Computable Foundations of Bounded Rationality*

Ying Fang Kao
School of Social Sciences/ASSRU,
Department of Economics,
University of Trento
Via Inama, 5, Trento, Italy

*This paper is a joint work with Prof. K. Vela Velupillai. Both authors are deeply grateful to our ASSRU colleagues in Trento, Professor Stefano Zambelli and V. Ragupaty, for innumerable conversations and discussions on the topics discussed here. Needless to say no one but ourselves are responsible for all remaining infelicities.
## Contents

1 **Bounded Rationality** 5  
   1.1 Simon’s bounded rationality .......................... 6  
       1.1.1 Satisficing and Optimizing ..................... 10  
       1.1.2 Procedural and Substantive Rationality .......... 11  
       1.1.3 Heuristics ........................................ 11  
   1.2 From Turing to Simon: decision making as problem solving .... 12  
       1.2.1 Man and Machine ................................. 13  

2 **Alan Turing and Solvability** 15  
   2.1 Solvable and Unsolvable Problems ..................... 16  
   2.2 Undecidable Decision Problems ........................ 18  
      2.2.1 Computability Theory .............................. 19  
      2.2.2 Hilbert’s Tenth Problem is Unsolvable .......... 19  

3 **Computational Complexity** 21  
   3.1 Time Complexity ....................................... 24  
      3.1.1 SAT .................................................. 25  
   3.2 Space Complexity ....................................... 25  
   3.3 Complexity of Combinatorial Games ................... 26  

4 **Computable Foundation of Bounded Rationality and Beyond** 27  
   4.1 Computable Economics .................................. 27  
   4.2 Human Problem Solving .................................. 29  
      4.2.1 Heuristics .......................................... 30  
      4.2.2 Near Decomposability: Divide and Conquer Heuristics .... 31  

5 **Concluding Remarks** 31  

A **(Formal) Language and Grammar** 33  

B **Asymptotic Complexity Notation** 33
Introduction

Bounded rationality is the central theme of behavioural economics, more precisely - Classical Behavioural Economics\(^1\). Bounded rationality is ubiquitously accepted as the formidable replacement for the otherwise infeasible notion of Olympian rationality, which Herbert Simon strongly disapproved. Contrary to popular understanding, Simon perceived bounded rationality as the more general notion when compared to Olympian rationality.

The aim of this chapter is to clarify, interpret and reformulate bounded rationality with only Simon’s definitions and emphasize that bounded rationality ought to be placed, studied and understood within a well structured context, which Simon had been advocating. Although bounded rationality has often been defined as an implication of limited cognitive computational capacity by Simon, models of bounded rationality are much complicated when compared to classical rationality in economics. However, it is important to note that classical notions (also Olympian or substantive) rationality are infeasible either in human or theoretical (recursion theoretic) senses. Bounded rationality, thus, is adequately equipped to describe the real life decision processes of concern in the decision sciences, broadly conceived, and encompass various notions of rationality used in other social sciences and psychology.

If we accept the premise that human thinking is a process of composing language out of a set of finite symbols, then the limitation of human thinking and consequently, humanly attainable procedural knowledge is bounded by computability. That is, there exist some things that human minds can not think, some problems that human minds can not solve, procedurally. Therefore, it is not enough to simply characterize or formulate the external environments, the constrains that a human mind is confronted with, and postulate theoretically that the best choices in such cases can always be found, completely disregarding the procedural aspects of solving such a problem. Given that there are difficulties in finding optimal solutions in the real world, often, it is theorized that people are expected to find acceptable approximately optimal solutions.

If one takes into account the characteristics of most situations of human decision making, then we can soon observe that knowing the alternatives and their characteristics in the choice set is not at all straightforward. Computational complexity is one of the measures that describes how difficult such a problem is. In a nutshell, the notion of bounded rationality is a natural outcome when one approaches this from computability theory and computational complexity theory.

In many possible applications of bounded rationality, Simon had devoted most of his time to human problem solving, which is a building block where one can find many analogies of human minds and computers. We are motivated to formulate the otherwise loosely and inadequately interpreted notion of bounded rationality -at least by variety of ‘orthodoxy’, but not so by Simon- in terms of computability theory, and strengthen it by invoking insights from computational complexity theory.

The main motivation of this chapter stems from a letter from Herbert Simon to Prof. Velupillai, who is one the pioneers of computable economics. The contents of this letter provide some clues on how Herbert Simon himself has travelled in the maze of

\(^1\)It is termed Classical to contrast it from Modern Behavioural Economics. The distinction and pioneers of these two traditions can be found in Kao and Velupillai (2012)
formalizing human decision making. Parts of the letter can be found in the following quotation:

“Dear Friends,

I want to share some first impressions on my reading of “Computable Economics”. (I confess that ‘reading’ did not include going through all the proofs.) I was delighted and impressed by the mileage you could make with Turing Computability in showing how nonsensical Arrow/Debreu formulation, and others like it, are as bases for notions of human rationality. Perhaps this will persuade some of the formalists, where empirical evidence has not persuaded them, of what kinds of thinking humans can and can’t do” – especially when dealing with the normative aspects of rationality.

... Turing computability is an outer boundary, as you show, any theory that requires more power than that surely is irrelevant to any useful definition of human rationality. A slightly stricter boundary is posed by computational complexity, especially in its common ‘worst case’ form. We cannot expect people (and/or computers) to find exact solution for large problems in computationally complex domains. This still leaves us far beyond what people and computers actually can do. The next boundary, but one for which we have a few results except some of Rabin’s work, is computational complexity for the ‘average case’, sometimes with an almost everywhere loophole. That begins to bring us closer to the realities of real-world and real-time computation. Finally we get to the empirical boundary, measured by laboratory experiments on humans and by observation, of the level of complexity that humans actually can handle, with and without their computers, and – perhaps more important – what they actually do to solve problems that lie beyond this strict boundary even though they are within some of the broader limits.

... I am sure that you will be able to interpret these very sketchy remarks, and I hope you will find reflected in them my pleasure in your book. While I am fighting on a somewhat different front, I find it greatly comforting that these outer ramparts of Turing computability are strongly manned, greatly cushioning the assault on the inner lines of empirical computability.

Once again, thank you very much for sending your fine book. Please continue to keep me in touch with your work. I’ll send along some recent reprints of mine.

Cordially,

Herbert A. Simon
Professor of Computer Science and Psychology”


Herbert Simon’s contributions to decision making and problem solving can be inferred by his life-time research in the cognitive science and, in particular, bounded
rationality and satisficing in multi-disciplinary ways in decision science. Both of them are very relevant to economics in what may be seemingly different, but in reality, interconnected fields. Among Simon’s numerous research contributions, Human Problem Solving theory incorporates the essential themes and the above stated notions. Although Simon never phrased his theories and concepts in terms of computability and computational complexity theories (with some caveats to be added here), explicitly, as he wrote in the letter, he devoted himself to constructing the more realistic boundary of human rationality, which in turn is strongly established and backed by rationality as encapsulated by Turing computability.

This chapter aims to reveal and elaborate the computability theoretic underpinnings of the concept of bounded rationality and then, further tie it to normative suggestions of modelling in economics. This chapter is structured along the lines and thought processes that one finds in the letter by Simon, i.e. Turing computability, computational complexity and empirical complexity. We argue that the appropriate underpinnings of bounded rationality are computability theory and computational complexity theory. While viewing bounded rationality in the context of problem solving in general, and human problem solving in particular, three aspects of problem solving become relevant: the existence of a method, the construction of a method, and the complexity of a method. We hope to use this interpretation of bounded rationality to demonstrate the impossibility and meaninglessness of optimization doctrines, which dominates economic modelling and analysis. We also argue that human rationality should be meaningfully formalized by other, appropriate and faithful formal tools. The construction of a meaningful outer boundary of rationality was established in Velupillai (2000). A more important message that this chapter hopes to convey is that the bounds to human rationality depends on the complexity of different problems that the problem solver encounters. The research program on Human Problem Solving initiated by Herbert Simon is along this direction.

1 Bounded Rationality

The term “bounded rationality” was first coined by Herbert Simon in the introduction to the fourth part of his collected works - Models of Man. He wrote:

The alternative approach employed in these papers is based what I shall call the principle of bounded rationality:

*The capacity of the human mind for formulating and solving complex problems is very small compared with the size of the problems whose solution is required for objectively rational behavior in the real world - or even for a reasonable approximation to such objective rationality.*

If the principle is correct, then the goal of classical economic theory - to predict the behavior of rational man without making an empirical investigation of his psychological properties - is unattainable. *Simon (1957), p.198-199, italics in the original*

Although the term appeared in 1957, the original idea of bounded rationality can be seen in both Simon (1955, 1956) and was earlier cultivated in Simon (1947) which is the revised from his Phd thesis. From the descriptions and postulations of bounded
rationality of Simon, it seems that the existence of a bound is simply due to the psychological nature of limitation in human decision making. In fact, this concept is a natural result of a context, in which both aspects of the environment and its decision maker are considered. Due to the natural characteristics of the environment and the decision maker, “satisficing” (first used in Simon (1956)) is the natural action to rational decision making, and furthermore, heuristics are the natural tools to make satisficing behaviors possible.

We will elaborate more on this in later subsections and sections. Especially, the vague idea of “environment” will be more precisely formulated into “problem space” the decision makers are confronted with. Furthermore, the “complex problems” appearing in the above quotation of Simon will be given a precise definition and context of computability theory and computational complexity theory. Later, we interpret the principle of bounded rationality with its inherent computable foundations, where it will be more obvious and straightforward to us that bounded rationality is neither irrationality (Simon, 1957, p.200) nor approximate optimality (Simon, 1972, p.170).

