ON THEORETICAL COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE

ANTTI J. YLIKOSKI, AALTO UNIVERSITY,
SCHOOL OF SCIENCE

May 7, 2015

Keywords: computational complexity, theory of computation, artificial intelligence.

Abstract: This dissertation concerns the novel and fruitful relationship between theoretical computer science and artificial intelligence, and in general two subfields in a field if scientific endeavour.

Dissertation for the degree of the Doctor of Technology to be presented with due permission for public examination and criticism in Auditorium X at the Aalto University on the Yth of Zapember 20ZZ, at 12 o'clock at noon.

Author's email:
antti.ylikoski@aalto.fi
antti.ylikoski@hut.fi
antti.ylikoski@gmail.com
antti.ylikoski@elisanet.fi
antti.ylikoski@windowslive.com

Author's URL address:
www.aalto.fi/~ajy
www.hut.fi/~ajy

Contents

Chapter 1

Foreword

Douglas Lenat has been quoted to have said that he was actually given an academic degree for doing something that he loved. This also holds for me and this work. I always have been fascinated by the Theoretical Computer Science. I also have considered myself an Artificial Intelligence researcher. So it is very appropriate that this work concerns the novel connection between Theoretical Computer Science and Artificial Intelligence.

Originally, I was creating a PhD thesis concerning predicate logic based inference systems. In order to educate myself, I read [Cormen-Leiserson-Rivest, 1995] and became very interested in the $\mathcal{P}? = \mathcal{NP}$ problem. For some time, this thesis involved that problem. The subject of the thesis was "On Nondeterministic Polynomial-Time Computation".

However, it eventuated that many interesting side issues arose from my work. Then, I changed the subject of the thesis to "Contributions to the Theory of Computation".

It turned out that the most important result which arose from my work was the connection between theoretical computer science and artificial intelligence. This finally became the subject of the thesis. Therefore, "On Theoretical Computer Science and Artificial Intelligence".

While creating this work I learned the skill to read and author formal discussions. This is indeed an independent skill which is indispensable to a computer scientist and actually similar to the skill of driving a car: it can be consciously learned, this requires nontrivial effort but it is not exceedingly difficult, and the skill does not go away with time.

Usually, a scientific study has a certain set of hypotheses; methods; goals; techniques; and findings of the research. This work is a mathematical-formal study

whose method of research is the formal mathematical method. ("The most important tool of the mathematician is the sofa.") The goal has changed during the course of the work; finally I decided to collect the scientific results under the umbrella of the relationships between theoretical computer science and artificial intelligence.

I recall a scientific AI paper such that one of its reference sources was published 27 years before the publication date of that paper. This does not authomatically imply that the author(s) had been working on their paper for 27 years, but it does give an impression of the amount of dedication, working hours and perseverance that is required to create a bit of real science.

Finally, I note that this work is rather old, and has been in the public Internet for a very long time – I was being ill and wanted to publish this long, long before there was there the opportunity for the actual public defence of the thesis – so I have seen that the idea of the cross-fertilization between AI and TCS, and generally speaking between two of several ones of the subfields of the big field of Computer Science has become more or less common and commonly applied knowledge in the trade.

Quotation # 1:

> Hine lo yanum v'lo yishan
> lo yanum, lo yishan
> Hine lo yanum v'lo yishan
> Shomer Israel
>
> (Neither does slumber nor does sleep
> neither slumbers, nor sleeps
> Neither does slumber nor does sleep
> He who keepeth Israel)

Quotation # 2:

AI wizards do it with resolution.

Quotation # 3:

As they go out, they will see the corpses of those who rebelled against Me, where the devouring worm never dies, and the fire is not quenched. All mankind will view them with horror.

—The end words of the Book of Yeshayahu

Quotation # 4:

And for all time to come, the name of the town shall henceforth be: THE LORD IS HERE.

—The end words of the Book of Yehezgel

1.1 Epilogue to the Prologue

In the accompanying Figure there is there an excerpt from the Internet news:, from the newsgroup news:comp.ai. It is the author's entry there in the year 2006.

So this idea of the cross-fertilization between these two subfields of Computer Science had been presented already in the year 2006 by me. Of course, in the history of science, innumerable researchers in numerous subfields of scientific endeavour have used and utlized the idea of the "cross-fertilization" – I'm one of the group of the scientists who have been presenting and furthering this idea.

So the central idea in my work can be summarized as follows:

In a particular field of science, there exist there a number of individual subfields. It is often a good idea to look for some cross-fertilization between these different individual subfields, in the most general case between $\mathbf{n}$ *subfields, where* $\mathbf{n}$ *integer and* $\mathbf{n} >= 1$.

The (corrected) manuscript of my PhD thesis can be accessed in

    www.hut.fi/~ajy/diss-1.pdf

The main scientific result of the work is that there is a connection between theoretical computer science and artificial intelligence. Therefore, the work may be of interest to the readers of this newsgroup.

I would like to note here that I'm not a crank, and the work is not garbage. I posted an earlier version of the work in the comp.theory, and a couple of days after that I was informed by ordinary mail that my name and biography are going to be published in two international reputable Who's Who books. If the reader would like to verify this then I recommend that he/she get the Marquis' Who's Who in the World 2005, 22nd edition, and find my biography there ("Ylikoski, Antti Juhani").

But -- the work is controversial!

Firstly, there are individuals who have a political motivation to attack me and my work; the scientific quality of the attacks is of no importance. (And -- No, I'm not a Russian agent!)

Secondly, I'm presenting some rather bold claims in the work, and I may have to repudiate them later... But I very much prefer receiving the criticism (valid criticism) here to receiving the criticism during the public defence of the work!

    Cheers, Mr Antti J. Ylikoski
    antti.ylikoski (at) hut.fi
    www.hut.fi/~ajy

Figure 1.1.1 The author's entry in news:comp.ai in 2006.

Chapter 2

On the Philosophy of Science

There is there a problem Pr, which I do not constrain here in other ways than that Pr is a real-world object, which can be described with the natural language, and this description has a finite length.

Here – as elsewhere – I note that it is remarkably easy to express various uncomputable objects with the natural language.

From Pr, we can form the formal description of the problem

$DSC = DSC(Pr)$

where DSC is a discrete symbol string, whose length is finite and whose symbol alphabet Σ is finite. DSC could for example be the description of the problem with predicate logic, or it could be the description with a LISP symbolic expression (S-expression). (In [Papadimitriou, 1981], the author skeptically notes that it may be impossible to ever formalize natural language.)

Now, from DSC we will form a description of the problem in such a form that it can be given to an information processing entity to solve Pr. This description could be for example

$LISP - DSC = LISP(DSC)$

ie. DSC having been coded so that (as a LISP S-expression) the problem Pr can be given to a LISP program to be solved, or the description could be eg.

$TM - DSC = TM(DSC)$

the description of the problem Pr represented as a finite symbol string, which can be written onto the tape of a Turing machine, for the solution of Pr.

In the above, it is really a question of describing the reality with a model. The description of the reality with a theory is a question which has been extensívely

studied by philosophy. Here this theory is a formal model – because the intention is to solve the problem with a machine.

2.1 Philosophy of Language, Epistemology, Subfields of Philosophy

I discussed the above question, ie. describing the reality with a model, with some scientists in the Internet, and was suggested to that I should "take a crash course in the philosophy of language".

The philosophy of language is such a voluminous and sophisticated subfield of philosophy that I will not attempt to give a summary of it here, or a thorough discussion of the relationship of the philosophy of language to software engineering. But I do note that such a relationship does exist, ie.

Thesis 2.1.1 *There is a connection between philosophy of language and general Computer Science.*

□

See

`http://en.wikipedia.org/wiki/Philosophy_of_language`

But, the reader can keyword search relevant books in the Amazon (http://www.amazon.com and http://www.amazon.co.uk) and in the Alibris (http://www.alibris.com and http://www.amazon.co.uk).

Two relevant sources are the references [Miller, 2007] and [Lycan, 2008]. But significantly more do exist.

Another (also voluminous and sophisticated) subfield of philosophy which is relevant to the question of describing the reality with a theory is epistemology, the science of science, and of knowledge. Again–

Thesis 2.1.2 *There is a connection between epistemology and general Computer Science.*

□

As above, the reader can keyword search relevant books in the Amazon (http://www.amazon.com and http://www.amazon.co.uk) and in the Alibris (http://www.alibris.com and http://www.amazon.co.uk).

See

http://en.wikipedia.org/wiki/Epistemology

Moreover, metaphysics, is relevant – the ontology of a particular metaphysics is relevant e. g. in forming taxonomies of concepts and beings (i. e. things); lists of things; and categorizations of objects.

See

http://en.wikipedia.org/wiki/Metaphysics

and

http://en.wikipedia.org/wiki/Ontology

For example, see the **OWL, the Ontology World Language** – see

http://en.wikipedia.org/wiki/Web_Ontology_Language

And, see the Web 2.0:

http://en.wikipedia.org/wiki/Web_2.0

2.2 Philosophy of Science

The subfield of philosophy which is particularly relevant with respect to forming models of the reality is the **philosophy of science.** Much of scientific endeavour indeed consists of building models of the reality!

Again, the philosophy of science is such a voluminous and sophisticated subfield of philosophy that I won't here attempt to summarize it, or give a detailed discussion of the relationship of philosophy of science to AI and TCS or Computer Science in general.

But we will note here that the relationship does exist, i. e:

Thesis 2.2.1 *The philosophy of science is related to Computer Science in general.*

□

2.3 Philosophy of Language

One more note about the philosophy of language:

Thesis 2.3.1 *The Philosophy of Language is relevant for Artificial Intelligence.*

□

This does not only concern NLP, Natural Language Processing, it also concerns subfields of AI such as Knowledge Representation.

Chapter 3

Definitions

This chapter contains mathematical definitions drawn from the Wikipedia, and professional literature. These points are presented rather tersely for the sophisticated reader. Blaise Pascal once noted in a letter to an individual that "this letter is longer than usually because I did not have enough time to make it short".

3.1 The Deterministic Turing Machine

See

`http://en.wikipedia.org/wiki/Deterministic_Turing_machine`

The *deterministic Turing machine, DTM* is the quintessential model of computation. It corresponds to a computation such that the process consists of discrete steps; each step requires a finite amount of work; and there are a finite number of steps in a halting computation. (And, an infinite number so to speak of steps in a nonhalting computation.)

3.1.1 Formal definition

This definition stems from [Papadimitriou, 1995].

Formally, a (deterministic) Turing machine is a quadruple $M = (K, \Sigma, \delta, s)$

where

- K is a finite set of *states* which correspond to instructions in a conventional computer;
- Σ is a finite set of *symbols* (we say that Σ is the *alphabet* of the machine M. We assume that K and Σ are disjoint sets. Σ always contains the special symbols $\sqcup$ and $\triangleright$: the *blank* and the *first symbol on the tape.*

- Finally, δ is the *transition function*, which maps $K \times \Sigma$ to $(K \cup \{h, \text{“}yes\text{”}, \text{“}no\text{”}\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}$.

As we noted, the set K is the instruction set of the Turing machine, and moreover the function δ is the *program* of the machine. The function δ specifies, for each combination of the current state $q \in K$ and the current symbol on the tape $\sigma \in \Sigma$, a triple $\delta(q, \sigma) = (p, \rho, D)$. p is the next state, ρ is the symbol that shall be overwritten over the symbol σ, and $D \in \{\leftarrow, \rightarrow, -\}$ is the Direction to which the cursor, i. e. the location of the symbol scanned on the tape shall move for the next instruction, i. e. the next discrete computation step.

The program starts as follows. Initially the state is s. The string is initialized to a $\triangleright$, which is followed by the input to the machine, a finitely long string $x \in (\Sigma - \{\sqcup\})*$.

Anything that operates according to these specifications is a deterministic Turing machine.

A *discrete state of a Turing machine* is called a *configuration*. Intuitively, a configuration contains a complete description of the state of the computation at the moment. We have

Definition 3.1.1 *A configuration of a Turing machine M.*

A configuration of a Turing machine M is a triple (q, w, u) where $q \in K$ is a state, and w, u are strings in $\Sigma*$.

q is the current state of the Turing machine, w is the string to the left of the cursor (== reading / printing head), including the symbol scanned by the cursor (which symbol is always the rightmost symbol of w), u is the string to the right of the cursor, possibly empty.

□

Let M be a fixed Turing machine. Next, we shall define what happens when a configuration (q, w, u) *yields* another configuration (q', w', u') *in one step*.

Definition 3.1.2 *A configuration of a certain Turing machine yielding another configuration in one step.*

A configuration (q, w, u) *yields another configuration* (q', w', u') *in one step*, denoted $(q, w, u) \rightarrow^M (q', w', u')$ iff one step of the transition function $\delta(p, \sigma)$ can take the machine from the previous configuration to the latter configuration.

□

After we have defined the relationship "yields in one step" we can define the "yields" relationship as its transitive closure, in the obvious manner. This relationship is written as $(q, w, u) \rightarrow^{M^*} (q', w', u')$.

After that, we can define the "yields in k steps" relationship – which shall later be utilized as the computation time of the Turing machine computation. A computation that takes the Turing machine from the first configuration (q, w, u) via k applications of the transition function $\delta(p, \sigma)$ to the last configuration (q', w', u') is said to yield the result (i. e. the latter configuration) in k steps. This concept is written as $(q, w, u) \rightarrow^{M^k} (q', w', u')$.

3.1.2 Deciding and Accepting a Language

Definition 3.1.3 *Deciding, and accepting, a language $L = L(Prb)$*

Let $L = L(Prb) \subset (\Sigma - \{\sqcup\})*$ be a language, corresponding to the problem Prb, i. e. a set of strings of symbols. Let M be a Turing machine such that, for any string $x \in (\Sigma - \{\sqcup\})*$, if $x \in L$, then $M(x) =$ "yes'' i. e. M on input x halts at the "yes" state, and, if $x \notin L$, then $M(x) =$ "no''.

Then M *decides* L.

If L is decided by some Turing machine M, then L is called a *recursive languare.*

We say that M *accepts* L whenever, for any string $x \in (\Sigma - \{\sqcup\})*$, if $x \in L$, then $M(x) =$ "yes'' – however, if $x \notin L$ then the computation $M(x)$ either halts in the "no" state, or will not terminate. (Often we cannot tell by waiting which alternative will the case.)

If L is accepted by some Turing machine M, then L is called *recursively enumerable*, abbreviated *r. e.*

□

3.2 The Nondeterministic Turing Machine

See

`http://en.wikipedia.org/wiki/Nondeterministic_turing_machine`

The *nondeterministic Turing machine, NDTM* is a generalization of the *deterministic Turing machine* such that the computation is nondeterministic: in a particular state with the head scanning a certain symbol, the transition relation Δ may direct the machine to several distinct states (and several different symbols may be printed on that tape square.)

3.2.1 Formal definition

This definition stems from [Papadimitriou, 1995].

Definition 3.2.1 *A nondeterministic Turing machine.*

A *nondeterministic Turing machine* is a quadruple:

$M = (K, \Sigma, \Delta, s)$

where K, Σ, Δ and s are as in the definition of the *DTM* above, except that Δ is a many-valued relation.

Now, the nondeterminism of the machine is manifested in the *relation* Δ – it is no longer a function from $K \times \Sigma$ to $(K \cup \{h, \text{“}yes\text{”}, \text{“}no\text{”}\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}$, but instead a *relation* $\Delta \subset (K \times \Sigma) \times ((K \cup \{h, \text{“}yes\text{”}, \text{“}no\text{”}\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\})$.

In other words, for each individual state – symbol-on-the-tape combination, there may be **more than one** *appropriate next steps – or* **none at all**.

□

The “yields in one step” relation is no more a function, it is a many-valued relation, but is it defined analogously to the DTM. Similarly for the “yields” and “yields in k steps” relations, which are defined analogously, and I don't want to repeat those definitions here for the sophisticated reader.

For a nondeterministic TM to decide a language, is defined as follows. I have not came across in the professional language the concept of a NDTM accepting a language.

Definition 3.2.2 *A nondeteministic Turing machine deciding a language.*

Let $L = L(Prb)$ be a language and N a nondeterministic Turing machine. We say that N *decides* L if for any $x \in \Sigma*$, the following is true: $x \in L$ iff $(s, \triangleright, x) \rightarrow^{N*}$ $(\text{“}yes\text{”}, w, u)$ for some strings w and u.

So an input is accepted if there exists *some* sequence of nondeterministic computation that results in “yes”.

□

Definition 3.2.3 *Deciding a language in a nondeterministic time $f(n)$.*

We say that a NDTM N decides the language L *in time f(n)*, where $f()$ is a function from $\mathcal{N}$ to $\mathcal{N}$, if N decides L, and, furthermore $\forall x \in \Sigma*$, if $(s, \triangleright, x) \rightarrow^{N^k}$

(q, u, w) then $k \leq f(|x|)$.

□

3.3 The Class $\mathcal{P}$

See

`http://en.wikipedia.org/wiki/Polynomial_time#Polynomial_time`

The computation time of a function is defined as the number of discrete steps that the computation needs to complete. (That is, if the computation halts. Otherwise the complexity is not defined.)

The complexity of the input to a Turing machine is defined as the number of squares on the tape that the input takes, i. e. $|x|$ if the input word on the tape of the Turing machine is x.

Therefore, the complexity of deciding a language $F(x)$, x is the input word, is defined as the number of computation steps $T(|x|)$ needed, as a function of the size of the input.

Definition 3.3.1 *The complexity class P.*

We denote by $TIME(T)$ the class of languages and problems $L = L(Prb)$ which can be decided by a deterministic Turing machine in time T.

We define $\mathcal{P}$ (for polynomial-time decidable) to be the class of languages

$\mathcal{P} = \bigcup\{TIME(n^d) : d \text{ integer and } d > 0\}$.

□

3.4 The Class $\mathcal{NP}$

See

`http://en.wikipedia.org/wiki/Nondeterministic_polynomial_time`

Definition 3.4.1 *The complexity class NP.*

Let T() be a function from $\mathcal{N}$ to $\mathcal{N}$. Let Prb be the problem to be solved, and let $L \subseteq \Sigma*$ be the language corresponding to Prb. Let

$M = (K, \Sigma, \Delta, s)$

be a nondeterministic Turing machine, as defined above.

