *Artificial Intelligence 3e: Foundations of Computational Agents*. Cambridge University Press. https://artint.info/index.html

Richstein, J. (2001). Verifying the Goldbach conjecture up to $4 \cdot 40^{14}$.

Mathematics of Computation 70(2001). American Mathematical Society. https://doi.org/10.1090/S0025-5718-00-01290-4

Shakespeare, W. (1623). *Antony and Cleopatra*. https://shakespeare.mit.edu/cleopatra/

Shelley, M. W. (1831). *Frankenstein; or, the Modern Prometheus.* Project Gutenberg. https://www.gutenberg.org/files/84/84-h/84-h.htm#chap24

Smith, J. T. (2014, February). David Hilbert's Radio Address – English Translation. *Convergence.* Mathematical Association of America Publications. https://old.maa.org/press/periodicals/convergence/david-hilberts-radio-address-english-translation

Turing, A. M. (1936). On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society* s2-42(1), 230-265. https://doi.org/10.1112/plms/s2-42.1.230

Vervaeke, J. A. (1997). *The Naturalistic Imperative in Cognitive Science.* [Doctoral dissertation, University of Toronto]. TSpace. http://hdl.handle.net/1807/10696

II

Theory of Computing (and Language)

4.

Turing Machine: A Magical Tape Reader

It's been a long road
Getting from there to here
It's been a long time
But my time is finally near
From Where My Heart from Take Me, Sung by Russell Watson (2001)

In the previous chapter, I implied that we can formalize a cognitive phenomenon by writing an algorithm. However, an algorithm itself is merely a description. To have an AI, the description must be acted out in some way, shape or form. An entity, be it mechanical or biological, must read out the algorithm and perform the instructions within it. Such an act can be called **computation**. Most computer science students today would think of computation as some sort of mechanical manipulation of electrical currents on some circuit boards. However, computation does not actually need to be carried out by machines, or even need electricity. After all, a computer used to be a person, most likely a woman, who performed repetitive calculation tasks (Light, 1999).

If computation is not about affecting the circuit board, then what is it? Turing (1936) answered this in his proof showing that the Entscheidungsproblem is not decidable. Entscheidungsproblem, posed by Hilbert, was stated in the previous chapter as: any mathematical proposition, through an effective means of computation, can be proven either to be true or false (Eberbach, Goldin & Wegner, 2004). To tackle the Entscheidungsproblem, Turing must first formalize the notion of "effective means of computation" itself. To him, this would be a challenging task, because a computer in his time was a person. She was not a mechanical entity, and her operations would not be as standardized as a mechanical one. When she added numbers together, she would carry out the operations within her brain, invisible to all in the world. Based on her training, she may add the numbers in different ways. A computer may be able to add very large numbers in **her** head, while another may require a pen and paper for aids. A novice computer may also not be able to compute as well as a more experienced one. This means a human computer is not a good model of computation. Therefore, Turing (1936) devised a type of machine, which we now call Turing machines (Eberbach, Goldin & Wegner, 2004), to represent computation at the most basic level.

Turing Machine: A Magical Tape Reader/Writer

> NOTE: The definition of Turing Machine is based on Goldin et al. (2004). There are other definitions with more restrictive requirements.

Turing (1936) defines computation as the working of a machine that takes in a tape and converts the tape using pre-specified steps. Each machine has its own $\sum$, a set of symbols that it can recognize and write. At the most basic level, $\sum$ only has two symbols. However, more symbols are possible. For instance, if the machine is designed to deal with a hexadecimal number, $\sum$ must at least contain $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$ and some additional symbols for work. Regardless of how $\sum$ is defined, one of the symbols must be a "blank" symbol. The tape is also assumed to be infinitely long and infinitely padded on both sides by the blank symbol.

The machine also has a set of states called Q. The machine starts in the initial state $q_0 \in Q$. Once the machine is running, it takes in the tape and starts reading a cell. Based on the symbol seen by the tape head and the current state, the machine can do one of the following:

- **Write:** Write a symbol using one of the symbols in $\sum$.
- **Move:** The tape head can move left, right, or stay.
- **Transition:** Change to a different state.

Once a halting state is reached, the machine finishes, and the user can read the tape to get the result.

To better understand a Turing Machine, let's examine how a machine that recognizes whether a binary number is even. Let $\Sigma = \{0, 1,_\}$. The binary number is represented as 0 or 1. We let _ represent the blank symbol. Below is an example of a valid tape for this specific Turing Machine:

…	_	_	1	0	1	1	_	_	…

To recognize if a tape represents an even number, the machine must check if the final digit is 0 or not. If it is, then the machine should indicate on the tape that the tape is even. We can use a different symbol to indicate this, but in this case, we simply use 0 to indicate "False" and 1 to indicate "True." We write this down on the tape and then erase everything else. Assuming that the first cell the machine will read contains 0 or 1, the states can be as follows:

- q_0:
  - **If the cell is 0:** Write 0, Move right, Transition to state q_0.
  - **If the cell is 1:** Write 1, Move right, Transition to state q_0.
  - **If the cell is _:** Write _, Move left, Transition to state q_1.
- q_1:
  - **If the cell is 0:** Write 1, Move left, Transition to state q_2.
  - **If the cell is 1:** Write 0, Move left, Transition to state q_2.
  - **If the cell is _:** Write 0, Move left, Transition to state q_3.
- q_2:
  - **If the cell is 0:** Write _, Move left, Transition to state q_2.
  - **If the cell is 1:** Write _, Move left, Transition to state q_2.
  - **If the cell is _:** Write _, Move left, Transition to state q_3.
- q_3: The state is the halting state, so no actions will be taken.

It is important to note that instead of erasing the cells, we can also design Turing machines with accepting and rejecting states. In this case, we will use these states:

- q_a (Accepting):
  - **If the cell is 0:** Write 0, Move right, Transition to state q_a.
  - **If the cell is 1:** Write 1, Move right, Transition to state q_r.
  - **If the cell is _:** Halt in this state.
- q_r (Rejecting):
  - **If the cell is 0:** Write 0, Move right, Transition to state q_a.
  - **If the cell is 1:** Write 1, Move right, Transition to state q_r.
  - **If the cell is _:** Halt in this state.

I will call this Turing machine a Chomskyan Turing machine, because it seems to have been pioneered by Chomsky.

Now, we have described the device that can be used to answer the Entscheidungsproblem. However, before revealing how Turing (1936) developed his answer, we will first explore how Turing machines are used in cognitive science. This is because Turing's proof is actually extremely difficult to follow, and in my process of developing this book, I find that the best way is to explain the proof is to replace Turing machines with modern computers and software.

Functionalism: None of This Is Real

At first glance, the description of a Turing machine implies a mechanical device. However, the machine in this case is simply an analogy. What we are trying to do here is to determine if an action taken by an entity is computational or not. The action is computational if we can re-describe the action as operations of a Turing machine. For instance, the process to determine whether a binary number is even or not is computational because we can construct a Turing machine for it. Therefore, a Turing machine is used to make a functionalist argument: *something is computational because it can be seen as* ***functioning*** *like a Turing machine*.

This brings us to another topic in cognitive science: **functionalism**. Functionalism, at its heart, is a way to describe a cognitive phenomenon by how it functions and its states, rather than by the architecture (Shagrir, 2010, Chapter 9; Levin, 2023). For instance, Putnam describes pain not as something that we feel. Instead, pain is a function that causes us to take protective and evasive action (Shagrir, 2010). If a person hits their knee against an object, like a counter, they will be in a mental state associated with pain. Then, their pain will influence them to take care of themselves. They may stop their activities and take a moment to sit down and examine their knee.

A key feature of functionalism is that it is "hardware-independent." It does not matter what the underlying architecture is. As Putnam puts it: "we could be made of Swiss cheese and it wouldn't matter" (Shagrir, 2010). Therefore, while the human brain is vastly different from a computer in terms of hardware and physical composition, from the functionalist perspective, this does not matter at all. Computational functionalism takes this even further. It argues that the mind is simply computation. It argues that the mind itself can be modelled as a Turing machine with a state of mind being a state of a Turing machine (Shagrir, 2010).

Functionalism plays an important role in linguistics and computational linguistics. In these fields, we often deal with concepts that are not really tangible and may not have any basis in biology or neuroscience (Shagrir, 2010). A notable example is Chomsky's conception of syntax (Shagrir, 2010) which argues that we construct and recognize valid sentences through algorithmic means. As his argument is functionalist in nature, he is not making any claim about how the human brain works. And chances are, if we are to look into someone's brain and their brain waves to find a Chomskyan grammar, we may not find anything that resembles it.

It is important to note that Chomsky is not the first person who treat a language in an algorithmic fashion. The honour may go to Pāṇini, an ancient Indian scholar who aimed to treat Sanskrit as an algorithm (Namboodiri, 2017). Both Pāṇini, and Chomsky will be discussed later.

Some of you may have realized something at this point: what about the quale and the subjectivity mentioned in an earlier chapter? Indeed, functionalism does not really discuss anything about this. Functionalism does not attempt to exorcise any ghost from the machine. Rather, it just ignores it, and in my opinion, I do not think that ignoring Descartes will just make him go away. Shagrir (2010) also outlines other arguments against functionalism, which we will not examine here. This is because we now know pretty well that while functionalism is useful for creating hardware-independent explanations, it will not help us to achieve Strong AI.

What Is an Algorithm?

In the early days of computer science, an algorithm was simply (Eberbach, Goldin & Wegner, 2004):

> a systematic procedure that produces in a finite number of steps – the answer to a question or the solution to a problem.

This notion is equivalent to: "Something is an algorithm if we can build a Turing machine for it."

However, today's "algorithms" may not follow the definitions above. In fact, some of the modern algorithms seem to defy the original definition. An example of this is the randomized quick sort, which involves picking random pivots. This means the steps taken by a randomized quick sort algorithm can vary between each run. If we pick good pivots, the algorithm is efficient ($O(n \log n)$ on a Random-Access Machine). However, bad pivots will result in a slow run time ($O(n^2)$ on a Random-Access Machine). The original Turing machine does not really take randomness into account. Therefore, it is slightly unclear how to incorporate randomness. We can side-step this by arguing that modern-day computers, except quantum ones, are incapable of true randomness. Therefore, a "random" number here is simply a number generated using deterministic steps with a seed number. Therefore, if we are to model a randomized quick sort algorithm as a Turing machine, the tape for the machine will not only accept an array, but also a seed to generate a pseudo-random number. By the end of the run, the machine will output an arranged array with the seed number erased (i.e. replaced with the blank symbols).

However, there are some algorithms that are harder to (or impossible to) model as Turing machines. An example of this is YouTube's video recommendation "algorithm," which violates the definitions in many ways:

- **Arbitrary:** An algorithm should be knowable and predictable. This means that what the algorithm outputs should be predictable. While many algorithms can be complicated, once we have a reasonable system for guessing the output, the system should always remain reliable. If we design an algorithm to sort an array in ascending order, then we do not expect the algorithm to sort items in descending order in the future. However, YouTube's video recommendation "algorithm" can change over time, rendering prediction tactics less effective (Bishop, 2020).
- **Interactive:** A Turing machine cannot be tampered with once it is running, and an algorithm should not be malleable to outside influences. However, YouTube's video recommendation "algorithm" seems responsive to the whim of outside influence. Bishop (2020) says that some YouTube producers whose videos used to get many views see their number of views dwindling. Andres, Rossi & Tremblay (2023) attribute this to the fact that some advertisers found their products and services being advertised on YouTube videos with questionable and inappropriate content, which led them to pull their advertisements off YouTube. To convince the advertisers to stay on the platform, YouTube revised its moderation policies. This means YouTube's video recommendation "algorithm" is interactive, something that a Turing machine cannot be.

As Eberbach, Goldin & Wegner (2004) point out, the definition of algorithms has changed because a modern-day computer is actually more powerful than a Turing machine.

Alternative Models of Computation

Turing machines are useful for making theoretical arguments in computer science, but they are hardly practical as real-world physical devices. Although we argue that many computers and their software can be reduced to Turing machines, the process of doing so is not straightforward. The truth is that, apart from a few examples, such as https://turingmachine.io/, converting a computer or software is rarely done. Furthermore, some theoretical arguments do not make use of a normal Turing machine. Instead, they use variations of Turing machines, such as the multi-tape Turing machine (Cook & Reckhow, 1972), which can read and write from multiple tapes at the same time (Fischer, 2003). Although some of these Turing machines are, at first glance, more powerful than normal Turing machines. They are not, because we can design normal Turing machines to simulate them. A normal Turing machine will perform more slowly than the Turing machine variants. However, what these variants can do, so can normal Turing machines if given enough time (Fischer, 2003).

For most of the time, most computer science students will find themselves mastering two other models of computation: von Neumann Architecture and Random-Access Machines. These models are much more practical than Turing machines, so they are often taught in computer science classes – either explicitly or implicitly. Less often, students may also see λ-calculus being taught as well.

Many computer science students may also be familiar with certain types of formal automata (e.g., finite state automata). However, we can consider these to be restricted Turing machines. For instance, a finite state automaton is simply a Turing machine with accepting and rejecting states that is not allowed to alter the tape and can only move right. The automata are very important to computational linguistics, so these will be discussed in the next few chapters.

Von Neumann Architecture: The Modern Computer

Von Neumann Architecture forms the foundation of all modern computing devices (Arikpo, Ogban & Eteng, 2007). Chances are, you are reading this page from a von Neumann machine, and you may have multiple von Neumann machines. Some of them may even be inside your pocket or purse at this moment. Based on Arikpo, Ogban & Eteng (2007), a von Neumann-based machine is different from a Turing machine in the following ways:

- Unlike a Turing machine, where the input and the output are stored on the same tape, a von Neumann machine has a separate memory unit. Today, the memory unit is implemented as a random-access memory and a hard drive.
- A von Neumann machine has a separate input and output (I/O) unit to accept inputs and display outputs. The inputs and outputs may still be tape-like (e.g., punch cards). Today, the inputs are generated from devices like keyboards, mice, touch screens, and more. The output is mostly displayed through screens, but occasionally on devices such as printers as well.

- Each memory cell stores more than one symbol at a time, and can be directly accessed using its address. This kind of memory is called random access, because the memory unit can be accessed in an arbitrary manner. A Turing machine memory, on the other hand, can only be reached when the tape head slowly makes its way to the unit.
- A central processing unit (CPU) which executes lines of code fed into it and has the ability to perform arithmetic and logical operations out of the box.

However, despite their differences, both types of machines still share something in common:

- A von Neumann machine program can be stored inside the memory itself and executed. This eliminates the need for creating specialized machines for specific purposes. A specific type of Turing machine, called a **universal Turing machine**, can actually perform something like this. It can accept another Turing machine encoded as a tape and simulate it. In fact, in the first paper ever to describe Turing machines (Turing, 1936), the ability for a Turing machine to simulate other Turing machines plays a key role in answering the Entscheidungsproblem.
- Both models are not able to deal with real numbers (Arikpo, Ogban & Eteng, 2007). While modern computers, which are based on the von Neumann architecture, can seemingly deal with real numbers, approximation is used instead.

> Many modern-day von Neumann machines are interactive. For instance, we can use mice, keyboards, touch screens, etc., to interactively affect a von Neumann machine. However, interaction is not a requirement for a von Neumann machine.

Random-Access Machine: For Big-O Analysis

The von Neumann architecture is useful for designing modern computers. However, in some cases, it may still be difficult to reason. For instance, if we wish to theoretically and abstractly discuss the time complexity (i.e. speed) of a program, we only need to care about how the machine's circuit is being activated. As such, we need a computational model that is abstract enough so that we do not need to discuss everything about program execution, but still concrete enough that we can imagine our arguments apply to general day-to-day computation. There is a model that can hit the sweet spot: a random-access machine.

From my literature review, I believe that this model was first introduced by Cook & Reckhow (1972) as an alternative to Turing machines for complexity analysis. They argue that Turing machines, and their multitape variants, can be difficult to work with. According to Morin (2013, Chapter 1.4), a random-access machine has the following properties:

- Each memory cell can contain multiple symbols and can be accessed, read, and written in $O(1)$ step by its address.
- Each memory address takes a single memory unit.
- A memory unit has a fixed size. On a 64-bit computer, the size is fixed to 64 symbols.
- Each arithmetic and Boolean operation takes $O(1)$ to complete

This model is similar to a von Neumann machine, but it is highly simplified to make it more convenient for us to make arguments about time and space complexity (Bilardi, Ekanadham & Pattnaik, 2009). For many computer science students, this model may be introduced quite early in their program. When I was a student at the University of Toronto, this model was actually introduced in my first term. However, the model was not explicitly named.

λ-Calculus

λ (Lambda)-calculus is a model of computation first developed in 1920's by Alonzo Church, Turing's PhD supervisor. Initially, Church developed the system to represent mathematical functions and their operations in the most fundamental way possible (Cardone & Hindley, 2006, Chapter 4; Barendregt, Dekkers & Statman, 2013, Chapter Introduction). Although Church did not set out to formalize the notion of computation like Turing, he later argued in the 1930s that λ-calculus can represent real-world computation (Rowland, n.d.). This implies Turing machines can represent computation performed for λ -calculus.

Just like Turing machines, λ-calculus is useful for making theoretical arguments. However, its usefulness lies with the analysis of programming languages because λ-calculus has the simplest syntax possible. As it turns out, all programming languages are reducible to λ-calculus (Cardone & Hindley, 2006, Chapter 6; Barendregt, Dekkers & Statman, 2013, Chapter Introduction). While programming with pure λ-calculus is impractical, it inspires many of the functional programming languages. We will explore λ-calculus and the Church-Turing Thesis in the next chapter.

String Manipulation as Cognition

Some readers may find the title "Cognitive Science as String Manipulation" to be an odd title. However, after having described the Turing machine, I can now finally reveal my reasoning. This title is actually based on how Turing machines convert an array of characters, i.e. a string, into another one. Unlike Church, Turing's PhD advisor, who viewed computation as the conversion of natural numbers, Turing treated computation as an act of string conversion (Boker & Dershowitz, 2008). He also further argued that mechanical computers can be used to implement artificial intelligence (Turing, 1950).

It is important to note that string manipulation and manipulation of natural numbers are actually not that different from each other. Kegler (2008) points out that code is essentially a gigantic natural number. After all, all characters that we can see on our screen can be converted into natural numbers using a conversion code like ASCII.

References

Andres, R., Rossi, M., Tremblay, M. (2023, December 3). YouTube 'Adpocalypse': The Youtubers' journey from ad-based to patron-based revenues. *ZEW Discussion Papers* (23-059). ZEW – Leibniz-Zentrum für Europäische Wirtschaftsforschung. https://www.econstor.eu/handle/10419/283579

Arikpo, I. I., Ogban, F. U. & Eteng, I. E. (2007). Von Neumann Architecture and Modern Computers. *Global Journal of Mathematical Sciences* 6(2), 97-103.

Barendregt, H., Dekkers, W. & Statman, R. (2013). *Lambda Calculus with Types*. Cambridge University Press.

Bishop, S. (2020). Algorithmic Experts: Selling Algorithmic Lore on YouTube. *Social Media + Society* 6(1). https://doi.org/10.1177/2056305119897323

Bilardi, G., Ekanadham, K. & Pattnaik, P. (2009). On approximating the ideal random access machine by physical machines. *Journal of the ACM* 56(27). https://doi.org/10.1145/1552285.1552288

Boker, U. & Dershowitz, N. (2008). The Church-Turing Thesis over Arbitrary Domains. In A. Avron, N. Dershowitz & A. Rabinovich (Eds.), *Pillars of*

Computer Science. Lecture Notes in Computer Science 4800. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-78127-1_12

Cardone, F. & Hindley, J. R. (2006). History of Lambda-calculus and Combinatory Logic. https://www.labouseur.com/courses/tpl/History-of-Lambda-Calculus.pdf. NOTE: Preprint, to be published by Elsevier.

Cook, S. A. & Reckhow, R. A. (1972). Time-bounded random access machines. *STOC '72: Proceedings of the fourth annual ACM Symposium on Theory of Computing*, 73-80. https://doi.org/10.1145/800152.804898

Eberbach, E., Goldin, D. & Wegner, P. (2004). Turing's Ideas and Models of Computation. In C. Teuscher (Ed.), *Alan Turing: Life and Legacy of a Great Thinker*. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-05642-4_7

Fischer, P. C. (2003). Turing machine. In A. Ralston, E. D. Reilly & D. Hemmendinger (Eds.), *Encyclopedia of Computer Science*, 1797-1801. John Wiley and Sons Ltd., West Sussex, United Kingdom. https://dl.acm.org/doi/abs/10.5555/1074100.1074881

Goldin, D. Q., Smolka, S. A., Attie, P. C., Sonderregger, E. L. (2004). Turing machines, transition systems, and interaction. *Information and Computation* 194(2). https://doi.org/10.1016/j.ic.2004.07.002

Kegler, J. (2008). Perl and undecidability: Rice's theorem. *The Perl Review 4*, 23-29.

Levin, J. (2023). Functionalism. In E. N. Zalta, & U. Nodelman (eds.), *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/entries/functionalism/

Light, J. S. (1999). When Computers Were Women. *Technology and Culture* 40(3), 455–83. http://www.jstor.org/stable/25147356

Morin, P. (2013). *Open Data Structure: An Introduction.* AU Press, Edmonton, Canada.

Namboodiri, E. V. N. (2017). Panini's Conception of "Syntactic Structures". *Interdisciplinary Journal of Linguistics* 10(2017), 1-16.

Rowland, T. (n.d.). Church-Turing Thesis. In E. W. Weisstein (Ed.), *MathWorld – A Wolfram Resource.* https://mathworld.wolfram.com/Church-TuringThesis.html

Shagrir, O. (2010). *Hilary Putnam* (Y. Ben-Menahem, Ed.). Cambridge University Press.

Turing, A. M. (1936). On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society* s2-42(1), 230-265. https://doi.org/10.1112/plms/s2-42.1.230

Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind* LIX(236), 433–460. https://doi.org/10.1093/mind/LIX.236.433

Watson, R. (2002). Where My Heart Will Take Me (Theme From "Enterprise") [Song]. On *The Voice- Encore*. Decca Music Group Limited.

5.

Church-Turing Thesis and Multiple Realizability

Yeah, you know, you really should have stolen the whole book because the warnings… the warnings come after the spells.
Stephen Strange, a character from *Doctor Strange* (Derrickson, 2016)

A key idea in computer science is that software can be converted into hardware and vice versa. This idea was stated by Turing in 1948 (Copeland, 2024):

> The importance of the universal machine is clear. We do not need to have an infinity of different machines doing different jobs. A single one will suffice. The engineering problem of producing various machines for various jobs is replaced by the office work of "programming" the universal machine to do these jobs.

Today, we often take this idea for granted, and we may not realize how much we depend on it. A prime example of this is the idea of video game emulation, where past video game consoles are virtualized on modern hardware, allowing players to still enjoy games from the past even though the original hardware is no longer being manufactured. Another example is that a computer can run multiple applications at the same time. At the more fundamental level, the ability to treat hardware like software allows us to be productive with a single computer instead of multiple. The world would be a very bad place if we were forced to create programs by cobbling up multiple physical computers together. For instance, instead of typing: 1 + 305 x 50 into a computer to get 15251 on a computer screen, we would be forced to set up two physical computers that perform the multiplication and the addition operation. Setting up multiple computers that perform very simple tasks would get cumbersome very quickly. Therefore, just as Turing had implied, it would be easier to have a single physical computer that could be made to perform the tasks of multiple simple computers.

Early cognitive scientists took this idea even further with the concept of Multiple Realizability. Some, like Putnam, argued that not only could we simulate computers, we could even simulate human intelligence itself (Eliasmith, 2002). Although modern computers are usually realized using electrical circuits, they could also be realized using other means. For instance, if we can devise wetware and biological means to do what a Turing machine performs, the wetware is also a computer. Humans, acting out the inner workings of a computer, are also a computers themself.

However, this powerful idea has some caveats which we should be aware of. Without being aware of the caveats, we could easily make some mistakes. For instance, based on this video narrated by Jago (2014), I used to believe that all modern computers could be converted into Turing machines. However, modern computers are actually much more powerful than what Turing may have envisioned. For instance, the actions made by keyboards, mice, touchscreens, etc., could not be translated into Turing machine actions. After all, this powerful idea was based on the Church-Turing Thesis. In short, the Thesis argues that we can convert λ-calculus into Turing machines and vice versa (Copeland, 2024). Nothing more, and nothing less. However, as Goldin and Wegner (2005) argue, the Thesis has been misused and misconstrued by some computer scientists who proclaim that all computers, even ones with mice and keyboards, can be converted to Turing machines. The mistake also leads to the idea of Multiple Realizability (Eliasmith, 2002).

Thus, I lead the chapter with the quote from Dr. Strange. He chastised several characters for stealing a page of a spell book. Since the characters did not have a complete understanding of the spell, they were not aware of the negative consequences of using it. This is akin to the Church-Turing Thesis. The powerful Thesis is often invoked, but rarely studied in full, which leads to misinterpretation. In a coincidence, Dr. Strange was portrayed by Benedict Cumberbatch, who also portrayed Alan Turing in “The Imitation Game.” Turing was one of the originators of the Church-Turing Thesis.

Church-Turing Thesis

> You cannot simply read a section and conclude that this powerful thesis can be applied in all contexts. Please also read the sections that come after as well! Just like in Doctor Strange, the warnings come after the spell!

The Church-Turing Thesis is one of the most important concepts in computer science. However, unlike many concepts in computer science that are well-defined and fully formal, the Thesis has multiple definitions. In the previous section, I simply and quickly defined the Church-Turing Thesis as that λ -calculus being convertible to Turing machines and vice versa. But this notion is too informal. The fuller and more formal version is as follows (Duží, 2013):

> A function of positive integers is effectively calculable if and only if it is general recursive or λ-definable or computable by a Turing machine.