After Simon proposed his simple models of rational behaviour, successive models of bounded rationality has been reviewed in March (1978). In that paper, it is shown that many different directions and mild extension or reinterpretation of bounded rationality were developed, among which, there are also Herbert Simon’s directions. The paper is in the strict lines which Simon emphasised, especially, we clarify the difference between modelling human rationality between normative economics and other models which take real human decision making into account. Earlier, the battles between bounded rationality and normative rationality were active and exciting until bounded rationality was incorrectly and superficially incorporated into modern behavioural economics. Often, inconsistent behaviours with respect to normative economics are considered as some mistakes of human beings. Bounded rationality has been explained as the mistake or short coming of human beings for about 50 years. Modern behavioural economics is not the only and the first field which considers bounded rationality as a compromise from normative rationality. We can find the signs in the following quotation:

“Alternatively, one can recall all of the deviations from normative specifications as stupidity, errors that should be corrected; and undertake to transform the style of exciting humans into the styles anticipated by the theory. This has, for the most part, been the strategy of operations and management analysis for the past twenty years; and it has had its successes. But it has also had failures.” March (1978), pp.597

It is necessary to understand the genuine concept behind bounded rationality from Simon’s postulates and formalize it in such a way that the boundaries of human reasoning are intrinsic in the models and the kind of rationality required in normative economics is only one special case in this formalization.

1.1 Simon’s bounded rationality

The models of rational decision making suggested in Simon (1955, 1956) does not require an utility function of the alternatives. They provide the primitive idea of how a boundedly rational entity could be modelled.
In Simon (1955), a simplified value function $V(\cdot)$ which takes only two values $(1,0)$ was introduced. The binary values can be associated with “satisfactory and unsatisfactory”, “accept, reject”, etc. The domain of function $V$ is $S$, the set of all possible outcomes which is mapped to $A$, a set of all behavioural alternatives. This is to distinguish means from ends. The rational decision-process in defined as

1. Search for a set of possible outcomes $S' \subseteq S$ such that the pay-off function is satisfactory ($V(s) = 1), \forall s \in S'$

2. Search for behavioural alternatives $a \in A'$ whose possible outcomes are all in $S'$ through the mapping.

This process doesn’t guarantee the existence and uniqueness of a solution until the sequentiality of alternatives and dynamics of aspiration levels (a psychological concept) are incorporated into it.

In real life, alternatives are often examined sequentially and the first satisfactory alternative evaluated is the one that is selected. The difficulty of discovering a satisfactory choice depends on the cost of obtaining better information about the mapping of $A$ on $S$. Thus, if the aspiration level grows when the cost of search is low, and declines when the cost of search in high, then this will lead to near-uniqueness and existence of a solution in the long run.

In Simon (1955), the focus is on suggesting a realistic process for decision makers without going into the details of the mapping between $A$ and $S$. However, in Simon (1956)\(^2\), the focus is more on the other aspect - the environment. The problem setting in this paper provides a platform for constructing more elaborate models. Here, the organism is assumed to have a single and a fixed aspiration level - it needs only food. But, the food heaps are located in such a way that it has to walk in a maze where there are branches after each nodes. Each node is a possible location for food. This is combined with the constraint that its vision is limited and therefore it cannot see as far as it would like. However, if it sees a food heap in the range of its vision, it knows the way to reach the food. It has to eat the food before it dies of starvation and there is a maximal number of moves it can make, each time after eating, before its energy runs out. Thus, there are some parameters regarding the environment that the organism is facing and the “physical” constrains that the organism has:

- $p$: $0 < p < 1$, is the percentage of branch points, randomly distributed, at which food is found.
- $d$: is the average number of paths diverging from each branch point.
- $v$: is the number of moves ahead the organism can see.
- $H$: is the maximum number of moves the organism can make between meals without starving.

The first two parameters are regarding the environment (problem space) on how the targets are distributed and how big the problem space is. The last two parameters are regarding the organism on how far it can search and the capacity it can spend on

\(^2\)This paper is where satisficing was used for the first time by Simon.
searching. With these parameters, Simon was able to demonstrate the probability that the organism can not survive.

This setting can be applied to a much broader class of problems. The parameters do not have to be limited only by physical needs and constraints. Especially, the probability $p$ is not central in many realistic cases of decision making for Simon. For example, in chess, a game that was studied intensively by Simon, the goals (some particular patterns) that a player might seek are not randomly distributed in the problem space. Instead, the goals have to be achieved by making certain exchange of pieces on the board. Experts of chess tend to have better ability to obtain information from the mapping of actions and outcomes as compared to the novices.

Integrating the models in the two papers mentioned above, we can summarize the situation of rational decision making postulated by Simon as the following: There are always two aspects of decision making - the environment and the mechanism of the decision maker. The two aspects are highly interrelated. The characteristics of the environment or the problem space are

- The alternatives have discrete values
- The alternatives or the offer come in a sequence, while the order is not necessarily known
- The alternatives are combinatorial in nature in some cases

The characteristics of decision makers are

- Satisficing. (They are influenced by aspiration levels)
- Limited computational capacities (such as time and memory)
- Heuristic search
- Some knowledge or clue regarding the stopping rule for searching (starting from any node)
- Adapting aspiration levels
- Knowledge of what to choose and what not to choose

The metaphor of the problem space being a tree and this tree is explored is opposed to the orthodox idea that all the alternatives considered by the decision maker are displayed on the table, simultaneously, in front of the decision maker. This problem step-up of 'searching in a tree' is probably inspired from the means-end schema proposed in chapter IV of Simon (1947). Simon's intuitive observations of the environment - which should be formulated by the needs, drive and goals of decision maker - seems to underpin his metaphor that many problems in life are like searching in a maze, which are of a combinatorial nature theoretically. Therefore, human decision making, which is part of human thinking activity, can be associated in various ways to many rich areas of study such as computer science, graph theory, formal logic, etc.
Informally, a maze is a set of rooms connected by one way corridors. Certain rooms are designated goal rooms and one room is designated the start room. Thus, a maze is a directed graph with certain nodes or rooms distinguished. The maze is threadable if there is a path from the start room to some goal room.” Savitch (1970), p.187

It is important to note that the probabilities used to calculate the probability of failing to survive or finding a solution are trivial or meaningless in many real life problems.

From a still a third standpoint, the chess player’s difficulty in behaving rationally has nothing to do with uncertainty - whether of consequences or alternatives - but is a matter of complexity. For there is no risk or uncertainty, in the sense in which those terms are used in economics or statistical decision theory, in the game of chess. As von Neumann and Morgenstern observe, it is a game of perfect information. No probabilities of future events need enter the calculations, and no contingencies, in a statistical sense, arise.

From a game-theoretical standpoint, the presence of the opponent does not introduce contingencies. The opponent can always be counted on to do his worst. The point becomes clear if we replace the task of playing chess with the task of proving theorems. In the latter task, there is no opponent. Nor are there contingencies: the true and the derivable theorems reside eternally in Plato’s heaven. Rationality in theorem proving is a problem only because the maze of possible proof paths is vast and complex. Simon (1972), p.170

Although, it is debatable whether the opponent can be counted on to do the worst in games like Chess, the search space of Chess or Go are already certain, but wait to be discovered. We can view that the chess player will use the heuristics in his own mind to decide what the possible reacting moves his opponent will make.

Furthermore,

What we refer to as “uncertainty” in chess or theorem proving, therefore, is uncertainty introduced into a perfectly certain environment by inability - computational inability - to ascertain the structure of that environment. But the result of the uncertainty, whatever its source, is the same: approximation must replace exactness in reaching a decision. In particular, when the uncertainty takes the form of an unwieldy problem space to be explored, the problem-solving process must incorporate mechanisms for determining when the search or evaluation will stop and an alternative will be chosen. (ibid)

Many problems have relatively closed and a pre-defined problem spaces, despite being a massive tree. However, there are many other problems which are far more complex, for example, finding a particular quotation amongst the books in a library. Most of the time, the material that one is looking for is just in the vicinity, but it is hard to find a good heuristic to reach it.
1.1.1 Satisficing and Optimizing

Satisficing is another pillar on which Simon’s behavioural economics stands on. Here the decision maker does not look for an optimal choice, where as the ‘procedure’ of searching will lead him/her to choose a satisfactory outcome as and when one encounters it. This would mean that although there might be an outcome that could yield a higher level of satisfaction, the choice process stops once a ‘good enough’ alternative that matches the aspiration level is met. Simon also comments on the relation between satisficing and optimizing.

“A satisficing decision procedure can often be turned into a procedure for optimizing by introducing a rule for optimal amount of search, or, what amounts to the same thing, a rule for fixing the aspiration level optimally.... Although such a translation is formally possible, to carry it out in practice requires additional information and assumptions beyond those needed for satisficing” Simon (1972), p.170

For a decision maker, the act of optimization would require a before-hand knowledge of what is optimal or a method to confirm whether a given alternative is optimal or not and to have a method to choose this optimal outcome. As Simon remarks, this is both unrealistic and excessively demanding.

“The central problem of this paper has been to construct a simple mechanism of choice that would suffice for the behaviour of an organism confronted with multiple goals. Since the organism, like those of the real world, has neither the senses nor the wits to discover an “optimal” path - even assuming the concept of optimal to be clearly defined - we are concerned only with finding a choice mechanism that will lead it to pursue a “satisficing” path, a path that will permit satisfaction at some specified level of all of its needs.” Simon (1956), p.136

The more subtle differences between optimization and satisficing can be inferred from the following remark which clearly brings out the distinction between approximate solutions of optimization and satisficing.