We say that M **accepts** L **(and solves** Prb**) in nondeterninistic time** T if the following holds. For all input words $w \in \Sigma*$,

$w \in L$ if and only if $(s, \triangleright, x) \rightarrow^{M^t} (h, u, w)$

for some $u, w \in \Sigma*$, and $t \leq T(\|w\|)$.

We say that L is **acceptable in nondeterministic time** T if there exists a nondeterministic Turing machine M that accepts $L = L(Prb)$ in nondeterministic time T. The class of languages acceptable in nondeterministic time T is denoted by $NTIME(T)$.

Finally, we define

$\mathcal{NP} = \bigcup\{NTIME(n^d) : d \text{ integer and } d > 0\}$

□

3.5 Alternative Definition of the Class $\mathcal{NP}$

This section presents another, very widely known definition of the class NP.

Definition 3.5.1 *A polynomially balanced language.*

Let Σ be an alphabet, and let ";" be a symbol not in Σ . Consider a language $L' \subseteq \Sigma*;\Sigma*$. We say that L' is **polynomially balanced** *if there exists a polynomial $p(n)$ such that if $x; y \in L'$, then $|y| \leq p(|x|)$.*

□

With that definition, we have

Theorem 3.5.1 *The class $\mathcal{NP}$ with polynomial-time certificates.*

Let $L \subseteq \Sigma*$ be a language, where the symbol $; \notin \Sigma$, and $|\Sigma| \geq 2$.

Then $L \in \mathcal{NP}$ if and only if there exists a polynomially balanced language $L' \subseteq \Sigma*;\Sigma*$ such that $L' \in P$, and $L = \{x :$ there exists a $y \in \Sigma*$ such that $x; y \in L'\}$.

Above the operator ; denotes the concatenation of languages.

Proof.

" $\Rightarrow$ "

If the language $L \in \mathcal{NP}$ then there exists a nondeterministic Turing machine M which decides the language L in a polynomial time. Then a certificate for an input word x is precisely (the trace of) any accepting computation of M on the input x. This accepting computation can be simulated in a polynomial time to verify x.

" $\Leftarrow$ "

If such a language as the language L' above exists, then a nondeterministic Turing machine could nondeterministically try all possible certificates (it is easy to nondeterministically guess a symbol string from a finite alphabet Σ .) After we have the guessed certificate, we can use the deterministic polynomial-time Turing machine which decides $L' = \{x; y\}$. Then the language L is in the set $\mathcal{NP}$.

□

This result has been characterized as, "the class NP consists of problems for which a solution candidate is easy to check (because it has a polynomial-time certificate) but computing the solution is difficult."

3.6 The Linear Bounded Automaton

See

`http://en.wikipedia.org/wiki/Linear_bounded_automaton`

The following text is an excerpt from the Wikipedia:

In computer science, a linear bounded automaton (plural linear bounded automata, abbreviated LBA) is a restricted form of nondeterministic Turing machine.

3.6.1 Operation

Linear bounded automata satisfy the following three conditions:

1. Its input alphabet includes two special symbols, serving as left and right endmarkers.
2. Its transitions may not print other symbols over the endmarkers.
3. Its transitions may neither move to the left of the left endmarker or to the right of the right endmarker.

As in the definition of Turing machines, it possesses a tape made up of cells that can contain symbols from a finite alphabet, a head that can read from or write to one cell on the tape at a time and can be moved, and a finite number of states.

It differs from a Turing machine in that while the tape is initially considered to have unbounded length, only a finite contiguous portion of the tape, whose length is a linear function of the length of the initial input, can be accessed by the read/write head. Instead of having potentially infinite tape on which to compute, computation is restricted to the portion of the tape containing the input plus the two tape squares holding the endmarkers.

This limitation makes the LBA a somewhat more accurate model of "realistic" computers that actually exist than a Turing machine, whose definition assumes unlimited tape.

3.7 The Deterministic Finite Automaton

See

`http://en.wikipedia.org/wiki/Deterministic_finite_automaton`

From the Wikipedia:

In the theory of computation and automata theory, a deterministic finite state machine – also known as deterministic finite automaton (DFA) – is a finite state machine accepting finite strings of symbols.

For each state, there is a transition arrow leading out to a next state for each symbol. Upon reading a symbol, a DFA jumps deterministically from a state to another by following the transition arrow. Deterministic means that there is only one outcome (i.e. move to next state when the symbol matches (S0 -> S1) or move back to the same state (S0 -> S0)). A DFA has a start state (denoted graphically by an arrow coming in from nowhere) where computations begin, and a set of accept states (denoted graphically by a double circle) which help define when a computation is successful.

DFA recognize exactly the set of regular languages which are, among other things, useful for doing lexical analysis and pattern matching. A DFA can be used in either an accepting mode to verify that an input string is indeed part of the language it represents, or a generating mode to create a list of all the strings in the language.

DFA is defined as an abstract mathematical concept, but due to the deterministic nature of DFA, it is implementable in hardware and software for solving various specific problems. For example, a software state machine that decides whether or not online user-input such as phone numbers and email addresses are valid. Another example in hardware is the digital logic circuitry that controls whether an automatic door is open or closed, using input from motion sensors or pressure

pads to decide whether or not to perform a state transition (see: finite state machine).

I feel that it is not necessary to write down the precise formal definition for the sophisticated reader.

3.8 Nondeterministic Finite Automaton

See

`http://en.wikipedia.org/wiki/Nondeterministic_finite_automaton`

The following is a quotation from the Wikipedia:

In the theory of computation, a nondeterministic finite state machine or nondeterministic finite automaton (NFA) is a finite state machine, where for each pair of state and input symbol there may be several possible next states. Moreover, there are there transitions on the empty input (often designated as ϵ. These points distinguish it from the deterministic finite automaton (DFA), where the next possible state is uniquely determined.

Although the DFA and NFA have distinct definitions, it may be shown in the formal theory that they are equivalent, in that, for any given NFA, one may construct an equivalent DFA, and vice-versa: this is the powerset construction. Both types of automata recognize only regular languages.

Non-deterministic finite state machines are sometimes studied by the name subshifts of finite type. Non-deterministic finite state machines are generalized by probabilistic automata, which assign a probability to each state transition.

I feel that it is not necessary to present the full formal definition here for the sophisticated reader.

3.9 The Pushdown Automaton

See

`http://en.wikipedia.org/wiki/Pushdown_automaton`

From the Wikipedia:

In automata theory, a pushdown automaton (PDA) is a finite automaton that can make use of a stack containing data

PDA do not occur in this work. Therefore, we will not consider the precise mathematical definition here.

3.10 The $\mathcal{P}? = \mathcal{NP}$ problem

See

`http://en.wikipedia.org/wiki/P_versus_NP_problem`

Let the complexity classes P and NP be as defined above in this setion. The P vs. NP problem can be stated as:

is the class P equal to the class NP, or is it a subset of NP?

For discussion about this point see [Cook] and [Cook, 1976].

3.11 On the Concept of the Infinite

See

`http://en.wikipedia.org/wiki/Infinite`

There can be two different kinds of infinities.

1. A *potential infinity* means a mutable quantity INF, which can become larger than any given fixed quantity.
2. An *actual infinity* is a fixed object AINF, which is greater than any given quantity, fixed or mutable.

There exist a denumerable set of actual infinities — the *transfinite numbers*.

See

`http://en.wikipedia.org/wiki/Transfinite_numbers`

3.12 The intension and extension of a predicate P() in a universe E

See

`http://en.wikipedia.org/wiki/Intension`

Following is an excerpt from the Wikipedia:

In linguistics, logic, philosophy, and other fields, an intension is any property or quality connoted by a word, phrase or other symbol. In the case of a word, it is often implied by the word's definition. The term may also refer to all such intensions collectively, although the term comprehension is technically more correct for this.

The meaning of a word can be thought of as the bond between the idea or thing the word refers to and the word itself. Swiss linguist Ferdinand de Saussure contrasts three concepts:

1. the signifier — the "sound image" or string of letters on a page that one recognizes as a sign.
2. the signified — the concept or idea that a sign evokes.
3. the referent — the actual thing or set of things a sign refers to.

Intension is analogous to the signified, extension to the referent. The intension thus links the signifier to the sign's extension. Without intension of some sort, words can have no meaning.

Also see

`http://en.wikipedia.org/wiki/Comprehension_(logic)`

from which the following quote stems:

In logic, the comprehension of an object is the totality of intensions, that is, attributes, characters, marks, properties, or qualities, that the object possesses, or else the totality of intensions that are pertinent to the context of a given discussion. This is the correct technical term for the whole collection of intensions of an object, but it is common in less technical usage to see "intension" used for both the composite and the primitive ideas.

On the extension of predicates, see

`http://en.wikipedia.org/wiki/Extension_(predicate_logic)`

The Wikipedia quotes as:

The extension of a predicate – a truth-valued function – is the set of tuples of values that, used as arguments, satisfy the predicate. Such a set of tuples is a relation.

Summa summarum, we have:

Definition 3.12.1 *The intension and extension of a predicate in a mathematical context.*

The intension of a predicate PRD() is its meaning, semantics, content in a symbolic sense. The extension of a predicate PRD()is the set of objects (eg. n-tuples) which satisfy the predicate in the base universe E.

□

3.13 A property of a function, of a predicate, of a Turing machine etc.

From the above, we easily get:

Definition 3.13.1 *The property of a function.*

A property PROPF() of a function F() is a predicate on a base universe E, which in this case is a set of functions, the default case being E = { the class of all functions }.

□

Definition 3.13.2 *The property of a predicate*

A property PROPP() of a predicate PR() is a predicate on a base universe E, which in this case is a class of predicates, the default case being E = { the class of all predicates }.

□

Definition 3.13.3 *The property of a Turing machine.*

A property PROPTM() of a Turing machine TM() is a predicate on a base universe E, which in this case is a class of Turing machines, the default case being E = { the class of all Turing machines }.

□

Chapter 4

The Classical Church-Turing Thesis Revisited

This chapter contains one more of my theorems which show the close relationship between theoretical computer science and artificial intelligence. In the chapter, we will see a novel characterization of that which can be computed, and its intimate relationship to artificial intelligence.

4.1 The Classical Church-Turing Thesis

The so to speak classical Church-Turing thesis states that the classical deterministic Turing machine corresponds to that which can be computed. The thesis has been augmented, by stating that the below mentioned models of computation all are equivalent, and, that they correspond to that which can be computed:

- mu-recursive functions, which can be extended to strings with the Gödel numbering [Lewis-Papadimitriou, 1981];
- Chomsky's general grammars [Papadimitriou, 1994];
- Turing's deterministic and nondeterministic machine [Papadimitriou, 1994];
- Post's finite combinatory processes [Post, 1936];
- Church's lambda calculus, extended to strings with the Gödel numbering [van Leeuwen, 1990] and [Lewis-Papadimitriou, 1981];
- Markov's theory of algorithms [Markov, 1954];
- the Random Access Machine [Papadimitriou, 1994];
- Cellular Automata [Atallah, 1999].

All the above models of computation have in principle an infnite memory. But, so to speak realistic computers have a finite memory whose size is bounded from above, and the time to execute the computation is limited from above.

As a theorem,

Theorem 4.1.1 *Finiteness of real computation*

All computations that a realistic computer carries out are finite and bounded from above, with respect to computation resources such as time and space.

Proof

The number of atoms on Earth is finite.

According to the modern cosmology, the universe is 13.7 plus/minus 0.2 billion years old. It is possible that (but it is not known if) the universe is actually infinite, and it will expand forever at an accelerating speed.

If the universe is infinite and we have in our use some kind of hyper-technology which makes it possible for us to carry out mining operations at distant stars, then we could in principle construct a true Turing machine which has an unlimited amount of memory. But, at the moment this is science fiction. I feel that it is currently scientifically safe to have the above definition of a "realistic" computer.

Moreover, the time to construct a computing device is limited from above. There is, in principle, a way to transcend this limitation. If we had a colony of nanotechnology machines which would multiply exponentially like bacteria and which would construct the computing device, then we could get rid of this limited construction time limitation. But, currently this is science fiction.

Furthermore, the space in which to house the computing device is limited from above. Theoretically, we could transcend this limitation with progressively better nano- pico- and so forth technology, but at the moment this is science fiction.

And, the limitation about the limited computation time could theoretically be coped with by devising processors with continually higher and higher clock frequencies ad infinitum; or, exponentially multiplying processors for a problem which is suitable for parallel processing; but this is science fiction at the moment.

□

In the Wikipedia article on the so-called **GOFAI, "Good Old Fashioned Artificial Intelligence research"** it is noted that the GOFAI is to a large extent based on discrete symbol manipulation.

See

`http://en.wikipedia.org/wiki/GOFAI`

Following is an excerpt from that article:

The approach is based on the assumption that many aspects of intelligence can be achieved by the manipulation of symbols, an assumption defined as the "phys-

ical symbol systems hypothesis" by Alan Newell and Herbert Simon in the middle 1960s. The term "GOFAI" was coined by John Haugeland in his 1986 book Artificial Intelligence: The Very Idea, which explored the philosophical implications of artificial intelligence research.

Also see

`http://en.wikipedia.org/wiki/Physical_symbol_systems_hypothesis`

These models of the Church-Turing thesis carry out precisely the operation of discrete symbol manipulation – which result has been discussed by us below. As I noted, the GOFAI is equivalent to finite, bounded from above discrete symbol processing. About this point, I moreover include in the following an excerpt from an Internet discussion:

```
Entry in the news://comp.ai newsgroup by:

Anders N Weinstein

Date of entry: 13th April 1998, 10:00

---- Start of quote by Weinstein

Fine, but you originally spoke of the central system doing "reasoning",
moreover using results of "probing for details of what they would
do under various circumstances". That suggests a rich conceptual
repertoire in which to reason.
>could be distributed in the interface with the real world.  My actual
>position is that it is better not to think of there being a central
>system at all.  In some sense, the brain coordinates perceptual and
>motor activities, but thinking is not a brain function at all.
>Rather, thinking is a whole person activity which makes heavy
>coordinate use of motor and perceptual operations by the brain.

OK, that's just dandy by me.
I guess my point was that the model of central reasoner, with its
symbolic concepts, propositions, inference rules connected to
sensory-motor peripherals (however complex, subtle, adaptive) is
basically the traditional one, the one we inherit from Descartes. As I
see it it is pretty much the model of GOFAI, e.g. Jerry Fodor's idea
``classical'' symbol processing architectures hooked up to modular input
systems just dresses it up in new clothing.
But that does not seem to be what you are pushing in the end.
```

>No, because the central system is too simple. It is little more than
>a data pathway between perceptual units and motor units.

OK. But again, your post spoke of the central system as *reasoning*.
>particular, it has no concepts that relate to the world. It only has
>concepts which relate to control functions. Concepts that relate to
>the world are virtual concepts which emerge from the system as a
>whole.

That's great. I suppose this means that whenever *I* speak of concepts in any connection I must be talking about what you call "virtual concepts", since I take it concepts are elements that have their role only in the rational life to be found at the level of the whole person.
My idea is that at the level of the whole person we will find many of the features traditionally ascribed to the rational Cartesian ego. But as long as we situtate these features at the right level, we can avoid entirely the embarassing metaphysics of a mysterious "self" conceived as an inner, possibly immaterial, homunculus-like central "executive" that receives and interprets inputs from sensory peripherals and sends outputs to motor peripherals.
>>Then perhaps one could say that something a bit like CYC could be an ok
>>model of the central reasoner, only, alas, one that is not in fact
>>hooked up to the peripherals (Fodor's modular "input systems") that
>No. CYC has far too much formal knowledge for that role. The
>central system needs instead to be a simple control system which can
>probe the device drivers for the type of knowledge that has been
>programmed into CYC. The results of the probing will be far more
>robust than any CYC knowledge could be, for they will come from a
>device driver which has been finely tuned to optimize its interaction
>with the world just right.

But this brings us back to my original problem about the interface, and what representational system must be used to pass information across it. The response from the driver must ultimately issue in the depositing of propositions (or at least offering up of candidates) into the knowledge base if it is play any role in the central reasoner's symbolic *inference* or *reasoning*. Then there could not be any "richer" information in the driver, beyond what is expressible in the central reasoner's language, that could make a difference to the inferences performed by the central system.

```
Again we come back to the problem that the only thing that could exert a
*rational* influence on the cognitive transitions of a inference
engine must be expressible in a language the inference engine understands
(can deal with).

------ End of quote by Weinstein
```

We have

Conjecture 4.1.1 *The models listed above, equivalent to the Turing machine, execute the operation of discrete symbol manipulation.*

To prove this, we must first create some kind of a formal model of "discrete symbol manipulation".

□

Definition 4.1.1 *Definition of "finite" and "bounded" computation.*

To say that a computation is finite, is equivalent to saying that it is terminating. A bounded from above computation is one whose all theoretical measures such as computation time, memory space, amount of I/O, etc etc. are smaller than a given fixed quantity C^{max}.

Theorem 4.1.2 *Good Old Fashioned Artificial Intelligence.*

Good Old Fashioned Artificial Intelligence is equivalent to finite, bounded from above discrete symbol processing. To prove this, we must create some kind of a formal model of current state-of-the-art symbolic processing architectures, here primarily software architectures.

□

4.2 A Formal Glance at Symbolic Processing

4.2.1 LISP

The rigorous (but not tightly formal) definition of the Common LISP can be found in the following three references:

- [Steele, 1990] and

- [Graham, 1996]
- [CL-Hyperspec, 2011]

There exists, as it is being mentioned in Graham's reference, an ANSI standard for the Common LISP.

4.2.2 Prolog

Some good references concerning the Prolog language are:

- [Clocksin-Mellish, 2003]
- [O'Keefe, 1990]
- [Sterling-Shapiro, 1994]
- [Bratko, 1986]

The reference by Clocksin and Mellish mentions that there exists an ISO standard for Prolog.

4.2.3 Smalltalk

Some good references on the Smalltalk language are:

- [Goldberg-Robson, 1983]
- [Hunt, 20##]

There exists there an ANSI standard for Smalltalk.

4.2.4 The Formal Definition of the GOFAI

Based on the abovementioned presentations of three AI languages, and also information on the use of those tools in e. g. machine learning, automated reasoning, expert systems, etc etc. it would be possible to build an entire formal, rigorous theory of the GOFAI symbolic processing.

This is, however, an extremely sizable and extremely sophisticated task, which we shall here take as being outside the scope of this work. But I believe that such a formal, rigorous theory is possible to devise, and there is here some space for future, creative, substantial basic research in theoretical computer science and artificial intelligence.

