Logical Instruments: Regular Expressions, AI and thinking about thinking

Christopher M. Kelty [1]

16 December 2008

[1]University of California, Los Angeles, Department of Information Studies and the Center for Society and Genetics

introduction

This chapter revisits the origins of AI in the context of the history of software and computer science. This might seem an obvious thing to do, but it is not. For one, the period from roughly 1940 to 1960 was a time of relentless fantasizing about the potential of the new calculating machines and the new science appropriate to them—alternately christened cybernetics, automata studies, artificial intelligence, mechanization of thought processes, and computer science, among others. Researchers from an incredibly large range of disciplines struggled to understand and define what computers, computation, and the control of mechanical logical processes would come to mean. Proposals for the future and research programs designed to achieved them were abundant and varied, and no one was certain what the computer itself would become: giant brain? industrial calculator? military weapon? automatic linguistic translator? cybernetic organism? experimental device? communications system?

It was a period of tremendous experimentation and exploration, and artificial intelligence of the form promoted by John McCarthy, Allan Newell, Herbert Simon and Marvin Minsky was but one fine thread. All of these men were involved in multiple attempts to define the computer, along with a wide range of others, from a youthful Noam Chomsky to John von Neumann from neuroscientist-philosophers to circuit-designing engineers and abstract object-loving mathematicians. All of them played with the ideas and technologies of the time with but vague ideas of what they would become. Hence it is worth looking back to this period from the perspective of the software,

networks and computing devices we have created, in order to ask: how does thinking about thinking proceed?

The period from 1940-1960 is also witness to a division that haunts computer science into the present period: is the computer a tool for thinking, or a replacement for it? Artificial intelligence proponents like Newell and Simon made the leap instantly from rudimentary logic problem solving programs to a autonomous thinking machine. Rather than speak of computers and computational science as aids to thought (as many at the time did, from Vannevar Bush to Warren Weaver to JCR Licklider), they leapt instantly to speaking of them as thinking entities. Later, critics like Dreyfus and Suchman (among many others) would argue that this reduction of human thinking to logical problem solving was philosophically impoverished and an impossible research program [Agre, 1997, Suchman, 1987, Dreyfus, 1992]. But close attention to this period reveals that not everyone confronting the computer equated logic and human reasoning. For many, the computer remained a tool to think with, a way to try out and test ideas in a mechanical form in order to generate concepts and results which could give meaning and form to the practice of mechanical reasoning. In short, many people in this period treated logical reasoning not as the basis of human thought, but a companion to it, as an instrument to be explored and put to use. As a logical instrument to reason *with* not *against*.

This chapter tells the story of one such "logical instrument": regular expressions. Regular expressions are (today) both a ubiquitous programming tool and a keystone in the edifice of theoretical computer science.[1] By

[1]Regular expressions are central to the construction of programming languages, es-

exploring the history of these arcane tools, it is possible to demonstrate how the development of logic and mechanical reasoning proceeds alongside, and in conversation with human reasoning and exploitation of these tools. As such, this story does not take sides for or against claims made about artificial intelligence (such as the critiques made by Dreyfus), but seeks instead to observe, historically and anthropologically, how thinking proceeds; I use the case of regular expressions to show how concepts evolve and are made concrete, even (and perhaps most significantly) concepts meant to capture the foundations of thought itself. It is this objectification, or "registration" [Smith, 1996] of objects as having a certain stability and tractability which I attempt to demonstrate through the historical analysis of the half-century long transformation of the idea of regular expressions. Such an approach is broadly pragmatic in its understanding of logic, thought and concepts as things concretely tested against reality.[2]

pecially in tools that partially automate the process, such as compilers, parsers, lexical analyzers and compiler compilers. By 1969, regular expressions had been incorporated into computer science curricula in most of the major programs as part of classes on formal languages and automata theory. The first textbook was Hopcroft and Ullman's *Formal languages and their relation to automata* [Hopcroft and Ullman, 1969]; current textbooks retain much of the same structure and material, e.g. Peter Linz's *An Introduction to Formal Languages and Automata* [Linz, 2001]

[2]In particular, the work Dewey undertook in Experimental Logic [Dewey, 2004, Hester et al., 2007]. Pragmatist approaches to AI seem few and far between, but see [Burke, 1995]. Recent work by Okrent has extended a Dewey-inspired reading of Heidegger that focuses on intentionality and tool-use as one way to reformulate the philosophical basis of rationality. Intentionality as a case of tool-use is close to what is sought here as an explanation in the case of regular expressions, though the present case does not fit neatly within Okrent's explanation of intentionality, in part because the actors discussed here do not always understand their own manipulations of logical systems as a tool-oriented, intentional activity. And yet, I argue, it is possible to understand what they have done within this frame, and that this re-description is a step towards articulating the understanding of intentionality necessary to remove logic from the domain of pure reason and show its function in the domain of practical reason, especially after the advent of software and programmable computers. See especially [Okrent, 2007, Okrent, 1991]

The range of transformations that regular expressions have gone through is impressive and surprising: they emerge out of the work of Rudolph Carnap on the syntax of logic, are transformed by McCulloch and Pitts into a logical calculi of nerve nets, picked up by John von Neumann for his description of the EDVAC computer, and later his theory of automata, formalized (and given the name "regular expressions") by mathematician Stephen Kleene, used to design state diagrams for circuits by Janusz Brzozowski, implemented in software for the first time by Ken Thompson (who also held a patent on their use for pattern matching), re-implemented in a variety of text editors such as emacs and vi,used in a wide variety of basic textbooks in computer science and built into nearly every programming language in use into the present.

In telling this story of transformations, I demonstrate how human reasoning develops in concert with (indeed, through) the logical tools for thinking that we create. Such a claim is increasingly obvious given our contemporary reliance on software and computation to guide thinking, and yet we have no clear understanding of how we have created such a possibility for ourselves. The period 1940-1960 is key to understanding how logics went from being the ultimate basis for all human and mathematical reasoning (in the sense of Hilbert's program) to domesticated creatures just barely under our control, living alongside us, keeping us company, thinking with us, thinking about thinking.