“The terms satisficing and optimizing, which we have already introduced, are labels for two broad approaches to rational behaviour in situations where complexity and uncertainty make global rationality impossible. In these situations, optimization becomes approximate optimization - the description of the real-world situation is radically simplified until reduced to a degree of complication that the decision maker can handle. Satisficing approaches seek this simplification in a somewhat different direction, retaining more of detail of the real-world situation, but settling for a satisfactory, rather than an approximate-best, decision. One cannot predict in general which approach will lead to the better decisions as measured by their real-world consequences. In chess at least, good players have clearly found satisficing more useful than approximating-and-optimizing.” Simon (1972), p.170
Whether satisficing is an achievable goal depends on the “aspiration level” which, in turn, is a psychological concept. Roughly, aspiration levels should rise when the search cost of a satisfying solution is higher and decline when the search cost is lower. Furthermore, this dynamic adjustment of aspiration levels ensures the existence of the satisfying solution.

1.1.2 Procedural and Substantive Rationality

The insistence on methods or procedures involved in choosing and its centrality in the theory of decision making is the one that fundamentally distinguishes Simon’s approach from the other theories that invoke behavioural traits such as the modern behavioural economics. The link between a procedurally rational choice and computation is present from the very outset. The insistence here is on the complexity of this decision making in terms of the effort devoted in doing it.

“The search for computational efficiency is a search for procedural rationality, and computational mathematics is a normative theory of such rationality. In this normative theory, there is no point in prescribing a particular substantively rational solution in there exists no procedure for finding that solution with an acceptable amount of computing effort.” (Simon (1976), p.133)

1.1.3 Heuristics

The question then is to ask, what these procedures are and how these procedures present themselves in the context of decision making, how they are discovered and develop dynamically over time. Simon calls these methods as heuristics, which are nothing but methods of achieving some goals. They can be discovered by oneself, taught by teachers, forced by regulations.

“Most weak methods require larger or smaller amounts of search before problem solutions are found, but the search need not be blind trial-and-error-in fact, usually cannot be, for the search spaces are generally far too vast to allow unselective trial and error to be effective. Weak methods generally incorporate Polya’s idea of “heuristics”-rules of thumb that allow search generators to be highly selective, instead of searching the entire space.” Simon (1983), p.4570

It is quite obvious that heuristics are heavily associated with one’s experience, knowledge and cognitive capacity. In terms of decision making among finite alternatives whose mapping between actions and outcomes is combinatorial, heuristics are used in the following three aspects.

- What to generate
- How to evaluate
- When to stop
These heuristics are in fact algorithms and this is evident from the general approach that underpins Newell and Simon’s theory of Human Problem Solving. It is clear from the quote below that the model of decision making that they had in mind was a computational model, where heuristics are formal algorithms, though they did not state it explicitly.

“Physical symbol system must use heuristic search to solve problems because such systems have limited processing resources; in a finite number of steps, and over a finite interval of time, they can execute only a finite number of processes. Of course that is not a very strong limitation, for all universal Turing Machines suffer from it. We intend the limitation, however, in a stronger sense: we mean practically limited. We can conceive of systems that are not limited in a practical way, but are capable, for example, of searching in parallel the nodes of an exponentially expanding tree at a constant rate for each unit advance in depth. We will not be concerned here with such systems, but with systems whose computing resources are scarce relative to the complexity of the situation with which they are confronted. The restriction will not exclude any real symbol system, in computer or man, in the context of real tasks. The fact of limited resource allow us, for most purposes, to view a symbol system as though it were a serial, one-process-at a time device. If it can accomplish only a small amount of processing in any short term interval, then we might as well regard it as doing things one at a time. Thus “limited resource symbol system” and “serial symbol system” are practically synonymous. The problem of allocating a scarce resource from moment to moment can usually be treated, if the moment is short enough, as a problem of scheduling a serial machine.” Newell and Simon (1976), p.120

To further explore this connection, we need to explore the interconnections between the approaches of Turing and Herbert Simon, which is attempted in the next section.

1.2 From Turing to Simon: decision making as problem solving

The ground where bounded rationality and computability meet is comprehensively presented in Newell and Simon (1972), where symbolic systems can be adopted to understand of human thinking, especially in the activities of information processing. It should be noted that the word “computer” in this paper is used interchangeably to refer to human computer and digital computer.

“Moreover, since Homo sapiens shares some important psychological invariants with certain nonbiological systems - the computers - I shall want to make frequent reference to them also. One could even say that my account will cover the topic human and computer psychology.” Simon (1990), p.3

The crucial element then is to capture the link between the intuitive notion of thinking involved in deciding or problem solving and the structured machine that can
replicate this. For this there should also be a formal notion that encapsulates this intuitive notion. This was achieved by the seminal works of Alan Turing, through a formal definition of algorithm and a mechanism to encapsulate intuitive notion of effective computation in the form of Turing machines. This in turn forms the intellectual backdrop in which Simon and Newell developed their theory.

“The machine’s process are mosaics of very simple standard parts, but the designs can be of great complexity, and it is not obvious where the limit is to the patterns of thought they could imitate.” Turing et al. (1952), p.500, by Newman

Since computability theory is built on the inspection of the processes used by human beings, digital computers which mimic human computers are built to help accomplish many human tasks and this can be further applied to understand human thinking.

“To my mind this time factor is the one question which will involve all the real technical difficulty. If one didn’t know already that thee things can be done by brains within a reasonable time one might think it hopeless to try with a machine. The fact that a brain can do it seems to suggest that the difficulties may not really be so bad as they now seem.” Turing et al. (1952), p.503, by Turing

1.2.1 Man and Machine

Thinking cannot be possible without a language, despite that the exact relationship between language and thinking is still controversial and vague. Languages are sets of strings which are composed by symbols from finite sets. For example, English is constructed by 26 letters (symbols), a space and few punctuations. Simon was aware of this and the possibility of linking that to the idea of problem solving in economic sphere, through a model of computation proposed by Turing.

“To build a successful scientific theory, we must have a language that can express what we know. For a long time, cognitive psychology lacked a clear and operational language. Advances in formal logic brought about by Giuseppe Peano, Gottlob Frege, and Alfred North Whitehead and Bertrand Russell around the turn of the century provided it.

The relation of formal logic to psychology is often misunderstood. Both logicians and psychologists agree nowadays that logic is not to be confused with human thinking....

How , then, could formal logic help start psychology off in a new direction? ... Symbols are the stuff of thought, but symbols are patterns of matter. The mind/body problem arises because of the apparent radical incongruity of “ideas” - the material of thought - with the tangible biological substances of the brain. Formal logic, treating symbols as material patterns (for example, patterns of ink on paper) showed that ideas, at least some ideas, can be represented by symbols, and that these symbols can be altered in meaningful ways by precisely defined processes....
Exploiting this new idea in psychology requires enlarging symbol manipulation to embrace much more than deductive logic... This crucial generalization began to emerge at about the time of World War II, though it took the appearance of the modern computer to perfect it.

Parallel to the growth of logic, economics, in close alliance with statistical decision theory, constructed new formal theories of “economic man’s” decision making. Although economic man was patently too rational to fit the human form, the concept nudged economics toward explicit concern with reasoning about action. But the economist’s concern only for reasoning that was logical, deductive, and correct somewhat delayed recognition of the common interests of economics and psychology.

In striving to handle symbols rigorously and objectively - as objects - logician gradually became more explicit about their manipulation. When, in 1936, Alan Turing, an English logician, defined the processor now known as a Turing Machine, he completed his drive toward formalization by showing how to manipulate symbols by machine.” Simon (1991), p.192-193, italics in the originals.

Once this view of relating thinking to the process of computation is adopted, then the other pieces of the theory fall in place. The notions of what is achievable, procedurally (algorithmically) solvable, the level of complexity that one can handle, all become clear.

“Like a modern digital computer’s, Man’s equipment for thinking is basically serial in organization. That is to say, one step in thought follows another, and solving a problem requires the execution of a large number of steps in sequence. The speed of his elementary processes, especially arithmetic processes, is much slower, of course, than those of a computer, but there is much reason to think that the basic repertoire of processes in the two systems is quite similar. Man and computer can both recognize symbols (patterns), store symbols, copy symbols, compare symbols for identity, and output symbols. These processes seem to be the fundamental components of thinking as they are of computation.

For most problems that Man encounters in the real world, no procedure that he can carry out with his information processing equipment will enable him to discover the optimal solution, even when the notion of ‘optimum’ is well defined. There is no logical reason why this need be so; it is simply a rather obvious empirical fact about the world we live in - a fact about the relation between the enormous complexity of that world and the modest information-processing capabilities with which Man is endowed. One reason why computers have been so important to Man is that they enlarge a little bit the realm within which his computational powers can match the complexity of the problems. But as the example of the traveling-salesman problem shows, even with the help of the computer, Man soon finds himself outside the area of computable substantive rationality.” Simon (1976), p. 135
It is legitimate to add one question to Simon’s argument quoted above. Even if there is a super human who lives for eternity and owns unlimited or the whole universe of resource and space, **Will this super human have a general procedure to solve the optimization problems when the optimum is well defined to him/her?** If the answer to this question is negative, then the infeasibility of solving optimization problem by human beings with a procedure is a self-evident result from the previous question. In principle, there can be no time and space limitation of Turing machines. Thus the rationality of Turing machine can be considered the upper bound of rationality. Even then, there are unsolvable problems by Turing machines. We would like to show the answer to the above question is negative by using the computation universality of Turing machine in later sections.