We have

Research program.

Devise a formal, rigorous, mathematical theory of symbolic processing along the guidelines given in the above subsection.

□

4.3 A Novel Characterization of That Which Can Be Computed

We have

Theorem 4.3.1 *The abstract description of that which can be realistically computed, in concordance with the classical Church-Turing thesis, can be characterized as* ***finite, bounded from above discrete symbol manipulation****, which is precisely the same thing as the* ***GOFAI, i. e. Good Old Fashioned Artificial Intelligence****.*

Proof.

The Turing machine is a discrete symbol manipulator. Because of the Church-Turing thesis, it is the most general discrete symbol manipulator, i. e. if there were any other discrete symbol manipulator model Z then it would be the case that the Turing machine would subsume Z.

Therefore, that which can be realistically computed can be characterized as finite, bounded from above discrete symbol manipulation.

The GOFAI is equivalent to finite, bounded from above discrete symbol processing. The fact that novel symbols can occur in AI systems such as LISP can be taken care of by programming a symbol table on a Turing machine.

Therefore, the definition of that which can be realistically computed according to the above version of the Church-Turing thesis is precisely the same thing as the GOFAI.

□

I note here that the AI systems described in the modern AI Magazine [AI Magazine] make the early ("GOFAI") programs really toy programs for toy problems.

It is possible to present a counterargument to this proof: it is an extremely well known result that these various models of computation listed above are all Turing-equivalent, and it is a very well known result that the digital computer is (to an extent) Turing-equivalent (i. e. it has a finite memory), and therefore there is nothing novel in this result.

However, there is a novel nontrivial concept here, the concept of **finite, bounded from above discrete symbol manipulation**, and the equivalence of all the Turing-equivalent models and the GOFAI to it.

We have

Corollary 4.3.1 *There is a connection between Theoretical Computer Science and Artificial Intelligence.*

Chapter 5

On $\mathcal{P}? = \mathcal{NP}$, and a Research Program

In this chapter, we first discuss solving problems in $\mathcal{NP}$ in a polynomial time by means of a well-known and general paradigm: search, and pruning the search space. We prove a theorem that this is impossible. This leads us to other paradigms to solve the $\mathcal{NP}? = \mathcal{P}$ question, and we are led to the idea of exhaustively classifying algorithm families to solve $\mathcal{NP}$ problems in a polynomial time: *if we can exhaustively classify these algorithms and make a proof for each individual algorithm family, then we will have solved the* $\mathcal{P}? = \mathcal{NP}$ *problem.*

5.1 The First Theorem: Solving an $\mathcal{NP}$ Problem by Pruning the Search Space

In this section, we will discuss some properties of a function, which I call $F()$, which concerns the $\mathcal{P}? = \mathcal{NP}$ problem. The function $F()$ pertains to the problem of solving a nondeterministic polynomial-time problem deterministically in a polynomial time, by means of a well-known AI paradigm. Based on the results, we will finally outline a research program to study the $\mathcal{P}? = \mathcal{NP}$ problem, and a research program for theoretical computer science. This leads to one more corollary about the relationship of artificial intelligence and theoretical computer science.

This section is based on my scientific paper published in the ACM SIGACT NEWS [Ylikoski, 2004].

As in the discussion on the philosophy of science, let there a real-world problem Pr such that the corresponding formal description, $PR-DSC$, and the language, to a Turing Machine,

$PR - L - TM = TM(PR - DSC)$

is in the problem class $\mathcal{NP}$. Let M be the corresponding nondeterministic Turing machine. Then, M decides $PR-L-TM$. Let $PR-L-TM-DTM$ be the language corresponding to $PR-L-TM$, for the deterministic Turing machine M': then, M' decides $PR-L-TM-DTM$.

Typically, there is a set of nondeterministic polynomial-time Turing machines that decide the language $PR-L-TM$ which corresponds to Pr. Let this set be $NDTM-LIST = \{M_i\}$. Let M be one of these machines. In this paper, we are discussing the ramifications of trying to solve the problem Pr with a deterministic polynomial-time Turing machine.

Now, let us have in our hands the NDTM M, and one particular $w \in PR-L-TM$. We would like to decide w in a polynomial time with a deterministic TM. It turns out that it is possible to prove a difficultness result which finally takes us to the science of Artificial Intelligence.

To the function $F()$ is given the machine M and an input word w, and the purpose of $F()$ is to decide w in a polynomial time:

Definition 5.1.1 *The property $\mathcal{P}$ of functions – solving the problem in a polynomial time*

Let M and w be as above. Then the property of functions $\mathcal{P}(F)$ denotes that, the function $F(M,w)$ decides w in a polynomial time

□

Note that the set of functions $F()$ satisfying the predicate $\mathcal{P}(F)$ may contain a number of functions, or it may be the empty set – if it is impossible to decide w given as the input to M, in a polynomial time.

In the following discussion, we try to compute the function $F()$, and we shall see what follows.

Definition 5.1.2 *A problem Prb being nontrivially in $\mathcal{NP}$.*

If the problem Prb is in the class $\mathcal{NP}$, and no deterministic polynomial-time solution for Prb is known, then the problem Prb is nontrivially in $\mathcal{NP}$. Here note what I have previously said about teorems expressed in a verbal form.

□

Let us examine the computation of $F(M,w)$, if $F()$ is intended to satisfy the property $\mathcal{P}(F)$.

Let the size of the input word w to a Turing machine be $\|w\|$.

Theorem 5.1.1 *A difficultness result concerning the function $F()$.*

Let the problem Prb, as above, corresponding to M be nontrivially in $\mathcal{NP}$. In the computation of $F(M,w)$, **if the computation is based on the AI paradigm of serarching in a search space,** *we have to prune a search space to asymptotically zero per cent when $\|w\|$ tends towards infinity.*

Proof.

Let us examine the computation of the result of $F(M,w)$ from the formal definition of M and the input word w.

The set of all computations of the nondeterministic Turing machine M constitutes a tree. The size of this tree is superpolynomial in $\|w\|$. Let the size of this tree (the number of nodes in it) be $= X(\|w\|)$. Then, the order of growth of $X(\|w\|)$ is superpolynomial in $\|w\|$.

(If the size of the tree were polynomial, then $\mathcal{P}$ would be trivially in deterministic polynomial-time, and it would not be the case that $\mathcal{P}$ is nontrivially in $\mathcal{NP}$.)

Now, in a time polynomial in $\|w\|$ we can examine only a polynomial number of objects.

In the following, the tree of the computations of M is considered a search space.

Any search algorithm that we could hope to solve $\mathcal{P}$ (decide L) from M and w in a polynomial time must carry out two operations:

1. it must prune the search space, and
2. it must examine the pruned search space.

The phase 2 must get as an intermediary result a set pruned to a polynomial size in $\|w\|$, because it only can examine a polynomial number of objects in a polynomial time.

Note that here we actually are proving a result *on all algorithms that apply.*

Let the size of the pruned search space be $Y(\|w\|)$. Then, $Y(\|w\|)$ is a polynomial function. It is the size of the output of phase 1 above.

Now, the quotient of the size of the pruned set $Y(\|w\|)$ to the size of the set to be pruned $X(\|w\|)$, the number $Y(\|w\|)/X(\|w\|)$, tends asymptotically to zero when $\|w\|$ tends towards infinity.

Therefore, to compute $F(M, w)$ we must be able to prune a search space asymptotically to zero per cent when the size of the problem tends towards infinity. This happens in the phase 1 above.

□

This is merely a difficultness result, it does not prove that $\mathcal{P} \neq \mathcal{NP}$. For example, an explicit symbolic solution to the problem $[P, w]$ could exist. We could arrive to such a solution by means of a symbolic inference algorithm, in a polynomial time.

Also, F could carry out – to make another example – some complicated algebraic transformations and operations based on them – (cf. multiplying large integers by means of the FFT, the Fast Fourier Transform [Schönhage-Strassen]. Also see Levin on *universal search*.)

5.2 Theoretical Computer Science and Artificial Intelligence

Next, I outline a research program towards solving the $\mathcal{P}? = \mathcal{NP}$ problem:

Research Program.

Study the formal theory of symbolic reasoning algorithms and apply it to the $\mathcal{P}? = \mathcal{NP}$ problem.

□

We have

Extended Research Program.

Study the formal theory of symbolic reasoning algorithms and apply it to the study of computer science.

□

Therefore,

Corollary 5.2.1 *There is an important connection between artificial intelligence, in particular the theory of symbolic reasoning systems, and theoretical computer science.*

□

5.3 On the final formal solution for the $\mathcal{P}? = \mathcal{NP}$ problem

If we want to formally solve the $\mathcal{P}? = \mathcal{NP}$ problem for the negative, i. e. $\mathcal{P} \neq \mathcal{NP}$ — which is what most theoreticians currently would seem to believe — we shall have to *prove a theorem on all algorithms (that apply), or a theorem equivalent to that.*.

Theorem 5.3.1 *The $\mathcal{NP}$ class and a theorem on all algorithms*

Proof.

The statement, "no algorithm can solve all $\mathcal{NP}$ problems in a polynomial time", is equivalent to the statement, "for all algorithms A(), the algorithm A() cannot solve the $\mathcal{NP}$ problems in a polynomial time". So the result is something that can be said to be immediately self-evident.

□

So we could group various solution attempts at this problem:

1. algoritms which utilize pruning the search space (which we have proved not to succeed)
2. algorithms which are given as the starting point a symbolic description of the problem, and which utilize Artificial Intelligence-style symbolic reasoning
3. algorithms which apply various algebraic transformations, analogous to the Schönhage-Strassen algorithm which multiplies very large integers by means of the Fast Fourier transform
4. Can the reader invent something more?

If we shall be capable of devising an exhaustive taxonomy of algorithms to solve the $\mathcal{P}? = \mathcal{NP}$ problem and we shall be able to prove that all the algorithm subgroups in question are unsuccessful — then we shall have proved the $\mathcal{P}? = \mathcal{NP}$ problem in the negative, which I understand is the current consensus amongst computer scientists.

However, we have

Conjecture 5.3.1 *Because the length of an algorithm, being written on a universal Turing machine tape, may be arbitrarily long, it may be the case that it is impossible to devise an exhaustive classification of all algorithms (that apply). However, it might be possible to prove a result such as, an $\mathcal{NP}$–complete problem cannot be solved in a polynomial time by an algorithm $\mathcal{A}$ whose length $\|\mathcal{A}\|$ is less than 10^{100} symbols.*

Chapter 6

Black Box Languages, and Symbolic Information

6.1 Formal Languages

See

`http://en.wikipedia.org/wiki/Formal_language`

The following has been quoted from the Wikipedia:

A formal language is a set of words, i.e. finite strings of letters, symbols, or tokens. The set from which these letters are taken is called the alphabet over which the language is defined. A formal language is often defined by means of a formal grammar (also called its formation rules); accordingly, words that belong to a formal language are sometimes called well-formed words (or well-formed formulas). Formal languages are studied in computer science and linguistics; the field of formal language theory studies the purely syntactical aspects of such languages (that is, their internal structural patterns).

Formal languages are often used as the basis for richer constructs endowed with semantics. In computer science they are used, among other things, for the precise definition of data formats and the syntax of programming languages. Formal languages play a crucial role in the development of compilers, typically produced by means of a compiler compiler, which may be a single program or may be separated in tools like lexical analyzer generators (e.g. lex), and parser generators (e.g. yacc). Since formal languages alone do not have semantics, other formal constructs are needed for the formal specification of program semantics. Formal languages are also used in logic and in foundations of mathematics to represent the syntax of formal theories. Logical systems can be seen as a formal language with additional constructs, like proof calculi, which define a consequence relation. "Tarski's definition of truth" in terms of a T-schema for first-order logic is an example of fully interpreted formal language; all its sentences have meanings that make them either true or false.

6.1.1 Words over an alphabet

An alphabet, in the context of formal languages can be any set, although it often makes sense to use an alphabet in the usual sense of the word, or more generally a character set such as ASCII. Alphabets can also be infinite; e.g. first-order logic is often expressed using an alphabet which, besides symbols such as $\wedge, \neg, \forall$ and parentheses, contains infinitely many elements $x0, x1, x2, \ldots$ that play the role of variables. The elements of an alphabet are called its letters.

A word over an alphabet can be any finite sequence, or string, of letters. The set of all words over an alphabet Σ is usually denoted by $\Sigma*$ (using the Kleene star). For any alphabet there is only one word of length 0, the empty word, which is often denoted by e, ϵ or λ. By concatenation one can combine two words to form a new word, whose length is the sum of the lengths of the original words. The result of concatenating a word with the empty word is the original word.

In some applications, especially in logic, the alphabet is also known as the vocabulary and words are known as formulas or sentences; this breaks the letter/word metaphor and replaces it by a word/sentence metaphor.

Definition 6.1.1 *A formal language.*

A formal language L over an alphabet Σ is a subset of $\Sigma*$, that is, a set of words over that alphabet.

□

In computer science and mathematics, which do not usually deal with natural languages, the adjective "formal" is often omitted as redundant.

While formal language theory usually concerns itself with formal languages that are described by some syntactical rules, the actual definition of the concept "formal language" is only as above: a (possibly infinite) set of finite-length strings, no more nor less. In practice, there are many languages that can be described by rules, such as regular languages or context-free languages. The notion of a formal grammar may be closer to the intuitive concept of a "language", one described by syntactic rules. By an abuse of the definition, a particular formal language is often thought of as being equipped with a formal grammar that describes it.

6.2 Black Box languages

When considering the Kleene star over an alphabet Σ, we can note that a language which is a subset of $\Sigma*$ can be defined by means of its *characteristic function* $\chi()$.

Let there be a symbol string $S \subseteq \Sigma*$.

Definition 6.2.1 *The definition of a formal language $\mathcal{LAN}$ by means of its characterstic function $\chi()$.*

The symbol string S is in the language $\mathcal{LAN}$ iff $\chi(S) = 1$. If $\chi(S) = 0$ then S is not in the language $\mathcal{LAN}$.

□

We can define

Definition 6.2.2 *A formal language as a black box language.*

Let S, $\mathcal{LAN}$ and $\chi()$ be as above. Then the language $\mathcal{LAN}$ is a **black box language** *if $\chi()$ is a "black box" whose internal functionings we cannot examine.*

□

6.3 Definition of the class $\mathcal{NP}$ with certificates

We have

Definition 6.3.1 *Polynomially balanced languages.*

This has been presented before.

□

Now we can present — sapienti sat — the definition of the class $\mathcal{NP}$ with certificates:

Theorem 6.3.1 *The class $\mathcal{NP}$ defined with certificates.*

Let $L \subseteq \Sigma*$ be a language, where $; \notin \Sigma$, and $|\Sigma| \geq 2$. Then $L \in \mathcal{NP}$ iff there exists a polynomially balanced language $L' \subseteq \Sigma*;\Sigma*$ such that $L' \in P$, and $L = \{x|$ there is a $y \in \Sigma*$ such that $x;y \in L'$ $\}$.

Proof

This has been presented before.

□

Moreover,

Theorem 6.3.2 *The class NP defined with the characteristic function $\chi()$.*

Let there be a language $\mathcal{LAN} \in \mathcal{NP}$. Let x be a word which we would like to decide – i. e. determine if $x \in \mathcal{LAN}$. We can do this by 1) finding out the certificate y and then 2) running the concatenation $x;y$ through the Turing machine which decides $\mathcal{LAN}$' precisely as above. Now we can see that the sequential running of Phase 1) above plus thereafter Phase 2) above together constitute the characteristic function $\chi()$ of the language $\mathcal{LAN} \in \mathcal{NP}$, if we define that $\chi()$ returns 1 if we have determined that $x \in \mathcal{LAN}$ and 0 otherwise.

□

Now we can present the main result of this section:

Theorem 6.3.3 *The $\mathcal{P}? = \mathcal{NP}$ problem and black box languages*

Let $\mathcal{L}$ be a language such that $\mathcal{L} \in \mathcal{NP}$. Let $\mathcal{L}$ be defined only with its characteristic function $\chi()$, and let $\mathcal{L}$ be a black box language so that the internal functionings of $\chi()$ cannot be seen outside.

Then $\mathcal{L}$ is in $\mathcal{NP}$ but not in $\mathcal{P}$.

It is a well known result that if the set $\mathcal{NP} - \mathcal{P}$ is nonempty then $\mathcal{P}! = \mathcal{NP}$.

Proof.

Because the only information available to the solver of the problem are the results that the black box outputs, the only algorithm to solve it is the generate and test algorithm, which will take exponential time in the average. Then $\mathcal{L}$ is in $\mathcal{NP}$ but not in $\mathcal{P}$.

□

6.4 A Counterargument, or Why We Cannot Collect Clay's Million Dollars

It superficially seems above that we have proved that $\mathcal{P} \neq \mathcal{NP}$. However — How could we present the above discussion with Turing machines?

It clearly is not a valid derivation for general Turing machines if we in the course of the proof modify the structure of TM's (so that e. g. the TM's cannot see certain information, this limitation being built into it in the construction of the proof.)

6.5 The $\mathcal{P}? = \mathcal{NP}$ problem and symbolic information

Even if the abovementioned attempt at solving the $\mathcal{P}? = \mathcal{NP}$ problem is flawed, I have included it in this work because it beautifully demonstrates the relationship between symbolic information and the complexity of computation.

Chapter 7

Notes on Kolmogorov complexity

See

`http://en.wikipedia.org/wiki/Kolmogorov_complexity`

7.1 Formal definition

The following has been quoted from the Wikipedia:

In algorithmic information theory (a subfield of computer science), the Kolmogorov complexity of an object, such as a piece of text, is a measure of the computational resources needed to specify the object. It is named after Soviet Russian mathematician Andrey Nikolayevich Kolmogorov.

Kolmogorov complexity is also known as descriptive complexity, Kolmogorov–Chaitin complexity, stochastic complexity, algorithmic entropy, or program-size complexity.

More formally, the complexity of a string is the length of the string's shortest description in some fixed universal description language. The sensitivity of complexity relative to the choice of description language turns out to be an issue which will not so to speak cause problems for the discussion.

It can be shown that the Kolmogorov complexity of any string cannot be more than a few bytes larger than the length of the string itself. Strings whose Kolmogorov complexity is small relative to the string's size are not considered to be complex. The notion of Kolmogorov complexity can be used to state and prove impossibility results akin to Gödel's incompleteness theorem and Turing's halting problem.