Regular Expressions, a brief history, with backreferences

Regular Expressions are widely known as a powerful and useful tool for matching patterns in text. As is clear from the very popular web-comic `xkcd` (See Fig 1), possessing knowledge of how to use these tools can be powerful, heroic even. The utility and ubiquity of Regular Expressions is a bit of a self-fulfilling prophecy; with the spread of the Internet has come the spread of structured text as a mode of interaction and transfer of data. HTML and XML (and its parent SGML) especially have created the possibility for, and the need for, the automatic processing, manipulation and reformatting of structured text. Regular expressions are the right tool for the job because they permit a programmer to write concise and general expressions that match a wide range of variable terms in structured text.

For example, a simple verbatim pattern like 'cat' will match the word *cat*, while a pattern such as [cat(.*)] will match any word beginning with *cat*. A regular expression can also match, for instance, the word *cat* at the beginning of a line or *dog* at the end of one, thus:

```
[(^cat|dog$)]
```

And sophisticated uses such as matching any email address are also possible:

```
{\b[A-Z0-9._%+-]+@[A-Z0-9.-]+.([A-Z]{2,4})\b}
```

Regular Expressions match only syntactic or lexical features; they cannot match semantic features unless they can by tied to some kind of lexical or

syntactic pattern. Nonetheless, they are widely used because they are built-in features of the most popular programming languages like C++, Java, Perl and Python. Perl in particular, was one of the first scripting (or 'glue') languages to make extensive use of regular expressions since it was used extensively in the early spread and development of the World Wide Web. Jeffrey Friedl's book *Mastering Regular Expressions* [Friedl, 2006] taught a generation of programmers how to apply regular expressions to problems of

Figure 1: "Regular Expressions" xkcd web-comic by Randall Munroe. Used with permission (Creative Commons Attribution-NonCommercial 2.5 License).

all kinds; from pattern matching to text searching, to processing and formatting streams of data like stock quotes or database queries. What's more, modern implementations include the ability to "back-reference" patterns, allowing sophisticated ways to search and replace bits and pieces of a patter (such as replacing the '.com' section of an email with '.edu'). Combined with other line-oriented UNIX-based processing tools, regular expressions are now part of an arsenal of text and data manipulation tools that allow us to control at least some of our symbol-saturated information environment today.

As a tool, its proliferation is tied to the proliferation of the UNIX operating system in the 1980s and the growth of Free Software in the 1980s and 1990s[Kelty, 2008]. Current versions of regular expressions trace their roots to Henry Spencer's implementation in perl in the 1980s which was widely disseminated via USEnet and the Internet. But prior to this, regular expressions were implemented in a variety of tools in the UNIX Operating System. `grep` is the most famous of these tools—so famous that it is used colloquially by geeks to mean "search" as in "I grepped for my socks." `grep` is standard on every UNIX operating system, including now the Macintosh OS. The modern forms of `grep`, known as `egrep` and `fgrep` owe their existence to the work of Alfred Aho, who created them for the UNIX operating system during his time at Bell Labs (1963-1990). Aho's versions of grep was combined with a tool called `sed` (Stream EDitor) by Larry McMahon, and led to the creation of a widely used programming language, with Peter Weinberger, and Brian Kernighan called `awk` (after the initials of its inventors). `awk` emphasized string data types, text processing and regular expressions.

`awk` was an inspiration for perl in the 1980s.[3]

But what most users of regular expressions as a tool do not know, except in outline, is the long and peculiar history of regular expressions. All of these implementations in software trace back to the work of Ken Thompson, one of the inventors of the UNIX operating system, who was the first to attempt to implement an algorithm for regular expressions in software. Thompson wrote the first UNIX version of what would become grep for a text editor called QED. QED was a rudimentary word processor, in an era before word processing—one of the very first tools that actually allowed people to directly manipulate a file on a screen or through a teletype; hence it was natural that some facility for searching text would be a goal [van Dam and Rice, 1971, Meyrowitz and van Dam, 1982]. Indeed, the name grep' itself derives from a set of commands that would have been issued in QED:

`G/re/P` (**G**lobally search for regular expression *re* and **P**rint it)

QED was written initially by Butler Lampson and Peter Deutsch [Deutsch and Lampson, 1967], two computer engineers at UC Berkeley working on the DARPA-funded Project GENIE, sister to Project MAC at MIT and one of the origins of many of the innovations that went into UNIX and modern time-sharing, multi-user operating systems [Lee et al., 1992]. That version of QED did

[3]There is no scholarly historical work on UNIX and its origins at Bell Labs that explores the range of activities underway there. Salus [Salus, 1994] reports on the development of the operating system in outline. Most of this work is available online, however, in various places. See Dennis Ritchie's web resources, for instance http://www.cs.bell-labs.com/who/dmr/

not include regular expressions, or even the ability to search for arbitrary fixed strings. Ken Thompson received his BS and M.Sc. at Berkeley (1962-1966) and worked with Lampson and Deutsch on Project Genie. When he was hired to work at Bell Labs, and was sent to MIT to participate in Project MAC, one of the first things he did was to create a version of QED for the famous Compatible Time-Sharing System (later to become Multics).[Vleck, 2008] Thompson went on to create versions of QED for Multics and ultimately for UNIX, for which he dropped the Q and called it simply `ed`. `ed` in turn inspired a generation of powerful and difficult to use text editing programs still in wide use today, such as EMACS and `vi` .

In 1968, Thompson also published a short "Programming Techniques" paper for the CACM in which he described the "Regular Expression Search Algorithm" he had implemented [Thompson, 1968]. What made regular expressions a more powerful tool for search was that they allowed for a kind of parallel processing. Rather than searching up and down a tree of possibilities through brute force, backtracking every time it fails, regular expressions allow the program to search more efficiently by "looking ahead" and figuring out which paths can be eliminated.[4].