“The human mind is programmable: it can acquire an enormous variety of different skills, behaviour patterns, problem solving repertoires, and perceptual habits.....There seems to be no escape. If economics is to deal with uncertainty, it will have to understand how human beings in fact behave in the face of uncertainty, and by what limits of information and computability they are bound.” Simon (1976), p. 144

It is not difficult to observe that Simon’s many approaches for performing the models of satisficing in bounded rationality are deeply influenced by Turing’s invention of the machine (possibly abstract) operating on symbols to understand the behaviour of human thinking. When Turing wrote “computer”, he really meant “human computer”; the first digital computer was not even born by then!

### 2 Alan Turing and Solvability

In order to understand the issue of solvability in this context comprehensively, we would like to trace back this to one of Hilbert’s problems - regarding decision problem, which was proposed in 1900.

> “Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: To devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers.” Hilbert,1900, Paris, Second International Congress of Mathematicians, italics added, Devlin (1988), p.141

This problem, in turn, was not answered without a formal definition of “a process according to which it can be determined by a finite number of operations” until 1936-1937 when the definitions of such a finite procedure were defined.

Turing’s question about problem solving is linked to the decision problem, which needs to be addressed before one starts to solve a problem. Is there a procedure to decide whether a given problem(puzzle) is solvable or not? This decision problem concerns all those problems which can be transformed into substitution puzzles. The answer to this was proved to be negative by Turing.\(^3\) The negative solution of this de-

---

\(^3\)The formal definition of substitution puzzle - algorithm - was defined during 1936-7 as mentioned in Turing (1954) by several people - Turing, Church, Post and others - at about the same time.
cision problem indicates that we need to develop specific procedure in order to decide for specific problems.

2.1 Solvable and Unsolvable Problems

In Turing (1954) he asks a simple question “Is there a systematic procedure by which one can tell whether a puzzle is solvable?” and aims to prove that the answer for it is negative.

In computability theory, this could be translated as “Is there a machine by which one can tell whether a set of languages is recognizable?”

In order to answer this question, Turing started with a statement:

“Given any puzzle we can find a corresponding substitution puzzle which is equivalent to it in the sense that given a solution of the one we can easily use it to find a solution of the other. If the original puzzle is concerned with rows of pieces of a finite number of different kinds, then the substitution may be applied as an alternative set of rules to the pieces of the original puzzle. A transformation can be carried out by the rules of the original puzzle if and only if it can be carried out by the substitutions and leads to a final position from which all marker symbols have disappeared.” Turing (1954) p.15, italics in the original.

He further wrote “In effect there is no opposition to the view that every puzzle is equivalent to a substitution puzzle.”

Let us have a look at what a substitution puzzle is:

“Another type of puzzle which we shall find very important is the ‘substitution puzzle.’ In such a puzzle one is supposed to be supplied with a finite number of different kinds of counters, perhaps just black (B) and white (W). Each kind is in unlimited supply. Initially a number of counters are arranged in a row and one is asked to transform the initial pattern into another pattern by substitution. A finite list of substitution rules that are allowed is given. Thus, for instance, one might be allowed the substitutions

\[(i) \text{WBW} \to B\]
\[(ii) \text{BW} \to \text{WBBW}\]

" Turing (1954) p.13

The production rules that are introduced here follow type 0 grammar (see definition in Appendix 20), though, the time at which Turing proposed it was before the Chomsky hierarchy was defined (Chomsky, 1956, 1959).

Type 0 grammars are essentially Turing machines; type 3 grammars, finite automata. Type 1 and 2 grammars can be interpreted as systems of phase structure description. Chomsky (1959), p.143

It is clear from that definition of the grammar that Type \(i\) grammar is the superset of Type \(i+1\) grammar; that is to say, Turing Machines are the most general kinds of symbol operators and they are capable of recognizing the languages generated from all types of grammar.
The task above can be carried out by applying the rules for substitution as the following:

\[ WBW \rightarrow WWBBW \rightarrow WWBWBBW \rightarrow WBBBW \]

In order to prove that there is no systematic procedure to decide, when given any puzzle, whether the puzzle is solvable or not, Turing claimed that there are two equivalences:

- The equivalence between the puzzles and the substitution puzzles.
- The equivalence between the substitution puzzles and the systematic procedures.

A substitution puzzle consists of its set of rules (substitution pairs) and starting position. And a systematic procedure is a puzzle in which there is never more than one possible move in any of the positions which arise and in which some significance is attached to the final result. The puzzle described for systematic procedure is also called "the puzzle with unambiguous moves."

Then this kind of “puzzle with unambiguous moves” is further applied into the argument to prove the negative answer to the existence of systematic procedure for deciding the solvability of a puzzle. Before he started to prove, he transformed the sentences of the set of rules into the same form of the starting position, i.e., a string of symbols. Therefore, there are many different strings that can actually represent the same set of rules of a substitution puzzle. We can represent a puzzle as \( P(R, S) \), where \( R \) is the row of symbols describing rules and \( S \) is the starting position. It is reasonable that the puzzle \( P(R, R) \) is also considerable, that is the starting position is the same string of the row of symbols describing rules. Provided that the puzzles being discussed are those with unambiguous moves, then these puzzles will be substituted with their rules, step by step, until no rules can be carried out, and report a certain result. In Turing’s example, it is either \( W \) or \( B \). That is to say, \( P(R, R) \) has as its final result either \( W \) or \( B \), it cannot have both possibilities.

- Class I is to consist of sets \( R \) of rules, which represent puzzles with unambiguous moves, and such \( P(R, R) \) comes out with the end result \( W \).

- Class II is to include all other cases, i.e., either \( P(R, R) \) does not come out, or comes out with the end result \( B \), or else \( R \) does not represent a puzzle with unambiguous moves. We may also, if we wish, include in this class sequences of symbols such as BBBBB which do not represent a set of rules at all\(^4\).

It is worthwhile in going through the actual proof in some detail to develop our own arguments. It is supposed that there exists a systematic procedure for deciding whether puzzles comes out or not. At the same time, this systematic procedure can be transformed into a substitution puzzle whose rule is \( K \). Naturally, \( K \) has unambiguous moves and it always comes out with whatever \( R \). Particularly it will come out with, say \( B \), when \( R \) belongs to class I, and \( W \) when it belongs to class II. Then, when we

\(^4\)Maybe the class should include the situation such as WW, which is a result of the puzzle when no rule can be further applied to it
look at the puzzle \( P(K, K) \) to be investigated, we will find inconsistent properties. That is, we should be able to classify that \( P(K, K) \) belongs to class I or II. But according to the substitution puzzle \( K \), it has the potential to come out both possibilities, as a result, we could not classify it into any two of these classes. That is contradictory.

This demonstration towards showing that there is no general algorithm for deciding whether a puzzle is solvable or not suggests that we need to seek for separate algorithms in order to decide whether a kind of problem is solvable or not, given the initial puzzle and the desired puzzle.

From this point, it should be possible to move on to introduce the formal definition of substitution puzzle which sheds light on Hilbert’s 10th problem.

### 2.2 Undecidable Decision Problems

One notable thing is that when Turing developed his intuitive idea of computation, he always took into account the natural limitations of human beings. A Turing machine can be seen as the mathematically formalized human computers.

In Turing (1954), he had stressed the importance of having formal definitions or representation of “a problem” and a “systematic procedure”.

> “It is possible to invent a single machine which can be used to compute any computable sequence. If this machine \( U \) is supplied with a tape on the beginning of which is written the S.D of some computing machine \( M \), then \( U \) will compute the same sequence as \( M \).” Turing (1936)

The “S.D.” appeared in the passage of Turing denotes the standard description which is a transformation of a sequence of quintuple instructions\(^5\), e.g. \( q_i S_i S_j L q_m \ldots \) (meaning the current state \( q_i \), the scanned symbol \( S_i \), printed symbol \( S_j \), move to the left, successive state \( q_m \)) by encoding the sequence into a sequence of letters. These alphabets can be further encoded into integers. The aspiration of inventing a universal Turing machine led to the project of building the first-general-purpose electronic machine - ENIAC, in which John Mauchly and John von Neumann were involved. It is useful to present the statement of the decision problem posed by Hilbert at this juncture.

**Definition 1** Decision problem: as to the existence of an algorithm for deciding the truth or falsity of a whole class of statements … A positive solution to a decision problem consists of giving an algorithm for solving it; a negative solution consists of showing that no algorithm for solving the problem exists, or, as we shall say, that the problem is unsolvable.

Turing had shown mathematically that there exists unsolvable decision problems by using both Cantor’s diagonalization method and letting the supposedly existing machine encounter itself and leading to a contradiction involving the halting problem. He was able to prove that the halting problem and the decision problem for whether any number that encodes a standard description (the substitution puzzle mentioned in the previous subsection) will produce infinite loops of symbols (satisfactory or circle-free) or not are both unsolvable.

\(^5\)The commonly used quadruple machines were formulated in Post (1947).
2.2.1 Computability Theory

"Concurrently with Turing’s work appeared the work of the logicians Emil Post and (independently) Alonzo Church. Starting from independent notions of logistic systems (Post productions and recursive function, respectively), they arrived at analogous results on undecidability and universality - results that were soon shown to imply that all three systems were equivalent" Newell and Simon (1976), p.117

Church defined algorithm with $\lambda$-calculus, and Turing defined it in terms of Turing Machines. They are proved to be equivalent definitions, thus, their definitions equating the intuitive notion of algorithm and its definition implies the so-called Church-Turing Thesis. If this thesis is true, then the halting problem for Turing machines is unsolvable.