Definition 7.1.1 *Kolmogorov complexity $K(s)$ of a string s.*

We shall here choose an encoding for Turing machines, where an encoding is a function which associates to each Turing Machine M a bitstring $< M >$. If M is a Turing Machine which on input w outputs the string x, then the concatenated string $< M > w$ is a description of x. For theoretical analysis, this approach is more suited for constructing detailed formal proofs and is generally preferred in the research literature. The binary lambda calculus may provide the simplest definition of complexity yet. In this context we will use an informal approach.

Any string s has at least one description, namely the program

```
function GenerateFixedString()
   return s
```

If a description of s, $d(s)$, is of minimal length—i.e. it uses the fewest number of characters—it is called a minimal description of s. Then the length of $d(s)$ — i.e. the number of characters in the description — (or, in the case of a Turing machine, the number of symbols in the description) is the Kolmogorov complexity of s, written $K(s)$. Symbolically,

$K(s) = \|d(s)\|$

where $\|S\|$ = the number of charactes in the string S.

□

7.2 Different descriptions of a symbolic object

Let there exist a symbolic object called $SOBJ$. I do not constrain the $SOBJ$ in any other way than that it must be possible to define or descibe the $SOBJ$ in a natural language, and the description must have a finite length.

In particular, the $SOBJ$ may be a finite or an infinite set or a class.

There may be three different definitions for the *information contents* of the formal definition for the $SOBJ$:

1. the information carried by the "raw" binary representation,
2. the information contents of the symbolic representation as a symbol structure;
3. the Kolmogorov complexity of the binary representation.

here I'm making the restriction that the lenghts of these all three descriptions must be finite.

7.2.1 Shannon information

As it is very well known, given the information I, if we can choose one object out of N because of gaining the information I, the (Shannon) information contents of the information I is ${}^2logN$. Equivalently, if we are given K binary digits of information, we have got K bits of classical Shannon information.

So if the symbolic object $SOBJ$ mentoned in the beginning of this section has been encoded in ENC bits, then the bit count ENC gives a measure of the information contects of the symboic object $SOBJ$. Here we however shall note that that is a quite crude measure of the information quantity contained in $SOBJ$.

This measure of the information contents of $SOBJ$ we shall here call by the name $SHANNON(SOBJ)$.

7.2.2 Symbolic information

Let the object $SOBJ$ be encoded in a formal symbolic manner – e.g.

1. with a computer LISP S-expression; or
2. with a predicate logic definition, with possibly a number of predicate formulas.

Elsewhere in this work, I have noted that the amount of information carried by symbolic processing is qualitatively larger than the information amount possessed by nonsymbolic, "raw" binary information.

So there remains a research problem:

Research problem.

Define formally the amount of knowledge contained by a symbolic object, with a certain interpretation which is actually given by the human(s) who defined the object.

□

This measure (which I have in this work left undefined) we shall call $SYM - INFO(SOBJ)$.

7.2.3 Kolmogorov information

Let the $SOBJ$ be symbolically defined as above. Now we can calulate the Kolmogorov complexity of the symbolic definition $DEF(SOBJ)$:

`Kolmogorov complexity` $= K(DEF(SOBJ))$

This is the third measure of the knowledge contents of the object $SOBJ$, here we call it by the name $KOL - INFO(SOBJ)$.

7.2.4 Research problem: the relationship of the three

So here remains a major reseach topic:

Research topic.

Define and calculate the relationship of $SHANNON(SOBJ)$, $SYM-INFO(SOBJ)$, $KOL-INFO(SOBJ)$. *Based on this research, devise* **a measure of symbolic information.**

7.2.5 The connection of the above to the $P? = NP$ problem

Now, let us consider what will happen when a (team of) humans or a computer system attempts to decide an NP problem in a polynomial time.

It is easy to see that the probem solvers(s) are given a set of information in a symbolic form. Therefore:

Theorem 7.2.1 *Symbolic information and the* $P? = NP$ *problem.*

When attempting to solve an NP problem as efficiently as possible, the amount and quality of symbolic information at hand is crucial.

Proof.

This is immediately self-evident ("by inspection").

□

So we have

Conjecture 7.2.1 *If the problem solver (human, or computer, or whatever) has enough symbolic information, then the problem solver can solve an* $\mathcal{NP}$ *– complete problem in a polynomial time. On the contrary, if the amount of symbolic information is not sufficient, the problem cannot be solved in a polynomial time.*

Chapter 8

On Symbolic Processing

8.1 What is a Symbol?

In the most general sense, a symbol is an object which refers to another object:

Definition 8.1.1 *The mathematical definition of a symbol.*

A symbol is an object NM (the name of the symbol) which refers to another object DN (the denotation of the symbol). There is a function $S : S(NM) = DN$, which defines the symbols of a computer program.

□

Because both NM and DN can be any object in the mathematical sense, both the domain and the range of S are uncountably infinite.

Therefore, this mathematical definition of a symbol is significantly more powerful than the classical binary nonsymbolic information.

Namely, the classical Shannon definition of information says that when possessing N bits of information one can choose one out of 2^N objects. But the abovementioned mathematical definition of a symbol makes it possible to choose one object DN out of uncountably many. I would not quite say that this enables the user to possess "infinitely many bits" but I feel that it is justified to say that symbolic processing it qualitatively stronger than so to speak conventional bit manipulation.

The Shannon definition of information says that if we can choose one out of N objects, then we possess $log2(N)$ bits of information. If here we substitute $N =$ uncountably infinite, then we nominally get the result that we possess infinitely

many bits of information. As I said, I'm not seriously saying that we actually possess "infinitely many" bits – but, as I note above, I feel that it is justified to state that symbolic processing is much stronger that nonsymbolic bit manipulation.

Moreover, I note that above the semantics'es of the symbols are given by the human(s) who authored the program. Then, it is indeed possible to have one object chosen from uncountably many. (If the semantics were given only by the program then it would not at all be evident that we could choose one from uncountably many!)

But – if we do not possess a computer-usable, binary representation of S, NM and DN, then the symbols are useless! There must be a way to represent them in the finite binary space of the digital computer. If they merely float in the Platonic space of ideas then they are totally useless.

First, we have

Definition 8.1.2 *The basic properties of a symbol.*

A symbol (in a piece of software) is a collection of information, in software a record, which has:

- an atom, the computer usable name of the symbol and all the information that is attached to it. Typically, the print name of this atom shows something about the semantics of the symbol NM (ie. the name is self-documenting) and it describes DN, the object which the symbol refers to, which may be external (YERSINIA-PESTIS) or internal (COMPILED-FUNCTION);
- it may possess a value or a function definition;
- it has a property list, a list of properties and their values;
- some bookkeeping data such as garbage collection bits;
- a list of pointers to it, and from it.

These constitute the **basic properties of a symbol**.

□

Note that with these basic properties we can do a lot with the symbols – such as the 1970's natural language processing.

Yet, we have to devise a definition which describes the genuine S in the finite binary memory space of a real computer.

We have

Definition 8.1.3 *The denotational properties of a symbol.*

Let $S : S(NM_i) = DN_i$ be a mathematical symbol as above.

Let $DEN - BIN(S, i)$ be a collection of digital computer representable data which describe the semantics of $S : S(NM_i) = DN_i$ described in a finite, bounded from above binary digital representation. Then, $DEN - BIN(S, i)$ constitutes the **denotational properties of the symbol** $S : S(NM_i) = DN_i$.

□

If the symbol corresponding to the basic properties of $S : S(NM_i) = DN_i$ were MYCOBACTERIUM-TUBERCULOSIS, then the denotational properties of the symbol could represent, in the CYC (see Lenat et al. in [Google et al.]) style, that $S : S(NM_i) = DN_i$ is a bacterium of the genus *Mycobacteriaceae*, that it causes an infection which is often fatal, that we can fight the bacterium with inoculations and certain antibiotics, and so forth.

Now, we can present the full scale computer processable definition of a symbol:

Definition 8.1.4 *A computer processable symbol.*

Let $S : S(NM_i) = DN_i$ be a symbol.

- The mathematical definition of the symbol (this could exist as the documentation of the symbol),
- plus the basic properties of the symbol,
- plus the denotational properties of the symbol,

constitute $COM-SYM([S, i]$, the computer representable symbol corresponding to $[S, i]$.

□

Here we can see that the amount of information contained by symbolic processing is very different from the amount of information contained in so to speak ordinary bit manipulation: if the data structure representing a symbol requires B bits, then it is not at all the case that the symbolic program has B bits of information.

In conventional symbolic programming languages such as LISP and Prolog, the name of a symbol is a finite character string which ordinarily reflects the semantics of the symbol. But, from the mathematical standpoint this is unsatisfactory, because the set of finite character strings is denumerable, and the class of all objects (mathematically, it is a class, it is not a set) $\{DN\}$ is uncountably infinite.

It would not help to allow for infinite character strings to stand for symbol names, which is easy to see.

Now, we can define symbolic and 'conventional' information processing:

Definition 8.1.5 *Symbolic information processing.*

A symbolic information processing system is one whose basic data types are a symbol, as defined above, and a list of objects which may be symbols or lists of objects.

The symbolic system also may have auxiliary conventional data types such as real, double precision, character string, or multidimensional arrays, attached in a suitable manner.

□

We have

Definition 8.1.6 *A symbolic processing program.*

A symbolic processing program utilizes symbols as defined above. Its data structures include but are not limited to LISP S-expressions (which are equivalent to the Prolog `[|]` objects). The program can be represented as a LISP S- expression.

□

Theorem 8.1.1 *Arbitrary program structures can be expressed as LISP S- expressions, as well as arbitary (unpacked) data structures.*

Proof.

In the reference [Davis-Sigal-Weuyker, 1994] these constructions are actually carried out in the context of program semantics: both denotational and operational program semantics. See [Davis-Sigal-Weuyker, 1994] for more.

□

Definition 8.1.7 *Nonsymbolic, conventional information processing.*

A nonsymbolic information processing system is one which is based on nonsymbolic binary data types such as real numbers, characters, multidimensional arrays, and so forth.

A nonsymbolic system does not utilize the concept of a symbol as defined above.

□

Now, we can present the main theorem of this section:

Thesis 8.1.1 *On the power of symbolic systems.*

A symbolic information processing system is significantly, qualitatively, more powerful than a nonsymbolic system.

Proof.

As has been discussed above, Claude Shannon's definition of information as bits says that (as is well-known) that knowing one bit of information enables one to choose one of two alternatives, and knowing N bits of information makes it possible to choose one of 2^N different alternatives.

But, with symbolic processing, the class of all objects (mathematically it is a class, it is not a set) is uncountably infinite, and therefore when knowing a symbol and its referent we can choose one object from uncountably many.

Then, as above, being able to choose one object from N objects we possess $log2(N)$ bits of information, and substituting $N =$ uncountably infinite we would get infinitely many bits of information.

I'm not quite saying that symbolic programs can contain "infinitely many bits" of information, but I feel that it if justified to say that symbolic processing is qualitatively significantly more powerful than nonsymbolic processing.

Furthermore, the semantics of the symbols are given by the human(s) who wrote the program, and therefore it is grounded to say that we actually can choose one from uncountably many.

Another consideration, to the same effect. Consider the square root of 2. With symbols, we can represent it with a few bits: $SQRT(2.0)$, or $\sqrt{2}$. But its non-symbolic, binary representation is an infinite object. When calculating with symbolic representations of numbers we can calculate with the exact values of variables, which is not the case when carrying out computations with non-symbolic objects.

Actually, it were possible to devise an entire symbolic calculus for computing with symbolic, exact values. To a large extent, this has actually been carried out in the symbolic mathematics systems: MACSYMA/Maxima [Google et al.]; Mathematica [Google et al.]; and Maple [Google et al.]. Also, the reference [Fateman] concerns this point.

□

8.2 On the Power of Knowledge Representation Languages

The First-Order Predicate Logic (FOPL) can be used as a tool to evaluate the power of knowledge representation languages. I'm under the impression that usually, the modern knowledge representation systems have a representational power which is equal to or less than the power of FOPL. (see ART (Automated Reasoning Tool), KEE (Knowledge Engineering Environment), SOAR (Search, Operator, And Result), OPS5 (Official Production System), CLIPS (C Language Integrated Production System), JESS (Java Expert System Shell), SNePS (Semantic Networks Processing System) in [Google et al.]).

As in the context of symbols, we can define:

Definition 8.2.1 *The mathematical definition of a symbolic predicate.*

A symbolic predicate is an object NM (the name of the predicate) which refers to the predicate $PR(X_1, X_2, \ldots, X_n)$. Here, the PR is equivalent to a set of n-tuples – the values for the variables $X_1, \ldots, X_n$ for which $PR()$ is true.

This information could reside in the computer system as the documentation of the predicate $PR()$.

There exists a function $S : S(NM_i = PR_i())$ which defines the predicates of a program.

□

Definition 8.2.2 *The basic properties of the predicate* $S(NM_i) = PR_i()$.

The same data as in the case of a symbol, such as the property list of the atom corresponding to NM, constitute the **basic properties of the predicate** $F(NM_i) = PR_i()$.

So:

- an atom, the computer usable name of the predicate and all the information that is attached to it. Typically, the print name of this atom shows something about the semantics of the predicate $S(NM_i) = PR_i()$ (ie.

the name is self-documenting) and it descibes $PR()$, the actual predicate; this atom may refer to an external object [(POWER-PLANT-RADIOACTIVITY-MEASUREMENT-IS-OK)] or to an internal object [(COMPILE-EVAL-LOAD Prog25)];

- this atom may possess a value or a function definition;
- the atom may have a property list, a list of properties and their values;
- some bookkeeping data such as garbage collection bits;
- a list of pointers to it, and from it.

are the basic properties of the predicate PR().

□

Definition 8.2.3 *The denotational properties of the predicate* $S(NM_i) = PR_i()$.

The semantic information of $PR_i()$, encoded within the finite bounded from above memory space of the digital computer, eg. in the CYC [Lenat et al.] style, constitutes the **denotational properties of** $F(NM_i) = PR_i()$**.**

□

Above, an example of the denotational properties of a predicate (PR) would be the explicit symbolic formula to calculate the truth value of $(PRX_1, X_2, ...)$ for a certain set of arguments X_i.

Another example of the denotational properties of a predicate $(IS - A - RETROVIRUSX_1)$ would be the representation in the CYC style [Google et al.] of the facts that the genome of X_1 consists of two RNA strands inside a protein shell; that the reverse transcriptase enzyme will convert the strands into DNA inside a living cell; that this DNA is spliced into the host cell's own DNA, thereby hijacking the cell's own biological mechanisms; that mutations often happen to retroviruses; and so on.

Now, we can present

Definition 8.2.4 *A computer representable predicate.*

The combination of

- the mathematical definition of the predicate $PR()$, (which could exist in the software as the documentation of the predicate)
- the basic properties of $PR()$, plus
- the denotational properties of $PR()$

constitute **the computer representable predicate** $CR - REP(PR())$.

□

Now, we can prove

Theorem 8.2.1 *A knowledge representation language which has at least the power of FOPL is qualitatively significantly stronger than a nonsymbolic programming system.*

Proof.

Similarly to the above discussions: First, let us consider predicate logic without negation, without the logical connectives **and** and **or** and without quantification. Suppose that a program has the knowledge of a computer representable predicate $P(Obj_1, Obj_2, \ldots, Obj_n)$. How much information does the program possess? Shannon's definition of information is that possessing one bit of information enables one to choose either one of two alternatives; possessing N bits of information enables one to choose one of 2^N alternatives. And, if we can choose one of N objects, then we possess $log2(N)$ bits of information. For 2^N this becomes N, as should be.

The class of all predicates (mathematically, it is a class, it is not a set) in mathematics and philosophy is uncountably infinite; and the knowing of the abovementioned predicate enables one to choose one object from uncountably many. I'm not quite saying that the program possesses "infinitely many bits" of information; but again, as in the above discussion concerning the power of symbolic information processing systems, we can see that a reasoning program which utilizes a predicate logic based representation, or an equivalent representation, has in its hands something much more powerful than a nonsymbolic information processing system.

Namely – if we can with the symbolic predicates choose one object from uncountably many, then the above formula ($log2(N)$ bits, when N = uncountably infinite) would indicate that we possess infinitely many bits. So, it is justified to say that the symbolic predicates contain more information than so to speak nonsymbolic models of computation.

And, as above, the semantics of the predicates are given by the human(s) who have written the program, so it can actually be stated that we can choose one out of uncountably many.

Then, consider the full scale predicate logic, with negation, logical connectives as above, and quantification. Trivially, this has a greater expressive power than the abovementioned restricted predicate logic.

□

Theorem 8.2.2 *Let there be a problem Pr, as in the discussion about the philosophy of science, such that the corresponding language to a Turing machine is in the class NP and let $L = L(Pr)$ be the language corresponding to Pr. Then, the existence of symbolic information concerning Pr can very much help the solution of Pr.*

Proof.

Let us consider the SAT problem, the satisfiability problem of propositional logic well formed formulas. This problem is known to be NP-complete, ie. from all the problems in NP there is a polynomial-time reduction to SAT.

In [Cormen-Leiserson-Rivest, 1994] it is proved that SAT is equivalent to the satisfiability problem of electronic digital binary logic circuits.

In the same reference, the authors mention that this problem is of great importance in the area of computer-aided hardware optimization: if a large digital circuit turns out to be unsatisfiable, it can be replaced with a single wire whose output is permanently hardwired to digital 0. But, all digital circuits which humans design in practice possess a semantics. Humans very rarely create totally random circuits. Now, suppose that the circuit in question is a part of a digital 32 bit + 32 bit addition circuit. Then, it is easy to see that it is essentially impossible that the circuit will invariably output 0, unless it has been intentionally designed so that its output is hardwired to 0.

Here is an example of an NP-complete problem which has the property that the semantics of the problem makes it possible to solve the problem analytically, without search. Yet this does not imply that $P = NP$.

Therefore, there are NP-complete problems which have the property that symbolic, semantical information concerning the problem can to an extremely large extent help the solution of the problem. It is easy to see that to a large extent, the abovementioned digital circuit analysis problem can be solved with Artificial Intelligence techniques, by utilizing the semantics of the circuits that are created. Therefore, we can see that the $P? = NP$ problem is AI related.