It is not at all obvious, however, how Thompson reached this point. Prior to his implementation of this algorithm (which was originally written in Assembly language for the IBM 7090, on which CTSS ran, and presented in

[4]Russ Cox has written a technical history of the different implementations of the matching engine that Thompson used, and those that are in use to day, demonstrating differences in efficiency and speed that have been lost over time [Cox, 2007]. The Algorithm that Thompson implemented also bears some similarity to the general AI approach to "reasoning as search" given the focus on pattern matching and backtracking. Thompson's use of regular expressions, however, seems to be the first such implementation and entirely distinct from the AI tradition of problem solving.

the paper in Algol), there were no such attempts to use regular expressions in this way. Indeed, not many people would have known of their existence outside a handful of now famous people who had explored them in different forms—Stephen Kleene, John Von Neumann, Walter Pitts and Warren McCulloch, and Michael O. Rabin and Dana Scott. The link between these individuals and Thompson came from Janusz Brzozowski, who taught for a semester at Berkeley while Thompson was there, and who researched the use of formal languages for circuit design in engineering. Brzozowski introduced a method for using regular expressions to create "state-transition diagrams" which were widely used in the 1950s as a mnemonic for designing and implementing programs in circuits and in the computers of the 1950s [Brzozowski, 1964]. Thompson adopted Brzozowski's ideas at a time when "software" was just emerging as concept and practice.

Brzozowski's work was part of an explosion of research in the late 1950s that forms the basis of abstract computer science today: the formalization of symbolic processing at the root of the logical operation of computers [Myhill et al., 1960, Nerode, 1958, Arbib, 1961, Copi et al., 1958, Hopcroft and Ullman, 1969]. Michael O. Rabin and Dana Scott's 1959 article "Finite Automatons and their Decision Problems" [Rabin and Scott, 1959] is frequently referenced as the apotheosis of this work. It was a formal description that might more accurately be understood as the theoretical ground for modern computers than Turing's work of 20 years earlier [Turing, 1937]. Though indebted to Turing's work, the formalization of computing in the late 1950s took the finite nature of existing computing devices as a realistic constraint in developing a formal language that would allow for the manipulation of these

devices. Such an approach took as its object of logical manipulation of symbols by *finite* automatons (whether human, machine or animal). It was based in no small part on the explorations begun earlier by Turing, Von Neumann, McCulloch and Pitts, and ultimately by Carnap and Whitehead and Russell.

What should be clear to the reader is that these earlier attempts where in no way intended to produce a simple tool for matching patterns in text, but instead to explore the very basis of logical reasoning itself. What I show in the next section is that this exploration of logical reasoning is conducted as if it is an investigation into the basis of human reason; but it proceeds instead through the manipulation of objectified "logical instruments" which can be formalized, manipulated, explored and constructed as if they were tools, as if they were material objects.[5] Attention to the media-specific creation of these tools (in their symbolic expression, their representation in diagrams, and ultimately in programming languages and in machines as running code) can help demonstrate how logical instruments have come into being and are progressively translated or modulated from medium to medium and mind to mind.

From Principia Mathematica to Deus ex Machina

Thompson's paper on Regular Expressions is frustratingly vague about the origins of his idea. It has only four references: The IBM 7094 Programmer's

[5]The meaning of materiality is obviously at issue: objectification is probably a better term, or concept-ification. It is not merely the symbolic character that is at stake, but the ability to grasp the instrument as a singularity, as something with parts that can be observed, moved, explored, replaced as if one were looking at a device.

manual, the paper by Brzozowski, a single paper about pattern matching in text (by Anthony Oettinger), and Stephen C. Kleene's "Representations of events in Nerve Nets and finite Automata" [Kleene, 1956]. Kleene's paper is a kind of hidden classic and the link between theories of automata and the language of "regular events and regular expressions" which Kleene introduced. Stephen Kleene was a graduate student at Princeton in the 1930s, where he worked alongside Alonzo Church and Alan Turing and made substantial contributions in the field of "recursive functions." Kleene wrote perhaps the single most famous textbook on the subject called *Introduction to Metamathematics* in 1952. Kleene's 1956 paper describes what he called an "algebra of regular events." It was published in a well-known volume called *Automata Studies*, edited by Claude Shannon and John McCarthy. Kleene's paper is widely cited in the computer science literature, and is some ways one of his most famous worksbut it was not central to his research project, and he never returned to the subject.

Kleene's interest in the subject was motivated by his own work on the algebra of recursive functions. In the paper, he introduces a trio of mathematical operators that represent regular events: `A + B` (or union), `A ∘ B` (or concatenation), and `A*` (or the "Kleene Star" or iterate). Kleene's paper is generally understood to be significant because it proves two theorems about the equivalence of regular events and finite automatons. But what makes it interesting in this context is where the idea to create such an algebra came from. In stating these theorems in the text, the origins of Kleene's interest become clear:

Theorem 3: To each regular event, there is a nerve net which represents the event by firing a certain inner neuron at time p + 2, when started with suitable states of the inner neurons at time 1. If the event is describable by a prepositive and positive regular set of tables, the representation can be by a net started with all inner neurons quiet.

Theorem 5: In any finite automaton (in particular in a McCulloch Pitts nerve net), started at time 1 in a given internal state b_1 the event represented by a given state existing at time p is regular. [Kleene, 1956, 0]

What Kleene meant by "regular events" was an event processed by a set of nerve cells–an event of perception or of thought. Kleene's paper says nothing about computers, programming, matching patterns in text or searching for text on a computerthe paper was not even composed on or near a computer, as the typescript would indicate. However, it is clear from the opening line of the article what "automata" are: "anything that receives stimuli, human, machine or animal." It's also clear that the inspiration for Kleene's paper was the famous 1943 paper "A Logical Calculus of the Ideas Immanent in Nervous Activity" by Warren McCulloch and Walter Pitts [McCulloch and Pitts, 1943]. Indeed, Kleene adopted the same graphic diagrammatic representation that McCulloch and Pitts used to present his own definitions (See Fig 2)

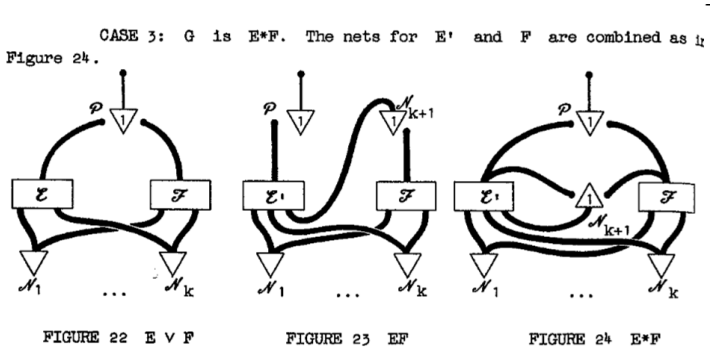
Kleene had written his paper in 1951, during a summer at RAND, after