Church in his paper Church (1938) mentioned that the intuitive notion of a effective procedure can be formalized into three different ways: Turing Machines (Turing, 1936), $\lambda$-definability (Church, 1936) and the most general recursive function (Kleene, 1936a). The equivalence of the three notions had been prove in Kleene (1936b) and Turing (1937). The equivalence of recursiveness and computability enables us to apply the definition of recursive function to prove more classes of computable functions. (see Davis, 1982, Ch.3)

In the following, we can see several definitions and theorems which are relevant for proceeding to comprehend the proof strategy of Hilbert’s tenth problem.

**Theorem 1** Let $S = x\mid P(x)$, and let $S \neq \phi$. Then the following statements are all equivalent:
1. $P(x)$ is $A$-semicomputable.
2. $S$ is the range of an $A$-primitive recursive function.
3. $S$ is the range of an $A$-recursive function.
4. $S$ is the range of an $A$-partial recursive function.

**Definition 2** A set $S$ is called $A$-recursively enumerable either if $S = \phi$ or if the equivalent conditions (1) to (4) of Theorem 1 hold.

**Theorem 2** $S$ is $A$-recursive if and only if $S$ and $\overline{S}$ are both $A$-recursively enumerable.

**Definition 3** A combinatorial system $\Gamma$ consists of a single non-empty word called the axiom of $\Gamma$ and a finite set of productions called the productions of $\Gamma$. A word is a finite sequence of symbols

2.2.2 Hilbert’s Tenth Problem is Unsolvable

**Definition 4** A set $S$ of ordered $n$-tuples of positive integers is called Diophantine if there is a polynomial $P(x_1, \ldots, x_n, y_1, \ldots, y_m)$ where $m \geq 0$, with integer coefficients such that a given $n$-tuple $< x_1, \ldots, x_n >$ belongs to $S$ if and only if there exist positive integers $y_1, \ldots, y_m$ for which

$$P(x_1, \ldots, x_n, y_1, \ldots, y_m) = 0$$

In order to answer Hilbert’s tenth problem, one needs a definition of “a process according to which it can be determined by a finite number of operations..”, that is an algorithm which was not formally defined until late 1930s. Before it was precisely defined, mathematicians had merely an intuition of the notion of an algorithm.
The “negative solution” of Hilbert’s tenth problem relies heavily on computability theory. The strong connection between computability theory and Diophantine decision problems was constructed by Martin Davis who had struggled many decades together with Hilary Putnam and Julia Robinson. Almost all the effort for the argument is spent on proving a conjecture proposed by Davis. If the conjecture is proved to be true, then the negative solution of Hilbert’s tenth problem can be deduced straightforwardly from the conjecture through a theorem in computability theory. Martin Davis’ conjecture is

Hypothesis 1  Davis’s hypothesis: every semidecidable set is Diophantine

Let us look at the following definition:

Definition 5  We say that the decision problem for a combinatorial system $\Gamma$ is recursively solvable or unsolvable, according as $T_{\Gamma}$ is or is not a recursive set. The set $T_{\Gamma}$ is a set of integers (Gödel numbers) determined by the system $\Gamma$

Martin Davis’s conjecture is to prove that a Diophantine set is recursively enumerable but not recursive. Therefore, according to the definition regarding combinatorial systems, the decision problem of Diophantine set is not recursively solvable.

This conjecture was eventually proved by Yuri Matiyasevich by constructing a Fibonacci sequence, whose elements are recursively enumerable, for the exponential Diophantine equations which Davis, Putnam and Robinson developed. In computability theory, it is proved that there exist an undecidable enumerable set of non-negative integers. Because Hilbert sought for an algorithm for deciding any given Diophantine equation, this problem is unsolvable.

The following quotation shows the pros and cons of proving the unsolvability of a problem:

“After all, showing that a problem is unsolvable doesn’t appear to be any use if you have to solve it. You need to study this phenomenon for two reasons. First, knowing when a problem is algorithmically unsolvable is useful because then you realize that the problem must be simplified or altered before you can find an algorithmic solution. Like any tool, computers have capabilities and limitations that must be appreciated if they are to be used well. The second reason is cultural. Even if you deal with problems that clearly are solvable, a glimpse of the unsolvable can stimulate your imagination and help you gain an important perspective on computation.”  
Sipser (1997), p.151

The solvability of Diophantine equation has been applied to many diverse problems. The first application of this in the context of economics was accomplished by Velupillai. One of the notable applications is the effective playability of Arithmetical Games.

“There is a whole tradition of game theory, beginning at the beginning, so to speak with Zermelo, linking up, via Rabin’s modification of the Gale-Stewart infinite game, to recursion theoretic formulations of arithmetical
games underpinned by the *axiom of determinacy* and completely independent of axiom of choice and eschewing all subjective considerations. In this tradition notions of effective playability, solvability and decidability questions take on fully meaningful computational and computable form where one can investigate whether it is feasible to instruct a player, who is known to have a winning strategy, to actually select a sequence to achieve the win. None of this is possible the orthodox traditions, which cons us into a somnambulance that there are no alternative mathematics for investigating, mathematically, adversarial situations in the social sciences” Velupillai (2010a), p.182

As Simon had already mentioned that he was happy to be cushioned by Turing Computability and he was looking for computational complexity in average cases of problems or even empirical complexity that is relevant for human problem solving. Now we have seen that there exist unsolvable decision problems, algorithmically, thus Turing computability should be the natural confinement of human rationality or machine computability. However, computability in theory takes no account of limitation on time and space which are required to solve a problem or compute a function. This only reinforces the conclusions and concepts that Simon advocated.

### 3 Computational Complexity

Rational decision making, for Simon, can be seen as having a problem and and trying to solve it. Before we start to look for the solution, it is natural for us to ask whether it is solvable or not, despite, in most of the cases, we have to try to solve it anyway. It is also intuitive that there are fairly easy problems and difficult problems. Equivalently, we can say that the problem space of a given problem is more complex then another. Unless we have better heuristics for the more complex problem, it is reasonable to say that the more complex problem is going to take more time and effort. Simon offered a clue on how to describe complexity which can be associated to computational complexity which is the product of computability theory:

> “How complex or simple a structure is depends critically upon the way in which we describe it. Most of the complex structures found in the world are enormously redundant, and we can use this redundancy to simplify their description. But to use it, to achieve the simplification, we must find the right representation.” Simon (1962),p.481

We would like to interpret the above fragment in terms of computational complexity theory, where the complexity of a problem is the complexity of the algorithm which is designed to solve it. Obviously, more rigorous definitions on algorithms and computational complexity are required. This leads to the next two sections.

There are three aspects of problem solving: the inherent solvability of a problem, the procedure to solve a problem and the complexity of the problem. Provided that we have Turing’s abstract model of computation, we can use the idea to construct an abstract machine for solving a particular problem. We can then analyse the number of steps or memory that the algorithm would require, approximately. This helps us
to have an idea of the associated difficulty of a problem we are dealing with before we really start to solve it. This provides another more solid and inner confinement of bounded rationality. Although the scale of time steps and space (memory) that computational complexity theory regards is normally pretty large, it is important to have a general idea of tackling a problem by knowing the complexity of the algorithm which solves it. In theory, they also use the reducibility among problems to study the complexity without actually construct a real algorithm.

As far as problem solving is concerned, according to Turing’s interpretation, a decision problem is to decide whether one can change a string of symbols to the desired string of symbols by only using a set of rules which are given in advance. Knowing a decision problem is unsolvable leads us to ask different questions and try to solve them, otherwise, it provides no practical help when we try to solve a problem. We need to find a set of rules for a substitution puzzle, that is “algorithm”, to solve our problem. However, even if we have an algorithm to solve a certain kind of problem, it does not guarantee that we can solve the problem within the desirable period of time and circumstance. Because the problem involved may be huge in scale, it can demand immense amount of computation from the problem solver. In order to have some brief idea of how much effort is needed for solving a problem, time complexity and space complexity are very useful and standard tools.

When we have an algorithm for solving a problem, we can more or less look at its general behaviour and analyse how many time steps and how much space or memory it would require. Time complexity tells the number of steps needed for running an algorithm, and space complexity takes care of the memory needed. Recall that these two aspects of resources never concern the Universal Turing Machine or Models of Turing Machines in general. Time and space complexity are the functions of size of input, for example, playing 3-disk Tower of Hanoi needs much less time steps than 10-disk Tower of Hanoi. In many cases, it is very difficult to obtain the exact reduced form of time and space complexity of an algorithm. Therefore, in computational complexity theory, $O(n)$ and $\Omega(n)$ (defined in the appendix) are used to present the asymptotic behaviour of an algorithm as the asymptotic approximation of the true function behind it. Normally, the $O(n)$ and $\Omega(n)$ represent work on large inputs.

There are several classes of complexity which gives an idea of how difficult a problem is. The classes are classified by the function type, for example, the time complexity $5n^3 + 4n^2 + 3$ is polynomial, and $3^n$ is exponential. When the input is big enough, exponential complexity grows much faster then polynomial ones, which makes problem solving become practically infeasible; although we have a method of doing so. It should be noticed that there exist always more than one method to solving a problem, therefore, the complexity of a problem is determined by the method that solves it. The complexity of a problem mentioned in theory are the most efficient algorithm ever found.

Precise computational complexity in time and space is very hard to attain, but it can be approximated. It can be approximately estimated by constructing an abstract machine for solving a problem and analysing the order of growth of complexity of that machine.