□

As a result, we have

Corollary 8.2.1 *We can see that when a professional computer scientist encounters an NP-complete problem, it is a good idea to search for the semantics of the problem, and other symbolic information which can be utilized to restrict the search space, or in some other way to make the solution more efficient.*

□

Therefore, we have

Corollary 8.2.2 *The P ?= NP question is to a large extent an Artificial Intelligence problem.*

□

Chapter 9

Prolegomena to a Future Theory of Symbolic Processing

9.1 What is an A.I., ie. an Actually Intelligent Program?

I intended to write an entire chapter in this work on literature references and our work on symbolic information processing – but finally decided to leave this subject for my later work in life. There is there a Finnish proverb "ruokahalu kasvaa syödessä" , ie. "one's appetite will grow while eating", and this chapter is a case in point.

Therefore, I have simply decided to list here the reference works which I collected for this task, and leave it for the industrious reader to delve into them, if he should be interested in further studying symbolic information processing. In addition I wrote a subsection about Douglas Hodstadter's work, applied to AI in LISP. This work is the war theatre program below.

In other words – I decided that the British Museum algorithm [Winston, 1992] is not a particularly good way to write an academic thesis!!

For the Further Reading list, see the first Figure in this section.

9.1.1 An Actually Intelligent Program as a Strange Loop

I invented the following example to clarify what I wish to say by Hofstadter-style using the expression "jumping out of the system".

Suppose that we are building an AI system in LISP to support human officers running a war theatre. The routine is input signal intelligence (SIGINT) from the field, the routine communicates with human officers (with AI, including a machine learning component) and after the decisions have been committed, it communicates the decisions to the actual war theatre.

- The Computer Revolution in Philosophy, by Aaron Sloman, [Sloman, 1978];
- Gödel, Escher, Bach: an Eternal Golden Braid, by Douglas R. Hofstadter, [Hofstadter, 1980];
- The Mind's I, Fantasies and Reflections on Self and Soul, by Douglas R. Hofstadter and Daniel C. Dennett, [Hofstadter-Dennett 2000];
- Metamagical Themas: Questing for the Essence of Mind and Pattern, by Douglas R. Hofstadter, [Hofstadter 1985]
- I Am a Strange Loop, by Douglas Hofstadter, [Hofstadter 2007]
- Le Ton Beau de Marot, in Praise of the Music of Language, by Douglas R. Hofstadter, [Hofstadter 1997]
- On Intelligence [Hawkins 1999]
- Hermeneutiikka: Ymmmärtäminen tieteissä ja filosofiassa, by Hans-Georg Gadamer, [Gadamer 2005]

Figure 9.1.1 Books for Further Reading for this Section

In LISP, that could be, as in the second Figure in this program.

Now suppose that the routine GUI-communicate-with-humans is intelligent enough to acquire some novel knowledge: it decides to modify the main program to start utilizing radar intelligence from AWACS planes. This change is presented in the third picture in this section.

Now the original program has become a new program: the program has by means of AI actually modified itself and improved its performance.

Moreover, let us suppose that the human officers want their program to begin using HUMINT as a knowledge source. So the routine GUI-communicate-with-humans again modifies the main routine, this time adding a function to it. This for the second time modified main program is presented in the last figure in this subsection.

Now the program has once again become a new program: the program has the second time augmented its capabilities and generalized itself.

After all this: is the computer "only doing what it was instructed to do"? Well of course it is running a LISP program – and it was programmed and instructed by humans!! But I feel that it has – as Douglas Hofstadter says – "jumped out of the system" – it is no more the program that was in the first place programmed into the machine. It is a Strange Loop in the sense of Hofstadter: it has modified its own sources.

```
(defun SIGINT ()
  "This routine gets signal intelligence from the war theatre."
  ........
  ........)

(defun GUI-communicate-with-humans ()
  "The routine negotiates with human officers and both commits decisions
   and gets novel information from the humans by means of machine learning."
  ........
  ........)

(defun communicate-orders-to-theatre ()
  "The routine sends the commands decided by the humans plus the
   software to the theatre."
  ........
  ........)

(defun main-routine()
  (progn
    (SIGINT)
    (GUI-communicate-with-humans)
    (communicate-orders-to-theatre))

(defun run-war-theatre ()
  "The main program."
  (compile main-routine)
  (loop do ; Trivial: LOOP without clauses runs an infinite loop
    (funcall main-routine))
```

Figure 9.1.2 Original form of war theatre program.

```
;;; The following happens as a result of running the routine
;;; GUI-communicate-with-humans.
;;; The routine will partially rewrite the source code of the main
;;; routine, i. e. the program will modify its own sources.

(defun main-routine()
  (progn
   (collect-radar-intelligence-from-the-AWACSes)
   (SIGINT)
   (GUI-communicate-with-humans)
   (communicate-orders-to-theatre))
```

Figure 9.1.3 The modified source of the program.

```
;;; And the main routine is run as follows:

(defun run-war-theatre ()
  "The main program."
  (compile main-routine)
  (loop do
    (funcall main-routine))
```

Figure 9.1.4 The main routine of the war theatre program.

```
(defun HUMINT ()
  "This routine communicates with knowledge sources in order to
  get HUMINT concerning the war theatre."
  .........
  .........)

(defun main-routine()
  (progn
   (collect-radar-intelligence-from-the-AWACSes)
   (SIGINT)
   (HUMINT)
   (GUI-communicate-with-humans)
   (communicate-orders-to-theatre))

(defun run-war-theatre ()
  "The main program."
  (compile main-routine)
  (loop do
   (funcall main-routine))
```

Figure 9.1.5 The second version of the self-modified program.

(Those of us who are versed in the history of AI will notice that above I have actually sketched something similar to John McCarthy's "Advice Taker".)

Chapter 10

Comparison of Our Research and the World Research

10.1 On the Modern Research on the Models of Computation

In the chapter, *The Classical Church-Turing Thesis revisited*, various Turing-equivalent models of computation are discussed. In the chapter, *On Finite, Bounded from Above Computation*, "realistic" models of computation are being discussed. In the chapter, *The P ?= NP Problem, and a Research Program*, we prove a theorem on all algorithms.

Therefore, it makes sense to discuss the modern research concerning the models of computation. This section has been written according to the papers [Lee-Sangiovanni-Vincentelli, 1996] and [Lee-Sangiovanni-Vincentelli,1998].

10.1.1 Lee–Sangiovanni-Vincentelli on the Models of Computation

There usually are two kinds of models of computation: operational and denotational. Operational models give a description of the computational processes that result from the computation which has been defined. Denotational models give the meaning of the computation as the (usually partial) function that the defined computation corresponds to. Lee-Sangiovanni give a notational framework, a meta model so to speak, with which models of computation can be described and compared.

Intro

The L-S model consists of tagged values. A tag can be a moment of physical time, but it need not necessarily be one. The authors use tags to model time, precedence relations, synchronization points, and other key properties of a model

of computation. A value is a Turing-equivalent computation. So this model gives the semantics of a computing object (Turing machine, DFA, digital computer, whatever) as a succession of snapshots as in the reference [Davis-Sigal-Weyuker, 1994].

We have

Definition 10.1.1 ***Signals.***

Given a set of values V and a set of tags T, we define an event e to be a member $T \times V$, ie. and event has a tag and a value. We define a signal s to be a set of events.

So a signal actually will denote a certain computation. The English noun "signal" refers to the sending of messages – it seems to the author that the authors of the L-S paper have been thinking of a modern object-oriented software system such as Smalltalk – in Smalltalk each and every computation indeed consists of the sending of a message to an object. CLOS [....], Java [....], Lazarus [....] and many, many others are such modern object-oriented software systems.

The book [Russell-Norvig, 2010] "Artificial Intelligence, A Modern Approach" defines Artificial Intelligence as the enterprise of designing intelligent objects.

Definition 10.1.2 ***Set of signals, tuple of signals.***

We call the set of all signals in a system S, where $S = 2^{(T \times V)}$ If is often useful to form a tuple $\mathbf{s}$ of N signals, where N is a natural number.

Above in a tuple of signals, integers serve as names of individual signals. It turns out that it is a very handy way to name signals by calling them integers.

The empty signal (one with no events) will be denoted by λ, and the tuple of empty signals will be denoted by Λ.

Processes

In the most general form, a *process* is a subset of S^N for some N. Ie. the process contains N ordered signals for the integer N. A particular $\mathbf{s} \in S^N$ is said to satisfy the process if $\mathbf{s} \in P$.

An $\mathbf{s}$ that satisfies the process as called a *behavior* of the process. Therefore, a *process* is a set of possible *behaviors*.

Above, it is easy to visualize a software system whose set of possible behaviors is defined – among other things – the program code that constitutes the system. Within this constraint the input data, interrupts, etc etc. define the actual behavior under the above constraint.

1 – Composing processes:

Since a process is a set of behaviors, a composition of processes is the mathematical intersection of the behaviors of the processes. Some care has to be used when forming this intersection. Namely, each process to be composed must be defined as a subset of the same set of signals S^N which is called by some researchers its *sort*.

The parallel composition of noninteracting processes is simply the cross product of the sets of behaviors. Since there is no interaction between the processes, the behavior of the composite process consists of any behavior of one composite process together with any behavior of the other process, in the case of two processes P_1 and P_2.

More interesting systems have processes that interact. A *connection* $C \subset S^N$ is a particularly simple process where two (or more) of the signals in the N – tuple are constrained to be identical. This usually means that the output of one subprocess is connected to the input of another process (because the signals which refer to them are constrained to be identical.)

We define the projection of a process as follows. It is actually identical to the concept of a projection in other mathematics – a projection of a n-tuple of variables $< v_1, v_2, \ldots, v_n >$ is a m-tuple of variables such that a certain set of variables v_i are included in the projection, and the other – the rest – of the variables have been left out. Therefore: Let $I = (i_1, \ldots, i_m)$ be an ordered set of indices in the range $(1 \leq i \leq N)$, and define the *projection* as follows:

Definition 10.1.3 *The projection* $\pi_I(\mathbf{s})$ *of* $\mathbf{s} = (s_1, \ldots, s_N) \in S^N$ *onto* S^m *by* $\pi_I(\mathbf{s}) = (s_{i_1}, \ldots, s_{i_m})$.

□

Therefore, the ordered set of indexes defines the signals that are part of the projection and the order in which they appear in the resulting tuple. The projection can be generalized to processes.

Definition 10.1.4 *Given a process* $P \subseteq S^N$ *define the projection* $\pi_I(P)$ *to be the set* $\{\mathbf{s}'$ *such that there exists* $\mathbf{s} \in P$ *where* $\pi_I(\mathbf{s}) = \mathbf{s}'\}$.

□

Projection facilitates the composition of processes – the combination processes need not be augmented to involve irrelevant signals, ie. signals which connect subprocesses to other subprocesses with no external connection.

If the two signals in a connection are associated with the same process, then the connection is called (within the L-S discussion) a *self-loop*. (The meaning of a "self-loop" is quite different in the context of Petri nets!!) A self-loop in this sense will make feedback possible – and iteration. Of course complex topologies of interacting processes can carry out highly nontrivial (Turing-equivalent) computations.

2 – Inputs and Outputs

Many processes (not all of them) have have the notion of inputs, which are events of signals that are defined outside the process.

Definition 10.1.5 *An input as a constraint.*

An input to a process in S^N is an externally imposed constraint $A \subseteq S^N$ such that $A \cap P$ is the total set of acceptable behaviors.

□

Often we wish to talk about the behaviors of a process for a certain set of possible inputs, which set we denote by $B \subseteq Powerset(S^N)$. This is the case eg. in complexity considerations. That is, any input $A \in B$. In this case, we discuss the process and its possible inputs together, (P, B).

With our definitions, we can easily assert the presence of an individual event in a particular signal. However, most often, the inputs define an entire signal or a set of signals.

Definition 10.1.6 *Input signal.*

We call any signal that is entirely defined externally an input signal.

□

A process and its possible inputs (P, B) is said to be *closed* if firstly $B = \{S^N\}$, and the set of actual inputs is a set with only one element, $A = S^N$. Since the set of behaviors is $A \cap P = P$, there are no input constraints in a closed process. A process and its poossible inputs are *open* if they are not closed.

Now, let us consider the act of determining the values of a computation, depending on the inputs.

Now, consider an index set I for m input signals, and and index set O for output signals. We have

Definition 10.1.7 *A Functional process.*

A process P is functional with respect to $(I,\ O)$ iff for every $\boldsymbol{s} \in P$ and $\boldsymbol{s'} \in P$ where $\pi_I(\boldsymbol{s}) = \pi_I(\boldsymbol{s'})$, it follows from the preceding that $\pi_O(\boldsymbol{s}) = \pi_O(\boldsymbol{s'})$.

□

For a functional process, there exists a single-valued mapping $F : S^m \to S^n$ such that for all $\mathbf{s} \in P, \pi_O(\mathbf{s}) = F(\pi_I(\mathbf{s}))$. In other words, the process computes a function of its inputs.

3) – Determinacy:

Denote by $|X|$ the size (cardinality, the number of elements in) the set X. We have

Definition 10.1.8 *Determinate processes.*

A process is determinate iff for any input $A \in B$ it has either exactly one, or exactly zero behaviors, ie. $|A \cap B| = 1$ or $|A \cap B| = 0$. Otherwise, the process is nondeterminate.

As mentioned above, an output of a process may be connected to one of its inputs. Then, whether determinacy is preserved, depends on the tag system and more details concerning the process.

Tag Systems

Frequently, a natural interpretation of tags is that they mark physical time in a physical system. Here we neglect relativistic effects and consider that there is a global time which is the same everywhere.

On the contrary, for *specifying* systems the global ordering of events in a timed system may be overly restrictive. A specification should not be constrained by one particular physical implementation, and therefore often the tags *should not* mark time. They should, instead, reflect ordering induced by causality.

Moreover, in a *model* of a physical system tagging the events with the time at which they occur may seem natural. Then, when modeling a large concurrent

system, the model should probably reflect the inherent difficulty in maintaining a consistent view of time in a distributed system. This difficulty appears even in relatively small systems, such as VLSI chips, where distributing the clock distribution is challenging.

The central role of a tag system is to establish ordering among events, ie. the values of tags. An *ordering relation* on the set T is a reflexive, transitive, anti-symmetric relation on members of the set. (The mathematical definitions of those concepts are so well-known that I'm not repeating them here.)

A set T with with an ordering realtionship is called an *ordered set*. If the ordering relationship is partial (there exist $t, t' \in T$ such that neither $t < t'$ nor $t' < t$), then T is called a *partially ordered set* or *poset*.

A. Timed Models of Computation

A *timed model of computation* has a tag system T where T is *a totally ordered set*. That is, for any distinct two t and t' in T, either $t < t'$ or $t' < t$ – here the "or" is the exclusive OR. In timed systems, a tag also is called a *time stamp*.

There exist several distinct flavors of timed models.

1 – Metric time:

In a relatively elaborate tag system, T has a *metric*, which is a function $d : T \times T \rightarrow \Re$, where $\Re$ is the set of real numbers, that satisfies the following conditions:

$d(t, t') = d(t', t)$

$d(t, t') = 0 \Leftrightarrow t = t'$

$d(t, t') \geq 0$

and

$d(t, t') + d(t', t'') \geq d(t, t'')$

for all $t, t', t'' \in T$. Such systems are said to have *metric time*. In a typical example, T is the set of real numbers and $d(t, t') = |t - t'|$.

2 – Continuous time:

Let $T(s) \subseteq T$ denote the set if tags in a signal s. A *continuous-time system* is a metric timed system Q where T is a connected set and $T(s) = T$ for each signal s in any tuple $\mathbf{s}$ that satisfies the system.

A *connected set* T is one where there do not exist two nonempty disjoint open sets (a topological concept here!) O_1 and O_2 such that $T = O_1 \cap O_2$. The continuum $\Re$ is an example of a connected set.

3 – Discrete evets:

Given a process P, and a tuple of signals $\mathbf{s} \in P$ that satisfies the process, let $T(s)$ denote the set of tags appearing in any signal in the tuple $\mathbf{s}$ Clearly, $T((\mathbf{s}) \in T$, and the ordering relationship for members of T induces an ordering relationship for members of $T(\mathbf{s})$. A *discrete-event model of computation* has a timed tag system, and for all processes P and all $\mathbf{s} \in P$, $T(s)$ is *order isomorphic* to a subset of the integers N. An *order isomorphism* is an order-preserving bijection.

The authors note that "order isomorphism" nicely captures the notion of *"discrete"*. It among other points expresses the intuitively appealing concept that between any two finite time stamps there invariably will be a finite number of time stamps. It also gives every individual event a unique, disrete "name", ie. the integer that the event is mapped onto.

4 – Discrete-Event Simulators

Discrete-event simulators often have an agenda architecture similar to many Artificial Intelligence systems – events explicitly include time stamps; there is a queue (list, agenda) of events sorted by the time stamp; the event with the smallest time stamp is removed from the list and processed; this may result in the creation of more events, which usually are constrained to have later time tags than the smallest time event; the additional events are added to the agenda.

In some discrete-even simulators, such as VHDL simulators, tags contain both a time value and a "delta time", a suitable tag system for such a simulation would be $T = \omega \times \omega$, where ω is the smallest infinite ordinal number, ie. the set of nonnegative integers with the "natural" numerical order..

Unfortunately – $T = \omega \times \omega$ is not order isomorphic with ω or any subset of it.

5 – Synchronous and Discrete-Time Systems

Two events are *synchronous* iff they have the same tag. Two signals (ie. tuples of events) are synchronous if all events in one signal are synchronous with an event in the other signal and vice versa. A process is synchronous of every signal in any behaviour of the process is synchronous with every other signal in the behavior. (Then, all of the events in the process are "clocked" with the same "clock". A *discrete-time system* is a synchronous discrete-event system.

6 – Sequential Systems

A degenerate form of a timed tag system is a sequential system. The tagged signal model for a sequential process has a single signal s, and the tags $T(s)$ in the signal are totally ordered. For example, under the Von Neumann model of computation, the values $v \in V$ denote states of the system, and the only signal

denotes the sequece of states during the execution of the program. The reference [Davis-Sigal-Weuyker, 1994] discusses such a "succession of snapshots" model of modelling a computation.

B. Untimed Models of Computation

When tags are partially ordered rather than totally ordered, we say that the tag system is *untimed*. A variety of untimes models of computation have been proposed.