Merrill Flood, his sponsor at RAND and fellow grad student at Princeton, had given him McCulloch and Pitts paper to read [Kleene, 1979]. Upon reading this paper, Kleene's first inclination as a mathematician was to push further the model it contained—a model of the brain as a logical calculus. Post-Gödelian logics being Kleene's metier, he naturally wanted to formalize what McCulloch and Pitts had proposed as a model of the brain in terms of an algebra of recursive functions. In a reflection from 1981, Kleene said:

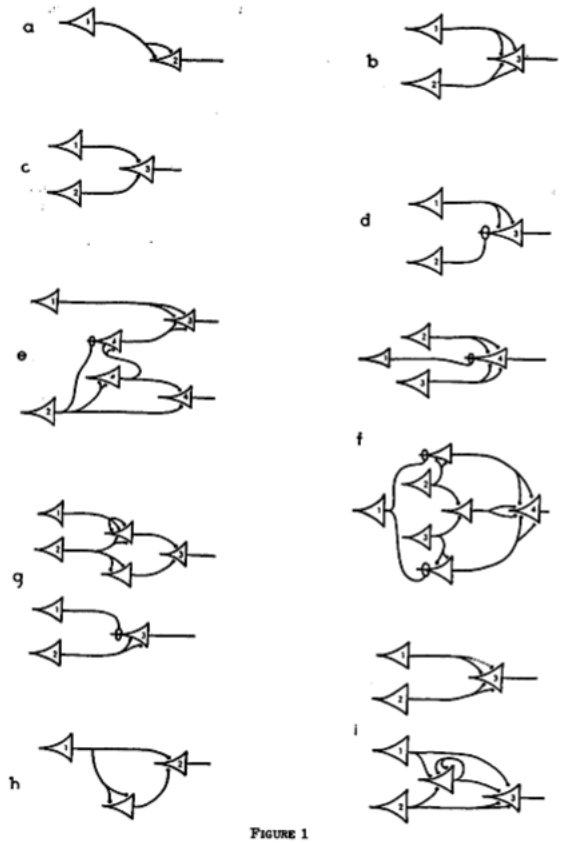
> I found [McCulloch and Pitts'] model to be very interesting—an original contribution—but their analysis of it to fall quite short of what was possible. So I did what cried out to be done with it (as it seemed to me) having newly in mind also the idea of a finite automaton, which came to me that summer at RAND through reading in printer's proof (?) Von Neumann's Hixon Symposium lecture.[Kleene, 1979, 0].

What "cried out to be done" however, was probably not what McCulloch and Pitts would have cried out for. It is worth reflecting here on the misfit of intentions between these two papers. Whereas McCulloch and Pitts were interested precisely in creating a representation of thought (if not neurons *per se*), Kleene was not:

> our theoretical objective is not dependent on the assumptions [about the neuro-physiological data] fitting exactly. It is a familiar stratagem of science, when faced with a body of data too complex to be mastered as a whole, to select some limited



(a) Kleene's Diagrams



(b) McCulloch and Pitt's Diagrams

Figure 2: Kleene and McCulloch and Pitts' diagrams of regular events

> domain of experiences, some simple situations, and to undertake to construct a model to fit these at least approximately. Having set up such a model, the next step is to seek a through understanding of the model itself. [Kleene, 1956, page].

Kleene was not interested in whether McCulloch and Pitts' model was accurate (that is, whether it really represented the state, function or structure of a brain, or of thought)—what he takes from the paper is a model that just happens to be defined in terms of neurons and sensory inputs and organisms, but that was, for him, not yet fully explored in mathematical terms. In fact, what Kleene proved in his paper was that it wasn't a model of human thought, per se, but a model of any finite automaton, whether animal, human, or machine, which is evident from his reference to von Neumann's Hixon Symposium lecture on automata. Von Neumann was among the first to explore the idea of a general theory of automata using the tools of logic. The Hixon symposium is clear that, whereas for centuries we have relied on the mathematics of continuous variation to analyze nature, the emergence of computing technology has created a need for a more powerful discrete mathematics that might be more amenable to the analysis of thought than that of classical mathematical analysis. Kleene takes this insight a step further: the implicit equivalence between the brains of animals, humans and machines in Kleene's approach holds no necessary connection to actual brains or neurons, or even to the actual facts of thinking. Von Neumann intuited this as well when, in writing the famous "First Draft of a Report on the EDVAC." Instead of simply describing the design of a

computer in terms of vacuum tubes, wires, plug-boards and relays, von Neumann used McCulloch and Pitts diagrams of neurons as its formalism (see Fig. 3).

6.2 The analogs of human neurons, discussed in 4.2–4.3 and again referred to at the end of 5.1, seem to provide elements of just the kind postulated at the end of 6.1. We propose to use them accordingly for the purpose described there: As the constituent elements of the device, for the duration of the preliminary discussion. We must therefore give a precise account of the properties which we postulate for these elements.

The element which we will discuss, to be called an E-element, will be represented to be a circle ◯, which receives the excitatory and inhibitory stimuli, and emits its own stimuli along a line attached to it: ◯—. This axon may branch: ◯—≺, ◯—≺—. The emission along it follows the original stimulation by a *synaptic delay*, which we can assume to be a fixed time, the same for all E-elements, to be denoted by τ. We propose to neglect the other delays (due to conduction of the stimuli along the lines) aside of τ. We will mark the presence of the delay τ by an arrow on the line: ◯→—, ◯→≺. This will also serve to identify the origin and the direction of the line.

Figure 3: McCulloch-Pitts Neurons in the text of the First Draft of the Report on EDVAC by John von Neumann

For von Neumann and Kleene, the formal equivalence demonstrated by McCulloch and Pitts between brains and logical languages was also a formal equivalence suitable to a machine. This is not a metaphor. It is not a suggestion that computers should function *like or as if* they are brains, but rather the recognition of a certain utility, testability, or pragmatic rectifiability to this formalism. With the introduction of an algebra of regular events, Kleene created a useful, practicable formalism that within a matter of years would dominate the nascent field of computer science. To a generation of people who would become the first luminaries of computer science, Kleene's paper was the first step towards a formal theory of actual and possible computer programs.