**Definition 6** Let $f, g: \mathbb{Z}^+ \to \mathbb{R}^+$. $f(n)$ is said to be $O(g(n))$, read “$f(n)$ is ‘big Oh’ of $g(n)$” – or just “$f(n)$ is ‘Oh’ of $g(n)$” – if and only if there exists $n_0 \in \mathbb{Z}^+$ and $c \in \mathbb{R}^+$ such
that we have
\[ f(n) \leq c \cdot g(n), \forall n \geq n_0. \]

The asymptotic complexity of functions can also be obtained by applying the following rules:

**Theorem 3**

- **Transitivity:** If \( f(n) \) is \( O(g(n)) \) and \( g(n) \) is \( O(h(n)) \), then \( f(n) \) is \( O(h(n)) \).

- **Scaling:** If \( f(n) \) is \( O(g(n)) \), then, for any \( k > 0 \), \( f(n) \) is \( O(k \cdot g(n)) \).

- **Rule of sums:** If \( f_1(n) \) is \( O(g_1(n)) \) and \( f_2(n) \) is \( O(g_2(n)) \), then \( (f_1 + f_2)(n) \) is \( O(\max(g_1(n), g_2(n))) \).

- **Rule of products:** If \( f_1(n) \) is \( O(g_1(n)) \) and \( f_2(n) \) is \( O(g_2(n)) \), then \( (f_1 \cdot f_2)(n) \) is \( O((g_1 \cdot g_2)(n)) \). Note that \( (f_1 \cdot f_2)(n) \) refers to the product of the values of the two functions at \( n \) and not to the value at \( n \) of the composition of the two functions.

Let us look at the examples (from (Sipser, 1997, chapter 7)) of analysing complexity of algorithms for solving the same problems. This is to show that the complexity of a language depends on the model of computation selected. For example, \( A = 0^k1^k \mid k \geq 0 \), there is a one-tape machine \( M_1 \) which decides \( A \) in \( O(n^2) \), and there is another one-tape machine \( M_2 \) which can decide \( A \) in shorter \( O(n \log n) \), and there is another two-tape machine \( M_3 \) which decides \( A \) in \( O(n) \). The procedures of these machines are the following:

**\( M_1 \):** On input string \( w \):
1. Scan across the tape and reject if a 0 is found to the right of 1.
2. Repeat the following if both 0s and 1s remain on the tape.
   (a) Scan across the tape, crossing off a single 0 and a single 1.
3. If 0s still remain after all the 1s have been crossed off, or if 1s still remain after all the 0s have been crossed off, then reject. Otherwise, if neither 0s or 1s remain on the tape, accept.

The stage 1 of \( M_1 \) runs in \( O(n) \) and stage 2 runs in \((n/2)O(n) = O(n^2)\), and stage 3 runs at most \( O(n) \). It total, \( M_1 \) in \( O(n^2) \).

**\( M_2 \):** On input string \( w \):
1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Repeat the following as long as some 0s and some 1s remain on the tape.
   (a) Scan across the tape, checking whether the total number of 0s and 1s remaining is even or odd. If it is odd, reject.
   (b) Scan again across the tape, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1.
3. If no 0s and no 1s remain on the tape, accept. Otherwise reject.

Stage 1 and 3 of $M_2$ together run in $O(n)$ and stage 2 runs in $(1 + \log_2 n)O(n) = O(n \log n)$, because after each round, the size of the problem is reduced to half. In total, $M_2$ runs in $O(n \log n)$.

The third machine is a two-tape Turing machine.

$M_3$: On input string $w$:

1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Scan across the 0s on Tape 1 until the first 1. At the same time, copy the 0s onto Tape2.
3. Scan across the 1s on Tape 1 until the end of the input. For each 1 read on Tape 1, cross off a 0 on Tape 2. If all 0s are crossed off before all the 1s are read, reject.
4. If all the 0s have now been crossed off, accept. If any 0s remain, reject.

Each of the states of $M_3$ runs in $O(n)$, therefore $M_3$ runs in $O(n)$ which is linear time.

### 3.1 Time Complexity

Let us look at the definitions regarding classes in time complexity and the reducibility of languages.

**Polynomial time:**

**Definition 7** $P$ is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words, $P = \bigcup_k \text{TIME}(n^k)$

**Definition 8** A function $f : \Sigma^* \rightarrow \Sigma^*$ is a polynomial time computable function if some polynomial time Turing matching $M$ exists that halts with just $f(w)$ on its tape, when started on any input $w$.

**Nondeterministic polynomial time:**

**Definition 9** A verifier for a language $A$ is an algorithm $V$, where $A = \{w | V$ accepts $<w,c>$ for some string $c$}

**Definition 10** NP is the class of languages that have polynomial time verifiers.

**Theorem 4** A language is in NP iff it is decided by some nondeterministic polynomial time Turing machine.

**Reducibility:**

**Definition 11** Language $A$ is polynomial time mapping reducible, or polynomial time many-one reducible, or simply polynomial time reducible, to language $B$ written $A \leq_P B$, if a polynomial time computable function $f : \Sigma^* \rightarrow \Sigma^*$ exists, where for every $w$,

$$w \in A \iff f(w) \in B$$

The function $f$ is called the polynomial time reduction of $A$ to $B$.

**Theorem 5** If $A \leq_P B$ and $B \in P$, then $A \in P$

**NP-complete**

**Definition 12** A language $B$ is NP-complete if it satisfies two conditions: 1. $B$ is in NP, and 2. every $A$ in NP is polynomial time reducible to $B$. 

24
3.1.1 SAT

Satisfiability problem is proved to be NP-complete and will be later applied to show its significance as a satisficing criterion. We start with the definition of Boolean formula.

**Definition 13** A Boolean formula is a formula over Boolean variables using operators \( \lor, \land, \text{and} \neg \).

**Definition 14** The problem Satisfiability (SAT) is defined as follows: Given a Boolean formula \( \phi \), determine whether there is an assignment that satisfies it (i.e., more formally, SAT is the set of all satisfiable Boolean formulas).

In Velupillai (2010b), it is demonstrated that SAT problem is the meeting ground of Diophantine problem and satisficing, in turn this connection leads to the conclusion that bounded rationality is the superset of Olympian rationality, which Simon had been advocating.

**Theorem 6** Cook-Levin Theorem. SAT is in \( P \) iff \( P=NP \); or in another form, SAT is NP-complete.

3.2 Space Complexity

Space complexity has attracted less attention and effort of research than time complexity. By default, when the complexity of a problem is discussed, time complexity is referred. Arguably, it is because whether \( P = NP \) is one of most popular unsolved problems. In this chapter, we claim that in the domain of human problem solving, space complexity is at least as important as time complexity. Although, there is no doubt that human minds have the potential to store huge amount of knowledge, the amount of information that minds can process at the given moment is severely limited. For example, it is very tough to calculate \( 4593 \times 3274 \) in the mind for an ordinary person, unless this person has pencil and paper at hand or he/she is an expert of arithmetic calculations. Such calculation requires certain amount of temporary memory which is a function of input size. In terms of time limitation, minds are constrained with attention span, apart from other external time conditions, e.g., a chess player has to make a move in 5 minutes. How minds are constrained by time and memory varies with different contexts and structure of the problems and different persons. Furthermore, these two dimensions should not be completely independent, i.e. the memory constraints affects the time which is needed for solving a problem and vice versa. Therefore, we believe that it is important to investigate the time complexity of a problem or an algorithm together with the space complexity; consequently, we will be able to know what kinds of heuristics are needed based on these two dimensions. In the following, there are important definitions and theorems regarding space complexity.

**Definition 15** Let \( f : \mathcal{N} \to \mathcal{N} \) be a function. The space complexity classes, \( \text{SPACE}(f(n)) \) and \( \text{NSPACE}(f(n)) \), are defined as follows.

\[
\text{SPACE}(f(n)) = \{ L | \text{L is a language decided by a } O(f(n)) \text{ space deterministic Turing machine} \}
\]

\[
\text{NSPACE}(f(n)) = \{ L | \text{L is a language decided by a } aO(f(n)) \text{ space nondeterministic Turing machine} \}
\]

The first theorem Savitch proved in his paper (Savitch, 1970) is very significant for space complexity. It says:

**Theorem 7** If a set, \( A \), is accepted by a nondeterministic Turing machine, \( Z_n \), within storage \( L(n) \geq \log_2 n \), then \( A \) is accepted by some deterministic Turing machine, \( Z_d \), within storage \( |L(n)|^2 \).
The theorem leads to the conclusion that $PSPACE = NPSPACE$.

**PSPACE-complete:**

**Definition 16** A language $B$ is PSPACE-complete if it satisfies two conditions:

1. $B$ is in PSPACE, and
2. every $A$ in PSPACE is polynomial time reducible to $B$.

If $B$ merely satisfies condition 2, we say that it is **PSPACE-hard**.

In spite of the fact that the time and space complexity of an algorithm can be analysed, human beings are constrained very differently from (digital) computers. We normally have only a certain amount of time to make a decision, and we have very limited working memory to process this task, regardless of the presumably unlimited long-term memory. We are forced to use those algorithms which will be able to halt within certain amount of time by applying the knowledge and experience we have in the long-term memory. Although, we are often assigned to a task like “find the best person for this job”, we are not able to solve it as an optimization problem. At best, we will have the criteria for appropriate candidates and consider only a small group of people. In complexity theory, the time and space of which an algorithm will take depend on the size of the inputs. That is, if we have only $O(n^6)$ time and space, then we can consider only at most 6 options.