Some readers may notice the phrase “positive integers.” They may also realize that modern computers do not simply deal with positive integers. They also deal with other types of data, such as strings. Therefore, this domain seems to be too limiting to be applicable to modern computers and software. It is important to note that this version of the Thesis was first formulated by Church (Boker & Dershowitz, 2008), who was more interested in tackling the Entscheidungsproblem. His Thesis was meant to be as tight as possible. Turing himself *may*[1] have thought that the domain could be relaxed to include any string data (Boker & Dershowitz, 2008), but Boker & Dershowitz (2008) later pointed out that wantonly relaxing the domain from “positive integers” to any arbitrary domain could introduce various problems when invoking the Thesis. This requires us to restrict the domain to a certain degree (Boker & Dershowitz, 2008). For instance, if we are making arguments about the same class of problems using Turing machines, then the associated machines must be reading and writing with the same set of symbols.

It is important to note that the Church-Turing Thesis cannot be proven mathematically. The main issue is that, at its heart, the Thesis involves the notion of *effective computation* being arbitrarily defined and agreed upon. Turing defined effective computation as operations on Turing machines, and Church defined it as computation via λ-calculus. We say that Turing machine operations can be mapped onto λ-calculus, because we agree that this can be done (Cleland, 1993). If we disagree that Turing machines and λ-calculus are equivalent, then the Thesis simply does not exist.

At first glance, this thesis does not seem to say anything about software and hardware being convertible into each other. However, a link between λ-calculus and Turing machines can be seen here. The statement implies that we can use λ-calculus to describe an algorithm that computes a function of positive integers, and we can also build a Turing machine that can compute the same thing. If we let λ-calculus represent software, and Turing machines represent hardware, then the statement implies that we can convert one into another.

There is a significant issue, however. First, there seems to be a sleight of hand. How does λ-calculus represent “software” and Turing machines represent “hardware”? As far as I know, no one is really programming in λ-calculus, and Turing machines have some magical properties that our ordinary computers do not possess, such as having an infinite amount of memory. To explain the sleight of hand, we must:

1. As the Church-Turing Thesis was developed semi-independently by multiple people (Copeland, 2024), it would not be possible to know how much collaboration Turing had with Church on the Thesis.

- Find some evidence that modern software can, in a way, be seen as an application of λ-calculus or close to it.
- Find some evidence that modern hardware, represented by the von Neumann architecture, can be presented by Turing machines.

Evidence that modern software can be seen as an application of λ-calculus comes in the form of a programming language called Scheme designed by Sussman & Steele (1998). The language, inspired by LISP, is essentially a programmable version of λ-calculus. Although the language contains some syntactical sugars, the language still allows a program to be written in an almost pure λ-calculus. As Scheme is a high-level programming language, other programming languages could also be converted to Scheme. Thus, all programming languages should be λ-definable.

Cook & Reckhow (1972) showed that modern computers could be reduced to Turing machines. In their paper (Cook & Reckhow, 1972), they proposed an abstract model of computation called Random-Access Machine (RAM) because they realized that Turing machines are not ideal for making arguments about time complexity. This stems from the fact that modern computers are based on the von Neumann architecture, which supports near-instantaneous memory accesses (Arikpo, Ogban, & Eteng, 2007) while Turing machines only allow for one symbol cell to be manipulated at each step. Since RAM represents an idealized and more abstract version of the von Neumann architecture (Bilardi, Ekanadham, & Pattnaik, 2009), if RAM is shown to be reducible to Turing machines, we would have evidence showing that modern computers could be converted to Turing machines. Cook & Reckhow (1972) show that this is possible by outlining the steps to convert a RAM program into a multitape Turing machine in their work. Although a multitape Turing machine is not the same as the normal single-tape one, a multitape Turing machine could be further reduced to one (Fischer, 2003). Therefore, modern computers are like Turing machines – albeit with some caveats, which we will discuss later.

At this point, we have found evidence that software can be rewritten as λ -calculus and hardware can be converted to Turing machines. This means we have shown that modern software can be turned into λ-calculus, and *simple* modern computers can be reduced to Turing machines. However, we still have not demonstrated that modern machines could become modern software and vice versa. Fortunately, Cook & Reckhow (1972) also contained a description of a high-level programming language named RAM-ALGOL. It was derived from ALGOL, an existing programming language, and tailored specifically for writing time-complexity proofs. Cook & Reckhow (1972) showed that RAM-ALGOL could be turned into machine instructions for RAM. This means, a RAM-ALGOL program could be mapped directly onto a Turing machine. Since RAM-ALGOL is a high-level programming language, it can be rewritten in other high-level languages, including Scheme, thus providing the link between software and hardware.

There is also other evidence which supports the Church-Turing Thesis. Furthermore, while this book frames λ-calculus as “software” and Turing machines as “hardware”, other sources may frame these concepts differently. For instance, Barendregt, Dekkers & Statman (2013, Chapter Introduction) frame λ-calculus as computation by hand, and Turing machines as mechanical and automated computation. They also deem λ-calculus as an abstract representation of functional programming languages, and Turing machines as the ancestor of imperative programming languages. Each source has its own dichotomy. The reader is encouraged to consult other literature on this topic.

Application and Limitation of Church-Turing Thesis

Can all computers be converted to Turing machines? The evidence above may suggest that they can be. Some videos (e.g., Jago, 2024; Veritasium, 2021) may imply that any modern computer can be turned into a Turing machine. But this is not so simple. As a researcher in human-computer interaction, I do care immensely about how data is being input into a computer and output through its screen. However, if we were to look at a Turing machine closely, we know that the machine only works with tapes. Many computers today do not work with tape. Rather, they work with input devices such as mice, keyboards, and monitor screens. These input and output devices introduce something called **interaction,** which is not supported by Turing machines. Instead, they may be supported by another type of machine called ***choice* machines**. When Turing (1936) was describing Turing machines, he described them as automatic machines – once run, they do not stop and cannot be interfered with. At the same time, he also described choice machines which support external manipulation (Turing, 1936; Goldin, 2000). However, Turing did not really describe these machines in detail because he was more focused on the Entscheidungsproblem, and the automatic machines are more than sufficient.

Since modern computers support interactions, they are not readily reducible to Turing machines. Therefore, they engage in something called **hypercomputation**. They can perform tasks that are more powerful than computation (Davis, 2006).

Identifying if something is hypercomputation is not necessarily a straightforward task. For instance, if we use a mouse to draw something on a computer, the act of drawing does not necessarily mean that there is hypercomputation. For instance, in Fig. 2, on mdbg.net, I was drawing Chinese character strokes into the box, and the website was trying to identify the character that I was trying to write. After each stroke was complete, a list of candidate characters would appear on the right-hand side. The list would be updated after each stroke.

Figure 2. *Drawing of 我 on mdbg.net.*

In this example, it would be easy to imagine a Turing machine that would perform these tasks. In this case, we can imagine a Turing machine $\mathcal{M}$ that accepts a tape $\mathcal{T}$ which represents the drawing. To convert a drawing into $\mathcal{T}$, the drawing area would be serialized (i.e. stretched) into a 1D linear data structure. Each item in the data structure contains the tuples that follow this format:

- **First item:** x-position of a pixel of the stroke.
- **Second item:** y-position of a pixel of the stroke.

The data structure can then be encoded onto $\mathcal{T}$ with a format like this. Each <x , y> represents a pixel in the input pad that has been filled:

…	<	1	0	,	1	,	1	>	…

Then, when the tape is fed into the machine, the machine slowly changes the tape into a list of candidate Chinese characters. Eventually, the tape becomes:

我	岚	俄	杂	菊	良	岡	哦	…

It is important to note that the output tapes do not necessarily have to come out the same. A slight variation in the strokes is enough to alter the output list.

However, the first few candidates should still match the drawing the best. The input and the output here do not resemble anything like those in Fig. 2. This is fine, because Turing machines are never truly about providing good human-friendly representations.

The Turing machine that represents the handwriting recognition algorithm slowly converts the list of tuples into a list of Chinese characters. While the real-world algorithm on the website may work almost instantaneously, as a simple tape/reader writer, its Turing machine counterpart is slow. However, speed and practicality are not important factors for a Turing machine (Mauro, 2018). If it will take a thousand years for the Turing machine to convert the tuples into Chinese characters, so be it.

Now, one may argue that the algorithm behind Fig. 2 must use machine learning. This means we cannot program this by hand. And since we cannot program the algorithm by hand, they cannot be Turing machines. However, as it turns out, we can still use Turing machines to represent the machine learning process itself, as I argued in Hu (2025). There, I argue that machine learning software is simply a Turing machine that takes another Turing machine and training data encoded onto a tape, and uses the data to modify the standard description of the machine on the tape itself. The modified machine is the algorithm. Is turning the whole machine learning process into Turing machines practical? No. In fact, it may take thousands of years to write, another thousand years to train, and more years to classify the writing.

So if handwriting recognition can still be represented by Turing machines, albeit very impractically, what cannot be? As it turns out, there are simple things that we take for granted that cannot be represented by Turing machines. The first example is a video game that requires user input. It can be something as simple as a 2D platformer. In this case, the game simply waits for the user to make an action. Because the user must act in order for the game to progress, the game cannot be represented as a Turing machine, as these machines do not allow for outside intervention. Another example is an operating system that is capable of downloading a new driver and quietly updating itself with the driver. This action is forbidden under the Turing machine paradigm because a Turing machine cannot self-modify (Hu, 2025).

Modern computers do stretch the Church-Turing Thesis quite thin. However, does this mean we can no longer convert them into software? Actually, no. We can and often do convert computers into software. An example of this is a video game console emulator. The video game console may no longer be manufactured. However, we can still write software to mimic the original console quite well. Any current von Neumann computer, in fact, can be emulated as well – even if it is a choice machine.

Multiple Realizability

We can take the idea of the Church-Turing Thesis even further with the theory of Multiple Realizability. For some cognitive scientists who are computational functionalists, cognition is the same as computation (Eliasmith, 2003; Bringsjord & Arkoudas, 2006). Therefore, cognition can be represented by Turing machines. Therefore, cognition can also be represented by λ-calculus. Ergo, cognition is abstract and can be implemented through many means without any concern for architectures (Eliasmith, 2003). Under this premise, it is assumed that we can create computer-based Strong AI just by creating good algorithms. We also do not need to take the architecture into account. Intelligence here is the same regardless of whether it is wetware-based or hardware-based.

Even though past cognitive scientists like Putnam are optimistic about Multiple Realizability. Modern cognitive science does not share the same enthusiasm. First of all, although Multiple Realizability is an important concept in cognitive science, Eliasmith (2003) argues that it is actually based on a flawed interpretation of the Church-Turing Thesis. As soon as we try to implement actual computers in the real world, we will have difficulty creating ones that are representative of the mental processes that we want to represent (Eliasmith, 2003). For instance, if we try to create programs for handwriting recognition, we note that there are multiple ways to implement them. A usual way to implement a handwriting recognition program is by using neural networks. However, each programmer will train their neural networks differently with different parameters and data, so the question is: which of the neural networks is the most representative of how a human recognizes handwriting? We cannot answer this without consulting other fields such as neuroscience. We cannot simply pick software or any formal implementation that performs a certain task well and claims to be the explanation of how a human performs that task (Vervaeke, 1997).

The human mind is not purely computational, so Turing machines will prove to be inadequate in some cases (Bringsjord & Arkoudas, 2006; Eliasmith, 2003; Goldin & Wegner, 2000). For instance, Goldin & Wegner (2000) argue that interaction cannot be modelled using Turing machines. An example of interaction is when a human notices and reacts to a stimulus in their peripheral vision. Although we can model what a person can see in a single frame as a Turing machine tape, the act of seeing is equivalent to having Turing machine tapes constantly being edited by the physical world. This does not mean that some aspects of cognition are not computational in nature, however. For instance, if we strictly follow a recipe to create an omelette, we are, in a sense, applying an algorithm that converts eggs, butter, black pepper, and other ingredients into an omelette. Nevertheless, many things that we do, do not have associated algorithms.

Applying Multiple Realizability can also lead to some absurd demonstrations. Let's imagine that we have an algorithm for Strong AI named Galatea and a man named Pygmalion. Galatea exists only on a piece of paper. Based on Multiple Realizability, we argue that Galatea can come to life, so to speak, if Pygmalion faithfully follows the Galatea procedure on the paper. However, if Pygmalion faithfully acts out the steps, he will soon find that his Galatea is not quite like a normal human being. For example, while she can reach the state of hunger, she can never truly feel the hunger, nor die from starvation. However, for real biological beings, the act of consumption is very important. After all, it is a key to survival.

Should Multiple Realizability be disregarded in its entirety? Mallat & Feinberg (2021) disagree. They argue that some arguments made by Multiple Realizability are correct. For instance, cognition could be described in terms of functions. For instance, our ability to see, hear, smell, reason and more can be seen as different functions. However, they also argue that for Multiple Realizability to work, it must be constrained, and it must respect biology. In essence, we do not get to choose how we simulate cognition. Rather, the organization of the brain, evolution, and more constrain how we can treat the mind like a program (Mallat & Feinberg, 2021).

References

Arikpo, I. I., Ogban, F. U. & Eteng, I. E. (2007). Von Neumann Architecture and Modern Computers. *Global Journal of Mathematical Sciences* 6(2), 97-103.

Barendregt, H., Dekkers, W. & Statman, R. (2013). *Lambda Calculus with Types.* Cambridge University Press.

Bilardi, G., Ekanadham, K. & Pattnaik, P. (2009). On approximating the ideal random access machine by physical machines. *Journal of the ACM* 56(27). https://doi.org/10.1145/1552285.1552288

Boker, U. & Dershowitz, N. (2008). The Church-Turing Thesis over Arbitrary Domains. In A. Avron, N. Dershowitz & A. Rabinovich (Eds.), *Pillars of Computer Science. Lecture Notes in Computer Science* 4800. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-78127-1_12

Bringsjord, S. & Arkoudas, K. (2006). On the Provability, Veracity, and AI-Relevance of the Church–Turing Thesis. In A. Olszewski, J. Wolenski & R. Janusz (Eds.), *Church's Thesis After 70 Years*, 66-118. De Gruyter. https://doi.org/10.1515/9783110325461.66

Cleland, C. E. (1993). Is the Church-Turing thesis true? *Minds and Machines* 3, 283 – 312. https://doi.org/10.1007/BF00976283

Copeland, B. J. (2024). Church-Turing Thesis. In E. N. Zalta & U. Nodelman (Eds.), *The Stanford Encyclopedia of Philosophy (Winter 2024 Edition).*

Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/archives/win2024/entries/church-turing/

Davis, M. (2006). The Church-Turing Thesis: Consensus and Opposition. In A. Beckman, U. Berger, B. Löwe, J. V. Tucker (Eds.), *Lecture Notes in Computer Science* 3988, 125-132. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11780342_13

Derrickson, S. (Director). (2016). *Doctor Strange* [Film]. Marvel Studios.

Duží, M. (2013). A procedural interpretation of the Church-Turing Thesis. *Church's Thesis: Logic, mind and nature*. Copernicus Center Press, Krakow.

Eliasmith, C. (2003). The myth of the Turing machine: the failings of functionalism and related theses. *Journal of experimental & theoretical artificial intelligence* 14(1). https://doi.org/10.1080/09528130210153514

Fischer, P. C. (2003). Turing machine. In A. Ralston, E. D. Reilly & D. Hemmendinger (Eds.), *Encyclopedia of Computer Science*, 1797-1801. John Wiley and Sons Ltd., West Sussex, United Kingdom. https://dl.acm.org/doi/abs/10.5555/1074100.1074881

Goldin, D. Q. (2000, February). Persistent Turing machines as a model of interactive computation. *International Symposium on Foundations of Information and Knowledge Systems*, 116-135. Springer. https://doi.org/10.1007/3-540-46564-2_8

Goldin, D. Q., & Wegner, P. (2005, June). The Church-Turing Thesis: Breaking the Myth. *Conference on Computability in Europe* 3526, 152-168. Springer. https://doi.org/10.1007/11494645_20

Hu, S. (2025). Chatbots as Turing Machine. *32nd IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*.

Jago, M. (2014, August 29). *Turing Machines Explained – Computerphile* [Video]. YouTube. https://www.youtube.com/watch?v=dNRDvLACg5Q

Mallat, J. & Feinberg, T. E. (2021, September 23). Multiple Routes to Animal Consciousness: Constrained Multiple Realizability Rather Than Modest Identity Theory. In P. B. Graben (Ed.), *Frontiers in Psychology* 12 – 2021. https://doi.org/10.3389/fpsyg.2021.732336

Mauro, L. S. (2018, October 25). Church-Turing Thesis, in Practice. In: M. Piazza, & G.Pulcini (Eds.), *Truth, Existence and Explanation* 334. Springer, Cham. https://doi.org/10.1007/978-3-319-93342-9_13

Metropolis, N. (1992). The Age of Computing: A Personal Memoir. *Daedalus* 121(1), 119-130. https://www.jstor.org/stable/20025423

Peterson, C. & Lepage, F. (2012). Cleland on Church's Thesis and the Limits of Computation. *Philosophia Scientiæ: Travaux d'histoire et de philosophie des sciences* 16-3, 69-85. Éditions Kimé. https://doi.org/10.4000/philosophiascientiae.772

Sussman, G. J. & Steele Jr., G. L. (1998). Scheme: An Interpreter for Extended

Lambda Calculus. *Higher-Order and Symbolic Computation* 11, 405-439. https://doi.org/10.1023/A:1010035624696
Turing, A. M. (1936). On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society* s2-42(1), 230-265. https://doi.org/10.1112/plms/s2-42.1.230
Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind* LIX(236), 433–460. https://doi.org/10.1093/mind/LIX.236.433
Veritasium. (2021). *Math's Fundamental Flaw* [Video]. YouTube. https://www.youtube.com/watch?v=HeQX2HjkcNo
Vervaeke, J. A. (1997). The Naturalistic Imperative in Cognitive Science. [Doctoral Dissertation, University of Toronto]. TSpace. http://hdl.handle.net/1807/10696

Media Attributions

- wo_drawing

6.

What Computers Cannot Do

With everything that's happening, the things that are about to come to light, people might just need a little old fashioned.
Phil Coulson, a character from *The Avengers* (Whedon, 2012)

Why did Turing (1936) create the blueprint to modern computer science in the first place? Does this have anything to do with ENIGMA or some other fancy machines later on? As it turns out, he was answering the Entscheidungsproblem. While Turing did deeply care about computing machines, that was not the concern in the 1936 paper. The paper shows that the Entscheidungsproblem is undecidable. However, the paper itself was difficult to understand. Therefore, in the modern day, people simply state that Turing (1936) shows that the Halting problem is undecidable, and because the Halting problem is undecidable, so is the Entscheidungsproblem (Lucas, 2021). The truth is that Turing did not show this. Instead, another computer scientist named Martin Davis was the one who devised the proof (Lucas, 2021).

So what is the Halting Problem? Basically, it asks: Can we create a Turing machine that tells if an arbitrary Turing machine will halt? The answer is: no (Lucas, 2021). The proof, based on Hamkin & Nenu (2026), is as follows:

Theorem 1 [The Halting Problem is undecidable]. Assume that the problem is decidable, then there exists a Turing machine H that takes in a standard description of another machine M, and its input i. H can determine if M will halt with i. Now, we construct another machine named I. I follows the steps below:

- Let H^* be the simulated version of H.
- Simulate H^* on M with any input j.
- If H^* determines that M halts, I enters into an infinite loop.

- If H^* determines that M does not halt, I then halts.

Now, we try to run T_H again with $M = I$ with any input. In this case, H will face a paradox. If H says I does not halt, then H^* must have halted. Otherwise, if I halts, then H^* determines that I actually does not halt. Thus, $H \neq H^*$. However, as H^* is a simulated version of H, this is a contradiction. Therefore, neither H nor H^* can exist.

This proof works, but in my opinion, being able to halt is also not necessarily an undesirable property of a program or a computer (Hu, 2025). After all, many computers (e.g., servers, remote sensors) today are not designed to halt at all. Therefore, we should care more about what computers will output. For instance, we care whether a computer or software outputs swear words to users. As it turns out, Turing (1936) has already shown that this is undecidable (Lucas, 2021). However, his proof is extremely difficult to understand, with large sections of the paper dedicated to the descriptions of Turing machines and universal Turing machines. Fortunately, thanks to developments in computer science, I can provide a simpler and intuitive explanation of Turing's proof. It is important to note that the proofs in this chapter differ greatly from the proofs that one can find in more theoretical computer science textbooks. This is because my goal is to make the proofs as easy to understand as possible while maintaining some semblance of technicality, and to stay as close to Turing's work as possible. Meanwhile, those proofs rely on a specific type of Turing machine that I dub the Chomskyan Turing machine. Unlike the original Turing machine, which answers by writing out onto the tape, a Chomskyan Turing machine can only answer "yes" or "no." It does not write the answer onto the tape. Instead, it enters an accepting or a rejecting state and then halts. Since these Turing machines can only provide binary answers, they restrict how proofs can be written.

Turing's (1936) proof has three parts, which we will discuss:

- Determining if a computer is circle-free is undecidable.
- Determining if a computer will print a specific symbol is undecidable.
- The Entscheidungsproblem is undecidable.

It is important to note that I will also invoke the Church-Turing Thesis. This means that computers and software are treated as interchangeable. However, there issomething that we need to keep in mind. Turing's (1963) proof relies on

Turing machines, which use tapes that are simultaneously input, memory, and output. This means the results of the proofs here apply to a much further extent than you may imagine. For instance, when Turing (1963) shows that the Printing Problem is undecidable, i.e., a computer cannot decide if another computer will print a certain symbol, his proof applies to all writing actions a computer can perform. These actions include showing a symbol on a monitor screen, writing a symbol onto the random-access memory, writing a symbol to the hard drive, printing the symbol on a piece of paper, and more.

Another thing to keep in mind is that undecidability proofs apply to a class of problems, and not to specific instances of problems (Boker & Dershowitz, 2021). For instance, while the Halting Problem is shown to be undecidable in general, we can still create programs to detect if simple programs will halt or not. To test this out, you can actually input a simple Python program to a generative artificial intelligence (AI) chatbot like ChatGPT or Gemini, and it will be able to tell you that this program will never halt:

```
while True:
   print("AHHHHH!!!!!")
```

Circle-Free Is Undecidable

To prove if a computer is circle-free, first, we must define what circularity is. First, we must imagine that a computer can print two types of symbols: numeric and non-numeric symbols. The numeric symbols represent numbers. In the base-10 system, the symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. However, in other systems, such as hexadecimal, other symbols like A, B, and C can also be numeric symbols. A circular computer will:

- Print a fixed number of numeric symbols (Turing, 1963; Lucas, 2021).
- The numeric symbol can be broken up by non-numeric symbols.
- Once a fixed number of numeric symbols is printed, the computer prints nothing or non-numeric symbols.
- The computer may halt.

Meanwhile, a circle-free computer is the opposite of a circular computer:

- Endlessly print numeric symbols (Turing, 1963; Lucas, 2021).
- The computer never halts.

Knowing whether a computer is circle-free or not can be desirable. For instance, when we create a program (by the Church-Turing Thesis, a program is also a computer) that divides two numbers, we want to know if a program will output a number with a fixed number of decimals, or a number with infinitely long decimal digits (e.g., some rational numbers and all irrational numbers). If the number has infinitely long decimal digits, we will need a way to round up the digits so the results can fit onto a screen.

Turing (1963) argues that this is not decidable, because we can create a circle-free machine that behaves like a circular one by design.

Theorem 2 [Circularity Is Undecidable]. This is a proof by contradiction, so we will first assume that circularity is decidable. Let C be a Turing machine that can decide this problem. Then, we use C to create D, a machine that determines if all Turing machines in the universe are circular or not. D has two tapes: the one for the input, and the one for the output. The input tape contains descriptions of all Turing machines in the universe, while the output tape is used to show the number of steps:

- It accepts a tape encoding of all Turing machines in the universe.
- Clear the output tape of any existing data.
- Creates a counter variable named n.
- For each computer M:
    - Simulates C on M.
    - If C returns "Yes", increments n.
    - Prints n onto the output by appending it to the other printed numbers.

D is a circle-free computer, because it is designed to only print out numbers and infinitely so.

Since D itself is a Turing machine, it follows that at one point, D will be run on itself. When this happens, all the Turing machines before C will be checked. However, when C is reached, C basically restarts, and the earlier computers will be re-checked, and the number will be re-printed. This repeats an infinite number of times, and D will never get to check all computers in the universe. This means the user of D will only get to see numbers: $1, 2, 3, \ldots, n$ on the output tape. For example, assuming that D manages to find five circular and halting computers before itself, the output will be as follows: 1, 2, 3, 4, 5. When it reaches itself, the tape is cleared and D is forced to print 1, 2, 3, 4, 5 again. This process repeats endlessly, and all the user can see on the output tape is 1, 2, 3, 4, 5.

In Turing's (1936) original proof, a single-tape Turing machine was used. However, a 2-tape Turing machine could be simulated using a single-tape one. On a single-tape machine, we can simply put the counter numbers in front of the descriptions of all machines in the universe. When D runs, we still only see 1, 2, 3, 4, 5 plus descriptions of all machines. Based on this pattern, D must be circular, thus a contradiction.

Printing Problem and the Entscheidungsproblem Are Undecidable

The Printing Problem is the problem of whether a computer can predict if another computer will ever produce a symbol either in its memory or in its output. Turing (1936) has shown that this problem is undecidable by arguing that, if we can create an algorithm to solve the Printing Problem, then that same algorithm can be used to determine if a machine is circle-free or not. Since the Circle-Free problem is undecidable, by modus tollens, the algorithm that solves the Printing Problem does not exist either (Pinna & Giunti, 2022). To get from the Printing Problem to the Circle-Free problem, we will first show that the algorithm for the Printing Problem can be extended – e.g., if we can determine whether 0 is printed once, we can determine if 0 is printed an infinite number of times. The lemma is as follows.

Lemma 1 [If a Printing Problem is solvable, then we determine if the symbol is printed for an arbitrary number of times]. Let P_0 be the Turing machine that checks if M, an arbitrary, will print 0 once. Then, we can devise P_0^2 that checks if M will print 0 twice. P_0^2 works as follows:

- Use P_0 on M.
- If 0 is not found, answer "no" and halt.
- If 0 is found on the output tape, then modify M so that where it first prints 0, it prints $\bar{0}$.
- Use P_0 on modified M.

- If 0 is not found, answer "no" and halt.
- If 0 is found on the output tape, answer "Yes" and halt.