In 1959, for instance, Michael Rabin and Dana Scott would publish "Finite Automata and Their Decision Problems" [Rabin and Scott, 1959] which

would become the locus classicus for future research into the classification of computer programs, their complexity and their solvability. What Thompson took from Kleene was not a model of a brain, or even an algebra, but a kind of logical toy, a structure with certain implications that could be explored, an object to think with that could potentially made to fit some new situation. Indeed, somehow Thompson recognized that this algebra was a tool for efficiently matching patterns in text and employed the algebra to create an algorithm and a piece of working software. One might say this is an obvious case of science being "applied" to create technology, but the problem is that whatever it is Kleene's paper sets out, proleptically, as science, is not the technology that Thompson creates. Rather it is a translation of Kleene's algebra into an algorithm, a paper and a program.[6] And Kleene had in turn translated McCulloch and Pitts' model into an algebra and a formalism now known as regular expressions for regular events, terms initially meant to capture the sensory input experience of an organism or an automaton.

Kleene's formalization of McCulloch and Pitts transformed their ideas from the domain of representations of thought to the domain of representations of automata, and hence made it useful as a tool—both a logical tool for thinking about thinking (i.e. thinking about logical reasoning) and a tool for designing thinking machines (i.e. designing and controlling circuits and eventually software). McCulloch and Pitts' paper had no such goals in mind. Indeed, the paper is claimed as an origin point variously by cybernet-

[6] And on top of that, a patent which represents the pattern-matching regular expression algorithm in yet another kind of formalism, that of the legal language of claims in a patent (3,568,156)

ics, neuroscience, mathematical psychology, cognitive science, mathematical biology, artificial intelligence, and the burgeoning subfield of computer science that studies neural nets and neural processing, quite in the absence it should be said of any brains other than those of the researchers[7].

McCulloch's interest was initially in finding what he hypothesized as a "psychon"—or atomic unit of neural activity, which he first sought in his physiological research conducted during the 1930s in partnership with Yale physiologist J.G. Dusser de Barenne. In the early 1940s, McCulloch was introduced to Walter Pitts by Jerome Lettvin, and thereby to Nicholas Rashevsky's Mathematical Biology group at the University of Chicago, where Walter Pitts had been actively working on models of neural activity with Rashevsky and mathematician Alston Householder. The collaboration between the two was lopsided, at best. McCulloch was in his forties, Pitts was 17; McCulloch had spent his career in physiology and philosophy, Pitts was by various and sometimes unreliable accounts a mathematical prodigy who had run away from his home in Detroit and met Bertrand Russell in a park in Chicago [Smalheiser, 2000, Schlatter and Aizawa, 2008]. Together, however, they managed to piece together something that met in the middle, a paper that demonstrated the formal equivalence between a plausible model of neural activity, and a logical calculus. Part of McCulloch and Pitts inspiration for their paper was Turing's machine. As Tara Abraham puts it "Turing was able to define the complicated process of computation in 'mechanical' terms, with the notion of a simple algorithm so exhaustive,

[7]Tara Abraham has recently documented much of the intellectual context and background of this paper [Abraham, 2003, Abraham, 2004, Abraham, 2007]. See also [Kay, 2002] and [Piccinini, 2004]

rigorous and unambiguous that the executor would need no 'mathematical knowledge' to carry out its task." [Abraham, 2003, 18] This identification of computation with an automatic procedure provided the inspiration for McCulloch and Pitts to model a set of nerves as something that could also calculate "in the absence of mathematical knowledge."

In hindsight, what McCulloch and Pitts achieved was far more influential in engineering, computer science and mathematics than it was in biology or neuroscience. Works that take McCulloch and Pitts logical calculus of nerve nets as a starting point have been extraordinarily bountiful in mathematics and computer science. Accounts of Artificial Intelligence almost inevitably refer to this paper as one origin point, whether as inspiration or as foundation.[8] By contrast, neurology and neurobiology have more or less dropped the formalization entirely, beginning at least with McCulloch and Pitts themselves, whose 1947 paper "How we know universals" and the 1959 paper they wrote with Lettvin and Maturana, "What the Frogs Eye Tells the Frog's brain" [Lettvin et al., 1959, Pitts and McCulloch, 1947] both abandon the strict formal equivalence with propositional calculi or the Turing machine, in favor of more complex biological models which are less amenable to logical manipulation.

The analogy between logic and thinking is obviously not unique to Mc-

[8]Lily Kay points out that even though their paper is routinely adopted as an origin point for AI, McCulloch was himself "not an AI enthusiast. He was far less interested in machines that try to think than in the mechanical principle of thought (and by mechanical he did not mean physicochemical or reductionist). He was attracted to 'Machines that Think and Want': how we know particulars and universals, the desire for food, woman and bed, music, poetry, mathematics. And he sought a mathematical corrective to the mindless blackbox of behaviorism and the tripartite Platonic conjectures of psychoanalysis." [Kay, 2002, 603-4]

Culloch and Pitts. In most of philosophy and mathematics in the early twentieth century, and especially under the influence of the Hilbert program, logic and mathematics represent the capacity for human reason at its pinnacle. McCulloch and Pitts materialized this assumption in a representation of neurons, adding a concrete formal specificity to an otherwise abstract set of claims about human reason. Like Thompson's paper on Regular Expressions, McCulloch and Pitts paper is exasperatingly short on citations: it has three, and none refer to previous work in biology or physiology, nor even to Turing's work, but instead to arguably the three most famous attempts to formalize scientific reasoning of the 20th century: Whitehead and Russell's *Prinicipia Mathematica*, Hilbert and Ackerman's *Foundations of Theoretical Logic* and Rudolph Carnap's *Logical Syntax of Language*.

Despite McCulloch's polymathic knowledge of physiology, the paper with Pitts is hopelessly imprecise (and avowedly so) in physiological terms. What drives the paper instead is the attempt to transform a logical calculus from one symbolic system to another—a symbolic system inspired by the observation that neurons are connected together in networks and tend to either be firing electrical impulses or not, are either "on" or "off".

Walter Pitts was a student of Rudolph Carnap and had clearly read his work (on the advice of Bertrand Russell himself, so the story goes). However, what was it that McCulloch and Pitts took from Carnap? Was it a particular logical system or a set of propositions and theorems? Or was it the symbolism? What was it that "cried out to be done," to echo Kleene, with Carnap's work? I would argue that Carnap provided McCulloch and Pitts with the warrant they were looking for to *creatively* formalize their

model.