### 3.3 Complexity of Combinatorial Games

Chess is the recurring example and an important one for Simon. Even though the problem space of chess is closed and certain, the massive complexity prevents human beings or even supercomputers to use brute search algorithm⁶. Let us take the complexity of Chess, which is approximated in Shannon (1950) as an example. If we want to know, from the beginning of the game, whether Black or White has a winning strategy, we have roughly $10^{120}$ variations to calculate; when each branch reaches its end, we can see whether that branches leads to a win, loss or draw. Suppose we have a high-speed computer which uses only one microsecond ($10^{-6}$ a second) for one variation, for searching the whole problem space, it will take $10^{100}$ million years! Obviously, in real life, different actions should be taken, that is why we can always find plenty of Chess tutorials in the bookshops.

“The common feature of all combinatorial problems is their discrete and finite nature: their parameters can only take on discrete values, and their solutions can only be drawn from a finite set of possibilities. The finiteness of these problems generally ensures the existence of some brute-force method of solution: simply generate all possible solution structures, always keeping the best one. Such methods, however, can only succeed with fairly small solution set.” *Moret and Shapiro (1991)*, p.1-2

The game of Go will be the paradigm of this thesis to go beyond Simon by programming information processing systems of solving it. This game has already been studied by combinatorial game theory. It is shown in Lichtenstein and Sipser (1980) that GO is in PSPACE-Hard, ⁶In game theory, the counterpart of optimization as in classical theory is min-max. A optimal strategy in chess will be to observe the entire game-tree, especially the terminal nodes which can be and only can be classified to win, lose, and draw, and to apply backward induction and choose an action which leads at least to a draw. In this case, chess is a trivial game to Von Neumann and Morgenstern. The approximate game-tree size, $10^{120}$, is an immediate fact to suggest that min-max has no practical use.
although later on, it is shown that no PSPACE algorithm can exist (Robson, 1983). Chess, for example, is not yet shown to be in PSPACE-hard. The problem asked in the previous paper is “given an arbitrary Go position on an $n \times n$ board, determine the winner”. They start by reducing truth quantified boolean formulas (TQBF) which is PSPACE-complete to Generalized Geography, which is also PSPACE-complete. Furthermore, they reduced the Generalized Geography problem to planar Generalized Geography, which is again PSPACE-complete. Then they encode the constructed planer generalized geography into Go positions. Following the argument, it is clear that all the problems in PSPACE can be polynomially reduced to TQBF, and further reduced to GG which is also PSPACE-complete. Now, all the problems in PSPACE can be polynomially reduced to GG, and further reduced to planar generalized geography which is also PSPACE complete. Now, they have encoded planar generalized geography into Go positions, which means all the problems in PSPACE can be reduced to Go. But Go is not yet known as PSPACE, therefore, at best we can conclude that Go is PSPACE-hard.

People then started to look for PSPACE algorithms for solving Go, and the existence of a PSPACE algorithm of Go is refuted by proving that Go is EXP-time complete (Robson, 1983), which means Go is exponential time. Problems in exponential time are considered to be among the most difficult problems. This result suggests that deciding whether Black or White has winning strategy at an arbitrary position is practically infeasible.

4 Computable Foundation of Bounded Rationality and Beyond

“Theories that incorporate constraints on the information-processing capacities of the actor may be called theories of bounded rationality” Simon (1972), p.162

4.1 Computable Economics

Velupillai (Velupillai, 2000, 2010a,b) has formalized decision making as choosing a subset from an finite, non-empty, countable set, as opposed to uncountably infinite sets, by using a choice function. Solving optimization of the later case is equivalent to solving linear programming problem. Velupillai transformed the linear programming problem in to the optimization problem of a combinatorial system, and then constructed abstract Turing machines to study the characteristics of the problem solving. His conclusion is that bounded rationality in the context of decision problem is more general than Olympian rationality. Optimization in Olympian rationality has no useful sense. This result is consistent with Simon in the following quotation.

“If a distinction is wanted between this very strict species of rationality and more general forms, the former maybe termed optimality, the latter adaptiveness or functionality. Simon (1982), p.405”

The conclusion is obtained by equating the notion of the behaviour of a suitably programmed (universal) Turing machine to the rational behaviour of the “economic agent” and by connecting adaptive process with dynamical system for studying the computation universality. And it is proved that only those dynamical systems which are capable of computation universality are consistent with Olympian rationality. He further proves that no trajectory of dynamical systems capable of computation universality can be usefully related to optimization in Olympian rationality. On the other hand, it was also demonstrated that a boundedly rational information processing system in the decision problem framework is capable of computation universality. In the decision-problem framework, optimizations are merely specially case.
The decision problem framework of a boundedly rational information processing system can be achieved through the channels, among systems of linear Diophantine equations, systems of linear equations in nonnegative integer variables, integer programming, and SAT. Solving the former three problems are equivalent in the sense that the method of one problem leads to the other two. The channel between integer linear programming and SAT can be made possible by translating one from each other. For example, by expressing the negation in conjunction normal as \( \neg x_1 = (1 - x_1) \), then the CNF can be written as a system of equations where the variables are boolean values. By adding an optimization criterion, the problem becomes linear integer programming, which is one of the special cases. SAT is in NP-complete, which means all the problems in NP can be polynomial time reduced to SAT; and SAT also runs in space \( O(n) \) which is linear. Through the theories and the mathematical channels of the problems, Velupillai showed the possibility and way to formalized satisficing into SAT or Diophantine decision problem. Moreover, linear programming problem has been proved to be in polynomial time (Khachiyan, 1979), which is more tractable than SAT.

The main definitions and theorems used in the above argument are the following:

**Theorem 8**  Boundedly rational choice by an information processing agent within the framework of a decision problem is capable of computation universality.

**Definition 17**  A dynamical system capable of computation universality will be called a universal dynamical system.

**Lemma 1** Dynamical Systems capable of Computational Universality can be constructed from Turing Machines.

**Definition 18**  A dynamical system - discrete or continuous - is said to be capable of computation universality if, using its initial conditions, it can be programmed to simulate the activities of any arbitrary Turing Machine, in particular, the activities of a Universal Turing Machine.

**Theorem 9**  There is no effective procedure to decide whether given class of decision rules are “steady states of (some) adaptive process.”

**Theorem 10**  Only dynamical systems capable of computation universality are consistent with rationality in the sense that economists use that term in the Olympian Model.

**Theorem 11**  No trajectory of a dynamical system capable of universal computation can, in any ‘useful sense’, be related to optimization in the Olympian model of rationality.

With this background on algorithms, we may hypothetically view the decision making in economics as searching the elements in a list. Presumably, we are looking for one element in the list which gives us the highest satisfaction. In the ideal case, we know the values regarding each element in the list, and the list is sorted. Then we only need to go to the bigger end of the list and pick up the first element in the list. If we have an unsorted list of values regarding all the elements, then we need to go through every element in the list; if we have \( n \) elements in the list, this action is going to take \( O(n) \) time steps and small memory just to keep track of the biggest value ever found. Again, if it is the case that we know all the values of the elements but the list is unsorted, will we be better off from sorting it in the first place? The most efficient sort algorithm ever constructed has \( O(n \log n) \) time, such as “Merge sort”, which was first written by John Von Neumann in 1945 (Knuth, 1970). It is obvious that sorting first and then pick up

---

The author of this series of book won the Turing Award in 1974, one year prior to Newell and Simon.
the biggest will need more time than simply walking down from the beginning of the list to the end; that is because $O(n \log n) > O(n)$.

If we are looking for a particular value in the list instead of the biggest one, then the time of search can be further reduced. Let us take an example of a sorted list: [1,3,5,14,35,29,40,58,96,120,234,475]. Suppose we are looking for ‘234’ in the list, we can simply start from the first element from the left, and after 11 steps, we will get what we want. The algorithm of $O(\log n)$ which will be demonstrated below is typically the most preferable class. It results from constantly discarding sufficient amount of choices during the course of search. The most typical example is “binary search”, which can be applied to a sorted list. We start at the mid way of the list, say 40, in the list. We know that 234 is greater than 40, so the elements at the left hand side of 40 including itself can be discarded. Now the size of the list has been reduced to 5. Again, we start at the midway, which is 120; we know that we also can discard some elements to the left of 120. As soon as we do so, we find what we want. Totally it takes 3 steps only! However, if the list is not sorted from the beginning, we benefit nothing from sorting the list first if we are going to look for a particular value in the list.

The decision making in the above discussion is made possible by the assumptions: (i) there are some elements in the list and (ii) the values regarding each element are given. This is what Simon had been objecting about. In many realistic circumstances, we do not even know what elements we should consider when we encounter some problem. Even when we have made a list of alternatives, we need to evaluate the choices, which may incur very complex dynamics which follow those choices. Therefore, in realistic problem solving situations, we need to consider the structure or architecture of the problems along with the capacity and experience that the problem solvers have.

Two rather striking theorems in Velupillai (2000) for objecting assumption (ii) in the previous paragraph:

**Theorem 12** “There is no effective procedure to generate preference orderings.”

**Theorem 13** Given a class of choice functions that do generate preference orderings (pick out the set of maximal alternatives) for any agent, there is no effective procedure to decide whether or not any arbitrary choice function is a member of the given class.