1 – Rendezvous of Sequential Processes

The CSP model of Hoare and the CCS model of Milner are key representatives of a family of models of computation that involve sequential processes that communicate with *rendezvous*. Similar models are realized, for example, in the languages Occam and Lotos. Intuitively, rendezvous means that sequential processes reach a particular point at which they must verify that another process has reached a corresponding point before proceeding. This is easy to capture in the tagged signal model.

2 – Kahn Process Networks

In a *Kahn process network* processes communicate via *channels*, which are (informally) one-way, unbounded, first-in-first-out (FIFO) queues with a single reader and a single writer. This notion also is rather easy to describe with the tags-values model.

3 – Dataflow: The dataflow model of computation

The *dataflow model of computation* is a special case of Kahn process networks. A *dataflow process* is a Kahn process which also is sequential, such that there are actors there which can process information and send/receive messages.

4 – Petri Nets

Petri nets also can be modeled in the framework. Petri nets are similar to dataflow, but the events within signals need not be ordered.

C. Heterogenous Systems

It is assumed above that when defining a system, the sets T and V include all possible tags and values. In some applications, it may be more convenient to partition these sets and to consider the partitions separately. For instance, V might be naturally divided into subsets $V_1, V_2, \ldots$ according to a standard notion of *data types*.

Similarly, T might be divided, for example, to separately model parts of a heterogenous systems that includes continuous-time, discrete-event, and dataflow

subbsystems.

Also, processes themselves can be divided into types, yielding a *process-level type system.*

The Role of Tags in Composition of Processes

The authors give two exmamples of the mathematical use of tags in composing processes. These two special cases are discrete-event systems and Kahn process networks.

A. Causality in Discrete-Event Systems

Causality is a key concept in discrete-event systems. Intuitively, it means that output events do not have time stamps less than the inputs that caused them. By studying causality rigorously, we can address a family of problems that arise in the design of discrete-event simulators. The discussion of the authors is mathematically rather deep; I have not referred to it here since I feel it would not any more be within the scope of this paper.

Monotonicity and Continuity in Kahn Process Nerworks

The definitions of monotonicity and causality are mathematically rather involved, I feel that they would not any more be within the scope of this paper.

Transformations Between Models

A transformation between two models could eg. describe the pratical implementation of a system from its description.

10.1.2 Lee–Sangiovanni-Vincentelli's Discussion and My Research

Lee–Sangiovanni's model consists of tagged values; values are snippets of Turing-equivalent computation; and the snippets are labeled with tags. A signal is a group of tag-value pairs. Therefore, their model mainly concerns:

- certain properties of the Turing-equivalent computation, and
- certain properties of communication between snippets of some Turing-equivalent computation

In the chapter, *The Classical Church-Turing Thesis Revisited*, the definition of computation by means of several Turing-equivalent models is being discussed. This research concerns a different research direction than L-S's discussion. My research is original and divergent with respect to the research topic, when compared to this Lee–Sangiovanni-Vincentelli's work.

In the chapter, *On Finite, Bounded from Above Computation* so to speak "realistic" models of computation are discussed. Again, my research is original and concerns a non-traditional research direction, when compared to L-S's work.

In the chapter, *The P =? NP Problem, and a Resarch Program*, we prove a theorem on all algorithms. Once more, my resarch is original and concerns a non-traditional research direction, when comparesd to this L-S's research.

10.1.3 Modern Models of Computation, and Smalltalk

The modern research on the theory of computation often concerns the object-oriented model. One example of this is the fact that the book [Russell-Norvig] defines Artificial Intelligence as the enterprise of devising intelligent objects.

As an example of modern computing models research, we in the following have a glance at the Smalltalk programming language – because it is totally object-oriented: everything in Smalltalk is an object, and all computation takes place by creating classes and their superclasses and subclasses, creating instances of classes, sending messages to classes and their instances (and executing the snippets of Turing-equivalent computation which are the operatonal semantics of certain messages.)

One of the creators of the Smalltalk-80 system, Alan Kay, in an interview noted that the main design goals of the Smalltalk-80 system were two: 1) late binding and 2) object orientedness.

Note that the following GNU Smalltalk program entirely consists of objects, and sending messages to objects.

The meaning of the exclamation mark (!) is similar to closing the last parenthesis and pressing ENTER in many iteractive LISP systems – it sends the preceding object for some routine to be evaluated. Actually, the GNU Smalltalk system has a READ-EVAL-PRINT loop that resembles (but is not identical to) many interactive LISP systems.

The text

```
Object subclass: #FinancialHistory
```

(roughly speaking) sends the message `subclass:` to the object named `Object`, with the argument

```
#FinancialHistory
```

thereby creating a subclass of the class `Object`.

In the same vein, a subclass `DeductibleHistory` of the class `FinancialHistory` is created.

```
initialBalance: amount
  | newHistory |
  newHistory := super initialBalance: amount.
  newHistory initializeDeductions.
  ^newHistory !
```

Figure 10.1.1 *Defining a method in GNU Smalltalk.*

Let us examine the text in the nearby Figure.

Here a new method is defined. The message that invokes this method is `initialBalance: 1000` in case that we are setting an initial balance of 1000 dollars (remember, the Smalltalk-80 book is an American book). The character string inside horizontal bars `| newHistory |` defines a local variable, whose name is – of course – `newHistory`.

The text

`newHistory initializeDeductions`

sends the message `initializeDeductions` to the object `newHistory`. The line

`^newHistory`

returns the value of the called method, the value of the variable `newHistory`.

The rest should be rather clear to the sophisticated reader.

```
"The FinancialHistory class from the Goldberg-Robson
1983 book on Smalltalk.

Converted to the GNU Smalltalk and debugged by Antti J. Ylikoski
01-21-2010 Helsinki, Finland."


Object subclass: #FinancialHistory
       instanceVariableNames: 'cashOnHand incomes expenditures'
       classVariableNames: ''
       poolDictionaries: ''
       category: 'First-exercise-in-Smalltalk'!


FinancialHistory subclass: #DeductibleHistory
```

```
        instanceVariableNames: 'deductibleExpenditures'
        classVariableNames: 'MinimumDeductions'
        poolDictionaries: ''
        category: 'Subclass-Demo'!


\^L

!FinancialHistory class methodsFor: 'instance creation'!

initialBalance: amount
  ^super new setInitialBalance: amount!

new
  ^super new setInitialBalance: 0! !

\^L


!FinancialHistory methodsFor: 'transaction recording'!

receive: amount from: source
  incomes at: source
          put: (self totalReceivedFrom: source) + amount.
  cashOnHand := cashOnHand + amount !

spend: amount for: reason
  expenditures at: reason
               put: (self totalSpentFor: reason) + amount.
  cashOnHand := cashOnHand - amount ! !

\^L

!FinancialHistory methodsFor: 'inquiries'!


cashOnHand
  ^cashOnHand!

totalReceivedFrom: source
  (incomes includesKey: source)
```

```
    ifTrue: [^incomes at: source]
    ifFalse: [^0] !

totalSpentFor: reason
  (expenditures includesKey: reason)
    ifTrue: [^expenditures at: reason]
    ifFalse: [^0] ! !

\^L

!FinancialHistory methodsFor: 'private'!

setInitialBalance: amount
  cashOnHand := amount.
  incomes := Dictionary new.
  expenditures := Dictionary new. ! !

\^L


!DeductibleHistory class methodsFor: 'instance creation'!

initialBalance: amount
  | newHistory |
  newHistory := super initialBalance: amount.
  newHistory initializeDeductions.
  ^newHistory !

new
  | newHistory |
  newHistory := super initialBalance: 0.
  newHistory initializeDeductions.
  ^newHistory ! !

\^L

!DeductibleHistory class methodsFor: 'class initialization'!

initialize
  MinimumDeductions := 2300 ! !
```

```
\^L


!DeductibleHistory methodsFor: 'transaction recording'!

spendDeductible: amount for: reason
  self spend: amount for: reason.
  deductibleExpenditures :=
    deductibleExpenditures + amount !

spend: amount for: reason deducting: deductibleAmount
  self spend: amount for: reason.
  deductibleExpenditures :=
    deductibleExpenditures + deductibleAmount ! !

\^L

!DeductibleHistory methodsFor: 'inquiries'!

isItemizable
  ^deductibleExpenditures >= MinimumDeductions !

totalDeductions
  ^deductibleExpenditures ! !


\^L

!DeductibleHistory methodsFor: 'private'!

initializeDeductions
  deductibleExpenditures := 0 ! !

\^L
```

Figure 10.1.2 *Modern object-oriented programming: a small GNU Smalltalk program.*

10.2 On the History and Status of the P ?= NP Question

This section has been written after the survey paper [Sipser, 1992].

The P ?= NP question arose from studies in mathematical logic and electronic tehcnology in the middle part of the 20th century. I do not wish to describe this process in detail to the sophisticated reader; I simply refer to [Sipser, 1992].

10.2.1 History

According to Sipser: The term P, for Polynomial Time, refers to the class of languages decidable on a Turing machine in time bounded above by a polynomial. The polynomial is ordinarily in |w|, where w is the input word to the Turing machine, and |w| is the length of w, the length of the input. The class NP, for Nondeterministic Polynomial Time, refers to the class of languages decidable in a polynomial time by a nondeterministic Turing machine. Another well-known characterization is that the class NP consists of those languages where membership is *verifiable* in polynomial time.

The question whether P is equal to NP is equivalent to whether an *NP-complete* problem X can be solved in polynomial time — then there is a polynomial-time transformation from all the NP problems to X, and P is equal to NP.

Brute Force Search

The "brute force" algorithm means, examining the entire unpruned search space. With many practical problems the size of the space grows so rapidly with the size of the problem that the brute force algorithm algorithm only works for toy problems.

John von Neumann – of the first computers fame – had written in 1953 a paper giving an algorithm avoiding brute force search for the assignment problem [von Neumann, 1953].

Edmonds [Edmonds, 1965] gave the first lucid account of the issue of brute force search appearing in the Western literature.

In the 1950's, researchers in the USSR were aware of the issue of brute force search, called *perebor* in the Russian language. Yablonski [Yablonski, 1959] described the issue for general problems.

P and NP

Several early papers [Sipser, 1992] proposed the notion of measuring the complexity of a problem as the number of steps required to solve it with an algorthm.

Cobham [Cobham, 1964], Edmonds [Edmonds, 1965], and Rabin [Rabin, 1966] suggested the class P as a reasonable approximation to the class of "realistically solvable" problems.

Cook [Cook, 1971] and Levin [Levin, 1973] first defined the class NP and proved the existence of complete problems. Karp [Karp, 1972] demonstrated the very wide extent of NP-completeness when he showed that many familiar problems have this property.

The Explicit P ?= NP Conjecture

Cook [Cook, 1971] first precisely formulated the P / NP conjecture.

10.2.2 The Status of the Problem

Diagonalization and Relativization

Superficially, it might seem possible to use diagonalization to prove that there exists at least one problem in NP which cannot be solved in a polynomial time.

We would devise a nondeterministic Turing machine NDTM1 which, while processing an arbitrary input, has an opportunity to run every single one of the deterministic polynomial time Turing machines and arrange to accept a different language. If NDTM1 runs the DTM D_n, then we can arrange for NDTM1 to accept a language which differs from the language accceptd by the DTM, ie. D_n, in one symbol position, as in the ordinary diagonalization argument.

In other words – the set of deterministic TM:s is denumerable, ie. it can be arranged in an ordered list, and then we would have a language acepted by NDTM1 which cannot be accepted by any one of the ordered list of the polynomial-time deterministic Turing machines. Then we would have one language in NP - P and it would be the case that P != NP.

This argument fails because the NDTM1, running in the time n^k, cannot in that time simulate a DTM D! whose running time is a larger polynomial. There is space for some creative research here!

A *relativized computation* means that the machine isprovided with a set, called an *oracle*, and the capability to decide membership in that set without computation cost. Baker, Gill, and Soloway [Baker, Gill, and Soloway 1975] demonstrated an oracle relative to which P is equal to NP, and another oracle relative to which they differ.

A step-by-step simulation of one machine by another, such as that used in a conventional diagonalization, still succeeds in the presence of an oracle – the

simulator can query the oracle whenever the simulated machine does. So any argument which relies only on step-by-step simulation to collapse or separate A and NP would therefore apply in the presence of any oracle. This would contradict the Baker, Gill, and Soloway theorem. So step-by-step simulation will not be sufficient to solve the problem. In particular, oracle considerations will not do.

Independence

Set theory constitutes the foundations of mathematics. Mathematical logic has been built on it. Topology is constructed on them. Algebra and real analysis, and complex analysis have been built on them, in turn.

The canonical axiom set that ordinarily is given in set theory is the ZFC axiom set, ie. the Zermelo-Fraenkel axioms plus the Axiom of Choice.

I feel that the precise, formal presentation of the Zermelo-Fraenkel axiom set, and related axiomatic set theories such as Tarski-Grothendieck set theory and von Neumann-Bernays-Gödel set theory would be beyond the scope of this discussion. But see [Wikipedia, Zermelo-Fraenkel] and [Wikipedia, von Neumann-Bernays-Gödel].

It has been proved that the Axiom of Choice is independent of the other ZFC axioms. Another landmark result is that the continuum hypothesis is independent from the ZFC axioms.

There has been there the speculation that the P ?= NP question may be unresolvable (i. e. independent) in a similar sense. In the direction of rather strong independence, a number of papers have attempted to offer a beginning, notably:

[Hartmanis, Hopcroft 1976]: Independence results in computer science; is giving an oracle under which the P versus NP question is independent of set theory.

The papers [Lipton, 1978]: Model theoretic aspects of complexity theory; [DeMillo, Lipton, 1979]: The consistency of P=NP and related problems within fragments of number theory; [Sazanov, 1980]: A logical approach to the problem "P=NP?"; [Buss, 1986]: Bounded Arithmetic; [Cook, Urquhart, 1988]: Functional interpretations of feasibly constructive arithmetic; [Ben-David, Halevi]: On the Inndependence of P versus NP; are establishing independence and related results within weak fragments of number theory or releated systems.

In [Sipser, 1992] the author wryly notes that "our current inability to resolve this question derives from insufficient brain power, rather than any lack of power in our formal systems".

Expressibility in Logic

A potential application of mathematical logic arises because it provides a different way of defining familiar complexity classes.

Fagin [Fagin, 1974] showed that the set of all properties expressible in existential second-order logic is precisely the complexity class NP. It is remarkable since it is a characterization of the class NP which does not invoke a model of computation such as a Turing machine.

Another way to state the same result is to say that NP equals the class of languages representing structures definable by Σ second-order sentences.

Fagin and Jones and Selman [JS74] showed that the class NEXPTIME equans the class of languages representing the collections of cardinalities of universes of such struectures.

Other notable results exist but so far they have not provided an answer to this venerable problem.

Restricted Systems

The difficulty of proving lower bounds on complexity stems from the richness of the class of algorithms. (NOTE that in the chapter "About the P?=NP Problem and a Research Problem" I actually succeeded in proving a theorem on a large set of algorithms [i. e. the ones which apply to the problem in hand].)

One approach that leads to partial progress is to restrict the class of algorithms considered. The restricted models in the literature fall into two categories, *natural models* and *handicapped models*.

With natural models, a model is selected that allows only for operations speccially tailored to the problem at hand. These include sorting models; polynomial evaluation models, various models towards data structures; etc etc.

With handicapped models, we seek to maintain the generality of the model but weaken it sufficiently so that an interesting lower bound is obtainable.

[Sipser, 1992] discusses restricted proof systems as a form of a natural model. Proof systems are related to the class NP. If all unsatisfiable formulas had polynomial length resolution proofs then it would easily follow that NP = coNP.

Circuit Complexity

Shannon [Shannon, 1949] proposed the size of Boolean circuits as a measure of the complexity of functions. Savage [Savage, 1972] demonstrated a close relationship between circuit size and execution time on a Turing machine.

In [Sipser, 1992] the author notes that circuits are an appealing model of computation for proving lower bounds. Most researchers consider circuit complexity to be the most viable direction for resolving the P ?= NP question.

A *circuit*, or a *a straight line program* over a basis (typically AND, OR and NOT) is a sequence of instructions, each producing a function by applying an operation from the basis to previously computed functions. If each function is used at most once, the circuit is called a *formula*.

By Savage's theorem, any problem in P has a polynomial size family of circuits. Therefore, to show that a problem in NP is outside of P it suffices to show that its circuit complexity is superpolynomial. Then, if the set NP-P is nonempty, we would have P != NP.

A simple counting argument shows that most Boolean functions on n variables require exponential circuit size [Shannon, 1949], [Muller, 1956].

Sipser notes in [Sipser, 1992] that it does not seem likely that the techniques used above to get this result will extend to significantly useful lower bounds.

In [Valiant 1990], Valiant suggests that the Boolean approach may not be the best approach. He suggests that we should examine the algebraic case first because it is more general. The "harder" problem may allow one to see more easily into the heart of the matter.

Bounded Depth Circuits

The next few sections treat handicapped variants of the circuit model. By placing various restrictions on structure, such as limiting the depth or restricting the basis, it has been possible to obtain several strong lower bounds. The survey paper [Sipser, 1992] discusses this point in more detail.

Monotone Circuit Size

A *monotone* circuit is one having AND and OR gates but no negations. A function is *monotone* if $X <= Y$ implies $f(X) <= f(Y)$ under the usual Boolean ordering. It is possible, and easy, to see that monotone circuits compute exactly the class of monotone functions. (I would attempt to prove this my means of induction on circuit size and formula size.)

A function is a *slice function* if for some k the value of $f(x)$ is 0 when the number of 1s in x is fewer than k, and the value is 1 when the number of 1s in x is more than k.

In [Sipser, 1992] the author shows that if one were able to prove a strong lower bound on the monotone circuit complexity of a slice function then this would show that P != NP.

Monotone Circuit Depth

The depth of a circuit is the length of the longest path from an input variable to the end result calculated by the circuit. Sipser gives in [Sipser, 1992] a number of individuals results concerning this definition,but they are not really highly significant with respect to our problem.

Active Research Directions

In [Sipser, 1992] three research dirctions are given which appear to be plausible means of separating complexity classes. These are the *approximation method*, which has been used succesfully in analyzing restricted circuit models; the *function composition method*, which may be useful to prove functions require large circuit depth; and an *infinite analog*, which has led to lower bounds in the bounded depth circuit model.