Carnap's book, *The Logical Syntax of Language*, is the pinnacle of his attempts to define a philosophical language suitable to the analysis of scientific statements. In it, he constructs two kinds of artificial languages, labeled Language I and Language II. Language I is a highly restricted language that includes only the most basic postulates of the arithmetic of natural numbers. Language II includes Language I as well as "indefinite" terms for all of mathematics and "the sentences of physics." McCulloch and Pitts chose for their formalism, Language II, a choice widely regarded as unfortunate because of Carnap's impossibly baroque symbolism. Nonetheless, it is not the formalism per se, nor the content, which they draw inspiration from so much as Carnap's insistence that *the choice itself is arbitrary.* Carnap's *Logical Theory of Syntax* is famous for this approach:

> In [this book], the view will be maintained that we have in every respect complete liberty with regard to the forms of language; that both the forms of construction for sentences and the rules of transformation (the latter are usually designated as 'postulates' and 'rules of inference') may be chosen quite arbitrarily.... For language, in its mathematical form, can be constructed according to the preferences of any one point of view represented; so that no question of justification arises at all, but only the question of the syntactical consequences to which one or another of the choices leads, including the question of non-contradiction. (*Logical Syntax of Language*, p. xv, 1954 ed.)

Carnap was justly famous, even among the logical positivists, for his vigorous hatred of metaphysics and of "meaningless" statements—a hatred most clearly visible in the 1932 article "The Elimination of Metaphysics through the Logical analysis of Language" [Carnap, 1932]. So it is perhaps oxymoronic to term this approach to the arbitrariness of logical language the "Principle of Tolerance," which states: "It is not our business to set up prohibitions, but to arrive at conventions. (p. 51)" and "In logic, there are no morals. Everyone is at liberty to build up his own logic, i.e. his own form of language, as he wishes. All that is required of him is that, if he wishes to discuss it, he must state his methods clearly, and give syntactical rules instead of philosophical arguments. (52)"

Only the principle of tolerance, the submission to the arbitrary, Carnap insisted, could lead us to "the boundless ocean of unlimited possibilities" that awaits us[SARKAR, 1996]. Hence, what McCulloch and Pitts picked up on was not that Carnap's formalism was the most realistic, or accurate, or universal, but that its arbitrariness freed it from any putatively metaphysical presuppositions that might constrain it. It could be made into an experimental tool. It could be made into a brain, a brain-machine, or a brain-language—and the slippage amongst these things could be deliberate and refer implicitly, if not explicitly and precisely, to the activity of human thought.

Like Turing's machine, Carnap's languages were meant to be the most mechanical and the most rigorous constructions of reason possible–but they were nonetheless constructions of reason. They are not, either in Turing's case or in Carnap's, instructions for the creation of machines with which hu-

mans might think. For Carnap, the elimination of metaphysics was explicitly about creating protocols that define the understandability and analyzability of statements; but it was also an implicit ontology of introspection and thinking. Logical analysis provides a way to construct systems by which it is possible to reach valid conclusions without recourse to any inaccessible form of introspection or any revealed knowledge. That is to say, to construct machines that will demonstrate the answer to multiple parties regardless of any mathematical knowledge.

Carnap, in a tradition stretching back to Leibniz, wanted his artificial languages to be the ultimate arbiters of dispute. No longer would human frailty, human weakness in its reliance on irrationality or its reference to the supernatural, be responsible for decision making when the best of all possible systems could be constructed to answer questions objectively and mechanically.[9]

McCulloch and Pitts seem to have recognized something slightly different in this liberation of logic; they seem to have appreciated that the design of a logical system should not be constrained by our assumptions or our metaphysical beliefs, but should instead be an act of pure construction whose implications and consequences for the task at hand are all that matter. But McCulloch and Pitts did not use Carnap's Language II to test the

[9]Dreyfus locates his original critique of AI in just this age-old desire, which he refers to as Platonic in origin. Dreyfus' book is not a critique of reason per se, however, but an alternative explanation of human reason drawing on concepts of embodiment and background know-how [Dreyfus, 1992]. It does not, however, make any assumptions about whether embodiment or background knowledge are themselves mediated by the presence and proliferation of machines and symbol systems, regardless of whether they were created with the intention of being able to reason autonomously. Thinking with machines seems to me to fit into our "background" know-how in ways that neither AI nor Dreyfus seems to give any credit.

soundness of their model, they were not making empirical statements about the brain which Carnap's logic would verify. Rather what McCulloch and Pitts did was to prove the formal equivalence of their model neuron and that of Carnap's logic; or to put it differently, they created yet another language—a brain-language or nerve-language—which they showed to have exactly the same expressive possibilities as Carnap's syntax-language.[10]

McCulloch and Pitts brain-language is represented in their abstract images of neurons which are carefully correlated with formal propositions from Carnap's calculus. These abstract neural-net diagrams show up again and again in subsequent formalizations, such as Kleene's work, and are ultimately central to neural network research of all kinds (See again, Fig 2). McCulloch and Pitts translation of Carnap (like Kleene's translation of McCulloch and Pitts and Thompson's translation of Kleene) doesn't try to preserve Carnap's universal language, nor reduce the brain to it, it invents a new one, it translates it or remediates it.

If one examines the work of Carnap, McCulloch and Pitts, Kleene and Thompson and later contributors to the development of regular expressions, one sees a continuity of an instrumental kind, even if that continuity does not map on to one of research programs, institutions or philosophical commitments. Indeed, if anything the "principle of tolerance" does not liberate reasoning from metaphysics at all, so much as it creates an entirely new kind of object to think with: logical instruments. McCulloch and Pitts' insights

[10]Lily Kay's analysis of McCulloch and Pitts confirms this, in part. She argues that what McCulloch and Pitts attempted to do was to bridge the formal analysis of the brain with that of the mind: "Neurophysiology could expand to embrace mind, since that mind—conceptualized within a new discourse of information—could now be embodied in neural nets and investigated experimentally" [Kay, 2002, 600].

are mediated through, literally only thinkable through, the liberated logical mechanisms of Carnap or Turing. Just as Kleene's are mediated through McCulloch and Pitts and Thompson's through Kleene's and so on. There is a stability and a coherence to the object registered by these researchers, even if the intentions and justifications for what they are doing seem to diverge or only converge on loosely and serendipitously.