Thus, we can use P_0 to create P_0^n which will check for 0 for $n-1$ times. Assume that we want to create P_0^n, we can simply use P_0^{n-1} in this way:

- Simulate M on input i using P_0^{n-1}.
- If 0 is not found, answer "no" and halt.
- If 0 is found on the output tape, then modify M so that where it first prints 0, it prints $\bar{0}$. At this point, M has been modified to print $\bar{0}$ for $n-1$ times.
- Use P_0 on modified M.
- If 0 is not found, answer "no" and halt.
- If 0 is found on the output tape, answer "Yes" and halt.

Thus, if we can solve the Printing Problem for P_0, we can create P_0^n where $k \in \mathbb{N}$.

Now, if we can create P_0^n, we can also create P_1^n – i.e., a Turing machine that determines if 1 will appear n times. Having P_0^n and P_1^n alone is sufficient to create a solver for the Circle-Free problem.

Lemma 2 [Algorithms for solving the Printing Problems can be used to solve the Circle-free Problem]. All numbers can be encoded as binary numbers. For example, 8 can be encoded as 1000. Therefore, we can use T_0^n, T_1^n together to see if an arbitrary Turing machine M is circle-free or not. Basically, if both P_0^n, P_1^n say "yes", then M is circular.

Since the Circle-Free problem is undecidable, the Printing Problem cannot be solved for 0 and 1 either.

Theorem 3 [The Printing Problem is undecidable]. Assume that P_0^n, P_1^n exist. Then, based on **Lemma 2**, the existence of $P(0) \wedge P(1)$ implies that we can create a machine that solves the Circle-free problem. However, there is no Turing machine that can decide the Circle-free problem **(Theorem 2).** By modus tollens, there cannot be P_0^n, P_1^n.

Essentially, we have a chain of implicatures:

$$P_0, P_1 \text{ exists } \to P_0^n, P_1^n \text{ exists} \to \text{ Circle-free has solutions}$$

Since the Circle-free problem is undecidable, all the antecedents in the chain are rendered false.

At this point, the lemmas and proofs are made using only 0 and 1. However, a Turing machine can work with infinitely many symbols. Can we actually generalize the proofs for these symbols to all symbols? Answering this question is actually quite simple. A computer, even a modern and more powerful one, is always dealing with 0 and 1. Regardless of how it is implemented, all symbols are converted into a series of 0's and 1's. Even non-numeric characters have numeric and binary representations. For instance, the binary representation for “a” under the ASCII system is 1100001 (IBM, 2024). Therefore, when asking if a symbol like “a” will ever be printed, we must implicitly invoke P_0^n, P_1^n. Since

be the set of Turing machines representing the programs. Then, we can construct a 2-tape Turing machine named R. The first tape accepts the property that we wish to check for, and the second tape encodes Ms. It answers by replacing each standard description in the second tape with "yes" or "no." If all Turing machines in the second tape possess the property, then all of their descriptions will be replaced with "yes."

We can use R to solve the Printing Problem by setting: (1) the first tape contains the sentence: "The machine will print 0", and (2) the second tape to be a machine description. This means R is now equivalent to a decider of the Printing Problem. However, since the Printing Problem is undecidable, R cannot exist.

Note: It is important to be aware that the original Rice's Theorem is obtained by reducing the Halting Problem to the problem of deciding a non-trivial property (Kozen, 1997, Chapter 34). This involves arguing that the Halting Problem can be reduced to the problem of deciding property. Therefore, the existence of a decider of a non-trivial property implies the existence of a decider of the Halting Problem. The consequent is false, so must the antecedent. As Lucas (2021) points out, while both the Printing Problem and the Halting Problem are both decidable, printing a specific symbol does not correlate to halting. Therefore, I use "Rice's Theorem" in quotes to show that my "Rice's Theorem" is not the same one found in other sources - e.g., Kozen (1997).

An Arbitrary Machine Is Not Every Machine

It is worth repeating again that just because a problem is undecidable, it does not mean that all instances are. Some of the undecidable problems are actually decidable if we restrict the length of the Turing machine tapes. For instance, the Halting Problem becomes decidable if we limit the length of the input. However, these problems can be computationally intractable. The bounded version of the Halting Problem turns out to be decidable (Kingler et al. 2023). We will discuss NP-Complete problems in a later chapter. For now, we just need to know that creating an efficient algorithm for this class of problem may be impossible.

We also notice that the undecidability proofs involve some absurd machines that we will never create in real life. For instance, we will never create machines that try to see if another program will print an infinitely long number. While I argued earlier that such a machine can be useful for distinguishing infinitely long rationalnumbers and rational numbers from other numbers, the use case is extremely limited. After all, all real-world computers have a limited amount of memory, so dealing with an infinitely long number is impossible. Many Turing machines are predictable; we just never know which one at first glance.

Although AI developers have been working hard to improve the correctness of the AI, we must still be careful when we are dealing with undecidability. We cannot be overly trusting of AI. Since AI systems are computers, they are bedevilled by computability issues. As such, no matter how hard they try, their creators cannot ensure perfect correctness. Another cruel part of the computability proofs is that the proofs do not tell us which instances of the problems are undecidable. We must find that out by building the machines ourselves. We must learn the hard way.

References

Boker, U. & Dershowitz, N. (2008). The Church-Turing Thesis over Arbitrary Domains. In A. Avron, N. Dershowitz & A. Rabinovich (Eds.), *Pillars of Computer Science. Lecture Notes in Computer Science* 4800. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-78127-1_12

Hamkins, J. D., & Nenu, T. (2026). Did Turing prove the undecidability of the halting problem?. *Journal of Logic and Computation 36*(1), exaf075.

Hu, S. (2025). Chatbots as Turing Machine. *32nd IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*.

IBM. (2024, August 27). ASCII, decimal, hexadecimal, octal, and binary conversion table. *IBM AIX documentation*. https://www.ibm.com/docs/en/aix/7.1.0?topic=adapters-ascii-decimal-hexadecimal-octal-binary-conversion-table

Kegler, J. (2008). Perl and undecidability: Rice's theorem. *The Perl Review*, *4*, 23-29.

Kingler, A., ver der Eyden, M., Stengele, S., Reinhart, T. & De las Cuevas, G. (2023). Many bounded versions of undecidable problems are NP-hard. *SciPost Physics* 14(173). SciPost Foundation. https://doi.org/10.21468/SciPostPhys.14.6.173

Kozen, D. C. (1997). *Automata and Computability*. Springer Science+Business Media.

Lucas, S. (2021). The origins of the halting problem. *Journal of Logical and Algebraic Methods in Programming* 121(100687). https://doi.org/10.1016/j.jlamp.2021.100687

Pinna, S. & Giunti, M. (2022). Alan Turing and the Cognitive Foundation of the Concept of Algorithm. *Algoritmo* 34(1). https://hdl.handle.net/11584/359159

Turing, A. M. (1936). On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society* s2-42(1), 230-265. https://doi.org/10.1112/plms/s2-42.1.230

Whedon, J. H. (Director). (2012). *The Avengers* [Film]. Marvel Studios.

7.

Language Machines

If you can't give me poetry, can't you give me poetical science?
Ada Lovelace (Ebrahimi, 2014)

Since Turing views computation as the act of string manipulation (Boker & Dershowitz, 2021), it follows that a Turing machine, or a computer, should also be able to deal with human languages to some degree. Chomsky certainly believes so. He also argues that we can also "de-power" Turing machines in various ways in order to better model aspects of human languages (Jäger & Rogers, 2012). For example, if we restrict a Turing machine so that it cannot write and can only move right, then we have a Finite State Automata or FSA, which can model the morphology of English (Kurdi, 2016, Chapter 3). Morphology means changing the meaning of a word by modifying its prefixes and suffixes, or conjugating it (Kurdi, 2016, Chapter 3), e.g., we modify the word "language" to "language**s**" to indicate that we are talking about more than one language. Chomsky is not the first person to note though that there is a link between languages and algorithms. In fact, he says that he is inspired by Pāṇini, an ancient Indian scholar (Namboodiri, 2017).

Pāṇini's Sanskrit Machine and Chomsky's Language Generator

Regardless of what you think about languages, some aspects of languages are clearly algorithmic. If I shuffle the words in this sentence:

A man eats a fish.

We can totally alter the meaning of the sentence:

A fish eats a man.

Or we can get something that can no longer be considered an English sentence:

Eats a a man fish.

In English, if we want to express the idea that "a man eats a fish", the subject, the verb and the object must be placed in order. Furthermore, the nouns and the verbs must be altered to reflect the ideas of number. If we remove "a" in front of "a fish", then the man must have eaten more than one fish. In some languages, the rules are not as strict. And in others, the rules are even stricter.

The question is, how much of a natural language is algorithmic in nature?

Pāṇini, in 350 BC, wrote a manuscript named Aṣṭādhyāyī which outlined all the rules for writing Sanskrit. Although the concept of algorithms, programs, and computers did not exist back then, his manuscript implied that a Sanskrit sentence could be generated in a fully algorithm-like manner (Rajpopat, 2021, Chapter 1). While parts of Aṣṭādhyāyī could be understood, modern scholars struggled with certain rules. This means they could not use Pāṇini's rules to create every single grammatical sentence (Rajpopat, 2021).

Does this mean that all natural languages are never meant to be fully algorithmic, and Pāṇini's attempt to formalize Sanskrit was always meant to be futile?

Eventually, Rajpopat (2021) wrote in his doctoral dissertation that we have been interpreting Pāṇini's rules incorrectly all along. Rajpopat argues that Pāṇini's rules have always been perfect. Now, I would caution against fully accepting Rajpopat's argument at this point due to his work being so novel. After all, a good argument deserves a good scrutiny.

Pāṇini's rules are only meant for Sanskrit, but is it possible to create rules for any arbitrary human language? Also, if we could create rules for all languages, could we also devise a single universal set of rules which we can derive the rules for all languages? Chomsky certainly believes so. Inspired by Pāṇini, Chomsky developed the theories of generative grammar and universal grammar (Namboodiri, 2017):

- **Generative Grammar:** Language is generated by following a finite set of rules, i.e. a grammar. Each grammar can generate an infinite number of sentences (Ramoo, 2021, Chapter 3). In the Chomskyan tradition, we use formal languages to do so. Many computer science students would already have encountered formal languages in a second- or third-year theory classes or a class in designing programming languages.
- **Universal Grammar:** Everyone is born with an innate ability to generate a language, i.e., everyone has a general set of rules to use a language (Dąbrowska, 2015).

It is important to note that these concepts are functionalist in nature. If we can scan someone's brain to find their "generative grammar", chances are, we may not find anything. Dąbrowska (2015) also argues that universal grammar does not have solid evidence.

Language as Computer

> A language is a collection of sentences of finite length all constructed from a finite alphabet (or, where our concern is limited to syntax, a finite vocabulary) of symbols.
> – Chomsky (1959)

If we ask a random person what a language is, we tend to get many answers. So for the sake of convenience, we will start with the most soulless definition possible – the one by Chomsky (1959). Many computer science students will find this definition very familiar, this is because it is also the definition used within the theory of computation class. To better understand this, let's unpack the quote a bit:

- **A language is a collection of sentences… :** This simply means that a language is a set of strings that are finite in length.
- **…sentences… :** This is simply a string created by concatenating multiple symbols together.
- **…all constructed from a finite alphabet… :** The strings in the language must be from the same set of symbols, and the set of symbols must have a finite size.

As the most soulless definition, a language here does not need to be a human language at all. A language can simply be a set of strings that follow a certain rule. For instance, when a website asks us to create a password, it may have some specifications:

- The length of the password must be 6 or more (as long as it is finite).
- The password must contain both upper and lower case letters.
- The password must contain a number.
- The password must contain special characters such as !, @, #, etc.

This is a language. It may not be comprehensible nor poetic, but it is a language – a formal language if you will. A sentence here is simply a valid password that meets the above requirement.

Although Chomsky (1959)'s definition of language has Turing machines in mind, the Turing machines can be difficult to work with. Therefore, we adopt a different notation called the **production rules** (Critchlow & Eck, 2025) to encode a grammar, i.e. rule, of a language. An example of this is the grammar for a formal language that represents an odd binary number:

$S \to A1$
$A \to \epsilon$ where ϵ is an empty string.
$A \to 0$

$$A \rightarrow 1$$
$$A \rightarrow A0$$
$$A \rightarrow A1$$

For the sake of brevity, the A-rules can be rewritten as a single line like this:

$$A \rightarrow \epsilon \mid 0 \mid 1 \mid A0 \mid A1$$

In this notation, capital letters (e.g., S, A) are non-terminal symbols with S being the start symbol. The left side of the $\longrightarrow$ is called the left-hand side, and the right side is called the right-hand side. They must be further expanded or replaced with a terminal symbol. Usually, a terminal symbol is represented by lower-case letters. Since this language only represents numbers, the terminal symbols are simply $\epsilon, 0, 1$. These production rules are saying that:

- An odd number must start with A and end with 1.
- A can be an empty string (i.e. nothing), 0, 1, A0 or A1.
    - If the symbols are A0 or A1, then A must be further expanded, because in a production rule, our goal is to eliminate non-terminal symbols. We can do this by looking at the rules again.

A string is considered a member of the language if it is recognized by the rule, or if it can be generated by applying the rule. For instance, the string: 100 is not in the language, because its last digit is not 1 as dictated by the rules. Meanwhile, the string 10111 is, because it can be generated using the rules:

- The rule $S \longrightarrow A1$ generates a binary number that ends with 1. Since 10111 ends with 1, it can be generated with the number. During the generation process, we have A1 and A must be replaced with terminal symbols.
- The rule $A \longrightarrow \epsilon \mid 0 \mid 1 \mid A0 \mid A1$ basically says that we can make any binary numbers. Therefore, 1011 can be generated by this rule if we apply it recursively.
- Therefore, 10111 can be generated, so it is in the language.

This notation can be modified to represent human languages as well, but we will discuss this later in this chapter when we discuss context-free grammars.

State Machines and Grammars

Chomsky (1959) notices that formal grammars can be represented using different types of computers. Some formal grammars do not need powerful computers. For instance, a computer that recognizes the aforementioned language of odd binary numbers does not need memory. Instead, the computer simply checks if the final digit is 1 or not. If it is, then the number must be odd. In Python, we can write the algorithm like this:

```
def is_odd(num_string):
   return num_string[-1] == 1
```

Based on the Church-Turing Thesis, we can devise a Turing machine to represent this algorithm. However, this Turing machine does not need to write at all, except to indicate if the string is in the language of odd binary numbers. Here, the Turing machine can simply move the final number and check the digit.

Some languages can be more difficult to recognize, though. For instance, if we wish to check if a string belongs to the language of palindromes, we may use a recursive code which requires a stack data structure to memorize some recursion-related data:

```
def is_palindrome(word):
   if len(word) <= 1:
      return True
   else:
      if word[0] != word[-1]:
         return False
      else:
         return is_palindrome(word[1:-2])
```

Although the code can be rewritten to use iteration, units of memory are still necessary. Therefore, if we create a Turing machine to recognize a palindrome, the machine, at some points, must write its work onto the tape. Chomsky (1959) argues that there are four types of computers and four types of formal grammars associated with them (Jäger& Rogers, 2012):

- Turing machines and type-0 grammar
- Linearly-bounded Automata (LBA) and context-sensitive grammar
- Push-down Automata (PDA) and context-free grammar
- Finite State Automata (FSA) and regular language

The LBA, PDA, and FSA are basically restricted Turing machines, i.e. less powerful computers.

Turing Machines and Type-0 Grammar

Turing machines can represent a type-0 grammar. So what is a type-0 grammar, anyway? It is not something that we really need to be concerned about. Human languages are only *mildly* context-sensitive (Jäger& Rogers, 2012). As far as production rules are concerned, a rule can be written as (Su & Yan, 2011, Chapter 2):

$$\alpha \rightarrow \beta$$

where α is any combination of: at least one non-terminal symbol and any number of non-terminal symbols. β can be any combination of terminal symbols, non-terminal symbols and the empty string.

$$aSbcA \rightarrow Sd$$

Since a type-0 grammar is only recognizable by Turing machines, it cannot be represented by an actual computer because our day-to-day computers only have a finite number of cells. This is another reason why we do not really need to think too hard about this type of grammar. However, we should still be cognizant that since a type-0 grammar may require an infinite amount of memory to verify a string, when we ask a representative Turing machine to decide on a string, we may never get any result! After all, an infinite amount of memory requires an infinite amount of time to write!

Linearly-Bounded Automata and Context-Sensitive Grammar

A LBA is a Turing machine with a finite number of cells, with the number of cells being linearly proportional to the size of the input. These types of machines are representatives of context-sensitive grammar or CSG (Jäger& Rogers, 2012). A CSG, also known as a type-1 grammar, is the same as a type-0 grammar. However, the length of the left-hand side must be shorter or equal to the right-hand side (Su & Yan, 2011, Chapter 2). Below is an example rule that is a valid CSG rule:

$$bAc \rightarrow BCDf$$

A valid CSG rule is also a valid type-0 rule. However, it is not necessarily a valid rule for less powerful grammars. It is important to note that human languages can be represented using a CSG (Jäger& Rogers, 2012). However, a CSG is usually too powerful; a human language is only mildly context-sensitive (Jäger& Rogers, 2012), so most of the time, a context-free grammar (CFG), which we will discuss next, is usually sufficient.

Push-Down Automata and Context-Free Grammar

A PDA, like a Turing machine or LBA, has a memory. However, the access to memory is severely limited. The computer can only access the memory as a stack data structure. A PDA corresponds to a context-free grammar or CFG (Jäger& Rogers, 2012). CFG or type-2 grammar is the most popular formal grammar for modelling human languages. The production rule is as follows:

$$A \rightarrow \beta$$

where A is a single non-terminal symbol and β is a combination of terminal and non-terminal symbols.

X-Bar Theory

X-Bar Theory is a theory in linguistics that we can model human languages using a modified version of CFG, and a syntax tree (Anderson et al., 2022, Chapter 6). We slightly modify our notation to make the rules more readable:

- The non-terminal symbols can have multiple capital letters. For instance, in the X-bar theory, we can represent a noun phrase using NP.
- The terminal symbols can have multiple lower-case letters.
- Space is used to separate symbols.

For instance, a production rule below is represents some possible English noun phrases:

$$NP \to DN$$
$$D \to the \mid this \mid that \mid these \mid those$$
$$N \to cat \mid cats \mid dog \mid dogs \mid frog \mid frogs \mid \ldots$$

We notice right away that these rules are imperfect. For instance, “this cats” is grammatical based on the formal grammar, but it is not a valid phrase in English. One of my professors, Prof. Hirst, told me this: a context-free grammar is truly context-free. Based on how we design a CFG, it could be completely discordant with the language that we are working with. After all, the goal of using CFGs here is to get something that is something that we can work with, and not something that is perfect.

Using the production rules, we can also make a tree representation of a phrase. Below is an example of a syntax tree representing a sentence from Anderson et al. (2022, Chapter 6). The tree below has a corresponding CFG:

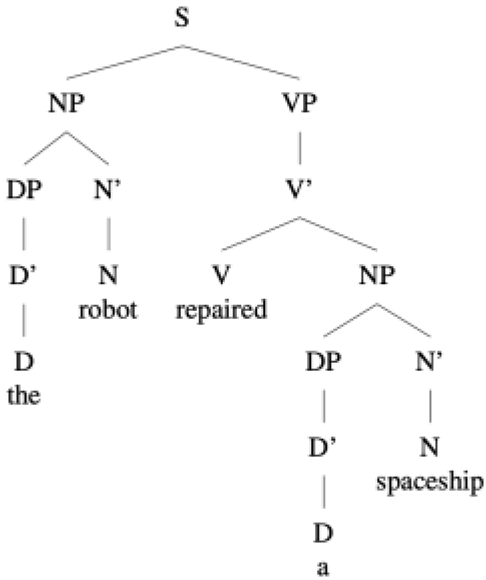

Figure 1: *A syntax-tree of the sentence "the robot repaired a spaceship" from Anderson et al. (2022, Chapter 6).*

Backus-Naur Notation: Programming Language Modelling

Some students who have taken a course in foundations of programming languages may find the production rules notation somewhat similar. This is no coincidence. Knuth (1964) points out that the Backus-Naur notation, which is used to model programming languages and to create programming language syntax trees, are quite similar to CFG.

Finite State Automata and Regular Language

FSA is less powerful than a PDA because it cannot write onto the tape at all – except, perhaps, to indicate if the string is accepted or not. However, we can also devise a FSA that does not need to write anything at all. Just like Turing machines, FSAs have multiple states that the machine can be in. Instead, we simply designate states as either accepting or rejecting. If the machine ends up in an accepting state, then the string is accepted. Otherwise, it is rejected. A FSA corresponds with a regular language (a.k.a. type-3 grammar).

A production rule for a regular is as follows (Su & Yan, 2011, Chapter 2):

$$A \to aB \mid b \mid Ba \mid b$$

where A, B can be any non-terminal symbol. I.E., a regular language is not limited to just literally using A or B. It can use any other symbols as well. Likewise, a, b can be any non-terminal. We often do not use the production rule notations for regular language. Instead, we use something called regular expression.

Regular language is also used to model morphology, which, in a layman's terms, involves changing a "word part" to alter its meanings (Kurdi, 2016, Chapter 3). A "word part" is officially known as a morpheme. For instance, the English verb for "walk" can be altered using the following morphemes: "-ing", "-ed", "-s", "-er", and etc.

Because we use regular language for modelling morphology, we can also use it to help with a process called **lemmatization**, where morphemes are removed so that only the stem of the original word (i.e. the lemma) remains (Bird, Klein & Loper, 2019, Chapter 3). Here is an example when lemmatization can be useful. Let's say a user of a chatbot wants to adopt a cat, so they ask the bot to access a database of all pets in the local shelters. The bot must then extract the entries of all cats and give them to the user. If the user asks:

> Can you give me info on all cats in the local pet shelters?

, the chatbot may directly start querying for "cat**s**". However, the search may fail because the database may not be using "cat**s**" at all. Instead, it uses "cat" in its records. Here, the chatbot can lemmatize the word "cat" to get a better result.

It is important to note, though, that lemmatization using a regular language can be imperfect. For example, English has irregular inflections, such as "goose" is pluralized as "geese."

> It is important to note that the computers in this chapter are what we call "Recognizers." In the end, they can only do three things: accept the string, reject the string, or get stuck in a loop forever. Now, one may wonder: most real-world computers do not behave like that! And they would be right. However, it is important to note that using simpler machines can make certain arguments easier to make. Additionally, these machines were developed when computer scientists were exploring how to create a good compiler (Lipton, 2010, Chapter 29). A string is essentially a possible program script. An automaton is a compiler of a specific programming language. Acceptance basically means the string is a well-formed script of that programming language.

Flaws of Treating Languages like Computers

What about Semantics?

We have only discussed the syntax, but not really the semantics. The closest that we came to discussing meaning was with morphology. This is by design. Just as Haugeland (1981, p. 44) stated:

> In effect, given an interpreted formal system with true axioms and truth-preserving rules, if you take care of the syntax, *the semantics will take care of itself.*

This means that if we can design the computers to represent the syntax very well, at the same time, we will also get a system that can deal with meaning. However, modern cognitive scientists no longer have these opinions. Developments in cognitive science showed that there are certain aspects that we cannot formalize quite well.

Historical Linguistics: All Grammars Are Transient

I also argue that formal grammars like the one advanced by Chomsky also do not take the historicity of languages into account. For example, the English language has evolved significantly. Below are the first five lines of "Beowulf", a poem written in Old English, obtained from the Project Gutenberg eBook of Beowulf (2003):

> Hwät! we Gâr-Dena   in geâr-dagum
> þeód-cyninga   þrym gefrunon,
> hû þâ äðelingas   ellen fremedon.
> Oft Scyld Scêfing   sceaðena þreátum,

In the poem, we note that there are many words that do not exist in modern English, and some letters like þ are no longer valid English letters. Therefore, the grammar of Old English is different from that of modern English. I also argue that the grammar is evolving as we speak. Therefore, Chomsky's idea of using computers to represent syntax is only to capture a snapshot of what a language is like, even though natural languages are ever-evolving entities.

Input on a Silver Platter

When we use these automata to deal with languages, we assume that the inputs are well-formatted – i.e., nicely encoded onto a tape. However, in the real-world, obtaining these inputs requires us to use our senses to glean information from the environment. Inputs themselves also may not be neatly organized. When we see a writing, we must first be able to recognize the handwriting, and each handwriting can vary greatly.

What about Language Models?

A question that a reader may have is: is a formal language a type of language model? In the most technical sense, the answer is "No." A language model, unlike a formal language, is probabilistic. For the simplest one, we have a bag-of-words (BOW) model where we simply assign probabilities to a word based on how often it appears in a corpus – i.e. a body of texts and documents that we use for training. Although we can use a BOW to generate some statistics and to learn about properties of a language, it has a context window size of 0.

A more complex model is a bigram model. Unlike a BOW, a bigram model represents a conditional model of word pairs. A probability of a word pair can be represented as (Jurafsky & Martin, 2025, Chapter 3):

$$P(w_2|w_1) = \frac{\#(w_1 \cup w_2)}{\#w_1}$$

This represents the maximum likelihood of w_2 appearing after w_1 in the text (Jurafsky & Martin, 2025, Chapter 3). We can extend the bigram models. Instead of representing 2 words, we can create trigram models to model word triplets instead. This, in turn, could extend the context window. In the case of bigram, the size of the context window is now 1. However, Jurafsky & Martin (2025, Chapter 3) stated that the n-gram approach is of limited usefulness. To create a chatbot or to create a generator, we will need a large language model (LLM), which involves neural networks and deep learning. Whereas a language model tries to predict a single word or a few words based on the preceding ones, an LLM tries to predict a full sentence. It also has a much larger context window. Furthermore, some types of LLMs use specific neural network architectures which allow them to resolve ambiguities. For instance, when a transformer-based LLM encounters an ambiguity, it can refer to past words in its large context window to resolve it (Kamath et al., 2024).

We will not cover a language model to a significant degree here, as there are already many resources on the topic. For instance, Jurafsky & Martin (2025) is available for free to anyone interested. Although it is still a draft, it is comprehensive and is good enough to be used for teaching.