### 4.2 Human Problem Solving

Standing within the theoretical premise built by Velupillai, we are able to interpret bounded rationality in the context of Turing computability. More importantly, Velupillai has created, at the same time, a bridge from computability to computational complexity through linking satisficing to decision problems. We then are able to study the complexity of a given problem as a general idea of how difficult a problem is. It is legitimate to ask whether there is absolute complexity which human mind can not handle any more. Note that the complexity of a problem depends on the algorithm for solving it, and brute-force algorithms are normally the benchmark, especially, when an optimization problem is proposed. Satisficing and heuristics now have very important roles to play in reducing the complexity of a problem. This is where and when we need some input from psychology and cognitive science, in other words, this is the meeting ground of economics, psychology the cognitive science. It is evident that human minds rely on very simple heuristics (algorithm) to solve a problem in a satisficing manner and also rely on the support of external memory and knowledge.

The theory of human problem solving where human beings are viewed as information processing systems is a well constructed paradigm of understanding practical problem solving of individual or organization. A critical step that Simon took is to ask different questions, i.e.
changing the focus from “What is my winning move?” to “What is my next good move?”. By
doing so, we are already able to abandon brute-force algorithms and ignore its astronomical
complexity. Reasonably, we might reach the next question - “what is my goal of this move such
that I can choose a good move?”, “what is the more attainable goal of the larger goal?” The
approach of dissecting a general problem into its means-end structure has been fully realized
is the rich paradigm that Simon had contributed as soon as he proposed his idea on bounded
rationality. A brief introduction on theory of human problem solving and information processing
system which is the computation model of the human problem solver can be found in (Kao

Therefore, we argue that it is meaningful and useful to bring in the measure of computa-
tional complexity into Human problem solving. Subsequently, we can bring in the context of
the task environment and look for the possible heuristics for reducing the complexity of a prob-
lem. No matter how difficult a problem might be, a physical system has to output an answer
with limited resource - time and space - or crash without achieving anything it is supposed to.
Heuristics are the methods or algorithms that are used to reduce the complexity of a problem
to levels that can be handled. They involve in generating a subset of alternatives, evaluating
the alternatives with the ability of pattern recognition associated with accumulated knowledge,
stopping evaluation. The first two aspects of using heuristics involve pattern recognition, and
the last one involves both pattern recognition and aspiration level.

The three aspects of heuristics are the core of HPS. The mapping between A and S - the
evaluation of alternatives - is supposed to by the most difficult one.

4.2.1 Heuristics

Heuristics are applied by almost everyone in almost every situation. For example, when a
small group of people want to decide what to do, it might be still easy to collectively make a
satisfactory solution. However, when the number of people gets too large, that is, when the
complexity of the problem is too big to handle, heuristics are needed. Good heuristics will later
on become regulations or law, so that if the same problems appear, people will know that the
good way is to consult the regulation and follow it. When we have a problem in mind - “what
is the good solution of this problem?” - we will need to narrow down the final goal to other
sub-goals and sub-sub-goals, etc. Therefore, a sub-goal may be a means to the final goal, but
it maybe the end of some sub-sub goal. Heuristics help the decision makers to travel between
means and ends. In our point of view, heuristics are algorithms.

Recall that in the section of computational complexity, three machines that are designed
to solve same problems were introduced. One is better than the other in terms of the smaller
complexity of the algorithm. This example can be an analogue in real situations that an expert
of a specific problem might have developed better heuristics of solving it. For example, the
most basic heuristics algorithms, in my point of view, are divide and conquer algorithms.

The negative evaluation on the achievement of Logic Theorist for proving a theorems in Principia
Mathematica (Newell et al., 1958) given by Hao Wang who failed to understand that heuristics are
algorithm is the following

“There is no need to kill a chicken with a butcher’s knife. Yet the net impression is
that Newell-Shaw- Simon failed even to kill the chicken with their butcher’s knife. ... To
argue the superiority of ‘heuristic’ over algorithmic methods by choosing a particularly
4.2.2 Near Decomposability: Divide and Conquer Heuristics

When we are facing a problem of huge size, we might not have an idea of how to deal with them in the first time. An intuitive heuristic is to divide the problem into smaller sub-problems and conquer them separately. Arguably, not all the problems can be amenable to divide and then conquer. This can be associated with a mathematical property defined by Herbert Simon - Near Decomposability. A brief history on how Simon came across the idea of near decomposability can be found in Kao and Velupillai (2012)

I will demonstrate some example to show why divide and conquer algorithms can help reduce the computational complexity. Of course in real human problem solving cases, divide-and-conquer can have many different forms. A pseudo code (in Matlab) of divide and conquer can be constructed as the following:

```matlab
function [output:Solution S]=Divide-And-Conquer(input: Problem P)
if P is small % if the problem is smaller than certain level
    solve the problem and output S;
else
    divide P into small problems of type identical to the original, P_1, P_2, ... P_k; % k ≥ 2 can be a constant of a function of P
    [S_1]=Divide-And-Conquer(P_1);
    [S_2]=Divide-And-Conquer(P_2);
    ...
    [S_k]=Divide-And-Conquer(P_k);
    combine S_1, S_2, ... S_k into S,
end
```

One of the examples of divide-and-conquer algorithm is the “mergesort” algorithm which is written by von Neumann. This is an algorithm of sorting. Its complexity is $O(n \log n)$ which is reduced from $O(n^2)$ - the complexity of “insertion.” (Moret and Shapiro, 1991, chapter 8) The heuristics in sorting are relatively extreme for human decision making, because in sorting, all the alternatives have to be reviewed at some point of time. On the contrary, in real-life decision making, we may divide the alternatives into sub-group and then find a satisfactory solution in one of the sub-group without going through all the alternatives.

Simon also considered a possible method which will reduce the problem to the smaller size

“One possible way would be to replace the actual problem space with a very much smaller space that approximates the actual one in some appropriate sense, and apply the classical theory to the smaller approximate space” Simon (1972), p.166

5 Concluding Remarks

It is not justifiable to assume that a choice made by an individual is a good, bad or best choice, a priori. Although we do try hard to make a good choice, when in the face of time constraints, it is also likely that a choice just made is a bad or a disappointing choice. It is important to know what to choose, as well as what not to choose. Furthermore, even though we may know our goals and sub-goals, it does not mean that we know how to attain them. Even if we have an abstract map of a city, and we would like to travel from A to B, it does not actually mean that we can reach B from A without any difficulty. We still have to work through the paths, follow the sings, check the map, etc.
If we view a piece of knowledge as an articulated paragraph composed by words which are composed by symbols from a finite set. It is never straightforward to understand or learn the knowledge by reading or memorizing it. We might have to follow the author with the same or different path on how the knowledge is reached. What matters is the knowledge generated in the mind, not the knowledge written. This is partly because a segment of words might have been a compression or definition of bigger segment of words.

Simon’s descriptions and definitions regarding bounded rationality were often very intuitive and straightforward. He thus left a big free room for others to build models based on bounded rationality. Simon’s intention can be interpreted more deeply, in the sense that how human rationality is bounded can be formulated with various theories and tools. More precisely and practically, different models should be constructed according to different situations and different minds which handle those situations. In this paper, it is emphasized that the two aspects of human problem solving - the task environment (problem space) and problem solver (algorithm) should be separately studied.

By appealing to computability theory, it is shown that bounded rationality is a superset of Olympian rationality. In this chapter, we further narrow down the boundary of rationality to the inner boundary - computational complexity. We suggest that heuristics are the method for us to further narrow down the boundary to Simon’s empirical boundaries.
A  (Formal) Language and Grammar

Definition 19  A language is a set of strings.

Definition 20  A grammar $G$ is a 4-tuple $(V_N, V_T, P, S)$ where $V_N$ is a finite set, the set of nonterminals or variables; $V_T$ is a finite set, the set of terminals. It is required that $V_N \cap V_T = \emptyset$. $S \in V_N$ is a distinguished symbol, the start symbol. $P$ is a set of semi-Thue (see Post, 1947) instructions over $V_N \cup V_T$, usually called productions (or rewriting rules) in this context.

Thus, $G$ is a semi-Thue instructions over $V_N \cup V_T$. The (formal) language defined or generated by $G$ is denoted by $L(G)$.

Definition 21  A type 0 grammar is one that satisfies Definition 20, with no other restrictions except those already mentioned in 20

A grammar is of type 1 $\iff$ for every $(x, y) \in P$, $|x| \leq |y|$, where $||$ is the length function on $(V_N \cup V_T)^*$ (context-sensitive)

A grammar is of type 2 $\iff$ for every $(x, y) \in P$, $|x| \leq |y|$ and $x \in V_N (|x| = 1)$ (context-free grammar)

A grammar is of type 3 $\iff$ for every $(x, y) \in P, x \in V_N, y = aw$, where $a \in V_T, w \in V_N \cup \Lambda$ (regular grammar)

A language $L \subset A^*$ is of type $i (i = 0, 1, 2, 3)$ $\iff$ there is a type $i$ grammar $G$ such that $L = L(G)$

B  Asymptotic Complexity Notation

Definition 22  Let $f, g : \mathbb{Z}^+ \rightarrow \mathbb{R}^+$. $f(n)$ is said to be $\Omega(g(n))$, read “$f(n)$ is ‘big Omega’ of $g(n)$,” if and only if there exist a constant $c \in \mathbb{R}^+$ such that, for any $N$, there exists an $n \geq N$ with $f(n) \geq c \cdot g(n)$.

Definition 23  Let $f, g : \mathbb{Z}^+ \rightarrow \mathbb{R}^+$. $f(n)$ is said to be $\Theta(g(n))$, read “$f(n)$ is ‘big Omega’ of $g(n)$,” if and only if $f(n)$ is $O(g(n))$ and $g(n)$ is $O(f(n))$.

References