Conclusion

Regular Expressions are everywhere today, which is a strange triumph if one sees in them a vestige of Carnap's program. The fact that his approach to understanding thought might lead, through a circuitous but very concrete path, to a tool used everyday to reformat symbolic information in our syntax-saturated, structured language-draped world is a curious fact. The continuity of regular expression raises a question that might be phrased in terms of Bruno Latour's concept of "immutable mobiles": what exactly is the "immutable" part of this mobile concept, ontologically speaking? What, if anything, remains of Carnap, in today's regular expressions?[11] Part of what is at stake is a creating a richer description of thinking about thinking; one in which logical instruments are included as part of our repertoire for exploring ideas, exploring arguments and claims, and constructing concepts. Regular Expressions are a strategic example, since they form the very basis of "formal language" theory in computer science, as well as a ubiquitous tool for matching patters in text. One can also show a similar progression

[11] Latour's work on immutable mobiles appears in *Drawing Things Together* [Latour, 1990]; See also Paul Rabinow on the concept of "remediation" [Rabinow, 2008].

with the related logical instrument "L-Systems" which similarly transformed from a model of biological growth, to a mathematical formalism, and ultimately to a widely used tool for generating computed visual forms, such as images of trees, flowers and neurons [Kelty and Landecker, 2004]. Similarly, one might look to other conventional features of contemporary computing: Bezier curves in illustration (and their origins in numerical controlled design of automobile bodies in the fifties), or filters used in audio-video editing software (and their origins in electrical engineering, communications theory, or speech-processing research). All of these are clear cases of tools based in logic (because based in the construction of effective procedures required by our computing devices) which have come to populate our world.

The success of logic, seen in this light, might also be measured opposite the putative "failure" of the classic AI program, especially in terms of the productive possibilities that result from research. Alan Newell and Herbert Simon's famous "Logic Theorist" [Newell and Simon, 1956] was a program designed to prove the theorems of the *Prinicipia Mathematica*. At first glance, this would seem to be another and particularly nice example of the kind of translations seen in the case of regular expressions. However, Newell and Simon's project made no explicit formal equivalence, it translated nothing. They did invent a new artificial language—a precursor to the programming language LISP—but they did not attempt to prove that that language was equivalent to any other. Rather, the implicit formal equivalence maintained in AI is that between the human capacity for reason and proof of theorems by logical inference. Newell, Simon, McCarthy, Minksy and others aimed at a second creation, a robot race of machines more intel-

ligent than humans. By contrast, the work of McCulloch and Pitts, Kleene, and especially Thompson led to the creation of something more like a companion species of logical instruments; logical instruments with and through which introspection and exploration is conducted. Rather than asking "Can machines think?" as AI relentlessly does, this minor tradition asks "Can humans think?" to which the answer is: not alone. We reason with logics, we form symbiotic relationships with them, we translate them from paper to machine, to patent to software.

If one sees AI as the attempt to create a second race of thinking beings, then one can only pronounce its failure; but seen from the perspective of "logical instruments" AI might also be seen as a similarly successful research program—not "degenerative" at all (Dreyfus) but proliferative. Game engines, recommendation systems, virtual world non-player characters, and net-based commerce and advertising applications are everywhere. Computer scientists routinely speak of "agents" today instead of humans or thinking entities. The Rodney Brooks-inspired approach of creating large coordinated networks of very simple processes has dominated the research imagination since the early 1990s at least.

Understanding AI as part of a tradition of logical instrument fashioning might also help explain the current explosion computer research traditions that surround us today: whatever the computer is, its much more than the machine on our laps or in our homes, and certainly much more than what Turing wrote about in 1936. Today there journals such as *Soft Computing*, *BioSystems Natural Computing*, *The Journal of Applied Soft Computing*, *the IEEE transactions on Evolutionary Computaion* and *The International*

Journal of Unconventional Computing. There are GRID computers, mesh computers, computers made of DNA, molecular computers and "moleware", reaction-diffusion chemical computing, quantum computers, spintronic computers, protein-based optical memories and processors, "computers" literally made of leech neurons and "membrane" computers (which are not made of membranes at all), tinker toy computers, "amorphous computers," neural networks and genetic algorithms as well as an increasingly vibrant array of programmable logic devices like the field programmable gate array, to say nothing of a new generation of biological sensors that measure and compute everything from light intensity to fish freshness. All these variations are designed with logic instruments and become logical instruments for designing. They proliferate in direct response to the playfulness with which they are engaged, and this should tell us something about the history of thinking about thinking. Some of these proliferations tend towards a fantasy world in which everything is a computer. Cells compute, DNA computes, molecules compute, waves compute—indeed, this version of computation-as-intelligence is so generalized that its seems impoverished to continue making the implicit connection between logic and thought, and instead start thinking of them as logical instruments without which we cannot think, creatively or otherwise.

Bibliography

[Abraham, 2003] Abraham (2003). From theory to data: Representing neurons in the 1940s. *Biology and Philosophy*, 18(3):415–426.

[Abraham, 2004] Abraham (2004). Nicolas rashevsky's mathematical biophysics. *Journal of the History of Biology*, 37(2):333–385.

[Abraham, 2007] Abraham, T. H. (2007). Cybernetics and theoretical approaches in 20th century brain and behavior sciences.

[Agre, 1997] Agre, P. E. (1997). *Computation and Human Experience*. Cambridge University Press.

[Arbib, 1961] Arbib, M. (1961). Turing machines, finite automata and neural nets. *J. ACM*, 8(4):467–475.

[Brzozowski, 1964] Brzozowski, J. A. (1964). Derivatives of regular expressions. *J. ACM*, 11(4):481–494.

[Burke, 1995] Burke, T. (1995). Dance floor blues: the case for a social AI. *Stanford Humanities Review*, 4(2):221–248.

[Carnap, 1932] Carnap, R. (1932). The Elimination of Metaphysics Through Logical Analysis. *Logical Positivism*, pages 60–81.

[Copi et al., 1958] Copi, I. M., Elgot, C. C., and Wright, J. B. (1958). Realization of events by logical nets. *J. ACM*, 5(2):181–196.