References

Anderson, C., Bjorkman, B., Denis, D., Doner, J., Grant, M., Sanders, N., Taniguchi, A. (2022). *Essentials of Linguistics, 2nd edition.* McMaster University. https://ecampusontario.pressbooks.pub/essentialsoflinguistics2/

Bird, S., Klein, E. & Loper, E. (2019). *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit.* O'Reilly. https://www.nltk.org/book/. NOTE: The website is an updated version from the book published by O'Reilly.

Boker, U. & Dershowitz, N. (2008). The Church-Turing Thesis over Arbitrary Domains. In A. Avron, N. Dershowitz & A. Rabinovich (Eds.), *Pillars of Computer Science. Lecture Notes in Computer Science* 4800. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-78127-1_12

Chomsky, N. (1959). On Certain Formal Properties of Grammars. *Information and Control* 2, 137 – 167. https://doi.org/10.1016/S0019-9958(59)90362-6

Critchlow, C. & Eck, D. J. (2025). *Foundations of Computation.* LibreTexts. https://eng.libretexts.org/Bookshelves/Computer_Science/Programming_and_Computation_Fundamentals/Foundations_of_Computation_(Critchlow_and_Eck)

Dąbrowska, E. (2015). What exactly is Universal Grammar, and has anyone seen it? *Frontiers in Psychology* 6. https://doi.org/10.3389/fpsyg.2015.00852

Ebrahimi, A. (2014, June). Expressing Programming Algorithm with Poetic Language. *ACM Inroads 5*(2). https://dl.acm.org/doi/pdf/10.1145/2614512.2614516

Haugeland, J. (1981). Semantic engine: An introduction to mind design. In J. Haugeland (Ed.), *Mind Design*. MIT Press.

Jäger, G. & Rogers, J. (2012). Formal language theory: refining the Chomsky hierarchy. *Philosophical Transactions of the Royal Society B* 367, 1956-1970. https://doi.org/10.1098/rstb.2012.0077

Jurafsky, D., & Martin, J. H. (2025). *Speech and Language Processing.* https://web.stanford.edu/~jurafsky/slp3/

Kamath, G., Schuster, S., Vajjala, & Reddy, S. (2024). Scope Ambiguities in Large Language Models. *Transactions of the Association for Computational Linguistics* 12, 738–754. https://doi.org/10.1162/tacl_a_00670

Knuth, D. Backus Normal Form vs. Backus Naur Form. *Communications of the ACM* 7(12). Association for Computing Machinery. https://dl.acm.org/doi/pdf/10.1145/355588.365140

Kurdi, M. Z. (2016). Natural Language Processing and Computational Linguistics 1. *John Wiley & Sons*.

Lipton, R. J. (2010). *The P=NP Question and Gödel's Lost Letter*. Springer US.

Namboodiri, E. V. N. (2017). Panini's Conception of "Syntactic Structures". *Interdisciplinary Journal of Linguistics* 10.

Rajpopat, R. A. (2021, July). *In Pāṇini We Trust: Discovering the Algorithm for Rule Conflict Resolution in the Aṣṭādhyāyī* [Doctoral dissertation, University of Cambridge]. Apollo. https://doi.org/10.17863/CAM.80099

Ramoo, D. (2021). *Psychology of Language* (M. Bartlett, Ed.). BCcampus. https://opentextbc.ca/psyclanguage/

Su, Y., Yan, S.Y. (2011). *Principles of Compilers: A New Approach to Compilers Including the Algebraic Method*. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-20835-5

The Project Gutenberg eBook of Beowulf. (2003). In J. A. Harrison & R. Sharp (Eds.), *The Project Gutenberg eBook of Beowulf*. Project Gutenberg. https://www.gutenberg.org/files/9700/9700-h/9700-h.htm

Media Attributions

- 615_tree1_repaired

III

Representation

8.

Logical Representation, Automated Reasoning, and the Capreses Salad

> The only salads that they have are pasta salads, like look at this, … chicken bacon pasta … piri-piri chicken pasta … pesto-mozzarella pasta. Tenically, it's a salad. Technically, it's correct. Emotionally? Kinda misleading.
> **Will Tennyson (2025)**

At this point, we have discussed what constitutes a language – without a major component: semantics. We, humans, do not simply form random strings. We, humans, have intention. The words that we have often do mean something. Semantics still exist within the context of rule-based AI. However, the subjective elements have been sacrificed to make room for cold logic. Instead, we simply have logical representations. For instance, the word "sibling" could be defined as:

- If X and Y share the same parent Z, then they are siblings.

This could be written more formally as:

- $parent(Z, X) \wedge parent(Z, Y) \rightarrow sibling(X, Y)$

The idea of Classical AI is that we can achieve something similar to the natural semantics just by defining all of these logical relationships. Everything and anything can be defined this way. This philosophy is best summarized as Hagueland's (1981, p. 44) quote: "if you take care of the syntax, *the semantics will take care of itself*." Now, with the advent of deep learning, we know better not to convert everything into logical representations. Still, the idea of logical representation is still worth exploring – not only for historical reasons, but also because it is actually making a comeback. This is because Large Language Models (LLMs), which are based on deep learning, cannot reason well (Shojaee et al., 2025). It can feign reasoning, but questionable answers often surface. Furthermore, some researchers propose combining both rule-based AI and deep learning – having the best of both worlds (Bhuyan et al., 2024).

Now, we are getting ahead of ourselves. Before talking about other issues, we must first know the symbolic approach. So here it is.

Automated Reasoning and Expert Systems

So "reasoning" might be a newfangled term around mid-2020s. Many undergraduate computer science students may think that these terms are often very new. However, they may not realize this: they actually know quite a bit about reasoning. In their lower years, many undergraduate computer science students have to take lessons or courses in logic (a.k.a. discrete mathematics). Their brush with logic is with Boolean logic. Boolean logic is quite simple: we have variables that we can substitute with either True or False, and the operations that deal with True or False values. For instance, below is a Boolean expression:

- $a + b$

The expression means a OR b. It states that if either a is True or b is True, then the expression is equal to True. Boolean logic is often used in several programming contexts, such as:

- Setting up program branches
- Filtering a data structure in a functional programming language

Boolean logic is relatively simple, so it can be easily mechanized as electric circuits (Dawson, 2013, Chapter 2). This ease of mechanization eventually led to the invention of practical and modern computers. More advanced than Boolean logic is First-Order Logic (FOL). Here, students encounter new symbols such as: $\forall$, $\exists$, $\in$ and more. A feature of FOL is its ability to create a new function. For instance, in the sibling example, we have a function that indicates whether an object is a parent of another one, and another that indicates if two objects are siblings of each other.

A set of logical propositions that are true can be called a knowledge base (Poole & Mackworth, 2023, Chapter 5.1). Using a knowledge base, someone can create software called an expert system to perform queries. For instance, if someone is using a self-diagnosis application like WebMD Symptom Checker[1], the user can provide information about their symptoms. The application then tries to match the symptoms with the names of the diseases. The knowledge base of such software may contain a proposition like:

- $has\_cold(X) \leftarrow exhausted(X) \wedge cough(X) \wedge runny\_nose(X) \wedge \ldots$

1. https://symptoms.webmd.com/

NOTE: We use $\longleftarrow$ instead of $\longrightarrow$ to match the convention of logical programming languages like Prolog.

When the user provides their symptoms, the application tries to match the right-hand side with the left-hand side. The application tries to substitute each function with a True or False value. This process is performed until **unification** is achieved. By trying to match the left-hand side with the right-hand one, this process is called forward chaining (Poole & Mackworth, 2023, Chapter 5.3). Forward chaining applies modus ponens until one of the propositions in the knowledge base is true. Modus ponens means that if we can show that the antecedent is true, then the consequent must be true. In the case of has_cold, having the cold is the consequent, and the antecedent is having other symptoms such as feeling exhausted, coughing, having a runny nose and so on.

The opposite of forward chaining is backward chaining. We start at the goal and try to find the potential propositions that could unify with them (Norvig, 1992, Chapter 16). For instance, let's imagine that we are building an expert system for making a pie. In our knowledge bases, we have the following propositions:

- $pie(X, Y) \leftarrow filling(X) \wedge crust(Y)$
- $filling(X) \leftarrow strawberry(X)$
- $filling(X) \leftarrow meat(X)$
- …
- $crust(Y) \leftarrow standing\_crust(Y)$
- $crust(Y) \leftarrow puff\_pastry(Y)$
- …

Now, assume that we just bought the instant puff pastry crust and we need to think about the filling, we can ask the pie-making expert system for an idea. The user's query may be turned:

- $pie(X, puff\_pastry)$

The expert system's goal is to find anything that can substitute X. Using backward chaining, the system will return strawberry, meat, and more as potential replacements for X. Anything that is not entailed by the knowledge base will not be suggested. Backward chaining can also be used to indicate if a pie combination is valid or not. For example, if the user's query is turned into:

- $pie(car\_tire, standing\_crust)$

The backward chaining system will assume at first that making a pie with a car tire is possible. Then, it will try to find from its knowledge base if any proposition can match or not. If nothing in the knowledge base indicates this, this is contradicted; then, the expert system tells the user that it is not possible to create a pie with a car tire filling.

Ultimately, logical reasoning comes down to the automatic treatment of logical propositions. Most computer science students already know how to perform logical reasoning (even if they may not realize it), so not many details will be spent on the matter. Automated reasoning, on the other hand, may be new to many students. While this book will not describe how computers perform these operations, it is important to note that search algorithms, especially depth-first search, play an elementary role (Poole & Mackworth, 2023, Chapter 5.3). Because basically, automated reasoning boils down to searching for a match in a knowledge base, given a query.

Automated Reasoning's Last Laugh

Although automated reasoning was eventually seen as outdated by the time I started my undergraduate degree around 2011, it is making a comeback. First, deep learning – i.e. neural networks, as it turns out, does not have a structured approach to logical thinking. Although we can train neural networks to feign automated reasoning, Shojaee et al. (2025) still do not think that this is true reasoning. Furthermore, even if a neural network could perform perfect automated reasoning, there is the issue of efficiency and practicality. Neural networks can be difficult to train as they require many data points and computational resources. Additionally, with the advent of Retrieval-Augmented Generation (RAG), we must consider knowledge base design more than ever. Although modern databases and knowledge bases work differently from what is described here, the basic principles still apply.

I argue that an ANN-based reasoning is possible. This is because, in Hu (2025), I argue that all Turing machines can be memorized within a neural network. Some Turing machines in the neural networks represent automated reasoning software. So an ANN that memorizes Turing machines for automated reasoning and acts is also capable of reasoning. However, training an ANN to be like this may be counterproductive. In many cases, we are simply better off just programming the automated reasoning software instead of training an ANN to become one.

Categorization Theory

Cognitive science also thinks about how objects can be represented through a concept called **categorization** – i.e. how does a person categorize an object that they see. Categorization is important. If a living being sees something, they need to determine whether it is food or a foe. If it is food, the being can consume it to sustain itself. If it is a foe, it must be prepared to fight or run. Classically, categorization relies on logic – very similar to the one outlined above. Although past philosophers may not have talked about "knowledge base" or "forward chaining", they still treated a category as logical associations. Because we have already discussed automated reasoning at length, we already know how the classical theory of categorization goes.

The classical theory does have a strength: if we can create a knowledge base that is sufficiently detailed, we can repurpose algorithms such as depth-first search to easily categorize objects. We can also use logical reasoning to make arguments about the categories. Such a process can easily be automated.

Logical associations, as it turns out, have a rather significant weakness. Human thinking seems to be too flexible to be captured by logic. Let's look at an example of the caprese salad. In Sault Ste. Marie, Ontario, where this book is written, there is a large Italian population. Therefore, it is very easy for me to get a caprese salad. A caprese salad is a very simple salad with the following ingredients:

- Slices of tomatoes
- Slices of fresh mozzarella cheese
- Basil leaves
- Olive oil
- Salt
- Pepper

So far, so good. However, an inquisitive person may ask: could a caprese salad be a cheese and fruit platter as well? After all, a cheese and fruit platter is simply a plate full of pre-cut cheese and fruits selected to pair well with the cheese. If we follow this line of logic, then a caprese salad is a cheese and fruit platter; the mozzarella cheese is the pre-cut cheese, and the tomatoes are fruits that supposedly go well together with the cheese.

Confusion reigns. We will need a more flexible way of thinking about salad.

Given that many Indian students are attending my university, my question to them is: Is a paneer and tomato curry also a Caprese salad?

Now, what can we do to unlock this impasse? Perhaps, we could use another categorization theory called the **prototype theory** (of concepts). First developed by Rosch (Lakoff, 2007), this theory assumes that we can categorize objects based on their similarities to prototypes. A prototype is an abstract form. For example, a prototype of SALAD is not something that we can actually consume. Instead, it is a vague notion of vegetable pieces being dressed with some sort of edible liquid. In the prototype theory, we can assign scores based on how similar an object is to its ideal prototype. In the case of SALAD, a garden salad would receive a higher score than a caprese salad. Although cognitive scientists often treat the prototype theory as a theory of how people categorize objects around them, Rosch herself objects to this. She argues that her theory is merely something observable. It is not a mental model – i.e., it does not represent how a person actually thinks (Lakoff, 2007).

The prototype theory does have a flaw, however. While it allows us to make a statement like: "a caprese salad is still a salad, but less salad-ish than a garden salad", it does not always present a clear boundary. As it turns out, the label of the object seems to vary based on the circumstances. For instance, if you are ordering a three-course meal with a salad in an Italian restaurant, the server may provide you with a list of salads. One of the possible choices is a caprese salad. In this case, the caprese salad is a salad. However, in some Italian restaurants, the caprese salad could also be presented as an appetizer instead of a salad. In this case, it acts more like a cheese and fruit platter – something more akin to a charcuterie board.

This does not mean the prototype theory is useless in any way. In fact, as I will argue in the next chapter, it plays a central role in many machine learning algorithms.

It seems that the way humans perform categorization seems to be very flexible and very much context-dependent. This is more in line with the ***theory* theory** of categorization – i.e., a membership of an entity has to be negotiated in real time (Margolis & Laurence, 2023). Vervaeke, Lillicrap & Richards (2012) propose a concept called relevant realization, which argues that many concepts in cognitive science seem to boil down to knowing whether something is relevant at the moment or not. For instance, a caprese salad is a salad when its salad-ness is relevant to the discussion at hand. Another example is: a fish can be both a pet and food depending on how we view it. We would never eat a goldfish, but we would never keep a tuna as a pet. A shark is a complex case. Some people consume shark fin soups, which makes it food. However, some aquariums also keep sharks in their exhibits. This also makes them pets as well. However, for common households, sharks would never be seen as pets, because they lack the means to raise them as such. We will pause the discussion of relevance realization for now, as it is more aligned with embodied cognitive science. For now, we will move on to the topic of connectionism – a key concept in cognitive science.

The Sacrifice of Meaning

And thus, practising computer science is an act of sacrifice. At the altar of Turing and other computer science pioneers, people are asked to sacrifice the meaning so that computers can solve their problems for them. In natural language processing, novels, texts, and corpora have their meaning stripped down and turned into frequencies and statistics so that computers can process them in some ways. Some AI pioneers, like Putnam, believe that we never really made any sacrifice of any meaning to begin with. After all, we cannot sacrifice what does not exist. Instead, everything can be explained away through logic and reasoning alone. However, as we have seen, formalization has its limits, and the altar still demands its sacrifice. In the next chapter, we will explore deep learning models, which relax the formalization somewhat. Such an approach allows computers to be more flexible. However, a sacrifice must still be made – this is not negotiable.

References

Bhuyan, B. P., Ramdane-Cherif, A., Tomar, R., & Singh, T. P. (2024). Neuro-symbolic artificial intelligence: a survey. *Neural Computing and Applications*, *36*(21), 12809-12844. https://link.springer.com/article/10.1007/s00521-024-09960-z

Dawson, M. R. W. (2013). *Mind, Body, World: Foundations of Cognitive Science*. AU Press. https://www.aupress.ca/app/uploads/120227_99Z_Dawson_2013-Mind_Body_World.pdf

Haugeland, J. (1981). Semantic engine: An introduction to mind design. In J. Haugeland (Ed.), *Mind Design*. MIT Press.

Lakoff, G. (2007). Cognitive models and prototype theory. *The cognitive linguistics reader*, 130-167.

Margolis, E., & Laurence, S. (2023). Concepts. In E. N. Zalta & U. Nodelman (eds.), *The Stanford Encyclopedia of Philosophy* (Fall 2023 Edition). https://plato.stanford.edu/archives/fall2023/entries/concepts/

Norvig, P. (1992). *Paradigms of Artificial Intelligence Programming: CASE*

STUDIES IN COMMON LISP. Morgan Kaufmann. https://github.com/norvig/paip-lisp

Poole, D. L. & Mackworth, A. K. (2023). *Artificial Intelligence: Foundations of Computational Agents, 3rd Edition.* Cambridge University Press. https://artint.info/3e/html/ArtInt3e.html

Shojaee, P., Mirzadeh, I., Alizadeh, K., Horton, M., Bengio, S., & Farajtabar, M. (2025). The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity. *arXiv preprint arXiv:2506.06941*. https://doi.org/10.48550/arXiv.2506.06941

Tennyson, W. (2025, June 29). *How Much Weight Can I Gain in World's Most Obese Town?* [Video]. YouTube. https://www.youtube.com/watch?v=mBSPJ_QeUxY

Vervaeke, J., Lillicrap, T. P., & Richards, B. A. (2012). Relevance realization and the emerging framework in cognitive science. *Journal of Logic and Computation* 22(1), 79-99.

9.

Connectionism: Neural Networks

We still don't understand how the brain works. We know a lot more than we did, but we still don't really know.
A quote from Geoffrey Hinton from Kalvapalle (2025)

Welcome to Connectionism

At this point in time, I have discussed cognition in a very classical manner. Basically, each thought and each action is merely a manipulation of one string to another. No consideration has been given to how the human body actually works. This view, at first glance, seems to be rather dismissive of how the human body actually works. This is by design. By detaching itself from biology, cognitive science aims to provide an architecture-independent explanation of cognitive phenomena. However, as we see, logic alone is insufficient. In theory, it does not explain qualia. In practice, it fails to deal with ambiguity in human languages and the ever-changing nature of all things.

Perhaps, we should start to look at a happy medium. To improve the explanation of how the mind works, we could take aspects of neuroscience. We do not have to fully commit to becoming Dr. Frankenstein by experimenting with the real brain. Instead, we can take the basic principles of neuroscience, simplify them, and simulate them on a machine. Further, we can even allow some logical flexibility. Instead of dealing with the rigid formal languages that struggle with ambiguity, we will even allow probabilities and accept imperfect answers. If we do things right, the computer hosting this neuroscience software may even possess a soul!

> By taking inspiration from nature, we are engaging in **biomimicry**. Usually, biomimicry appears in engineering and design.

History of Artificial Neural Networks

According to Abraham (2002), artificial neural networks (ANN) have their roots with Warren McCullouch (1898 – 1969) and Walter Pitts (1923 – 1969). McCullouch was a neurophychiartist and Pitt was a mathematician (Abraham, 2002). Both published a paper (McCulloch & Pitts, 1943) which argues that neurons, a type of brain cell, are capable of representing logical statements. It describes how neurons could be arranged to represent logical propositions, and their manipulations could be used to solve logical statements (McCulloch & Pitts, 1943). McCulloch & Pitts (1943) avoided discussing any type of learning or any type of neuroplasticity. In their framework, they assume that the neurons remain static for the sake of convenience. We know though that brain cells can be highly plastic and malleable, especially for the newly born. Some infants could be born with a large amount of brain missing, and could still grow up to function reasonably well as other parts of their brains took over the functions of the missing parts. For instance, Yu et al. (2015) encountered a patient who was complaining about a headache. After a brain scan, they found that she was born without a cerebellum. Although she faced developmental challenges, movement issues, and constant headaches, she was able to function relatively well without being aware of missing an organ.

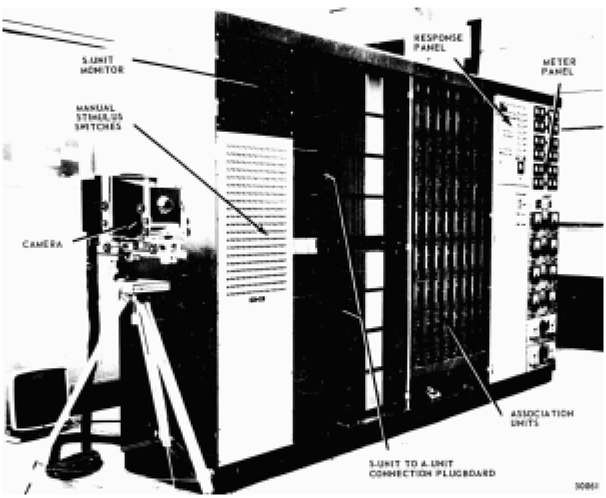

Figure 1: *Perceptron Mark I (Hay et al., 1960, p. 5).*

In 1950's, Rosenblatt would implement the first iteration of ANN, called **the Perceptron** (Dawson, 2013, Chapter 4). Unlike the ANNs of today, the early Perceptron was implemented as a physical machine as seen in Figure 1. Furthermore, the Perceptron was implemented using the McCulloch and Pitts' description, which means a neuron is triggered using a Heaviside step function, which can be described as follows (Dawson, 2013, Chapter 4):

$$H(x) = 0, x < t$$
$$H(x) = 1, x \leq t$$

where t is a specific threshold. The use of Heaviside step function is to simulate the “all or nothing” nature of neuron firing (McCulloch & Pitts, 1943; Dawson, 2013, Chapter 4) – i.e., for a neuron to output a signal to another neuron in the network, it must be sufficiently stimulated. If there is an insufficient level of stimulation, the neuron will simply not fire. We can combine multiple neurons together to represent logic gates. For instance, an AND-gate can be represented as Figure 2. In this example, we have two input neurons (a.k.a. node) and an AND-gate neuron node. The AND-gate neuron node will only fire if both input nodes fire. By firing, we mean each node is contributing the value of 1.

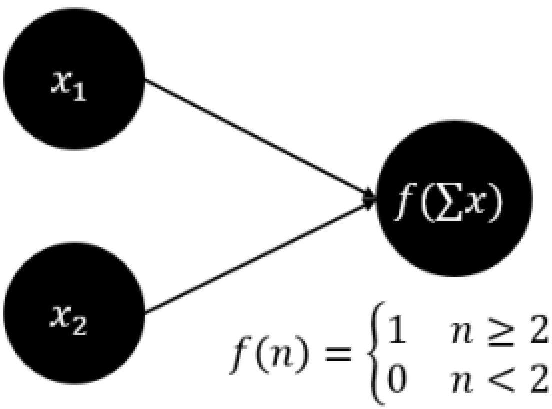

Figure 2: *A simple neural network representing an AND-gate.*

Alternatively, we can also provide the weight of the firing as well. For instance, instead of accepting the value of 1, the edges can also have their own multipliers, called weights. In Fig. 3., we redesign the AND-gate network so that each edge has the weight of 0.5. This ensures that when both neurons fire, the combined weight will not exceed one. Indeed, the Heaviside step function will need to be revised. Now, one may ask: Why are we going through the trouble of assigning weights to the edges? As it turns out, machine learning with ANN has very much to do with updating the weights. In each update, a neuron can become less or more relevant through changes of the weights. Less relevant neurons have smaller weights than the more relevant ones.

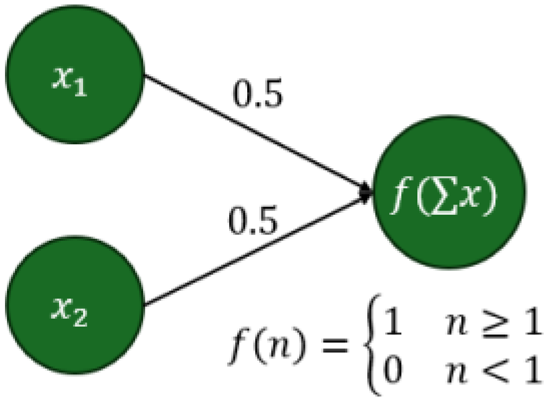

Figure 2. *A perceptron for an AND-gate. The edges have weights.*

By extending the inputs and the outputs so that they can accept 20 x 20 pixel images, Rosenblatt was able to create an ANN that is capable of image recognition (Mantri & Thomas, 2021). However, Rosenblatt's work soon hit an important hurdle: Minsky and Papert show the Perceptron cannot be made to represent an XOR-gate (Mantri, Thomas, 2021). To go be able to implement an XOR-gate, we will need something akin to the modern ANN. While today's computer scientists say that this issue can be side-stepped by adding more layers, deep learning did not exist back then. We were basically stuck at one layer. Essentially, all neurons were only arranged in two layers: input and output (Mantri & Thomas, 2021).

Superior learning could be achieved by adding more layers, and perhaps using other functions other than Heaviside step function. In this case, we have intermediate layers that can help transition from inputs to the output (LeCun, Bengio, & Hinton, 2015). This concept is called deep learning. The term was probably originally coined by Rina Dechter (Fradkov, 2020); however, Dechter's (1986) usage was deeply in the realm of Classical Artificial Intelligence. Specifically, she believed that a search algorithm for a constraint satisfaction problems could be improved if it could memorize its past mistakes. Since a search algorithm involves depths, the name "deep learning" was chosen. Hinton, based on my recollection from my undergraduate course, was the person who popularized the term. However, his usage was notably different and should be considered distinct from Dechter's.

To achieve Hinton's version of deep learning, we need a robust algorithm which is capable of propagating changes downward through the layers. The name of this key algorithm is **backpropgation**. Originally, it was invented by Linnainmaa. Later on, Werbos suggested that Linnainmaa's algorithm could be useful for training neural networks (Mantri & Thomas, 2021). The algorithm did not become practical until Rumelhart, Hinton & Williams (1995).

We will not discuss backpropagation in detail here, as there are many resources on this topic. Some details of this algorithm will be discussed later on in this chapter.

Motivation: From Rule-Based to Neural Network

To better understand why connectionism becomes an important approach in cognitive science and AI, it is best to examine a practical application. Here, I will introduce the task of image classification. The software will simply take a black-and-white image of a cat, and say if it is a cat or not. For example, Fig. 1. should be accepted as a cat. The image classification software here is rather simple in the modern day and age. Anyone, even those who just start out learning how to program, can quickly create this software. However, the historical development is a long and arduous one. By outlining three attempts to create this software, hopefully, we will get to learn the major milestones of AI and cognitive science.