10.2.3 Sipser's Discussion and My Research

We can see that my research is quite novel and original. I even have proved a theorem on a large set of algorithms, which is related to a problem considered difficult by Sipser – Sipser mentions proving a theorem on all algorithms.

10.3 On the Modern Research on Symbolic Information Processing

10.3.1 Withgott-Kaplan on Unrestricted Speech

The papers [Withgott-Kaplan, Xerox PARC] and [Withgott-Kaplan, Xerox PARC 2] describe a research project whose thrust is both theoretical and practical in nature. The applied core technology involves machine learning and statistical theory as well as fundamental linguistic and phonetic theory. The study is intended to further the understanding of requirements for future speech recognition systems, and developing strategies for extracting significant information from noisy and/or large quantities of language data, ie. text in various formats and from various sources.

Advances in both the theoretical and practical sides include:

- phonetic regularities have been discovered:
- phonetic processing architectures and parameter tracking methods, and the CDT algorithm (the Clustered Decision Tree algorithm) have been developed;
- the above take into account contextual factors associated with phonetic variation –
- a distance metric has been developed for co-channel speech-interference (ie. by an another source disturbed speech recognition);
- progress has been furthered in understanding information extraction in unrestricted language data; and
- part-of-speech annotation has been demonstrated.

To the abovementioned points I have a comment: as it is well-known, good understanding and translation of human-produced text necessitates the understanding of the semantics of the text. Understanding context constitutes another important point.

As a case in point, consider the translation of the well-known text snippet

"Time flies like an arrow".

In order to properly translate it, both context and semantics are needed: the text in question is a proverb concerning time.

The (when the paper was being written) most recent accomplishments include:

- Developed the Clustered Decision Tree algorithm (an n-ary classification induction method) which makes use of machine learning and statistical techniques to organize data into *structures* representing the contextual factors associated with phonetic variation.

- Using the CDT methodology, developed a program to create a probabilistic pronounciation models in the SRI RULE format. (SRI denotes, the Stanford Research Institute.)
- Discovered a phonetic point concerning identically transcribed phones from different phoneme sources, reported at the 116th meeting of the Acous. Soc. Amer.
- Developed an LPC-based distance metric for recognition in the presence of competing speech.
- Applied Markov random fields to (1) extract speech formants and (2) "restore" formants
- Developed automatic annotation for text – achieved 95% correct annotation in a test text unrelated in form or content to the training document.

The objective of the researchers has been to develop theories and computational technologies resulting in information interpretation for multi-media documents and database retrieval.

In concert with work supported by Xerox and NSF (National Science Foundation) the researchers have planned to study the integration of explicit signal knowledge representation with information-theoretic approaches to recognition. The intention was to use language models (both precise and statistical) to constrain the recognition process. The work was intended to involve a speaker-independent phonetic classification system for continuous speech as well as text-based recognition studies.

10.3.2 Fateman on Symbolic and Algebraic Systems

This is based on Fateman's paper on symbolic mathematic systems [Fateman, Berkeley].

There is a crucial difference between "profane" numeric and "high-level" symbolic computation. Symbolic computation concerns calclulating with the exact, symbolic values of objects – I have written about this in the corresponding Chapter in my work.

Symbolic math brings us to the edge of reseach areas in several fiels of mathematics, and presents us with problems in representation and communication of knowlege which border on the hardest problems in computer artificial intelligence – natural language processing, representation of complex data, heuristic programming.

Furthermore, searching for the most efficient methods possible for tacklng fundamental arithmetic problems (eg. multiplying two polynomials) brings us into analysis of the commplexity of computation.

The stduy of symbolic algorithms eg. is of interest to researchers – there exists an algorithm for inndefinite symbolic integration, even though calculus students do not learn it!! Andm of course it is of interest to scientists who need a extensive and efficient symbolic math engine for their work in their fields.

10.3.3 Williams on the MUSHROOM Machine

The MUSHROOM architecture (Manchester University Software and Hardware Realisation of an Object Oriented Machine) [Williams, Manchester] is a computer achitecture intended for the efficient exeecution of symbolic processing environments such as the *Smalltalk-80*.

The project was developing a high-performance objct-oriented system as the paper was being written. This concerns a shared-memory multi-processor oriented towards the efficient excecution of symbolic processing environments (Smalltalk-80 in particular.) Languages such as LISP and Smalltalk-80 run much slower than conventional languages (C, FORTRAN) on the same machines.

Performance

Performance difficulties while running symbolic languages on conventional machines include (but are not limited to, hi!)

- **Memory structure and granularity** In LISP and Smalltalk-80, memory is allocated in very small objects (ie. symbols!) which are accessed via a pointer and offsets (ie. symbols are stored as records (à la Pascal) or structs (à la C)). This causes a disparity problem between the page sizes of conventional computers and the units of locality in symbolic processing environments – which leads to poor memory performance.
- **Run Time Type Checking** As is well known, in LISP and Smalltalk-80 the types of the objects in an expression must be checked in run time. These exists a efficient approach for this problem – tagging the data objects, which has been done in the MUSHROOM environmnents.
- **Dynamic Binding** A common feature of symbolic processing environments is the run-time binding of procedure calls. Often, such *late binding* invoves potentially large searches of the appropriate tables and often dominates procedure call overheads.
- **Garbage Collection** This may be a major overhead. Tagging cells as used/unused is a known remedy.

Architecture Outline

MUSHROOM is based on a high performance bus which allows the sharing of a single memory space which is distributed amongst the processors. The memory system is equipped with cache coherency hardware.

Virtual Memory

MUSHROOM provides an object oriented virtual memory. It is not a contiguous linear memory space, it is a collection of up to 2^{32} objects having different sizes.

Tagged Architecture

The MUSHROOM word is 40 bits in length; 32 bits are used for representing the data and adresses, whilst 8 bits are used to represent the 'tag' of the word. Tages are used to distinguish the types of words – for example checking that both operands are numbers in a `(* OPER1 OPER2)` expression.

Many garbage collection algorithms can make extensive use of these tagging facilities. For example, an ephemeral-object garbage collector might use the tags to identify different ages of objects.

Performance Considerations

Maximum execution performance is achieved by optimizing the equation:

```
EXETIM = #-of-instructions X #-of-cycles/inst X cycleTime/cycle
```

The equation is apparently simple, but it contains many inter-term crossdependecies.

So in the MUSHROOM, much effort was being put into compiler optimization. The CPU pipeline is visible to the compiler, which helps speed up the execution.

All frequently executed instructions execute in a single cycle, whilst less frequently executed operations may take longer.

The basic cycle time is kept low by careful analysis of critical paths in any potential implementations of the architecture.

10.3.4 Wah-Lowrie-Li on Computers for Symbolic Processing

This subsection is based on the [Wah-Lowrie-Li, 1989] paper on computers for symbolic processing. W-L-L define symbolic processing a little differently from my definition in this discussion. They note that recent advancements in applications

of computers suggest that the processing of symbols rather than numbers will be the basis for the next generation of computing.

Knowledge representation and knowledge processing are two important characteristics of the solutions to a symbolic processing problem.

Referring to my definitions about symbolic processing, I note that W-L-L present A. Newell's and H. Simon's general scientific hypothesis – the Physical Symbol System Hypothesis:

Thesis 10.3.1 *The Physical Symbol System Hypothesis.*

A physical symbol system has the necessary and sufficient means for general intelligent action.

□

W-L-L explicate that "*Although empirical in nature, the continuous accumulation of empirical evidence on the above hypothesis in the last 30 years has formed the basia of mmuch research on AI and cognitive science.*"

W-L-L divide the computation being done with a computer in several cathegories:

- Analog. This class of computations usually concerns continuous variables. Often they contain parameters from the environment (such as temperature, wind speed etc.)
- Numeric. The computation concerns magnitudes. May applications fall into this cathegory – functions on integers, functions on floatingpoint numbers, numerical analysis....
- Word. The parameters of functions are binary words which do not necessarily have a quantitative value – such as text processing.
- Relational. The primary unit of storage to be operated on are groups of words that have some relational interpretation. Here I note that mathematically, n-object relations and n-ary predicates are equivalent!!
- Meaning. W-L-L note that "very little research has been done on techniques for automated computation at the meaning level." I wrote a proposal for a Research Plan to study the mathematical theory of symbolic information processing!!! – The primay unit of evaluation is an interrelated series of relations (ie. predicates) that represent semantics and meaning.

W-L-L define **symbolic processing** as follows:

Thesis 10.3.2 *Wah-Lowrie-Li on symbolic processing.*

Symbolic processing is defined as those computations that are performed at the same or more abstract level than the word level.

□

Characteristics of Symbolic Processing Applications

- *Incomplete knowledge.* W-L-L write that many applications are *nondeterministic*, ie. it is not possible to predict the flow of computation in advance. W-L-L write that this is due to incomplete knowledge and incomplete understanding of the application. Here I must note that the definition of **nondeterminism** in Computer Science is really quite different – it does not denote, uncertainty because of incomplete knowledge!! This lack of complete knowledge may lead to *dynamic execution*, ie. new data structures, new functions, annd new bindings begin created during the actual solution of the problem.
- *Knowledge processing.* The programs operate on knowledge and meta-knowledge (sic!)
- *Symbolic primitives.* The programs operate on symbolic and symbol-valued functions (for a very good repertoire see the builtin functions in the Common LISP, or the builtin classes and messages in Smalltalk.)
- *Parallel and distributed processing.* Many symbolic applications exhibit a large potential for parallelism. Parallelism may be cathegorized into AND-parallelism and OR-parallellism. In AND-parallelism, a set of necessary and independent tasks are exexuted concurrently. In OR-parallelism, a *shortcut* may happen – if one argument of an OR is `T`, then the entire OR is T, or, True. This can be used to shorten the processing time in nondeterministic computations by evaluating alternatives at a decision point simultaneously.

Points Concerning Symbolic Processing

1. *Knowledge Representation* There are four criteria to evaluate a knowledge representation scheme: fleixibility, user friendliness, expressiveness, and the efficiency of processing.
   Although a number of knowledge represenatation schemes have been proposed, no one of them is clearly superior to the others for all applications.
   - *Features of Knowledge Representations*
     a) *Local versus distributed representations* In a local representation the failure of a hadrware unit may result in the loss of crucial data. If the representation is distributed and data have been distributed

over many hardware units, the failure of one unit may not result in catastrophic loss of data.
b) *Declarative versus procedural representations* A program written in a declarative representation consists of a set of domain-specific facts or statements and a tehcnique for deducing knowledge (both factual and procedural) from these statements.
The authors claim that "Declarative representations are user-oriented and emphasize correctness and user-friendliness of programs. They are *referentially transparent,* ie. the meaning of the whole can be can be derived solely from the meaning of the parts, independent of the historical behaviour of the whole."
I happen to disagree with this. I always experienced programming with LISP as easier and more natural than with eg. Prolog.

- *Classical Knowledge Representation Schemes:*
  a) Predicate logic: Logic is a super good tool for that which it was intended to be used – the representation of axioms and theorems and the derivation of new conclusions. But it has its shortcomings: the representation has been separated from processing; theorem provers may produce precise solutions but they may require too much processing time; logic may be a poor way to solve practical programs; logic is weak as a representation of certain kinds of knowledge [Winston, 1992].
  b) Production systems: Production systems apply collections of antecendent-consequent rules to solve problems. Many years earlier, is was mentioned in the Internet that the Frame Problem in practize destroys the performance of rule-based systems at the size of some 400 rules. However, the reference [Russell-Norvig, 1995] claims to contain the solutions for the two kinds of the Frame Problem that will occur in practical programs. Unfortunately, the expressive power of production systems is limited.
  c) Semantic networks: The basic inference mechanism here is "spreading activation". Its meaning would be rather evident. Several authors have shown that semantic networks can be extended so that they have the same expressive power than predicate logic.
  d) Frame representation: Frames employ a data structure for representing stereotypical situations. It consists of types (called frames) in terms of their attributed (called slots).
  The authors note that the basic idea appears promising and has appeared in various forms in many conventional languages (eg. the objct-oriented programming paradigm, such as CLOS, Smalltalk, C++, Java etc. etc.).

Representation	Level of Representation	Characterization
Logic	Variable	Local/Declarative
	Statement/Relation	Local/Declarative
	Program	Distributed/Declarative
Production Syst	Variable	Local/Declarative
	Statement/Relation	Local/Either
	Program	Distributed/Declarative
Semantic Netwks	Node	Local/Declarative
	arc/relation	Local/Declarative
	Network	Local/Procedural
	Program	Distributed/Declarative
Frames	Variable	Local/Declarative
	Statement	Local/Either
	Slots	Local/Either
	Frame	Local/Declarative
	Program	Distributed/Declarative
Procedural	Variable	Local/Either
	Statement	Local/Procedural
	Program	Local/Procedural
Connectionist	Connection Strength	Local/Distributed
	Propagation Technique	Local/Procedural
	Data and Knowledge	Distributed/Declarative

Figure 10.3.1 Examples of Knowledge-Representation Schemes

e) Procedural representations: The knowledge base is viewed as a collection of modules in a procedural language, such as LISP or C. This paradgm – as is well known – can be computationally efficient. But, conventional FORTRAN or Pascal have been found to be inadequate for effficient symbolic processing.
f) Connectionist representations: A connectionist representation is a form of distributed representation: concepts are represented over a number of modules or units.

2. *Knowledge Processing*
In the figure above there are shown the classical knowledge representation paradigms and their respective reasoning techniques.
It has been argued that humans use logic-like reasoning in the domain of rational knowledge and apply memory-based reasoning for perceptual actions. In the news:comp.ai newsgroup, one CD Jones in the 1990's

Representation	Typical Reasoning Technique
Logic	Resolution (w/ unification)
Production rules	Forward/Backward chaining
Semantic networks	Spreading activation
Frames	Procedural attachments
Procedural	Control flow
Connectionist	Propagation of excitation

Figure 10.3.2 Reasoning Techniques

strongly argued that all human activity consists of fetching recollections from a huge memory. I to a certain extent disagree with this: driving a car fits this pattern but proving a mathematical theorem would not seem to fit it.

- *Uncertain, Incomplete, and Inconsistent Knowledge Processing:*
  a) Uncertain knowledge
  Methods and theories of capturing uncertainty have been researched in recent years. Probability and Bayesian statistics are the most fundamental approaches to the problem. Approaches and techniques include fuzzy logic; confidence factors; Dempster and Schafer's theory of plausible inference; odds; and endorsements.
  b) Incomplete and inaccurate knowledge
  Here the authors note that "A key feature of symbolic computations in nodeterminism, which results from the fact that almost any intelligent activity is likely to be poorly understood." I disagree with this in two ways: 1) symbolic computation does by no means imply nondeterminism; and 2) nondeterminism is a totally different concept from a computation being "uncertain" because it is poorly understood!!!
  c) Inconsistent knowledge processing
  Traditional logic is monotonic; nonmonotonic logica have been devised, even formally sound ones.
  Default reasoning can be broken into two areas: exceptions to the rules, and autoepistemic logic.
  Autoepistemic logic (aka. circumscription and the closed-worl assumption) allows for conclusions to be reached about relations for which there´exist no facts in the database.
  The distinguishing feature of different rtechniques for dealing with inconsistent knowledge is the method for handling correction of the

knowledge base. These methods include explicit encoding, sstem applied inference, semantic networks, and truth maintenance.
McCarthy and Hayes have indicated how actions might be described using modal operators like "normally" and "consistent".
de Kleer created the ATMS, the Assumption-Based Truth Maintenance Systems. In the IBM YES/MVS (Yorktown Expert System / Multiple Virtual Storage), inconsistent deductions are automatically removed and new consequences are then computed in accordance with the new, changed facts.

- *Parallel Knowledge Processing:*
  The human brain operates as – so to speak – a large number of parallel running processors, being main controlled by the frontal lobes. These parallell running processors themselves utilize a high degree of parallelism. So it has been suggested that the way to go in symbolic processing is to build parallel symbol processors.
  Unfortunately, early experiments with symbolic multiprocessors architectures have shown that parallel symblic programs exhibit snall speedups. This point is an area for future creative research!!!

Architectural Concepts for Symbolic Processing

1. *Software Architectures*
   * *The design of software languages:* The objective is to provide software support and implementation for the knowledge representation and knowledge processing employed. The choice of a logic representation dominates the characteristics of the Prolog language; LISP was devised as a procedural language for symbolic processing (trivial though this sentence may be...) Conventional languages which were designed for numeric processing such as Fortran have not proven to provide adequate support for symboilc processing.
   - *Functional Programming Languages*
     In a functional language, the meaning of an expression is independent of the history of any computation performed prior to the evaluation of the expression (ie. referential transparency). Side effects to not occur in purely functional programs.
     Programming in functional languages facilitates specification or prototyping, prior to the development of efficient programs. See [Graham, 2010].
     Unfortunately, the property of referential transparency is lost in most practical Lisp implementations.

In Multilisp [Halstead, 2005], concurrency is introduced by means of the *pcall* and *future* constructs. Both utlize an implicit *fork-join*. *(pcall A B C)* will result in the concurrent evaluation of the expressions A, B and C. *(future X)* immediately returns a pseudo location for the value of the expression X, and creates a task to concurrently evaluate X.

- *Rule-Based Languages*
  a) Logic
  The motivation of logic programming is to separate knowledge from control. However, logic programming implementations often include extralogical primitives to improve their run-time efficiency and flexibility in specification. [and call them by the buzzword *procedural attachment*, the addition by the author.]
  Concurrency has been succesully added into the Prolog language in multiple ways.
  b) Production systems
  The other major form of rule-based language which promotes the separation of knowledge is based on the production-system representation. A production system consists of a set of data elements and a set of rules with a left-hand side (LHS) and the right-hand side (RHS). This is so extremely well-known that I decided not to write more about it.....
  One of the popular environments for the implementation of a production-system representation is the OPS5 system ("Offical Production System"). OPS5 employs a Rete match algorithm which keeps up a decision tree of matching conditions and updates them at every recognize-act cycle.
  Production systems provide a natural paradigm for *"if-then"* problems, such as expert systems. Unfortunately, algorithms with iterations and recursions are difficult to encode. The Frame Problem earlier was said to prevent the building of expert systems larger than about 400 rules – but the book [Russell-Norvig, 2005] purports to contain the solutions for the two kinds of a frame problem that may occur.
  c) On functional versus rule-based languages
  There have been attempts to combine functional and logic programming – but these have not attained the popularity of LISP or Prolog as such.
- *Object-oriented languages*
  Object-oriented languages are the logical and philosophical descendants of *data abstraction*, ie. hiding the procedures and data of the

Lisp	Prolog
Data typing	Condition matching
Function calls	Database functions
Recursion	Search (strategy,
List structures	backtrack)
Garbage collection	
Individual commands	Unification
(car, cdr, #' etc.)	
Parallelism	Parallelism
(future, etc.)	(modes, guards)
Application support	Application support

Figure 10.3.3 Features of Two Example Languages That Can Be Supported by Hardware

objects from the users of the objects (id est, who send messages to them); *abstract data types*, ie. associating a set of operations with a certain data type; and *inheritance*, ie. the objects belonging to classes who constitue a hierarchy, and objects being able to inherit methods, data and properties from other classes.
The *Actor* model by Hewitt at the MIT is a formalization of the ideas of object-oriented languages that also considers the added effect of parallelism.