[Cox, 2007] Cox, R. (2007). Regular expression matching can be simple and fast. http://swtch.com/~rsc/regexp/regexp1.html.

[Deutsch and Lampson, 1967] Deutsch, L. P. and Lampson, B. W. (1967). An online editor. *Commun. ACM*, 10(12):793–799.

[Dewey, 2004] Dewey, J. (2004). *Essays in Experimental Logic*. Dover Publications.

[Dreyfus, 1992] Dreyfus, H. L. (1992). *What Computers Still Can't Do: A Critique of Artificial Reason*. The MIT Press, 1st edition.

[Friedl, 2006] Friedl, J. (2006). *Mastering Regular Expressions*. O'Reilly Media, Inc., 3rd edition.

[Hester et al., 2007] Hester, D. M., Talisse, R. B., and Burke, T. (2007). *John Dewey's Essays in Experimental Logic*. Southern Illinois University, 1st edition.

[Hopcroft and Ullman, 1969] Hopcroft, J. E. and Ullman, J. D. (1969). *Formal languages and their relation to automata*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Kay, 2002] Kay, L. E. (2002). From logical neurons to poetic embodiments of mind: Warren s. McCulloch's project in neuroscience. *Science in Context*, 14(04):591–614.

[Kelty, 2008] Kelty, C. (2008). *Two Bits: The Cultural Significance of Free Software*. Duke University Press, Durham, N.C.

[Kelty and Landecker, 2004] Kelty, C. and Landecker, H. (2004). A theory of animation: Cells, l-systems, and film. *Grey Room*, -:30–63.

[Kleene, 1956] Kleene, S. (1956). Representation of events in nerve nets and finite automata. *Automata Studies*, 34:3–41.

[Kleene, 1979] Kleene, S. (1979). Origins of recursive function theory. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science (sfcs 1979)-Volume 00*, pages 371–382. IEEE Computer Society Washington, DC, USA.

[Latour, 1990] Latour, B. (1990). Drawing things together. In Lynch, M. and Woolgar, S., editors, *Representation in Scientific Practice*, pages 19–68. MIT Press, Cambridge, MA.

[Lee et al., 1992] Lee, J., Rosin, R., Corbato, F., Fano, R., Greenberger, M., Licklider, J., Ross, D., and Scherr, A. (1992). The Project MAC interviews. *Annals of the History of Computing, IEEE*, 14(2):14–35.

[Lettvin et al., 1959] Lettvin, J., Maturana, H., McCulloch, W., and Pitts, W. (1959). What the Frog's Eye Tells the Frog's Brain. *Proceedings of the IRE*, 47(11):1940–1951.

[Linz, 2001] Linz, P. (2001). *An Introduction to Formal Languages and Automata*. Jones and Bartlett, Boston, 3rd ed edition.

[McCulloch and Pitts, 1943] McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5(4):115–133.

[Meyrowitz and van Dam, 1982] Meyrowitz, N. and van Dam, A. (1982). Interactive editing systems: Part ii. *ACM Comput. Surv.*, 14(3):353–415.

[Myhill et al., 1960] Myhill, J., Laboratory, E. T., of Pennsylvania, U., and Division, W. A. D. (1960). *Linear Bounded Automata WADD Tech. Note.* Wright Air Development Division, Air Research and Technology Command, United States Air Force.

[Nerode, 1958] Nerode, A. (1958). Linear automaton transformations. In *Proc. Amer. Math. Soc*, volume 9, pages 541–544. JSTOR.

[Newell and Simon, 1956] Newell, A. and Simon, H. (1956). The logic theory machine–a complex information processing system. *Information Theory, IRE Transactions on*, 2(3):61–79.

[Okrent, 1991] Okrent, M. (1991). *Heidegger's Pragmatism: Understanding, Being, and the Critique of Metaphysics.* Cornell University Press.

[Okrent, 2007] Okrent, M. (2007). *Rational Animals: The Teleological Roots of Intentionality.* Ohio University Press, 1st edition.

[Piccinini, 2004] Piccinini (2004). The first computational theory of mind and brain: A close look at mcculloch and pitts's "Logical calculus of ideas immanent in nervous activity". *Synthese*, 141(2):175–215.

[Pitts and McCulloch, 1947] Pitts, W. and McCulloch, W. (1947). How we know universals the perception of auditory and visual forms. *Bulletin of Mathematical Biology*, 9(3):127–147.

[Rabin and Scott, 1959] Rabin, M. and Scott, D. (1959). Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125.

[Rabinow, 2008] Rabinow, P. (2008). *Marking Time: On the Anthropology of the Contemporary*. Princeton University Press, Princeton.

[Salus, 1994] Salus, P. H. (1994). *A Quarter Century of UNIX*. UNIX and open systems series. Addison-Wesley Pub. Co, Reading, Mass.

[SARKAR, 1996] SARKAR, S. (1996). "the boundless ocean of unlimited possibilities": Logic in carnap's logical syntax of language. *Logical Empiricism at Its Peak: Schlick, Carnap, and Neurath*.

[Schlatter and Aizawa, 2008] Schlatter, M. and Aizawa, K. (2008). Walter pitts and "a logical calculus". *Synthese*, 162(2):235–250.

[Smalheiser, 2000] Smalheiser, N. R. (2000). Walter pitts. *Perspectives in Biology and Medicine*, 43(2):217–226. Volume 43, Number 2, Winter 2000.

[Smith, 1996] Smith, B. C. (1996). *On the Origin of Objects*. MIT Press, Cambridge, Mass.

[Suchman, 1987] Suchman, L. A. (1987). *Plans and Situated Actions: The Problem of Human-Machine Communication*. Cambridge University Press, 2nd edition.

[Thompson, 1968] Thompson, K. (1968). Programming techniques: Regular expression search algorithm. *Commun. ACM*, 11(6):419–422.

[Turing, 1937] Turing, A. (1937). On computable numbers. *Proceedings of the London Mathematical Society*, 2(42):230–65.

[van Dam and Rice, 1971] van Dam, A. and Rice, D. E. (1971). On-line text editing: A survey. *ACM Comput. Surv.*, 3(3):93–114.

[Vleck, 2008] Vleck, T. V. (2008). Multicians.org. http://multicians.org/.