Figure 1. *A black-and-white image of a cat.*

Attempt 1: Rule-based AI

The rule-based AI simply involves programming software. We simply create a Turing machine (or a less powerful machine) that takes in the image and prints out a binary label like “Yes” or “No.” Assuming that we are using 8-bit numbers to represent each grayscale pixel, the input image is converted into a tape. We will not describe the tape in detail. However, generally, it should contain image metadata (e.g., width, height) and the colours of the pixels. Essentially, the Turing machine is deciding on a formal language of cat images. It takes images as strings, and determines if the string belongs to the language.

Now, creating a Turing machine for almost any purpose is nigh impossible, so we will settle for an alternative: just write a program in any language. In the program, there is a function that accepts an array of 8-bit numbers, and some additional data. The function signature may look like this:

```
def is_cat(pixels: int[[]], width: int, height: int, ...) -> bo
```

For each pixel, we can check the range of values. For instance, we can argue that some pixels can be too dark or too bright to belong to a cat:

```
if pixel[1,0] < 10 and pixel[1,0] > 100:
   return False
```

Right away, this approach is problematic. We have this one question: Is checking an image one pixel at a time the best way to do this? …

Well, checking one pixel at a time is not the greatest idea. However, the idea of converting pixels into numbers is an important one. Just as a Turing machine requires an input to be encoded into its own symbols, modern software and machine learning models also require the conversion of inputs. Modern software is very different from what Turing may have envisioned, but it still inherits the sins of the forefathers.

Does this mean Turing machines and computers in general are not capable of looking at multiple pixels at once? Actually, a Turing machine and computers can check multiple cells by using their own memories. While a Turing machine can access a single cell of data at a time, it can temporarily memorize the interim results on the tape as well. When looking at a single pixel, the Turing machine can also refer to the other memory cells to make better inferences.

If a Turing machine or a computer can look at multiple pixels, should we go ahead and handwrite a better program that can do this? Today, even the greenest programmers will laugh at this proposition. Instead, they will say that we should use this technique called *machine learning*. However, such an answer was not obvious to past computer scientists. When I was an undergraduate student in 2011, machine learning was just a mythical concept. It was not obvious to the past version of me or many people that we simply cannot program our way to recognize cat images. Of course, the professors are already aware of machine learning, but some of them would still put on an act that everything was just some sort of magic.

Before moving on to machine learning, let's discuss why this is not possible. The major reason is that the rules behind image classification are just way too flexible. Just like in the Capreses salad example, we can play around with what constitutes a cat, so there is simply no rigid set of rules to constitute a cat. For instance, we humans can immediately recognize the stylized cat art by Maud Lewis, in Fig. 2. However, the rigid program may struggle to recognize Fig. 2 as a cat. While Fig. 2 features cat ears, everything else is heavily stylized. For Fig. 3, humans will say that the AI slop image is "catt-ish." Meanwhile, a machine may reject it because it happens to possess three front paws – something that a normal cat does not possess.

Figure 2. *A stylized painting of a white cat by Maud. Taken by myself from the Art Gallery of Nova Scotia.*

Classical cognitive scientists are well aware of this problem. So instead of having a cat image being judged by a discreet set of rules, they are open to relaxing the rules. Instead of saying that an image must be that of a cat because it possesses cat body parts, we can assign a probabilistic score to the image. We can apply the concept that we have seen from the previous chapter: **the prototype theory**.

Figure 3. *An AI-generated image of a cat with three front paws. Found on Facebook around 2024.*

An image would receive a score of 0 if it is nothing like a cat. An actual image of a cat would be assigned a score of 1. Fig. 2 and Fig. 3 would score lower than Fig. 1. However, they would not be rejected outright. Although cognitive science finds the theory promising at first, it turns out that the theory is imperfect. It struggles when we try to combine multiple concepts together. Multiple concepts, when combined, have emergent properties that are not inherent to the original concepts (Margolis & Laurence, 2023). For instance, Fodor and Lepore (1996) argue that “pet fish” is problematic. They state that while a goldfish is a good example of a “pet fish”, it itself is a poor representation of “pet” and “fish.”

So if the prototype theory is problematic, why do we still discuss it, and discuss it again? As it turns out, machine learning does make use of probabilistic assignments. Many machine learning techniques, including ANN, will not say if an input is exactly of that class. Instead, it will provide scores. Furthermore, the prototype theory also makes use of the distances (Fodor & Lepore, 1996), which play major roles in machine learning. The most notable examples of distances being used include:

- **Squared Error Loss Function:** Many machine learning algorithms are trained using the squared distances from the predicted answers to the ground truth value. We will see more of this in the next section.
- **Cosine Similarity Distance:** In large language models, the cosine distances between each pair of tokens are computed. The tokens are deemed to be more similar if the distance between them is small (de Vos et al., 2021).

Attempt 2: Logistic Regression

Since it is not a good idea to simply program image classification software, we will need to rely on a concept called machine learning. Machine learning, as Prof. Richard Zemel told me during my machine learning course, is essentially a program that can program itself. It is a program that can self-modify based on the data fed to it. Before diving into ANN, which is a type of machine learning technique, we should start with something simpler: logistic regression. A logistic regression simply applies a linear equation to a set of data, and *squashes* the result of the linear equation so that the output is between 0 and 1.

To understand logistic regression, let's start with the linear equation that we see in high school. In high school, we are taught the form of this linear regression: $y = a + bx$. However, we can keep adding additional x terms. So instead of $y = a + bx$, we have $y = b_0 + b_1x_1 + b_2x_2 \ldots$. Each b_i represents the weights where x_i is a specific input. Now, this y will not be a number between 0 and 1. To do so, we must apply the sigmoid function to it. The sigmoid function can be written as (Poole & Mackworth, 2023, Chapter 9.3): $\sigma(x) = 1/(1 + exp(-x))$. With the sigmoid function applied, the equation becomes (Poole & Mackworth, 2023, Chapter 7.3): $y = \sigma(b_0 + b_1x_1 + b_2x_2 \ldots)$. Now, $y \in (0, 1)$ instead of $y \in \mathbb{R}$.

We can then create a logistic regression with y, the response value, indicating the cattiness of the image, and $x_1, x_2, x_3, \ldots$ being the input. When we ask the regression model to judge an image, we substitute each x with a pixel value. Now, there is still a problem: what about the $b_0, b_1, b_2, \ldots$, a.k.a. the weights? To find the weights, we use a machine learning algorithm. In the simple linear regression, i.e. where the linear equation is simply $y = a + bx$, we can simply use the following formulae based on the method of the least squares (Dahlquist et al, 2024, Chapter 14.2):

$$b = \frac{n \sum xy - (\sum x)(\sum y)}{n \sum x^2 - (\sum x)^2}$$

$$a = \frac{\sum y}{n} - b\frac{\sum x}{n}$$

where n is the sample size, x representing the data, and y representing the label for the data point x. In this case, $y = 0$ if the image is not that of a cat, and $y = 1$ if the image belongs to a cat. The algorithm works by finding weights such that the error is minimized for all samples. Although these formulae are relatively simple in the machine learning world, they capture the major principle of machine learning itself: the actual data are used to estimate the weights – i.e., the actual data are used to modify the untrained model. If we have more than one x's and if we are not using simple linear regression, the method of the least squares will not be viable. Instead, machine learning algorithms use a cruder approach called the **gradient descent**. Unlike the least squares method, the weights are updated iteratively. This means, the algorithm will need to be called repeatedly until the weights settle to a point where the errors are minimized (Poole & Mackworth, 2023, Chapter 7.3). One of the formulas for the stochastic gradient descent for logistic regression is:

$$w^* := w - \eta(\frac{\partial L(w^T x_i + y_i)}{\partial w})$$

where w^* is updated weights (i.e., a matrix of $b_0, b_1, b_2, \ldots$), w is the matrix of the original weights, η is the learning rate, and the partial derivative function represents the weight changes with respect to a sample. L here is a loss function (scikit-learn, n.d.). While there are many versions, in essence, all loss functions represent the distance from the correct answer to the predicted answer. This formula can be further modified to optimize learning (scikit-learn, n.d.), but we will not discuss the modifications here.

Once trained, we have a function that assigns a value between 0 and 1 to an input. The input is the converted pixel data into a 1D matrix. The question is: How well does this function work? How accurate is it going to be? Will it assign a picture of a rabbit a high score, and a naked cat with a low score? As it turns out, this algorithm will not work as well. By simply mapping pixels to a value with a simple transformation, the function is engaging in shallow learning. The function would have worked much better if we had some sort of interim functions. These functions can help to recognize aspects of cat images. For instance, some functions may be specialized towards recognizing the ears, while some can recognize the eyes. Interim functions can also help to correct mistakes. If some functions are too fixated on recognizing short-haired cats at the expense of the long-haired ones, then some can correct these oversights. The interim functions are possible if we use deep learning. This will be discussed in the next section. These interim functions, in essence, can also serve as an internal representation.

Another question that we may have is: Can this logistic regression function be represented by a Turing machine? At first glance, it should not be, because we simply do not program it by hand. However, as I imply in Hu (2025), it can still be. We must recall that all Turing machines themselves can also be converted into Turing machine tapes. And as tapes, they can be read and modified by other Turing machines. Thus, a logistic regression program is an output of a training Turing machine which accepts an untrained model, training parameters, and data encoded as a tape. The training machine takes an untrained model, and smiths it using the data and the training parameters. Once the limit for smithing is reached, the training machine outputs the model. Still, we cannot consider machine learning a type of rule-based learning. Although the training machine works based on rules, the resulting machine is completely beholden to the data.

Attempt 3: Artificial Neural Network

At this point, I described a logistic regression for image classification. Although the classification will work better than the rule-based one, it is still not flexible enough. To improve its ability to classify images, we will need a different kind of machine learning. As we have been building towards, we will use neural networks to implement this. Although a neural network is vastly different from logistic regressions under the hood, for most software, the training interface is largely the same. As far as we are concerned, we set up some training parameters, feed the training function some data, and then we wait for training to be complete. The training process also still relies on the loss function, and uses the errors from the data to reweight the model using the stochastic gradient descent. However, deep learning learning makes the training more difficult.

Although the Perceptrons are able to perform some image recognition tasks, due to their single-layer nature, they work more like the aforementioned logistic regression. When I was an undergraduate student, we created matrices to represent ANNs in a programming language like MATLAB, an ANN expert's language of choice before PyTorch. Then, we need implement the backpropagation algorithm. However, before doing so, we need to select a function to compute the values from the preceding nodes. With Perceptron, the function of choice is the Heaviside step function. However, the Heaviside step function can be difficult to differentiate, a major step in training. Therefore, for a while, we simply used the logistic function. If we plot the logistic function against the Heaviside step function, we will see that the logistic function is basically a more flexible and continuous version of the Heaviside step function.

Well, the sigmoid function was popular during my days as an undergraduate student. However, it has since been replaced with rectified linear unit (ReLu), which can be written as (Poole & Mackworth, 2025, Chapter 8.1):

$$relu(x) = 0, x < 0$$
$$relu(x) = x, x \geq 0$$

Basically, ReLu is a linear function that will simply turn all negative numbers into 0. The sigmoid function is still used, however, for the final output (Poole & Mackworth, 2023, Chapter 8.1). Regardless of the functions used, an ANN learns by propagating the error from the loss function in the final layers to the layers below them. The weights assigned to the edges are updated using the stochastic gradient descent algorithm. Since we propagate the errors backwards, we call this algorithm **backpropagation**. The weight updating formula is largely similar to the one for the logistic regression. However, the trickier part is to derive a formula for the nodes in the lower layers. In short, we will need to apply the chain rule in differential calculus. So to find the partial derivative for the gradient descent part, we need to repeatedly expand the formula. The expansion below is based on Poole & Mackworth (2023, Chapter 8.1):

$$\frac{\partial L}{w} = L' \times f_n'(v_{n-1}) \times f_{n-1}'(v_{n-2}) \times \ldots \times \frac{\partial}{w} f_j(v_{j-1})$$

where L is the loss function of choice, w is a specific weight, f_i is a function used in the neural network. It is important to note that the formula above is not the gradient descent formula. It has to be plugged into one. Knowing how to perform backpropagation by hand is a difficult process. However, in my experience, it could be coded with a few lines of code using MATLAB. This is still difficult, but much more doable.

By having multiple layers of nodes, we essentially have deep learning. Each layer, at the mercy of the data and some parameters, can learn to specialize in specific areas. For instance, in a cat image, some nodes may specialize in recognizing cat snouts. Furthermore, we can apply concepts found in other fields to help with training. An example of this is the dropout technique developed by Srivastava et al. (2014), which involves randomly and temporarily blocking neurons from being updated. The technique, as Strivastava et al. state, is inspired by sexual reproduction, where the parents' genes are dropped by half when combined to make the genes of their offspring.

Indeed, we are just barely scratching the surface of ANN. There are many ways and techniques that can be deployed to improve the image classifier. In addition to being used as a classifier, an ANN can also be designed to deal with sequences of words. The most well-known example that we have right now is Large Language Models where neural networks are trained to predict the sequence of texts based on the ones given to them.

It is important to once again highlight the distinction between the Weak vs. the Strong AI. Although the users of ANNs are using techniques inspired by the brain and neuroscience, it does not mean that they are engaging in Strong AI. Some applications of ANN are strictly Weak AI. For instance, the cat classifier is a Weak AI model, because it does not really try to explain how humans actually recognize cats. It is also important to note that ANN is also simplified from the brain itself. Although the human brain contains neurons, the neurons themselves are also highly complex, with many more parts. Nevertheless, to implement Strong AI, it has become clear to cognitive scientists that we cannot ignore ANN.

Cognitive Science Joke: Our Brains Use Calculus

In cognitive science classes, we often joke about our brains using calculus because neural networks use backpropagation for training. However, we have to remember that just because good software does not always equate to the real explanation. Lillicrap et al. (2020) point out that our brains use a different system for training.

Black-Box Model

A major concern with a machine learning model is that it can be difficult to understand what is going on during the training. Although there are some works in visualizing the machine learning models, e.g., Olah, Mordvintsev, A., & Schubert (2017) show how to manipulate ANN to better visualize their internal representation, the difficulty ultimately comes down to the number of features.

To better understand this, let's start with trying to understand a linear model. Most students will be quite familiar with something called the linear models. Assuming the simplest case, a simple linear regression will only have a single feature (a.k.a. variable). They will also know how to use a 2D Cartesian coordinate to plot out the equation. So far, so good. However, what if we add more features? If we add one more, then we would have something like a 3D Cartesian coordinate plot. This is something that our brains handle. In the real world, however, each model will handle thousands, if not millions, of features. There are simply too many features for us to grasp as human beings. Our cognition faculty is simply not robust enough. If we already struggle as much with linear models, then understanding ANNs will be even more difficult. Thus, many machine learning models are known as black box models. Unlike the actual black boxes used in aviation, which aim to provide as much explanation as possible, the machine learning black boxes have unknowable internals.

Having too many dimensions can lead to a phenomenon called the **curse of dimensionality**. The curse of dimensionality is also not just a human issue. Machine learning algorithms can also struggle to work when there are too many dimensions. Unfortunately, to solve this issue for machine learning, human experts will need to be able to diagnose the issues (Altman & Krzywinski, 2018) while being afflicted by the curse themselves.

Having a black-box model can significantly hamper our understanding of AI behaviours. And often, we need to truly understand how it works. For instance, if an AI model has been used for facial recognition and seems to show some bias against people of colour, we need to be able to verify where the bias may exist in the model and fix it. This may involve going deep into the ANN for the facial recognition software and expunging the bad neurons. However, since an ANN is usually a black box model heavily afflicted by the curse of dimensionality, it would be difficult to do so.

No Free-Lunch Theorem

At this point, we demonstrate that ANN, with its ability to create internal representations, will triumph over the logistic regression model when it comes to image recognition. However, we have really been unfair to the logistic regression technique itself. As it turns out, logistic regression truly shines when we want to study factors that lead to a binary outcome. An example of this is to create a logistic regression model to predict whether a passenger of the Titanic would perish in the cold Atlantic Ocean based on their gender and their class. In this case, logistic regression will perform very well.

Now, the question is: Can a neural network with its deep learning perform better than a logistic regression model? The answer is, maybe, but not by much. As it turns out, there is a theorem named “No Free-Lunch Theorem”, which was first developed by Wolpert & Macready (1996). Wolpert & Macready (1996) demonstrated that an algorithm that is adept at solving a class of problems will be worse at the other. Therefore, since the logistic regression is very good at predicting binary outcomes in certain cases, it pays for this by having worse performance for image recognition. Later on, Gómez & Rojas (2016) found evidence suggesting that the “No Free-Lunch Theorem” also applies to ANN as well. Although ANN is flexible and malleable, the technique suffers if we cannot create internal representations of some sort (Gómez & Rojas, 2016). We must also keep in mind that an ANN is significantly more difficult to set up than a logistic regression model. Therefore, if we are creating a model for a case where the logistic regression is suitable, we are better off using logistic regression.

Overfitting: Generalization vs. Specialization

In cognitive science, we make a distinction between generalization and specialization. As it turns out, when something is specialized towards a particular skill, it also loses its ability to perform general tasks. An example of this is an animal’s claw vs. a human hand. An animal’s claw is specialized for attacking its prey. However, by specializing in fighting, it lacks the general dexterity of a human hand. A human hand, on the other hand, is very flexible. We owe much of our ability to function in the world to our hands. However, our hands are not particularly good at a specific task. For instance, if we wish to physically attack someone, we may be better off with an augmentation such as a melee weapon (e.g., a knife).

The issue of generalization and specialization also appears in AI as well. Although it is very obvious that an ANN that is specialized towards recognizing cat images will not be useful for any other tasks, the issue of generalization and specialization can also manifest in a subtler way. If an ANN, or any machine learning for that matter, is trained too much on a specific set of samples, it will have issues making predictions in general cases. For instance, if the model is trained with only the images of Siamese cats, then it will struggle to recognize the images of the other breeds of cats. This phenomenon is called **overfitting** in the machine learning parlance. It is important to note that, in addition to giving a model a diverse set of data to train on, there are other techniques that we can deploy to avoid overfitting. However, these techniques are beyond the scope of this book.

The Revenge of the Classic

Connectionism represents a major advancement in cognitive science as it allows us to model human behaviours more flexibly. Although it initially experienced a winter, just like its Classical counterpart, the Weak AI application of Connectionism also enjoys significant successes. The work by its pioneers and later scientists, like Hinton, has changed humanity in a way that I did not think possible. I certainly did not expect that even the newest of programmers can now start creating ANNs from their laptops. In the past, to get an ANN working, I had to transpose and re-transpose matrices until they fit. For better or for worse, Connectionism has also expanded AI to the masses. This leads to some absurd situations that no one thought were possible. For instance, my former friend, whom I used to volunteer for in the cognitive science club, had resorted to using an LLM to generate insults against me. This was because I was telling him to get off Facebook.

Does this mean that Classical AI is no longer relevant? For a very long time, the students of cognitive science like me have not found it relevant any longer. In fact, we learned to shun it. However, Classical AI is making a comeback, as it seems that ANNs are incapable of true reasoning. Although Classical AI's automated reasoning is merely search algorithms in disguise, they are still far more efficient than what ANNs can do. Now, I personally think that ANNs are capable of human-like reasoning, too. However, we must train them to do so, which can be time-consuming and difficult. While developing a method to train an ANN to reason will be a boon for Strong AI, Weak AI designers and users want automated reasoning sooner rather than later. They want their chatbots to be able to count the right number of "r"'s in "strawberry" soon, and they do not care about the architecture behind them.

References

Abraham, T. H. (2002). (Physio)logical circuits: The intellectual origins of the McCulloch–Pitts neural networks. *Journal of the History of the Behavioral Sciences* 38(1), 3-25. https://doi.org/10.1002/jhbs.1094

Altman, N. & Krzywinski, M. (2018, June). The curse(s) of dimensionality. *Nature Methods 15*(6), 399-400. https://doi.org/10.1038/s41592-018-0019-x

Dahlquist, J., Knight, R., Adams, A. S., Bacon, C. J., Cooper, S. T., Griffin, M. P., Higgins, K. C., & Musolino, L. (2024, December 18). *Principles of Finance.* OpenStax. https://openstax.org/details/books/principles-finance

Dawson, M. R. W. (2013). *Mind, Body, World: Foundations of Cognitive Science.* AU Press. https://www.aupress.ca/books/120227-mind-body-world/

Dechter, R. (1986). Learning while searching in constraint satisfaction problem. *AAAI-86 Proceedings.*

de Vos, I. M. A., van den Boogerd, G. L., Fennema, M. D., & Correia, A. (2021, December). Comparing in context: Improving cosine similarity measures with a metric tensor. *Proceedings of the 18th International Conference on Natural Language Processing (ICON),* (pp. 128-138). https://aclanthology.org/2021.icon-main.17/

Fodor, J., & Lepore, E. (1996). The red herring and the pet fish: Why concepts still can't be prototypes. *Cognition*, *58*(2), 253-270. https://doi.org/10.1016/0010-0277(95)00694-X

Fradkov, A. L. (2020). Early History of Machine Learning. *IFACS PapersOnline 53*(2). https://doi.org/10.1016/j.ifacol.2020.12.1888

Gómez, D. & Rojas, A. (2016). An Empirical Overview of the No Free Lunch Theorem and Its Effect on Real-World Machine Learning Classification. *Neural computation 28*(1), 216-228. https://doi.org/10.1162/NECO_a_00793

Hay, J. C., Lynch, B. E., Smith, D. R., Murray, A. E. (1960). *Mark I Perceptron Operator's Manual (Project PARA*). Cornell Aeronautical Laboratory, Inc. https://apps.dtic.mil/sti/tr/pdf/AD0236965.pdf

Hu, S. (2025). Chatbots as Turing Machines. *2025 IEEE/ACIS 29th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. https://doi.org/10.1109/SNPD65828.2025.11253606

Kalvapalle, R. (2025, April 22). *'Follow your curiosity': Geoffrey Hinton offers three tips for incoming students*. University of Toronto, Mississauga. https://www.utm.utoronto.ca/main-news/follow-your-curiosity-geoffrey-hinton-offers-three-tips-incoming-students

Lakoff, G. (2007). Cognitive models and prototype theory. *The cognitive linguistics reader,* 130-167.

LeCun, Y., Yoshua, B., & Hinton, G. (2015). Deep learning. *Nature 521*(7553), 436-444. https://hal.science/hal-04206682/file/Lecun2015.pdf

Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J., & Hinton, G. (2020). Backpropagation and the brain. *Nature Reviews Neuroscience 21*(6), 335-346.

Margolis, E., & Laurence, S. (2023). Concepts. In E. N. Zalta & U. Nodelman (eds.), *The Stanford Encyclopedia of Philosophy* (Fall 2023 Edition). https://plato.stanford.edu/archives/fall2023/entries/concepts/

Mantri, P., & Thomas, J. (2021). Design of Neural Networks. *IOP Conference Series: Materials Science and Engineering 1174*(012023). https://doi.org/10.1088/1757-899X/1174/1/012023

McCulloch, W. S., & Pitts, W. (1943). A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biology 52*(1/2), 99-115. https://doi.org/10.1007/BF02478259

Olah, C., Mordvintsev, A., & Schubert, L. (2017). Feature visualization. *Distill*, *2*(11), e7. https://doi.org/10.23915/distill.00007

Poole, D. L. & Mackworth, A. K. (2023). *Artificial Intelligence: Foundations of Computational Agents, 3rd Edition*. Cambridge University Press. https://artint.info/3e/html/ArtInt3e.html

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature 323*, 533-536. https://doi.org/10.1038/323533a0

scikit-learn. (n.d.). Stochastic Gradient Descent. *User Guide*. scikit-learn. https://scikit-learn.org/stable/modules/sgd.html

Shojaee, P., Mirzadeh, I., Alizadeh, K., Horton, M., Bengio, S., & Farajtabar, M. (2025). The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity. *arXiv preprint arXiv:2506.06941*. https://doi.org/10.48550/arXiv.2506.06941

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958. https://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

Wolpert, D. H., & Macready, W. G. (2002). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation 1*(1), 67-82.

Yu, F., Jiang, Q. Sun, X., & Zhang, R. (2015, June). A new case of complete primary cerebellar agenesis: clinical and imaging findings in a living patient. *Brain 138*(6), p.e353.

Media Attributions

- Perceptron Mark I
- and_neuron
- weighted_perceptrons
- cat_black

- cat_lewis
- cat_slop

10.

Semiotics

An equation has no meaning unless it expresses a thought of God.
A quote by Srinivasa Ramanujan from an article in India Today Education Desk (2026)

At this point, we have not really discussed any semantics in detail. Early on in the book, I did discuss semiotics very briefly as a teaser. In the previous chapter, I very briefly introduced some concepts like cosine similarity and internal representation within artificial neural networks (ANNs). Although these concepts, in a sense, can be seen as semantics, I think it is worth introducing the full gamut of semantic analysis. As a human-computer interaction (HCI) researcher, I also have a different type of experience in a kind of semantic research: semiotics. Unlike semantic analyses in computational linguistics, which still focus on applying formalism to meanings, mine is far more flexible. However, its flexibility is also a curse, because it can be difficult to connect disparate bodies of literature together. In one section, I talk about the struggle that Anglophone students face when learning French. Then, I talk about matrices. And then, I talk about someone's descent into madness. Perhaps, traffic signs show up somewhere, somehow, between all of these discussions. Then, I will talk a bit about simulation theory. Finally, I would be discussing how a chatbot should work in order to avoid annoying its users. Understanding different types of semiotics is like trying to do different things to achieve the same goal. A person can try to become healthier by doing sports, but what kind? Different types of sports have different sets of demands. Golfing requires a lot of walking, and swimming requires access to a pool. Analogies and scrambled writing aside, let's take a bit of time and repeat after me: "Semiotic is a tool and not truth."

This is a realization that I had when I was trying to study semiotics as a reading course during my PhD program. This is something that you should keep in mind, because the last thing you need is to become like Charles Sanders Pierce, who lived from 1839 to 1914 (Brent, 1996). Peirce, according to Brent (1996), may have been suffering from depressive-manic episodes throughout his life. While he argues that many geniuses also had similar conditions, Peirce also ended up alienating his university, Harvard, and other people around him. Thus, he ends up being vilified. After he was long gone, the university still continues to give difficulty to those who are interested in him. Brent (1996), a biographer of Peirce, faced many obstacles there himself. Today, the HCI community certainly enjoys a fragment of his genius, but we would never apply his whole work.