- *Mapping Applications into Software*
  The art of building software applications is the domain of software engineering. Software engineering environments can be classified into four generations: discrete tools; toolboxes; lifecycle suppport and knowledge-based tools; and intelligent life-cycle support.
  Discrete tools were typical in the 1960's and the 1970's; the tool set of the Unix (a commercial trademark) operating system are a characteristic example.
  Toolboxes refer to integrated packages of tools, the most prevalent exampe being the Interlisp.
  Life-cycle support and knowledge-based tools have been under development in the 1980's.
  At the moment of the writing of the source paper [W-W-L] intelligent life-cycle support was a topic for future research.

2. *Hardware Achitectural Support for Symbolic Processing*
   - *Microlevel Hardware Features*

In the following, five architectural features are discussed: stacks, data tags, garbage collection, pattern matching, and unification.
a) Hardware stacks and fast stack-access techniques: Stack acrchitectures support function calls. This is particularly useful for LISP annd other functional languages.
In the Symbolics 3600 computer, the stack buffer contains all of the current function environment (ie. stack frame) plus as many of the older frames as fit.
Caches have been applied and found beneficial.
b) Tagged memory: The speed of symbolic computers is often linked to how effectively they emulate a tagged-memory architecture. Hardware for datatype tagging has been estimated to increase system performance by as much as an order of magnitude in LISP computers.
c) Garbage collection: Inital techniques were centered around reference counts. Generation scavenging is an important technique which reduces the overhead rate of garbage collection – different aged memory areas (ie. different generations) are handled differently.
More recent research has focused on parallel garbage-collection methods. They can be designed with very simple and fast components, without becoming a bottleneck of the system.
d) Pattern matching hardware support: Empirical results show that 90 percent of execution time in a production system can be spent in the matching phase.
e) Unification hardware: Unification is the fundamental technique in logic programming. With restpect to parallel unification hardware – unfortunately – the unification problem has been proved to be log-space-complete in the number of processors. However, it has been shown that lear linear speedup can be achieved in parallel unification. (How the preceding two sentences relate, only the authors of the original [W-W-L paper] will know. – The author.)
In sequential logic programs, up to 60% of execution time is being run in the unification algorithm.
f) VLSI and emerging technologies: The high degree of space and time complexity in AI and symbolic computation has necessitated the use of both parallel processing and VLSI technology. Such microelectronics is among the major objectives of the Japanese Fifth Generation Computing project; the MCC (the Microelectronics and Computers Technology Corporation); and the DARPA Strategic Computing projects.

- *Subsystem Level Architectures*

a) Database architectures.
Early studies emphasized the use of parallelism. Some later systems, such as the Connection Machine, are designed with massive parallelism for symbolic applications, and can be applied for a number of specialized database functions. — The bottleneck in database retrieval has been found to be disk I/O and not processor cycles.
b) Knowledge-base architectures.
The objectives and requirements of a knowledge-base computer are different from those of a classical database architecture.
* An evolving knowledge-base subsystem should have a mechanism for either rejection of, or truth maintenance for, the insertion of inconsistent data or rules.
* The contents are of a higher level of abstraction.
* The items to be stored are classes and instances of objects.
* Stored items are complex relations.
* Knowledge changes less frequently than classical data.
* The use of a knowledge-base system is mainy analysis, planning, whereas a classical database is used for operational purposes.
c) Hardware support for inference engines.
The structure of an inference engine is highly dependent on the knowledge representation and programming language employed.
Deduction and search have been the dominant paradigms for machine inference over the last 30 years (at the moment of writing the reference article.)
d) Artificial neural networks.
A neural network can be viewed as containing eight components.
* processing units;
* state of activation;
* output function;
* pattern of connectivity;
* propagation rule;
* activation rule;
* learning rule;
* and, an environment.
There have been numerous industrial efforts to build neural networks in hardware.

Complete Systems

Systems have been designed with the top-down; bottom-up; and middle-out strategies.

1. *Single-Processor Symbolic Computers*
   A (current, not-very-recent) trend is that LISP development environments such as KEE (Knowledge Engineering Environment, Intellicorp, commercial trade mark) and the ART (Automated Reasoning Tool, Inference Corp, commercial trade mark) can be executed efficiently on high-speed (GHz class) general-purpose workstations. This trend to a large extent killed the special-purpose LISP and Prolog worksations.
2. *Parallel Symbolic Processors*
   1) Communication and Synchronization
   Message passing, marker passing, and value passing are the three predominant communication methods in parallel symbolic processors.
   Synchronization refers to the control of concurrent accesses to shared items in a parallel processing system. It is important in a message-passing system because asynchronously arriving messages may result in contention for shared resources.
   2) Parallel Functional Programming Computers
   The majority of special-purpose parallel processors designed to support functional languages are oriented towards LISP. At the time of writing the reference article there were some ten of them there.
   3) Parallel Logic Architectures
   Two key features in evaluating logic programs are unification and (logic) database search. At the time of the writing the article there were some ten architectures intended for the hardware accelerated execution of logic programs.
   4) Parallel Systems for Production Systems
   At the time of the writing the article there were some half a dozen architectures for this end. The PSM computer is a large-grain machine which is specifically designed to support the OPS5 system and a parallel Rete match system.
   5) Parallel Object-Oriented Architectures
   At the time of the writing the article there were some three such systems. This topic has been more closely discussed in connection with my discussion concerning the MUSHROOM machine.
3. *Connectionist Processing*
   The connectionist implementations focus on the correspondence between nodes in a graph containing the knowledge and have been designed primarily for semantic networks. (Semantic networks have the disadvantage that the only can immediately represent binary relations. This difficulty can be overcome by *reifying* the n-ary relation in question. – The author.)
4. *Summary*
   Sequential, parallel and connectionist processing are three fundmental

approaches to processing techniques that are appropriate for various knowlegde representations. Interestingly, it is easy to prove that when a problem has an exponential complexity and the computers speed up and their memory systems grow exponentially, then the problem has with respect to the time a linear speedup.

Research Directions

- *Technologies*
  The candidated for the future include GaAs circuits; wafer-scale integrated circuits; analog-digital VLSI circuits; and optical computing technology.
- *Algorithms*
  Research for algorithms will have the greatest potential for speeding up the solutions.
- *Knowledge Representations*
  Most new knowledge represenations have emphasized declarative and distributed features in order to reduce programming complexity (the authors do not say, complexity in the human-written solution, or complexity of the digital computation.) The addition of temporal reasoning and nonmomotonicity would help.
- *Software Architecture*
  The problems of program verification and validation are important related topics. Interstingly, it can be proven that all nontrivial properties of programs are uncomputable.... But what significance this has to program verifiction, is a nontrivial question.
- *Hardware Architectures*
  Hardware architectures are often based on known design tehcniques such as parallel processing and pipelining.
- *System Design*
  The proper mix of control, data, and demand flow is one area for research that may impact systems for symbolic processing.

10.3.5 Comparison of My Research and the World Research

Chapter 11

Conclusions

The abstract theory of computation and the theory of Artificial Intelligence have been advanced by proving a number of theorems. One of the original subjects of this work was "On Nondeterministic Polynomial-Time Computation", and these results either concern the $P? = NP$ problem or they are side issues which have arisen from my work with the $P? = NP$ problem and theoretical computer science.

The main result of this work is the connection between Theoretical Computer Science and Artificial Intelligence, mainly with the theory of symbolic information processing systems. I have proven that there is a connection with these two branches of Computer Science, and this is a significant novel result in Computer Science.

11.1 On, On the Philosophy of Science

A number of individual points about the cross-fertilization of philosophy and computer science are discussed. This cross-fertilization is not altogether novel – but however rather new in this context.

11.2 On, On Metaknowledge, Metalanguage, etc

This is my personal note about the popular use of co-called buzzwords.

11.3 On, On Theorems and Theses

This is there to clarify some terminological points which arose in the course of this work.

11.4 On the Chapter on Definitions

All mathematical-formal works contain a set of formal definitions used in the work.

11.5 On, The Classical Church-Turing Thesis Revisited

We devise an entirely novel way to characterize that which can be computed, namely **discrete symbol manipulation** and its "realistic" variant **finite, bounded from above discrete symbol manipulation**. This is a significant novel result in TCS.

11.6 About, On Finite, Bounded from Above Computation

We prove the theorem that the venerable Halting Problem is unsolvable only for infinite computers. This is a significant novel theorem in TCS.

11.7 On, The $P? = NP$ Problem, and a Research Program

In this section, we discussed some properties of a function, which I call $F()$, which concerns the $P? = NP$ problem. The function $F()$ pertains to the problem of solving a nondeterministic polynomial-time problem deterministically in a polynomial time, with the AI paradigm of search space pruning. Based on the results, we finally outline a research program to study the $P? = NP$ problem, and a research program for theoretical computer science. This leads to one more corollary about the relationship of artificial intelligence and theoretical computer science.

11.8 On, Black Box Languages, and Symbolic Information

A novel vision unto the $P? = NP$ problem is presented, and a novel vision about symbolic information and the complexity of computation.

11.9 On, Notes on Kolmogorov Complexity

A novel research problem is created, the question of **the measure of symbolic information**.

11.10 About, On Symbolic Processing

We create the fundamentals of the formal theory of symbolic information processing systems, and among other things prove two theorems which show that symbolic information processing is genuinely, qualitatively "more powerful" than nonsymbolic bit crunching.

These results are sigificant novel results both in TCS and in AI.

11.11 On, Prolegomena to a Future Theory of Symbolic Processing

This is my task I'm leaving for the researcher world to come. I decided to cut the amount of work contained in this work at this point. (Cf. Pierre de Fermat: "I have a miraculous proof that I cannot fit in the margin of this book".... I do not have the huge investment of time to thoroughly research into this problem field.)

11.12 On, Comparison of Our Research and the World Research

This was suggested by Professor Ilkka Niemelä. This discussion is essential for all full-scale creative scientific works.

11.13 On, This chapter

1. If you feel bored, discontinue and skip to the next chapter.
2. Otherwise, see this chapter.
3. (PROGN (PRINT 'IMPOSSIBLE) (ABORT))

11.14 Epilogue to the Epilogue: Future Research

- Probabilism and Judea Pearl
- Hofstadter's musings
- geometric approach
- Razborov and his research, in general van Leeuwen et al.
- See CACM 09/2009 about the status of P vs. NP

Chapter 12

Acknowledgements

I thank the advisor of my PhD work, Professor Markku Syrjänen, for his excellent advice for my work.

This creation could not have been possible without him.

I also thank Professor Ilkka Niemelä, who read a manuscript of this work, for his highly helpful comments.

I very much thank Professor Pekka Orponen for this intelligent comments. This work would not have been scientifically waterproof without him reading and commenting the manuscript.

Chapter 13

References

[AI Magazine] The AI Magazine, The American Association for Artifical Intelligence, 445 Burgess Drive, Menlo Park, California, USA, ISSN 0738-4602, www.aimagazine.org

[Atallah, 1999] From: Mikhail Atallah: Algorithms and Theory of Computation Handbook, CRC Press LLC 1999, ISBN 0-8493-2649-4.

[Björklund-Lilius 2002] Dag Björklund, Johan Lilius: A Language for Multiple Models of Computation, CODES '02, ACM, USA.

[Burkholder] Leslie Burkholder: The Halting Problem, CDEC, Carnegie Mellon University, Pittsburgh, PA, USA.

[Cook a] Stephen Cook: The P Versus NP Problem.

[Cook b] Stephen Cook: The Importance of the P versus NP Question, JACM Vol. 50, No. 1, January 2003,pp. 27-29.

[Cormen-Leiserson-Rivest, 1994] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest: Introduction to Algorithms, The MIT Press 1994, ISBN 0-262-03141-8.

[Cormen-Leiserson-Rivest-Stein, 2003] Thomas H. Cormen, Charles E. Leiserson, Ronald R. Rivest, Clifford Stein: Introduction to Algorithms, 2nd edition, The MIT Press 2003, ISBN 0-262-03293-7.

[Davis-Sigal-Weyuker, 1994] Martin D. Davis, Ron Sigal, Elaine J. Weyuker: Computability, Complexity, and Languages, Second Edition, Morgan Kaufmann, ISBN 0-12-206382-1.

[DeMillo-Lipton 1980] Richard A. DeMillo and Richard J. Lipton: The Consistency of "P = NP" and Related Problems with Fragments of Number Theory, ACM 1980.

[Fateman] Richard J. Fateman: Symbolic and Algebraic Computer Programming Systems, University of California, Berkeley

[Google et al.] international search engines in the Internet WWW:
www.google.com
www.altavista.com
www.yahoo.com
www.lycos.com
www.excite.com
www.hotbot.com
www.infoseek.com
Ask Jeeves
Dogpile
Look Smart
Overture
Teoma
Find What

[Hawkins, 2004] Jeff Hawkins: On Intelligence (with Sandra Blakeslee), St. Martins Griffin 2004, New York, NY, USA. ISBN 978-0-8050-7853-4.

[Hunt, 20##] John Hunt: Smalltalk and Object Orientation, An Introduction, http://www.jaydeetechnology.co.uk

[Laplante] P. A. Laplante: The Heisenberg Uncertainty Principle and the Halting Problem, Fairleigh Dickinson University, Madison, NJ.

[Lee-Sangiovanni-Vincentelli, 1998] Edward A. Lee, Alberto Sangiovanni: A Framework for Comparing Models of Computation, in: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 17, No. 12, December 1998.

[Lee-Sangiovanni-Vincentelli, 1996] Edward A. Lee, Alberto Sangiovanni-Vincentelli: Comparing Models of Computation, ICCAD 1996, 1996 IEEE.

[Levin] Levin: On Universal Search.

[Li-Vitanyi] Ming Li, Paul M. B. Vitanyi: Two Decades of Applied Kolmogorov Complexity, IEEE 1988.

[van Leeuwen, 1990] Handbook of Theoretical Computer Science, Volume A, Algorithms and Complexity, edited by Jan van Leeuwen, ISBN 0 444 88075 5. Volume B, Formal Models and Semantics, ISBN 0 444 88074 7; Set of Vols A and B, ISBN 0 444 88075 5.

[Lewis-Papadimitriou, 1981] Harry R. Lewis, Christos H. Papadimitriou: Elements of the Theory of Computation, Prentice Hall 1981. ISBN 0-13-273417-6.

[Markov, 54] A. Markov: Theory of algorithms, Trudy Math. Inst. V. A. Steklova, 42, 1954. English translation: Israel Program for Scientific Translations, Jerusalem, 1961.

[Massey, 1995] In the newsgroup `news://comp.ai` in 1995 an entry by Robert L. Massey, of Colorado, USA; `http://www.csn.net/~pidmass` and `rmassey@entertain.com`. The exact text of this entry could not any more be found in the Internet archives, regrettably.

[Maté 1990] Attila Maté: Nondeterministic Polynomial-Time Computations and Models of Arithmetic, JACM Vol. 37 No.1, January 1990.

[McDonley 1986] Alan P.McDonley: Ada(TM) Symbolic Processing for Information Fusion, INCO, Inc. 1986, CO, USA.

[Minsky, 1967] Marvin Minsky: Computation, Finite and Infinite Machines, Prentice-Hall 1967.

[Papadimitriou, 1994]. Christos H. Papadimitriou: Computational Complexity. Addison-Wesley 1994. ISBN 0-201-53082-1.

[Post, 1936] E. Post: Finite combinatory processes: Formulation I, J. Symbolic Logic, 1, pp. 103-105, 1936.

[Razborov-Rudich, 1994] Alexander A. Razborov, Steven Rudich: Natural Proofs, STOC 1994, ACM, USA.

[Russell-Norvig, 2003] Stuart Russell, Peter Norvig: Artificial Intelligence, A Modern Approach. Prentice-Hall 2003, ISBN 0-13-080302-2.

[Sipser, 1992] Michael Sipser: The History and Status of the P versus NP Question, the 24th Annual ACM STOC, ACM 1992.

[Wah-Lowrie-Li, 1989] Benjamin W. Wah, Matthew B. Lowrie, Guo-Jie Li: Computers for Symbolic Processing, Proceedings of the IEEE, Vol. 77, No. 4, April 1989.

[Williams] Ifor Williams: The MUSHROOM machine – An Architecture for Symbolic Processing, Dept of Computer Science, The University, Manchester, the UK.

[Wilson] D. M. Wilson: The Halting Problem for Turing Machines, Department of Computing Science, University of Aberdeen, Scotland, the UK.

[Winston, 1992] Patrick Henry Winston: Artificial Intelligence, Addison-Wesley 1992, ISBN 0-201-53377-4.

[Withgott-Kaplan a] M. Margaret Withgott, Ronald M. Kaplan: Analysis and Symbolic Processing of Unrestricted Speech, Xerox Palo Alto Research Center.

[Withgott-Kaplan b] M. Margaret Withgott, Ronald M. Kaplan: Analysis and Symbolic Processing of Unrestricted Speech, Xerox Palo Alto Research Center. (Has the identical name to the a) paper.)

[Ylikoski, 2004] Antti Ylikoski: Some Simple but Interesting Results Concerning the P ?= NP Problem, ACM SIGACT NEWS Vol 35 Number 3, September 2004, ISSN 0163-5700

[Ylikoski, 2005] Antti Ylikoski: The Halting Problem on Finite and Infinite Computers, ACM SIGACT NEWS Vol 36 Number 1, March 2005, ISSN 0163-5700