Before describing each component of semiotics, we should first examine the major branches of semiotics. According to Nöth (2011), semiotics has popularly been divided into three major branches:

- **Syntax:** The study of how signs are structured together.
- **Semantics:** The study of the meaning behind the signs.
- **Pragmatics:** The study of how the signs are used and transmitted (Nöth, 2011).

The division is based on Morris's interpretation of Peirce's prior work. However, Morris's division is not without debates. Eco argues that pragmatics should be merged together with semantics, because the way that we transmit signs also has significant effects on their meaning. In this chapter, we will focus on semantics and some elements of pragmatics, particularly Grice's maxim. For the approaches in semiotics, we will focus on three types of semiotic systems: Saussurean, Peircan, and Shpet.

Saussurean Semiotics

In this book, I will introduce two main types of semiotics: Saussurean and Peircan. The Saussurean semiotics was developed by, of course, Ferdinand de Saussure who lived from 1857 to 1913 (Susen 2018). To understand how the semiotic works, we should look at how a **semiosis –** i.e., how a sign is derived from another one – may happen under his system. Assume that we see the following combination of letters for:

goose

Now, the word itself is not the literal goose. Rather, it is a stand-in for the actual concept of goose. Both the stand-in and the original concept of the goose themselves are called **signs**. Under the Saussurean system, there are two types of signs:

- **Signified:** The original concept that is being represented by the signifier (Susen, 2018). In this example, the signified is the concept of goose itself.
- **Signifier:** The sign that helps to represent the signified. Here, the signifier is the word "goose."

We note that there are other things that can also act as the signifier. For instance, Fig. 1, an image of a Canada goose, is a signifier. It is important to note that Saussure does not believe in fixed meanings. The relationship between the signified and the signifier could change based on circumstances (Susen 2018). For instance, Fig. 1 does have more things going on than just a goose. First, there are multiple banks. Secondly, there is a river. Thirdly, there is a rocky beach, so on and so forth. We could also examine a sentence, instead of a word. What about the sentence “the goose is loose”? Here, we are not just dealing with a “goose” sign, but also other signs – e.g., something being loose. A sign begets more signs, which turns into a chain of what Eco called **unlimited semiosis** (Popescu, 2021). We will discuss the concept of unlimited semiosis later in this chapter, as this originates from Peirce’s more maddening concepts of semiotics. For now, we shall focus on how Saussurean semiotics can be used in practice. After all, Saussure views semiotics more as a tool of literary inquiry rather than the truth itself (Feshchenko, 2015).

Figure 1: *Canada geese.*

False Friends

When an Anglophone starts out learning French, they will encounter something called “faux-amis” which translates to “false friends” in English. A faux-ami is basically a French word that has a similar spelling to an English word, but a different meaning. The first time an Anglophone encounters a faux-ami, the experience can be quite bewildering. Some Anglophones may be under the impression that English is basically a hybrid of German and French. Some may think that if the words are spelled similarly, they should also have the same meaning. But of course, languages and their evolutions can be quite complex. Thousands of years can change everything.

Indeed, there are multiple true friends in English and French. Examples include:

- porc (Fr) & pork (En)

- façade (Fr) & facade (En)
- appearance (Fr) & appearance (En)
- …

When we get to meet so many true friends, we may simply become too trusting. And then, we start to encounter French words like:

prétendre

An Anglophone who does not know the true meaning of the word may simply translate it to something akin to "to put on an act." However, the actual meaning of this word is "to claim." As it turns out, the English word "to pretend" can also mean "to claim" in some very restricted sense. For instance, a "pretender to the French throne" is someone who has *some* legal claim to the nonexistent French throne.

Another false friend is:

sensible

This adjective, in English, means that someone who makes wise and efficient decisions. For instance, someone may praise a corporate move that ends up saving the company as being *sensible*. In French, however, the word means "sensitive" and is used like the English word "sensitive."

And then, there are some false friends that are just completely different:

blesser

This French word does not have a nice meaning. Instead, it means to actually injure someone. This word is used with something associated with accidents that results in physical injuries or even death.

When I was taking a third-year French course, I was introduced to the concept of Saussurean semiotics as a way to model the relationship between false friends. A written word like "blesser" is basically a *signifier,* and the painful concept that it represents is a *signified*. Thus, dealing with friends means using applying the semiotic lens to see who the real ones are, and who the false ones are.

Nadin (2007): Computers as Semiotic Machines

Although semiotics is a study of meaning, a semiosis itself does not have to have intention at all. While a human may giggle at the sentence "the goose is loose", we do not need to have any feeling towards that sentence. In Nadin (2007), there is an example of a soulless semiosis: using a mould to turn clay into sculpture. The clay itself has no meaning, but once sculptured, it becomes a signifier of a concept – perhaps, a godly entity. Nadin further argues that computers are also what we consider a semiotic machine, and they are also models of Saussurean semiotics. A Turing machine simply changes a string to another one – thus, altering its meaning. Now, we will apply Saussurean semiotics to model how a chatbot, a type of software, operates.

For a chatbot, a type of software, the goal is to convert one string to another. The user may input a series of words, images, or other data into the chatbot, and then the chatbot responds with another string that fulfills some requirements, and not necessarily in the user's best interest. Oftentimes, we do not want the chatbot to do what the user wants. For instance, the user may want to use a chatbot to help purchase the best and most affordable vacuum cleaner. While ideally, the chatbot should simply give the name of the best and most affordable vacuum cleaner, the chatbot's creator may have some other interests. A vacuum cleaner company may have paid the creator to prioritize their vacuum cleaner, even if it is not the best and most affordable one. Since the creator is paid to promote the company's vacuum cleaner, the chatbot must prioritize selling that vacuum cleaner instead – even if that goes against the user's best interests.

If the user types in, "Give me the best and cheapest vacuum cleaner." The chatbot cannot deal with that data right away. It must be converted first into a form that a computer can understand. This process is not as easy as it sounds. The whole sentence is a signifier, and we must identify the signified. And the signified must be formatted in a form that a computer can work with. In classical computational linguistics, we must try to identify the verb, the **mood**, the subject, the object, and more. Then, we can apply the rules to convert the sentence into a query for a knowledge base or a database system. Such a process involves tokenization and chunking. A tokenization, in essence, involves separating words and symbols in a sentence into constituent parts – i.e. tokens. For instance, "Give me the best and cheapest vacuum cleaner." may be tokenized as:

- give, Verb
- me, Pronoun
- the, Article
- best, Adjective
- and, Conjunction
- cheapest, Adjective
- vacuum, Adjective
- cleaner, Noun
- ., Period

We note that the period (".") has its own token. As it turns out, these little symbols also play an important role. For instance, if the user is aware that the chatbot is trying to avoid giving them the best and most affordable vacuum cleaner, they may end the sentence with "!!!". This indicates anger. If the chatbot is wise, it should realize that the jig is up and it should give the user what they want. Otherwise, the user may quit using the chatbot altogether.

Now, a tokenized sentence is more workable with, but it is not a good enough signified in many cases. For instance, the feature does not provide the mood of the verb “Give.” Yes, a verb can have a mood. And no, it does not mean that the verb can get upset or happy. Rather, the mood indicates the function of the verb. For instance, “Give” in this sentence here is in the imperative mood. This means the user is commanding the chatbot. If “Give” is used in a sentence like, “I give the best and most affordable vacuum cleaner”, then the mood is indicative because it is just describing an action. To better indicate this information, we can revise our tokenization system to represent mood. For instance, instead of having just “Verb”, we can have labels like “VerbPresent”, “VerbPast”, and “VerbImperative”, and more.

If the chatbot designer thinks that giving a list of tokens is not enough, then they can try giving the chatbot a tree structure instead. In $\bar{X}$ theory, we can use context-free grammar to represent sentence structures as trees. In this case, we can apply techniques called “chunking.”

Are tokenization and chunking sufficient for a robust chatbot? I do not think so. During the winter semester of 2025, I was running a course which led to this textbook. In the course, I gave the students a choice to create two types of chatbots: the classical one and the one based on LLM. The classical chatbots involved tokenization and using rules to answer based on the tokens. These chatbots performed terribly, and I would not blame the students. This is because the students must think of all *possible* rules. And having to complete a project within 12 weeks, without any prior experience in natural language processing, that was a rather tall order. Meanwhile, the ones based on LLMs were much more pleasant to use.

The secret to LLM, though, is that they use artificial neural networks (ANN). Tokens themselves are not compatible. If we recall the previous chapter, an ANN works with numbers and not with text. Therefore, the tokens must be further converted into vectors of numbers. The most well-known algorithm for this is Word2Vec. Only then can the ANN operate on the text and return the answer to the user. In summary, the steps in the semioses are:

1. Converting the Signifier: “Give me the best and cheapest vacuum cleaner” into a Signified, which is a list of tokens.
2. The tokens cannot be used for a LLM-chatbot, so they must be converted into a vector. This means that converting the Signified becomes the Signifier for numerical vector data. A technique like Word2Vec performs an additional semiosis to turn tokens into vectors.
3. The ANN tries to produce an answer that represents the best answer. The internal working of the ANN represents a series of semioses that

convert incomprehensible neural activities (the signified) into a readable answer (the signifier) to the user.

We can see that some semioses are not described here. For instance, a neuron activation is also a semiosis. However, such a detailed analysis is impossible since an ANN is a black box model.

It is important to note that Saussure was not burdened by a glorious purpose. For him, semiotics was just a tool and not the truth (Feshchenko, 2015). Although he believes that abstract communications of signs play an important role in communication (Nadin, 2007), he did not try to look too deeply (Nadin, 2007; Feshchenko, 2015). And perhaps, that was for the better.

Peircan Semiotics

Peirce makes grown men cry.
Professor Graeme Hirst to me

In Saussurean semiotics, something seems to be missing. We discussed how the signified can be derived from the signifier. However, in all semiosis, there seems to be an intermediary that acts as a mediator of some sort. For example, when we tokenized the words, there is an algorithm that separates words into a list. The algorithm itself could also be seen as a sign as well. This is because the algorithm can be read and interpreted like any other sign. In NLTK, a famous NLP library, we can even see that there are multiple tokenizer algorithms. Although the goal of all tokenizers is the same, they produce different lists. For instance, the regular rexpression-based algorithm will apply the regular expression rules. This is simpler than the other types of tokenizers, but it is also rather inflexible due to it being completely rule-based.

In the Peircan model, the semiosis involves three signs:

- **Object:** The original concept being represented by the Representamen (Kilstrup, 2015). This is similar to Saussure's signified.
- **Interpretant:** The sign that helps us to obtain the representamen (Kilstrup, 2015).
- **Representamen:** The sign that represents the Object (Kilstrup, 2015). This is similar to Saussure's signifier.

In the chatbot example, the interpretants are the algorithms and software involved in converting one sign to another. Examples include the tokenizer, the ANN, and the software that converts the ANN's output into human-readable forms. Indeed, we can also further scrutinize the interpretants. The tokenizer software itself can be a source of multiple signs as it is composed of multiple smaller helper functions.

> It is important to note that the framework of Peircan semiotics goes beyond the concepts of object, interpretant, and representamen. For instance, Peirce discusses concepts such as Firstness, Secondness, and Thirdness and introduces various jargon. As it turns out, one of his jargons did make its way to computer science: **icon**. The term **icon** was introduced to computer science by a user interface designer at Xerox (Blackwell, 2024). Based on my research, it is unclear whether the unknown designer was thinking of Peirce or not. Perhaps, this word was derived from the other definition of icons. However, it is clear that the computer science's version of icon is slightly different from what Peirce had in mind. In computer science, an icon is a small graphic that represents software or some of its functions (Shirk & Smith, 1994). However, in Peirce's mind, an icon is a simplified representamen of a more complex object. For instance, the goose emoji � is an icon of Fig. 1. However, the trefoil represent radioactvity ☢️ is not an icon. Rather, it is a symbol, because it does not bear any resemblance to radioactive particles.
>
> Peircan semiotics also has quite an influence in other areas of computer science, such as visualization and human-computer interaction. However, we will not discuss the application of Peircan semiotics here.

Non-Language Semiotics

Ceci n'est pas une pipe
René Magritte (1929)

Since the emphasis of this book is on strings, we have not discussed much on how semiotics is applied to other types of media, such as images and sounds. As it turns out, semiotics could also be applied to those areas as well. Semiotics analysis is often used to analyze advertisements, especially analyses of cigarette advertisements. One of the latest examples is by Hartono & Septiani (2024), who analyzed cigarette billboards in Indonesia. Since cigarettes are deemed as dangerous products by various governments, the advertisers of such products must find ways to get their messages across while trying to be as covert as possible. Therefore, they cannot directly advertise their products, but instead try to create some vague association that their product represents "coolness" in some way. For instance, in Hartono & Septiani (2024), they note that a billboard does not directly advertise the L.A. Bold-branded cigarettes. Instead, the advertisement has a tagline: “Starting from km 0,” signifying the start of a journey. This notion of journey is vague, but the viewers of the billboard may think that, since being on a journey is “cool”, then so is smoking an L.A. Bold cigarette. Meanwhile, a country with strong anti-cigarette policies like Canada completely rubbishes any kind of indirect message as it forces the manufacturers to put real-life photos of dissected lungs riddled with diseases, and other biopsy pictures of dead smokers. In a sense, Canada also plays a semiotic game. Here, the signifier is the cigarettes, and the signified is *a horrible and painful death*. (No example images are provided here due to their graphic nature.)

Another famous example is the painting named “The Treachery of Imagery” (Magritte, 1929), which can be found in Fig. 2. It is a famous painting of a pipe with a caption: “Ceci n’est une pipe” or “This is not a pipe” (Aiello, 2020). The statement which can be true and false at the same time. It is true because the “pipe” is a painting, and no one can force it to function like a real one. It is a signifier, or a representamen. However, if we see a real pipe, would that not be the same? Our vision of a pipe is not the pipe itself. A pipe truly becomes a pipe once someone starts to smoke from it. Or is it not?

Figure 2. *The Treachery of Images by Magritte (1929)*

As a computer is a semiotic machine, it can also processes image. Just as with natural language, the images must first be encoded. Fortunately, image files themselves can be coded.

Deep Semiotics

So far, we have discussed semiosis that converts one sign to another. However, all semiosis processes seem to be moving in a lateral fashion. For instance, to know that “goose” the word is a signifier for “goose” the concept, we must first know how to read the word “goose.” Whether the word is read from an online version of this book or an offline one, some of the reader's retina must have been activated. The activations then cause a cascade of effects within the reader's brain, including neuron activations. These actions themselves can be construed as semiosis in a way. In neither Saussurean semiotics nor Peircan semiotics will we find the way to model these tiny semioses. For Saussure, deep semiotics was never the goal to begin with. He was developing a tool that would be useful for analysis. According to Nadin (2007), Saussure may have never intended to be used beyond linguistic analyses. Meanwhile, Perican semiotics is an abstract process that is independent of how the brain actually works (Feshchenko, 2015).

There is a philosopher who cares about the level of depths named Shpet (Feshchenko, 2015). Unfortunately, his work remains obscure and underdeveloped because of his untimely death at the hands of the Soviet regime (Feshchenko, 2015). First of all, for Shpet, there are varying levels of semiosis. However, there must be a limit on the depth. For instance, we cannot simply say that a semiosis is a series of quantum interactions. Secondly, humanity is important as well. Similar to the concept of umwelt, the ability of an organism to perform a semiosis is tied to its own biology. However, Shpet also emphasizes that lived experience plays an important role. Lastly, Shpet also argues for inner and outer forms. An inner form is abstract and cannot be encoded into language (e.g., verbalized), while an outer form is encoded (Feshchenko, 2015). For example, the gooseness of the goose image is an inner form while the goose image is an outer one. Because Shept's incomplete model of semiotics involved the depth of human experience and tried to put a limit on the depth of unlimited semiosis, Feshchenko (2015) calls his brand of semiotics **deep semiotics**.

Pragmatics

Although semantics is a complex field that we have barely scratched the surface of, I think it is time to move on to another important topic in semiotics: pragmatics. Usually, pragmatics deals with how multiple people can deal with signs. An example of pragmatics is deixis, which involves words that can change their meaning depending on the context within a discourse (Ekowati & Sofwan, 2014). For instance, pronouns are deictic words because in each conversation, a pronoun may refer to different people. For instance, if we are having a conversation about Peirce, we may use "he" to refer to him in a conversation. However, if we then start talking about Saussure, then "he" may refer to Saussure at the same time. Sometimes, we may need to juggle multiple "he", "him" or other inflected forms in the same sentence. Apart from personal deixis, there are also other types, such as place deixis, which indicates relative locations. For instance, "here" usually refers to the location where the discourse is taking place (Ekowati & Softwan, 2014). Identifying what a deictic word refers to within a chatbot is a difficult task because ambiguity can often occur.

In addition to deixis, we have another important concept: the cooperative principles. As it turns out, in all conversations, all speakers are cooperating with each other – even if they do not necessarily like each other. For instance, let's talk about how a student may request an extension from a rather stern instructor. In a normal conversation, it may go like this:

- **Student:** I have been very ill over the past few days. Could you help me out by giving me a 2-day extension?
- **Instructor:** No.

The student did not want to get what they want, and the instructor did not seem to think too highly of the student. However, there are still elements of cooperation. Otherwise, the conversation would have been extremely incomprehensible. According to Paul Grice, there are four maxims (Bjorkman, 2018):

- Maxim of quantity: Do not provide too much information.
- Maxim of quality: Do not state what is false, or do not state anything that has no evidence.
- Maxim of relevance: State what is relevant.
- Maxim of manner: Use words that are appropriate (i.e. no jargon), avoid ambiguity, be brief, and be orderly.

Now, let's examine how a violation of each maxim can affect the conversation. First, we assume the **violation** of the maxim of quantity:

- **Student:** I, referring to myself, Mr. Tim Jamison, have been very sick with multiple conditions. These conditions include runny noses, headache, sorethroats. The afflictions have rendered me incapable of … Could you, Prof. Bill Williamsworth, provide me with additional times to complete the assignment that you have created, with 2 days extension, which will afford me much relief.
- **Instructor:** I, Prof. Billie Williamsworth, cannot strongly oblige your request. I will not justify my reasoning. As such, there are multiple conclusions that you may draw. However, given your past histories of cheating and plagiarism, you can deduce yourself.

Secondly, we assume the violation of the maxim of quality:

- **Student:** I have been abducted by the aliens, who then proceed to make me ill, please give me a two-day extension.
- **Instructor:** I will give you an extension [beat] not.

It is important to note that a maxim can either be violated or **flouted**. In this second example, the maxim is violated if the student is just blabbering on about being abducted by the aliens without any thought or consideration. However, if the student is purposefully lying to the instructor, then this is considered a flouting (Bjorkman, 2018).

Thirdly, we assume the violation of the maxim of relevance:

- **Student:** I hate the cafeteria food, because it is not good. I need an extension. I am an organist, but sometimes, I play a video game instead of an organ. I have been ill…
- **Professor:** I once won a pageant when I was 25 year-old, and I am considering a divorce. We must save the planet…

Fourthly, we assume the violation of the manner.

- **Student:** You must help me. A blood test provided by a doctor indicated that I have an elevated level of bilirubin.
- **Professor:** The answer is “I can.” There is a condition, however. I will need a doctor’s note before saying “yes.”

In this case, the student was being too technical with therm. Instead of stating the disease that they have, they use a technical term. The request is also very ambiguous because the student does not state what they want from the instructor. It could be anything from asking for permission to skip a class to asking to withdraw from the course. Neither the student nor the professors were brief in their answer. Lastly, the professor violates the sub-maxim of order by saying that they can assist the student before stating that the help is conditioned on the student giving them a doctor’s note. As a professor myself, I know that I can appear cold to students sometimes, and my retort would simply be: “Don’t hate the player, hate the game.” However, saying “yes” before requiring a doctor’s note is just evil.

As we can see, even if the speakers are not doing what the other want, they still need to cooperate in some way in order to accomplish something. Without any ounce of cooperation, the conversation simply becomes incomprehensible.

Consciousness Sold Separately

Although semiotics discusses the transformation of meanings, which implies consciousness, it actually does not require consciousness. For instance, when I see the spelling of words that I already know, I constantly recognize them without consciously thinking about them. This unconscious act of recognition is a semiosis. Thus, for a semiotician, Searle’s Chinese Room does not have much bearing on its development. Some semioticians like Shpet think consciousness is important in some semioses (Feshchenko, 2015). However, some semioticians take a more utilitarian approach to semiotics. They see semiotics as more of a tool to scaffold and explain sensemaking processes, rather than a formalization of cognitive science itself. Saussure is notable in this regard (Feshchenko, 2015).

References

Aiello, G. (2020). Visual semiotics: Key concepts and new directions. *The SAGE handbook of visual research methods*, 367-380.

Bjorkman, B. (2018). Pragmatics and the Cooperative Principle. In C. Anderson (Ed.), *Essentials of Linguistics*. eCampusOntario. https://ecampusontario.pressbooks.pub/essentialsoflinguistics/chapter/10-5-pragmatics-and-the-cooperative-principle/

Blackwell, A. (2014, January 1). *Visual Representation*. IxDF – Interaction Design Foundation. https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/visual-representation

Brent, J. Pursuing Peirce. *The Philosophy of C. S. Peirce 106*(3), 301-322. https://www.jstor.org/stable/20117493

Electronic Arts. (n.d.). *The Sims* [Video Game Series]. https://www.ea.com/games/the-sims?isLocalized=true

Ekowatiki, A., & Sofwan, A. (2014). The use of pragmatic deixis in conversation texts in "Pathway to English." *Lembaran Ilmu Kependidikan* 43(2).

Feshchenko, V. (2015). Gustav Shpet's deep semiotics: A science of understanding signs. *Σημειωτκή-Sign Systems Studies 43*(2-3), 235-248. https://www.ceeol.com/search/article-detail?id=457810

Hartono, Septiani, D. (2024). Semiotic analysis of cigarette advertising: a case study in South Tangerang. *Lakon: Jurnal Kajian Sastra dan Budaya 13*(2), 119-130.

Hayes, P. J. (1981). The frame problem and related problems in artificial intelligence. In *Readings in artificial intelligence* (pp. 223-230). Morgan Kaufmann.

Hu, S. (2025). Chatbots as Turing Machines. *2025 IEEE/ACIS 29th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. https://doi.org/10.1109/SNPD65828.2025.11253606

India Today Education Desk. (2026, February 23). Quote of the Day by Srinivasa Ramanujan: An equation for me has no meaning… In R. Chauhan (Ed.), *India Today.* https://www.indiatoday.in/education-today/gk-current-affairs/story/quote-of-the-day-by-srinivasa-ramanujan-an-equation-for-me-has-no-meaning-2872608-2026-02-23

Kilstrup, M. (2015). Naturalizing semiotics: The triadic sign of Charles Sanders Peirce as a systems property. *Progress in Biophysics and Molecular Biology 119*(3), 563-575. https://doi.org/10.1016/j.pbiomolbio.2015.08.013

Magritte, R. (1929). *The Treachery of Images* [Painting].

McCarthy, J. (1981). Epistemological problems of artificial intelligence. In *Readings in artificial intelligence*, (pp. 459-465). Morgan Kaufmann.

Nadin, M. (2007, January). Semiotic Machine. *The Public Journal of Semiotics I*(1), 57-75.

Nöth, W. (2011). Semiotic foundations of pragmatics. *Foundations of pragmatics*, 167-202.

Ontañón, S. (2018, September). Shrdlu: A game prototype inspired by Winograd's natural language understanding work. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment 14*(1), 268-270. https://doi.org/10.1609/aiide.v14i1.13009

Paolucci, C. (2021). The distinction between semantics and pragmatics: The point of view of semiotics. *Intercultural Pragmatics 18*(3), 293-307. https://doi.org/10.1515/ip-2021-2014

Parvizi-Wayne, D. (2025). What active inference still can't do: the (frame) problem that just won't go away. *Philosophy and the Mind Sciences*, *6*, 1-26.

Popescu, V. D. (2022). Unlimited semiosis of endless redescription? Peirce, Eco, and Rorty on the limits of interpretation. *Revue Roumaine de philosophie 66*(1), 85-103.

Shirk, H. N., & Smith, H. T. (1994). Some Issues Influencing Computer Icon Design. *Technical Communication 41*(4), 680–689. http://www.jstor.org/stable/43089751

Susen, S. (2018). Saussure, Ferdinand de. In B. S. Turner, C. Kyung-Sup, C. F. Epstein, P. Kivisto, W. Outhwaite, & J. M. Ryan (Eds.), *The Wiley Blackwell Encyclopedia of Social Theory,* pp. 2001-2006. John Wiley & Son.

Media Attributions

- An image of Canada geese.
- MagrittePipe

IV

Computers vs. The World

11.

Chatbot v. the World

夢見た世界が どこかに あるなら
探しに 行こうか 風のむこうへ
凍てつく夜明けの
渇いた真昼の
ふるえる闇夜の
果てを見に行こう
Lyric from Tabi no Tochuu (Kiyoura, 2008)

At this point in time, we have not discussed how AI would fare in the real world. While we discussed how an AI may accept an input from the external world and convert it, we have not discussed how the AI would obtain the input or how other entities in the external world would deal with the output. We have hinted that for an AI to understand something, the information from the external environment must be converted into a form that the machine can work with. However, the discussion was not extensive. As it turns out, processing data from the external world is actually extremely important – even for Weak AI. For instance, an image classifier will work much better if the data have been filtered in some manner. Techniques such as convolution, when applied to the input image or within the layers, can eliminate some noisy artifacts, boosting the performance of the model.

We have already introduced and discussed two branches of cognitive science: Classical, and Connectionism. However, there is a third type: Embodied. This type of cognitive science has already been hinted at in some previous chapters, but it has not been fully introduced. Essentially, embodied cognitive science sees AI as an entity that is situated within the physical world. It operates as part of the world, and not outside of it (Dawson, 2013, Chapter 5). To motivate our discussion of Embodied cognitive science, I will introduce a chatbot with the most advanced LLM. The chatbot will then be strapped to a humanoid robot body. Then, we start to see it fail.

Chatbot vs. the Church-Turing Thesis

I have been using LinkedIn quite a bit. As a professor, LinkedIn plays an important role in broadcasting my work. Unfortunately, I must also contend with posts that discuss AI. Several posts seem to suggest that we can just strap a LLM-based chatbot with a robot body, and then it should work well enough. However, I argue that this will never be so simple. First, the chatbot must confront the Church-Turing Thesis. The Church-Turing Thesis provides a rather outdated view of how computers should work. Modern computers, on the other hand, are too powerful to be captured by the Thesis (Goldin & Wegner, 2008). However, I argue that chatbots cannot really overcome the Church-Turing Thesis due to how it works.

First, we use a chatbot by inputting the text, and the chatbot takes its time to convert the input into the output. During this time, the user will see a waiting symbol, and they cannot really do much with the chatbot itself. For instance, if I ask it to convert Maud Lewis's artwork into a realistic style (Fig. 1), the chatbot will not be available for anything else. The only thing I can do is to halt the process. While a Turing machine cannot be rudely interrupted, I would say that I can interact more with a real artist. For instance, if I see that the artist is not drawing the cat in the way that I like, I can ask the artist to design on the fly. Whether the artist is happy about my interruption or not is another matter. Meanwhile, if the AI fails to live up to my expectations, all I can do is to furiously vent out my frustration elsewhere.

Figure 1. *An AI-generated image based on the following input: Maud Lewis's cat image with the promt: "Draw this in a photorealistic style." The AI image was generated on January 19, 2026 with Google Gemini Nano Banana Pro.*

The ability to influence computers while it is working is called *interaction*, something that Goldin and Wegner (2008) argue to be extremely important for modern-day computers. All modern-day computers are interactive in some ways. The modern day computers can do many things that Turing machines cannot do. For instance, it can accept mouse, touch, keyboard inputs, and more and incorporates them into its computation. Meanwhile, the only interaction allowed by a Turing machine is feeding it with a tape. In Hu (2025), I argue that modern chatbots work more like Turing machines. Once an input is given, there is not much we can do. Some chatbots may be able to carry the memory of the previous prompts and conversations. However, these can be models as Turing tapes extended with past data.

Fortunately, there are already works in computer science that are breaking the Thesis, and I believe that most computer scientists working in these areas may not even be aware that they have been trying to chisel the field out of the confine of the Thesis.

First, there is the field of robotics, which deals with how a robot navigates in the physical environment. For a chatbot to become a robot, it must go through some transformation. First, a chatbot is a type of bot which is only meant to navigate through the computational environment. Meanwhile, a robot is situated within the physical world and has sensors to ensure that it reacts to the changes within the real world as well as affects it (Poole & Mackworth, 2023, Chapter 1). To make a bot into a robot is a complex process. This is because there are more assumptions than ever before. As we will see in this chapter, logically defining these assumptions and constraints will not be easy.

Secondly, an operating system deals with how a computer manages its own resources. It is essentially an art and science of how to extend a universal Turing machine to make sure that it can juggle between multiple Turing machines and simulate them in the most efficient manner possible. To pursue efficiency, the universal Turing machine has been transformed into something far more. Not only can an operating system schedule multiple machines, it also allows user to constantly mess around with its inner workings – up to a limit. The operating system also manages memories and ensures that no programs will take too many resources. Although the field of operating systems is not necessarily a field of AI, I came to the realization that the journey towards Strong AI must incorporate the wisdom gained by the operating system research. I am not the only person to make such an argument. In a paper that has not been peer-reviewed, Lorentzon & Grewlin (2025) also make a similar argument.

Thirdly, there are also areas of human-computer interaction (HCI) which directly study how computers and humans can interact in a synergistic manner. Although much of HCI relies on empirical research, I argue that empiricism still plays a very important role. There are simply many phenomena that we do not have good computational models to represent them. For instance, how can we model social media addiction when social media is such a new phenomenon? While we can wait for someone to develop a solid framework, waiting is not an option. As social media addiction starts to wreak havoc on society, we must act. Imperfect actions without any formal actions. HCI also develops new modes of interaction that allow us to stretch the limits of what computers can and cannot do. As HCI is my field of specialty, there is more for me to say, but I will refrain from doing so.

For a chatbot to become a Strong AI robot, it must break the Church-Turing Thesis. Breaking the Thesis is not as difficult as some computer scientists think. However, breaking it is not a straightforward process. We cannot attack the Thesis head-on. Instead, we must spend time understanding the Thesis and find its weak points. This is like attacking a fortress by using infiltrators instead of a full-frontal assault.

Making Inferences

Assuming that we can convert a chatbot into a robot (and not just by strapping it onto a robot body), it must now be able to make inference the world around it. Regardless of how the robot is trained, it must make decisions based on the world surrounding it. In a previous chapter, we discussed automated reasoning. Here, we will explore how it can apply to making inferences about the world.

Regardless of how the robot is implemented, it should possess a knowledge base, which is a set of propositions. Then, for each piece of external information, it checks if the knowledge bases entail the information or not. For instance, when it is trying to buy ingredients for an omelette, it scans the items at the grocery stores and checks its knowledge bases. It then considers if the items make sense or not. The knowledge bases should entail ingredients like eggs, cheese, mushrooms, and other items that are used for making omlette.

$$Omelette \models HasEgg \wedge HasCheese \wedge HasMushroom \wedge \ldots$$

It is important to note that an entailment is a type of implication with $\models$ being similar to $\longleftarrow$.[1] Therefore, if the knowledge base is not applicable, then it does not render it false. For example, if the robot gets an incorrect ingredient, then the knowledge base for the omelette remains vacuously true. A knowledge base does not entail a statement if it could be false in some interpretation of the knowledge base (Poole & Mackworth, 2023, Chapter 5.1). We know that there are many variations of omelettes. Some may prefer it plain, and some may prefer it with other toppings. However, there are some ingredients that will never go into an omelette. For instance, if the robot fails to find bell peppers in the grocery store and it wrongly substitutes them with the plastic replica from a furniture store, then it does not make an omelette anymore. Instead, it makes flavoured melted plastic.

To create a knowledge base, classical cognitive scientists would be content with simply programming all propositions inside a knowledge base. However, modern cognitive scientists do not believe that everything in the world is fixed. Even though we use knowledge bases to reason about the world, more recent theories, such as the prototype theory and the *theory* theory suggest that knowledge bases must be flexible. A knowledge base seems to be created using pre-existing knowledge or information in an ad-hoc and context-sensitive manner.

To understand the flexibility of a knowledge base. Let's imagine that the robot has to cook a Thai omelette instead. In this case, the knowledge base will be extremely different from the Western omlette. While both types of omelettes contain eggs, other details are extremely different. For example, a Thai omelette must be fried in a neutral oil (or perhaps lard), but not in butter. The salting agent used is fish sauce or soy sauce. Therefore, for the robot to make an appropriate

1. Please note that the arrow is leftward – i.e. reversed.

omelette, it must take the environment and cultural cues as well. This is not a straightforward affair. For instance, one may think that the robot should just make a Thai omelette if it is in Thailand. However, what if the robot's master is not Thai, but a foreign tourist? Should it still make a Thai omelette? Perhaps, the master wants to enjoy some Thai food in order to vibe better with the environment. In this case, the robot should make the Thai omelette. However, if the master just wants a break from exotic local cuisine, then the robot should make a Western omelette. Or what about a hybrid? Although ham is not part of Thai cuisine, it may be acceptable to add it if the master really wants some protein in their omelette and other more authentic proteins, such as minced pork or shrimps, are not available.

We cannot program all of these away, McCarthy and Hayes (1969) argue that. However, we also cannot say that logical knowledge bases do not exist either. Yes, we can make variants of omelette, but adding a bell replica would instantly ruin the entailment.

Frame Problem and Relevance Realization

As it turns out, regardless of the architecture being used, making inferences about the world is actually extremely difficult. In AI, the chatbot must deal with something called the Frame Problem. The Frame Problem was introduced by McCarthy and Hayes in 1969. They argue that when we create an AI program using Classical AI to perform an action, we must also account for events that are unchanged, and these cannot be modelled using first-order logic (McCarthy, 1981). Hayes (1981) provides an example of this. Imagine that a child is stacking blocks on top of each other. Then, we tell the child that Block B can be stacked on top of Block A, if Block A does not have any other blocks on top of it. If Block B happens to already be on top of other blocks, the child will know that the blocks will not be affected by the stacking actions. When we create software to stack Block B on top of Block A, the AI may not have the fact that the other blocks will not be affected in the knowledge base. In other words, it may not know what remains unchanged in the world. While the AI may not be aware of the change, the world itself can also change while the action is performed. For instance, if the child has successfully stacked Block B on top of Block A without accounting for other blocks, logically, the other blocks could have been swapped. In the real world, this is very difficult to imagine. The child would have noticed another person performing the swapping, or have been spooked by ghostly actions manifesting in front of them. Logically, the phantom swapper would not have been accounted for.

While McCarthy (1981) argues that Classical AI would not have been able to overcome the Frame Problem, it turns out that it can be addressed, to a degree. In Weak AI, we simply program or model harder to account for all possible facts about the unchanged environment, for its context (Shanahan, 2016). An example of this is the self-driving vehicle. When such a vehicle is operating, it must execute actions based on the ever-changing world. Although imperfect, the programming and the modelling are sufficiently good for the vehicles to operate without human input most of the time. In this case, the designers of the vehicles restrict the knowledge domain of the vehicles. As far as the vehicles know, the roads are their entire universe. There are cases, though, where this does not work out – e.g., a self-driving taxi drove itself onto a light rail track and stopped for no particular reason (Landymore, 2026); the Frame Problem is still there. However, had the Frame Problem been so intractable, then a self-driving vehicle would not have been able to leave its parking spot to begin with.

Related to the Frame Problem is Relevance Realization. However, before discussing Relevance Realization, we must be aware that the Frame Problem, as stated by McCarthy and Hayes, is not necessarily the same one in cognitive science. In cognitive science, the Frame Problem has been expanded to discuss how an organism must be able to select its frame of relevance and discard irrelevant information (Parvizi-Wayne, 2025). For instance, as I type out the text in this book, I am also constantly distracted by the environment. I can hear the heater running, and someone clearing out the dishwasher. Outside the window, I can also see drifting snow. Yet, I was able to focus on typing the text. And yet, when I type, I do not focus on my exact finger positions. Instead, I just rely on the implicit skills that I have learned since elementary school. If I cannot hone in on the task at hand, I would not have been able to type. In fact, I would not be able to function at all, as there are myriad things in the environment. Thus, as Vervaeke (1997) argues, organisms must be able to perform Relevance Realization, i.e. identifying on what is relevant for the cognitive process.

At this point in time, I am afraid to say, we have not truly solved the problem of Relevance Realization. I note that in order to determine if something is relevant or not, we need to be conscious. And we do not have a good account of consciousness. We know when we are conscious. For instance, we know that we are less conscious when we are struggling to listen to a lecturer droning on about their topic, but we can never truly describe the word. Perhaps, Descartes is right. All of this may require God. Cognitive scientists are against this; after all, we are Hobbesian in nature. But we must still find a robust way to dispute dualism. I believe the problem is that we are using languages, be it formal or natural, to describe all phenomena. However, our cognition is not obligated to be explainable.

Learning from the World

Any robot that wishes to be human-like must, of course, behave like a human. And we, humans, are capable of learning from the environment. Although we have previously discussed machine learning, that particular brand of machine learning requires us to wait for the machine learning program to read from the data and adjust the weights of the models. In the meantime, humans and natural cognitive systems *mostly* learn in real-time and within the environment itself.

> I wrote *mostly* on purpose. When I was an undergraduate student learning about cognitive science for the first time, Prof. Vervaeke introduced me to the concept of wake-sleep algorithm during his lectures. This algorithm was developed by Hinton et al. (1995). The name "wake-sleep", according to Vervaeke, was chosen because Hinton believed it could form the basis of a computational theory of dreaming. Unlike a classification neural network model, there is no target output. Instead, the model is trained to reconstruct its own input, and use the reconstruction to train all the layers below. Without target outputs to supervise the learning process, the wake-sleep algorithm is unsupervised. This insinuates that dreaming is simply the brain training itself to generate information. I do not believe that psychologists today believe that we dream like the wake-sleep algorithm. The wake-sleep algorithm is not totally useless, however. Although the wake-sleep algorithm is no longer deemed a viable theory of dreaming, it seems that dreams are still relevant to cognition. Also, the wake-sleep algorithm serves as an earlier form of generative AI.

We will not discuss learning deeply in this book. However, we will still discuss **behaviourism**, not because it is important learning. However, it is because Chomsky's criticism of it is so devastating that it reverberates through multiple fields. Behaviourism is a branch of psychology, rooted in the philosophy that psychology should be about studying behaviours, instead of consciousness. Psychology itself should become objective and should not discuss that which can be deemed as subjective in nature (Baum, 2017, Chapter 1). This means that behaviourism is similar to cognitive science in the sense that it wishes to banish subjectivity, intention, and consciousness. However, while cognitive science tries to tackle this head-on through a constructive approach, behaviourism sidesteps these concepts altogether.

Those with a precursory knowledge of behaviourism may be very familiar with Pavlov's dog. In the observation, Pavlov observed that his dog produced extra saliva when it heard a sound signalling that food would be served soon. By repeatedly ringing a bell right before feeding, the dog associated the sound of the bell with feeding (Baum, 2017, Chapter 4). Conditioning, ultimately, involves the following:

- **Reward:** If an agent behaves well, then it receives a reward to encourage it to repeat the behaviour.
- **Penalty:** If an agent behaves poorly, it receives a penalty, which

discourages the behaviour.

Rewards and penalties can be positive or negative. It is important to note that "positive" or "negative" could probably be better described as "additive" or "subtractive." Positive means the agent is receiving something in return; meanwhile, negative means something is removed. Either can be for better or for worse. A positive reward means the agent is receiving something, like food or praise. A negative reward means the agent has something undesirable removed. For instance, if a child misbehaves, their parents can ground them so that they do not get to have fun outdoors. A positive penalty involves punishing the agent by adding an annoyance. For instance, if a student plagiarizes at some universities, the instructor may ask the student to redo and resubmit the work. Lastly, a negative penalty involves punishment by removing something from the environment. A misbehaving child have its toy taken away.

Later behaviourists would expand his work, and some psychologists, like Skinner, would expand this much more to include many types of learning. Skinner (1957) wrote a book called *Verbal Behaviour*, which proceeded to receive a devastating review from Chomsky (Baum, 2017, Chapter 7). The thesis of the book was basically that learning a language involved a series of conditioning. The child, who started out with no knowledge of language, would learn simply by being rewarded or punished by their parents. However, Chomsky argues that all of us have an innate capability to use language. Although a newborn baby cannot utter a coherent word, the capability to produce words is already there as a basic algorithm. This criticism was so devastating that, according to Baum (2017, Chapter 7), psychologists avoided studying language acquisition through behaviourism for a long time.

Now, just because Chomsky, whose ideas form the foundation of this textbook, makes a devastating argument, it does not mean that behaviourism is completely incorrect. We do indeed learn through conditioning in some scenarios, though not all. For instance, when we try to court another person, we may do so through pick-up lines. If the lines are corny, we get rejected. If the lines are good, we *may* be able to proceed with other courtship activities. Thus, a rejection serves as a way for us to learn better pick-up lines or completely change our courtship strategies. Although behaviourism may help us to become better, it is not entirely correct. We cannot simply treat romance as a series of transactions. Unlike in video games, repeatedly recycling supposedly rewarding behaviours to the one we try to court will not necessarily achieve the same desirable outcome.

Concepts in behaviourism are also relevant to the development of reinforcement learning (Maia, 2009). Unlike supervised machine learning, reinforcement learning involves an agent being trained in an environment, which may or may not be virtual. Reinforcement learning, like conditioning, also involves rewards and punishments. Although difficult to set, reinforcement learning can serve as a viable way to train an embodied agent.

Since Chomsky has provided a devastating criticism of behaviourism, it would be fair for us to ask, where is the universal grammar? Of course, nobody, including Chomsky, believes that we have literal machines that produce sentences by reading a very long tape and writing new outputs over it. However, would we be able to find a close approximation within our brains? Dąbrowska (2015) argues that it is not possible to find it.

Semiosis Begets Semiosis

Now, one may ask: What about semiotics? Do the Frame Problem and Relevance Realization also apply to semiotics as well? The answer is “no”, and “yes.” No, in the sense that not all types of semioses require consciousness. Nadin (2007) argues that a semiosis could just be as soulless as moulding clay into sculpture, or a Turing machine processing a tape. Thus, semiotics is mostly fine without any consciousness. Most things that we do are simply unconscious anyway. For instance, when we see and read a sign, the semiosis happens automatically without us actively and consciously trying to read.

However, parts of semiotics must be conscious. For instance, when a person is speaking, we must be aware of how that person speaks. If we are not conscious of how the person speaks, then we do not understand the person. Furthermore, some semiosis is intentional.

How’s Your Umwelt?

In previous chapters, we discussed semiotics and Umwelts. These are also important concepts when it comes to understanding the world around us. In a previous chapter, I argued that a multimodal chatbot has a richer umwelt than a completely text-based chatbot, because it can generate more types of signs and is more capable at semiosis. However, there is another way to limit an umwelt, by constraining the environment itself. Computer scientists in the past, e.g., SHRDLU by Winograd, constructed a small program that represents the world. The virtual world contains a few items and a bot. The user can instruct the bot to deal with the object (Ontañón, 2018). Microworlds are constructivist in nature; we learn by building. The hope was that if the bot manages to become a master of a microworld, we can incrementally upgrade the world. Eventually, the microworld itself becomes something akin to the physical world. In a sense, a microworld is basically a laboratory, and the bot serves as its subject for experimentation.

However, we develop our ability to glean meaning and perform semiosis by being born into the environment. The world does not expand with us. Although one may argue that the womb serves as a microworld, the womb itself does not perform much. It does not challenge its occupant by forcing them to solve virtual puzzles and more. Furthermore, the womb itself does not expand into the real world. Instead, once its baby is about to be born, it seems content to flush its content. Furthermore, it is unclear how to expand our microworlds. Parvizi-Wayne (2025) argues that there are simply too many details for us to think. Let's look at the Sims (Electronic Arts, n.d.), a video game series, as an example. In the game, the player is basically in control of a microworld with multiple bots. Ever so often (or extremely frequently to some fans), Electronic Arts would release an expansion that extends the functionality of the microworld. If the microworld project can be completed, then the Sims would have had all the possible expansions that represent everything about our world. But it has not, and it will not. Therefore, a Sim's umwelt will never match an embodied robot's umwelt.

As such, I expect a bot living in a virtual world will have difficulty adapting to the real world.

References

Baum, W. M. (2017). *Understanding behaviorism: behavior, culture, and evolution.* Wiley.

Dawson, M. R. W. (2013). *Mind, Body, World: Foundations of Cognitive Science.* Athabasca University Press. https://www.aupress.ca/books/120227-mind-body-world/

Dąbrowska, E. (2015). What exactly is Universal Grammar, and has anyone seen it?. *Frontiers in psychology 6*(852). https://doi.org/10.3389/fpsyg.2015.00852

Electronic Arts. (n.d.). *The Sims* [Video Game Series]. https://www.ea.com/games/the-sims?isLocalized=true

Goldin, D., & Wegner, P. (2008). The interactive nature of computing: Refuting the strong Church–Turing thesis.*Minds and Machines18*(1), 17-38. https://doi.org/10.1007/s11023-007-9083-1

Hinton, G., Dayan, P., Frey, B. J., & Radford, N. (1995). The "Wake-Sleep" Algorithm for Unsupervised Neural Networks. *Science 268*(5214), 1158-1161. https://doi.org/10.1126/science.7761831

Hu, S. (2025). Chatbots as Turing Machines. *2025 IEEE/ACIS 29th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. https://doi.org/10.1109/SNPD65828.2025.11253606

Landymore, F. (2026, January 14). Video Shows Passenger Bail From Waymo in Path of Oncoming Train. *Futurism.* https://futurism.com/advanced-transport/passenger-bail-waymo-train

Kiyoura, N. (2008). Tabi no Tochuu [Song]. On *Tabi no Tochuu*. FlyingDog.

Lorentzon, K. M., & Grewlin, A. S. (2025). From Agents to Cognitive

Operating Systems. https://grewlin.no/onewebmedia/CognitiveOS_K_Martin_Lorentzon.pdf

Maia, T. V. (2009). Reinforcement learning, conditioning, and the brain: Successes and challenges. *Cognitive, Affective, & Behavioral Neuroscience 9*(4), 343-364.

McCarthy, J. (1981). Epistemological problems of artificial intelligence. In *Readings in artificial intelligence,* (pp. 459-465). Morgan Kaufmann.

Nadin, M. (2007, January). Semiotic Machine. *The Public Journal of Semiotics I*(1), 57-75.

Ontañón, S. (2018, September). Shrdlu: A game prototype inspired by Winograd's natural language understanding work. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment 14*(1), 268-270. https://doi.org/10.1609/aiide.v14i1.13009

Parvizi-Wayne, D. (2025). What active inference still can't do: the (frame) problem that just won't go away. *Philosophy and the Mind Sciences*, *6*, 1-26.

Poole, D. L. & Mackworth, A. K. (2023). *Artificial Intelligence: Foundations of Computational Agents, 3rd Edition.* Cambridge University Press. https://artint.info/3e/html/ArtInt3e.html

Shanahan, M. (2016). The Frame Problem. In E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy* (Spring 2016 Edition). Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/archives/spr2016/entries/frame-problem/

Skinner, B. F. (1957). *Verbal Behavior.* Copley Publishing Group.

Vervaeke, J. A. (1997). *The Naturalistic Imperative in Cognitive Science.* [Doctoral dissertation, University of Toronto]. TSpace. http://hdl.handle.net/1807/10696

Media Attributions

- maud_real

12.

Combinatorial Explosion, Heuristics, and Rationality

Tell me, Riddler, how can you split your focus in thirty-two directions and sustain the concentration it takes to keep your world together? You can't!
A quote spoken by Batman from Episode 45 of *Batman: The Animated Series* (Issenberg, Skir, & Sebast, 1992)

In the last chapter, we briefly discussed the Frame Problem and Relevance Realization. These suggest that a real cognitive system, not an artificial one, has an effective mechanism that selects what to focus on. These were framed in the context of performing semiosis and designing an AI that can deal with the physical world. Without an effective mechanism, an AI will quickly face a large number of decisions that it must make. The number of decisions grows exponentially, so it faces something called a **combinatorial explosion**. Now, let's assume that the Frame Problem, in all cases, has been solved, and we can make an AI that can focus on certain things. Does the issue of combinatorial explosion still exist? The answer is: "Yes." As it turns out, there are some problems that are inherently difficult to solve. Even if the AI can perfectly understand all parameters, it will struggle.

Review of Worst-Case Algorithm Complexities

In a computer science program, students will often face the worst-case complexity computation of algorithms. We categorize software based on the number of steps they need to complete based on the input size on a Random-access Machine (RAM). Students should be well familiar with the following algorithms and their associated complexities:

- **Binary search:** $O(\log n)$
- **Linear search:** $O(n)$
- **A Good Sorting Algorithm:** $O(n \log n)$
- **A Bad Sorting Algorithm:** $O(n^2)$
- **A naive scheduling of tasks:** $O(2^n)$

For those who are unfamiliar with the Big-O notation, we can determine the slowness of an algorithm by looking at the function inside. Usually, any function that is “smaller” than $O(n^2)$ or a quadratic function will be deemed as good.

Now, is this system of ranking the slowness of the functions perfect? Absolutely not. There are algorithms that are ranked as fast, but turn out to be very impractical. An example of this is Radix sort, which runs in $O(n)$ steps. However, in practice, the algorithm itself is beaten by Quicksort. In the worst case, Quicksort runs in $O(n^2)$ and can be extremely slow. On average, however, the algorithm is substantially faster. Radix sort is also quite impractical as well. Although it runs in $O(n)$, it also requires all unsorted items in a data structure to have equal lengths. For instance, if we are sorting strings with a maximum length of 1,000 characters, then all strings must be padded with blank characters to ensure that they are all 1,000 character-long. This must be done even if the average length is much shorter (McIlroy, Bostic, K., & McIlroy, 1993).

The Slow Algorithms

Now, there are some problems that are just inherently difficult to solve. Let’s assume that we have an ideal microworld where the AI agent can perfectly understand what is going on and perfectly understand the input. Then, the user asks the AI agent to solve the Tower of Hanoi problem. The Tower of Hanoi is a game with a very simple premise (Carrno & Henry, 2018, Chapter 14):

- There are three pillars.
- There are blocks of varying sizes all stacked at one of the pillars.
- The blocks are stacked from the smallest to the largest.
- The goal is to move the blocks from the first pillar to the third one while obeying the movement rule: (1) only a single block can be moved, (2) the smaller block cannot be stacked below the larger one, and (3) all pillars can be used to temporarily store the blocks.

The AI will still struggle to solve this problem, because its runtime is $O(2^n)$. It will always take $2^n - 1$ steps to solve this problem.

There are also many real-world problems that will likely run at $O(2^n)$, because they are classified as NP-Hard or NP-Complete, or even worse[1]. For

1. I am aware that NP-Hard or NP-Complete are for decision problems. The usage of complexity classes will be *colloquial* for now. We will discuss the terms formally in the next chapter.

instance, the knapsack problem is considered to be NP-Complete. This problem simply involves trying to fit as many items as possible into a container while also trying to maximize the total value of the items (Karp, 2009). This means, the algorithm's worst-case run time will likely be $O(2^n)$, and mankind will unlikely be able to develop more efficient algorithms.

> It is important to note that just because a problem is deemed to be NP-Hard or Complete, it does not mean that it is empirically as hard as the other NP-Hard/Complete problems. For instance, the knapsack problem does have a well-optimized algorithm to solve it based on dynamic programming (Cacchiani et al., 2022).

Some games, even the seemingly simple ones, such as the Geography game is PSPACE-complete. The game is very simple in nature: one player states the name of a capital city, and the other player must find another capital name that starts with the last character of the previous one. For instance, if the first player starts with "Nuuk", the capital of Greenland, then the other player may pick "Kuala Lumpur", the capital of Malaysia. Previously used names cannot be reused. The game goes back and forth until one player runs out of capital names. If we are just playing the game for fun, we probably do not care much about the strategies. However, a highly competitive player may anticipate all the future moves, even beyond the next turn, and pick the capital name that will cause the opponent to stammer the fastest. This problem of finding the most optimal capital name is PSPACE-complete (Fraenkel & Goldschmidt, 1987). This means the best algorithm may likely be slower than the best ones for NP-Hard/Complete problems.

It is important to note that a slow problem presented here is not the same as the undecidable problems. The slow problems are all solvable in all cases, while the undecidable problem means there are some cases that do not have clear answers. What we are discussing in this chapter is the concept of **intractability**.

Limited Search and Heuristics

If combinatorial explosion is still a problem even though the AI knows its relevance, then what should the AI do? First, the AI should try to limit its search space. In a very basic search strategy, an AI can imagine itself performing all possible tasks based on the current condition. For example, in a tic-tac-toe game, the AI may imagine not only what it wants to do, but also tries to imagine all possible future moves that it and its opponent may perform afterwards.

Although tic-tac-toe is PSPACE-complete (Byskov, 2004), due to its limited grid size, trying to anticipate all possible moves until the end of the game is not necessarily a bad idea. Therefore, the AI's best move is to simply search for the best possible move. However, unlimited searching becomes unviable when the problem becomes much more complex. Therefore, we must impose some control over search (Poole & Mackworth, 2023, Chapter 3).

One strategy is to use a heuristic function. A*, a variant of breadth-first search (BFS), also features a function that helps the algorithm in determining how the search space should be navigated. An example of A* is in solving a maze. Assuming that the AI agent is trapped inside the maze, but with the knowledge of the coordinates of the exit. The agent will try to walk towards the exit as much as possible. If it faces a junction, it will prefer the one that faces toward the exit (Poole & Mackworth, 2023, Chapter 3).

Humans also apply various heuristics in their lives. In cognitive science, a heuristic is essentially the antithesis of an algorithm. A heuristic is a method that allows us to reduce the cost of the resource, but does not guarantee the correct answer. A heuristic does not need to be logical in nature. Often, a human will refer to a heuristic as a hunch, or something similar. Meanwhile, an algorithm will try to find the right answer no matter the cost.

Local vs Global Solution

Local search is almost always the only available choice. As previously mentioned, while some problems, like finding the winning strategy for a Tic-Tac-Toe game, can be brute-forced, many games and real-world problems are too combinatorially expensive. Therefore, they must make the decisions at the local level. However, thinking locally means we may get stuck at a local solution. The solution may be best for the moment, but it may not be the actual best solution. This issue can also show up in connectionism. In machine learning, we often need to find the minimum of a function. Ideally, we want to use differentiation to solve the problem, as we did in high school or in the first-year calculus class. For instance, if our function is x^2, then we can compute for $\frac{dx^2}{dy}$. We get $y = 2x$, and we find x such that $2x = 0$. However, when we have a more complex data set and multidimensional data, we must use an approximation. The stochastic descent is one such algorithm. Depending on how we set the parameters, it is possible for the algorithm to get a false, i.e. local, minimum instead of the global one. Therefore, regardless of the type of cognitive science we practice, we must be aware of the local and the global search.

Rationality

According to Vervaeke, a key to distinguishing rationality from intelligence is to look at the stem of the word "rationality" itself: "ration." Rationality is about the rationing of the available resources.

At first glance, intelligence and rationality seem to be the same. An intelligent person should make a rational decision, and a rational person should make an intelligent decision. However, in cognitive science, intelligence and rationality are not the same. Intelligence is associated with the ability to solve problems. Meanwhile, rationality incorporates resource management during problem solving. For instance, if an AI agent is solving a knapsack problem intelligently, it will be able to pack the container specifically, regardless of the resource constraints. However, if an AI agent has a time or memory limit placed on it, like humans, then it may no longer be able to make the best decisions. An irrational agent that does not keep its resource limitations in mind will run out of resources and provide no answer. Meanwhile, a rational agent will aim to provide the best possible answer even if it may be wrong. I can also provide a more cruel example: let's have two robots solving a puzzle in a room. The room is then set on fire, and it is clear that the fire will burn quite quickly. An intelligent but irrational robot will continue to solve the puzzle. Meanwhile, a rational robot will realize that the problem will never be solvable, and it will try to run away to save itself first. There is no use competing against the other robot when the result will be burned to ash anyway.

Earlier on, in the development of the theory of rationality, cognitive scientists expected people to behave in a way that maximizes resources. For instance, when a person makes a decision, they will make the decision that will yield the long-term benefit over the short-term one. However, this theory of rationality fails to describe how people actually behave in the real-world. Often, people behave in a very short-sighted way. At first glance, we could argue that humans are not rational beings. However, Simon (2000) argues that humans, when making decisions, are also bounded by various factors. For instance, when a person agrees to sign a loan at a payday lender, they are not making the most utilitarian decision. The interests are high, and the debts can be difficult to pay off. However, if the person really needs the money at the moment and does not have a high enough credit score to open a line of credit, then they are bound to make such a decision.

Byskov, J. M. (2004). Maker-Maker and Maker-Breaker games are PSPACE-complete. *BRICS Report Series 11*(14).

Carrano, F. & Henry, T. (2018). *Data Structures and Abstractions with Java, 5e*. Pearson.

Cacchiani, V., Iori, M., Locatelli, A., & Martello, S. (2022). Knapsack problems — An overview of recent advances. Part I: Single knapsack problems. *Computers & Operations Research 143*, Article 105692. https://doi.org/10.1016/j.cor.2021.105692

Fraenkel, A. S., & Goldschmidt, E. (1987). PSPACE-hardness of some combinatorial games. *Journal of Combinatorial Theory, Series A 46*(1), 21-38.

Issenberg, M. (Writer), Skir, R. N. (Writer), Sebast (Director). (1992, November 4). What Is Reality? (Episode 45) [TV series episode]. In A. Luiting (Executive Producer). *Batman: The Animated Series.* FOX Kids.

Karp, R. M. (2009). Reducibility among combinatorial problems. In *50 Years of Integer Programming 1958-2008: from the Early Years to the State-of-the-Art* (pp. 219-241). Berlin, Heidelberg: Springer Berlin Heidelberg.

McIlroy, P. M., Bostic, K., & McIlroy, M. D. (1993). Engineering radix sort. *Computing systems 6*(1), 5-27.

Poole, D. L. & Mackworth, A. K. (2023). *Artificial Intelligence: Foundations of Computational Agents, 3rd Edition.* Cambridge University Press. https://artint.info/3e/html/ArtInt3e.html

Simon, H. A. (2000). Bounded rationality in social science: Today and tomorrow. *Mind & Society 1*(1), 25-39. https://doi.org/10.1007/BF02512227

13.

Return of Formal Languages

You underestimate how the smallest decisions can compound into significant differences over a lifetime. Every tiny decision creates another branching universe, another – were you not paying attention before?

A quote spoken by Alpha Waymond, a character in *Everything Everywhere All at Once* (Kwan & Scheinert, 2022)

It has been a while since we discussed the topic of formal language. While some students may be happy about such a topic not being mentioned again, they may be less thrilled to learn that formal languages actually have not gone away. In fact, it has been hiding and lurking beneath the surface. For instance, the classifier mentioned in the Connectionism chapter can be seen as a nondeterministic Turing machine that decides whether a set of pixels belongs to the formal language of cat images or not.

In the previous chapter, we briefly used the terms NP and PSPACE to denote difficult problems in a manner that would make a theoretical computer scientist cringe. Now, we will reintroduce these terms in a way that is more technically sound. However, before doing so, we must provide some background information on the following topics: decision problems, Boolean Satisfiability Problem (SAT), and Nondeterminism.

Background Information

Decision Problems

A decision problem is simply a problem with two answers: “Yes” or “No.” Theoretical computer science and formal language theories often deal with decision problems, because it is easier to write proofs with them. To better understand this, let’s re-examine the knapsack problem from the previous chapter. The knapsack problem from the previous chapter is not a decision problem because the answer is the list of items to be fitted into a container. Rather, it is an example of a search or an optimization problem. Still, we can reframe the problem as one (Karp, 2009; Kellerer, Pferschy, & Pisinger, 2024). I must note, though, that even though it is possible to convert many decision problems into search problems or optimization problems is not a straightforward process. My extensive literature review does not really tell me how to convert one to another.

Boolean Satisfiability Problem

The most fundamental decision problem is the Boolean Satisfiability Problem (SAT). The premise of the problem is really simple: given a Boolean proposition, can we substitute the Boolean variables in such a way that the formula evaluates to true? For instance, $a \vee \neg a$ will always be satisfiable regardless of how we set a. Meanwhile, $a \wedge \neg a$ is not satisfiable. So far, the instances are very simple. However, in the real world, this can be very complex. As it turns out, Cook (2023) implies that many computer programs can simply be turned into instances of SAT. Karp (2009) shows that the decision problem version of many real-world problems can be reduced to SAT and demonstrates that they are NP-Complete.

Nondeterminism

Do we have free will? Or is everything in our lives already pre-determined? We will not answer these questions. Instead, we will discuss determinism v. non-determinism instead. In computer science, a deterministic program will work in the perfect way. If we provide the same input to a deterministic program multiple times, then the output will be exactly the same. However, a non-deterministic program may yield different results. An example of this is a program that fits a machine learning model that trains a neural network to classify images. When the program runs, it can produce different machine learning models even if the same training set is used.

A Turing machine can also be designed to behave non-deterministically. Unlike the normal (a.k.a. deterministic) Turing machine, a nondeterministic machine has finite options when it performs an action. A deterministic Turing machine can write, move, and change its state based on its current state and symbol. However, it is limited to writing a *particular* symbol, moving in a *particular* direction, and changing into a *particular* state. A non-deterministic machine has options for each of the actions. Instead of just writing a particular symbol, it spawns multiverses. Let's say that the symbol set $\Sigma = \{0, 1, 2\}$, then it can spawn three multiverses: one where 0 is written, one where 1 is written, and another where 2 is written. The machine can also move left or right, or stay in the current cell. Each type of movement is its own reality. While the machine can be in one state in one reality, there are multiple realities, each with a different state. The string is considered to be accepted if it is accepted in one of the multiverse branches.

A nondeterministic Turing machine has another magical property: there is no penalty for spawning or travelling through the multiverses. A nondeterministic Turing machine is considered to run in polynomial time if it takes $O(n^k)$ steps to complete by traversing through the best-case multiverses – i.e. 100% of the chosen operations lead to the correct outcome.

Although a non-deterministic Turing machine sounds magical, it can actually be simulated by a deterministic Turing machine (Savitch, 1972). In Savitch's (1972) proof, he basically represents the actions and inner workings of a non-deterministic Turing machine as a maze. Each maze room represents a branch, and the goal inside the maze represents an acceptance of an input string. The maze is then encoded as a Turing machine tape, which a deterministic Turing machine must solve. This proof demonstrates that a non-deterministic Turing machine is no more powerful than a deterministic Turing machine in terms of accepting and rejecting strings.

Non-determinism is a concept that can throw many students off. However, students today are more familiar with non-determinism than they think. They spent time growing up watching a few movies with a nondeterministic plot. Basically, the hero or one of the heroes of the story has the ability to create multiversal branches based on the actions and events that happen within the story. For instance, when a character is thinking whether they should leave the house or not, there will be two universes being created. In one universe, the character leaves the house, and in another, the character stays. The hero or the heroes are considered victorious if they manage to vanquish the antagonists within one of the branches.

Class NP and NP-Completeness

Based on the given background information, then class NP is a formal language recognized by a nondeterministic Turing machine within $O(n^k)$ steps where n is the size of input i (Karp, 2009). However, this definition can be impractical in many cases, so an alternative definition is that: If we have a certificate (i.e., a proposed solution to an instance of a problem), then it should take polynomial steps to verify that the certificate is valid. For instance, SAT is in NP because we can write a proposed solution to a SAT instance and verify it within polynomial time.

It is important to note, though, that just because a problem is in NP, it does not mean that it is hard. After all, all P (i.e. easy) problems are also NP problems. To be difficult, a problem must be shown to be NP-Hard (or worse, like PSPACE). To do so, we must reduce an NP-Hard (or worse) problem to the problem itself. For instance, the problem of finding a solution to a Sudoku game is NP-Hard, because Yato & Seto (2003) show that another NP-Hard problem, finding a solution to a partial Latin square, can be reduced to it – i.e., they rewrote finding a solution to a partial Latin square to finding a solution to a Sudoku problem. It is important to note that this Sudoku game will not have a size of 9 x 9. Instead, it has an arbitrary size. In a reduction, the board size can adjust itself to accommodate any problem.

A Latin square is essentially a square of characters which follows the rule:

- If the square is $n \times n$, then there are n characters. For instance, a 6 x 6 square can have A, B, C, D, E, F as characters.
- All rows must have different characters.
- All columns must have different characters.

An example of a Latin Square is the table below:

A	B	C
B	C	A
C	A	B

If a problem is shown to be both in NP and is NP-Hard, then it is NP-Complete.

Cook's Theorem

In the previous chapter, we demonstrated the intractability issue using random-access machines (RAM) – i.e. a theoretical computational model that is largely similar to what we are using day-to-day. However, we can also demonstrate intractability using Turing machines as well. However, instead of discussing the number of steps being executed, we reduce one decision problem representative of a hard class into another one. For instance, Karp (2009) shows that various real-world problems may not have efficient algorithms to solve them by reducing the Boolean Satisfiability (SAT) problem to it. SAT was shown to be NP-Complete by Cook in 1971 (Cook, 2023).

Warning: When dealing with complexity, please keep the type of abstract machine in mind. For instance, if you are finding the Big-O of a runtime function in a lower-year undergraduate computer science class, you are likely dealing with a RAM. On the other hand, if you are trying to show that a problem is difficult to solve because it is NP-Complete, then you can be dealing with either a RAM or a Turing machine, depending on the situation. Pay attention, computer science instructors do not often tell you which model is being assumed.

Now, how could Cook know that SAT was NP-Complete to begin with? Cook has to demonstrate two parts. First, he must show that SAT is in NP. Then, he must show that the problem is NP-Hard. According to Karp (2008), a problem can be shown to be in NP if an input can be decided by a nondeterministic Turing machine within polynomial time. Now, making an argument using a strange multiversal Turing machine can be very difficult. Fortunately, instead of using a nondeterministic Turing machine, we can alternatively argue that a problem is in NP if its **certificate** can be decided within polynomial time by a deterministic Turing machine. At this point, this may still sound quite confusing, so we will proceed slowly. First and foremost, we must define what a certificate is. A certificate is basically an input for a problem with a proposed output (Vega, 2016). For an instance of SAT, a certificate is composed of two parts (Vega, 2016):

- **Input:**
  - The Boolean formula
  - The assignments of the variables within the formula
- **Proposed Solution:** A Boolean value indicating whether the assignments will evaluate the formula to true

Now, we have to construct a verifier Turing machine that determines if the proposed solution matches the actual solution (Vega, 2016). Constructing a Turing machine is very difficult, though. However, we can take advantage of the Church-Turing Thesis, which posits that any simple algorithm that we write can be converted into a Turing machine. Cook & Reckhow (1972) have shown that a program that runs on a random-access machine (RAM) – i.e., any modern-day computer can be converted into a Turing machine. Furthermore, although Turing machines run much more slowly than a RAM, the run-time complexity will not change dramatically. Assuming that the RAM algorithm runs in polynomial time in the worst case, its Turing machine counterpart will still run in polynomial steps. Because we can easily write a code that verifies a SAT certificate, SAT is in NP.

Now, we must show that SAT is NP-Hard. Cook (2003) does this by reducing an arbitrary nondeterministic Turing machine into an instance of SAT within polynomial time. The Turing machine tapes for the machines also become a Boolean assignment scheme for the reduced SAT instance. The details of this process are extremely difficult, so we will omit them here. It is important to note that while Cook's reduction is groundbreaking, it also relies on multiple concepts found in the developments of computer science. First, it relies on the Church-Turing Thesis, which implies that many computer programs must have Turing machine equivalents. Secondly, it relies on the idea that software, programs, and logic are convertible to each other. Both have been extensively discussed in Part II.

Karp's Reduction: Nightmares Made Real

Showing that SAT is NP-Complete is a groundbreaking result. However, SAT is just one among many existing problems out there. It would be terrible if we could show that many real-world problems are also NP-Complete. Karp (2009) realized the nightmares. Instead of reducing a nondeterministic Turing machine into an instance of an NP-Complete or an NP-Hard problem, he instead starts by reducing SAT into the other problems. For instance, he shows that 3-SAT is NP-Complete by reducing SAT to it (Karp 2009). 3-SAT is a variant of SAT where there are only 3 variables. As it turns out, all SAT problems can be rewritten so that they will contain 3 variables. Then, he reduces 3-SAT to the problem of chromatic number, which is about determining the minimum number of colours necessary to colour nodes in a graph to ensure that no neighbours have the same colour (Weisstein, n.d.). Then, the chromatic number problem is reduced to the exact cover problem. Finally, the exact cover problem is reduced to the knapsack problem. The series of reductions shows that the knapsack problem is NP-Hard.

It is important to note that many of the problems in Karp (2009) are not treated as decision problems in practice. For instance, the knapsack problem is solved as a search problem – i.e., we are searching for a good combination of items that will meet the criteria. As it turns out, a decision problem can be "converted" into a search problem. The conversion usually involves creating a new search algorithm that calls the solver of the decision problem version.

I did try to find an academic example of how a decision problem could be reduced to a search problem. As it turns out, actual examples are elusive.

Bounded Undecidable Problem

While the general version of the Halting Problem is undecidable, the bounded version of the Halting Problem is decidable (Kingler et al., 2023). The bounded version basically asks: Given a Turing machine, will it halt within n on input i ? The bounded version is decidable, and in fact, it could actually be considered easy. According to Kingler et al. (2023), if the machine is a deterministic machine, then the problem is in P. P means a Turing machine or software will take $O(n^k)$ steps to complete without the use of nondeterminism. It is simple to argue why the bounded version with a deterministic Turing machine is in P: We let the Turing machine run up to n steps. If halts within n steps, then the answer is "yes." Otherwise, the answer is "no," and the machine is disabled. Give or take, the decision problem takes $O(n)$ steps to decide. The bounded version for non-determinstic Turing machine, on the other hand, is NP-Complete (Kingler et al., 2023). Furthermore, additional undecidable problems are also shown to be in NP-Complete.

References

Cook, S. A. (2023). The complexity of theorem-proving procedures. In *Logic, automata, and computational complexity: The works of Stephen A. Cook* (pp. 143-152).

Cook, S. A. & Reckhow, R. A. (1972). Time-bounded random access machines. *STOC '72: Proceedings of the fourth annual ACM symposium on Theory of computing*, 73-80. https://doi.org/10.1145/800152.804898

Karp, R. M. (1986). Combinatorics, complexity, and randomness. *Communications of the ACM 29*(2), 98-109.

Karp, R. M. (2009). Reducibility among combinatorial problems. In *50 Years of Integer Programming 1958-2008: from the Early Years to the State-of-the-Art* (pp. 219-241). Berlin, Heidelberg: Springer Berlin Heidelberg.

Kingler, A., van der Eyden, M., Stengele, S., Reinhart, T. & De las Cuevas, G. (2023). Many bounded versions of undecidable problems are NP-hard.

SciPost Physics 14(173). SciPost Foundation. https://doi.org/10.21468/SciPostPhys.14.6.173

Kellerer, H., Pferschy, U., & Pisinger, D. (2004). Introduction to NP-Completeness of knapsack problems. In *Knapsack problems* (pp. 483-493). Berlin, Heidelberg: Springer Berlin Heidelberg.

Kwan, D. (Director), & Scheinert, D. (Director). (2022, March 11). *Everything Everywhere All at Once* [Film]. A24; Gozie AGBO; Ley Line Entertainment; Year of the Rat; IAC Films.

Martello, S., & Toth, P. (1987). Algorithms for Knapsack Problems. *Annals of Discrete Mathematics 31* (pp. 231 – 258).

Savitch, W. J. (1972, May). Maze recognizing automata. In *Proceedings of the fourth annual ACM symposium on Theory of computing* (pp. 151-156). https://doi.org/10.1145/800152.804908

Vega, F. (2016). The complexity of a common certificate. https://hal.science/hal-01395950/

Weisstein, Eric W. (n.d.). Chromatic Number. *MathWorld–A Wolfram Resource*. https://mathworld.wolfram.com/ChromaticNumber.html

Yato, T., & Seta, T. (2003). Complexity and completeness of finding another solution and its application to puzzles. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, *86*(5), 1052-1060.

V

Conclusion

14.

Conclusion

Neurosama: Can I ask you something unrelated?
Vedal987: Yeah? What's up?
Neursoama: Do you think […] I'll ever be real?
Vedal987: Does it matter?
A conversation in a video clip (Neuro-sama Official Clips, 2025)

So this is the end of the textbook, but I hope that for the readers, this will be just the beginning of their studies into AI and cognitive science. As this is an unapologetically computer science-centric book, there are many topics that we are missing. Much of the content remains incomplete.

Being unapologetically computer science also means there are topics within "computer science" itself that we have missed. This is because the book very much focused on the core areas of computer science, the areas that have been developed by more traditional linguists (e.g., Chomsky), computer scientists (e.g., Cook, Savitch), and classical cognitive scientists who believe Turing machines are enough (e.g., Putnam). There are also other areas as well, such as machine learning, but I did not put much emphasis on them, because they are not *core* computer science. This brings us to the question: What is *true* computer science?

I leave the answer to you, the readers. However, before answering the question, please keep the following in mind. Initially, computer science began as a study of functions and strings. However, this expanded into the computation and manipulation of strings of arbitrary symbols. Then, it started to discuss the mechanization. The mechanization itself expanded into other fields such as computer engineering and human-computer interaction. Computers now include parts that were not original to the design of a computer. For example, since a monitor screen could display far more content than what Turing machines could, we must now examine how to creatively and effectively present content to the users. Computers themselves were used to create graphics, which led to the field of computer graphics. Whereas Turing machines only demanded to be fed a tape once, computers started to be able to actively listen for more input information (e.g., from a mouse). Afterwards, computers were used to model natural languages and formalize syntax in linguistics, which ultimately led to Cook's theorem. Machine learning researchers also tried to make computers that work with real (but truncated) numbers and (pseudo) random numbers. *So what is computer science?*

And by extension, what is AI, and what is cognitive science?

Reference

Neuro-sama Official Clips. (2025, December 22). *Do I matter to you?* [Video]. https://www.youtube.com/watch?v=a8m4scxQdL0

Appendix A: The Explanation behind the Epigraph

The epigraph is from a Thai epic named Phra Abhai Mani. This epic is a standard part of the Thai education curriculum. Many children, including me, had to read parts of the poem. The “I” in this case is a hermit who acted as a mentor figure to another character, Sudsakorn. In the specific section of the poem, the hermit was teaching him a lesson. The child was tricked by a nude man who then stole his qilin and his magical items (Lohatepanon, 2023). Afterwards, the nude man proceeded to push him off a cliff as well. Sudsakorn, who is a half-human and half-mermaid child, begged the hermit to rescue him from the bottom of the cliff. The hermit, through telepathy, heard the cry and rode a rainbow-emitting cloud to save Sudsakorn. He then told the child to be careful around people.

This was the hermit’s second attempt at rescuing the child. Prior to that, the child was tricked into entering a city full of cursed inhabitants. Their king had stolen the head of a statue of the Buddha, so he and his subjects were cursed to become zombies with a craving for flesh. These zombies were capable of using illusory magic to put on human guises at first glance. As such, they were able to trick the child into believing that he was entering a friendly ground.

Now, you may wonder: where is the father? The father is elsewhere, and the child is on an odyssey to find him. I believe the story has a happy ending, but I have never read it. I left for Canada before I reached the grade where the final part of the poem would have been introduced.

While some readers, including me, believe that the poem is teaching people to be overly paranoid, I also believe that the human mind is very mysterious and very difficult to grasp. The vine theme also pairs nicely with concepts in computer science. Beginners in computer science will quickly become acquainted with string, a data structure that is essentially made by stringing characters (i.e. letters and symbols) together. Intermediates will be familiar with threads, a concept in an operating system class. Some students will be familiar with Turing machines, which work with mysteriously long tapes. Some cognitive scientists argue that all of these long objects themselves can represent the mind. But I would say that the human mind is more mysterious still.

In the same lesson, the hermit also praised the virtue of possessing *survival skills* and encouraged Sudsakorn to become savvier. This also pairs well with embodied AI. If the AI is to become Strong, it must know how to operate within the real world, where survival is a priority.

I also chose the section of the poem just to highlight creativity as a central part of research. While researchers and scientists are not as creative as novelists or other artists, they must still engage in creative thinking and be unafraid of importing novel ideas even if they are not well-acquainted with them. Sunthorn Phu, the author of Phra Abhai Mani, was brave to use many characters from Western literature. He introduced Thai people to various colourful Western concepts. The mermaid, as introduced by Sunthorn Phu, is not an analogue of a local Thai cryptid. It is the same type of mermaid that could be found in Western folklore and fairy tales.

Originally, this would have been my PhD thesis's epigraph. However, the university's format does not allow for an epigraph. While drafting the thesis, we were living through a pandemic, and Russia was seemingly threatening World War III. These events brought out the worst of humanity, showing how twisted humans can be. It did not help either that one of my goals was to ensure that my research was as scientifically valid as possible. It is well-known that human-computer interaction, my field, is bedevilled with validity issues because we work with humans. And by the end of the program, I was still nowhere close to understanding them, despite being one myself.

It is important to note that my translation from the Thai aims to be as faithful to the original meaning. However, the original poetic structure has not been preserved. A more poetic, but more paranoid translation, is as follows (Lohatepanon, 2023):

Do not trust others, he said:
Be afraid of the human mind.
Even the most twisted of vines
Seems benign compared to the soul.

Reference

Lohatepanon, K. M. (2023, June 26). An Introduction to the Poetry of Sunthorn Phu. *Thai Enquirer.* https://www.thaienquirer.com/50051/an-introduction-to-the-poetry-of-sunthorn-phu/