International Journal of Research
International Journal of Research (IJR)
e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 05, May 2015
Available at http://internationaljournalofresearch.org

Limits of Algorithmic Computation

Nitin Garg
((6th Semester) Electronics and Computer Engineering, Dronacharya College of Engineering)
ngarg1910@gmail.com

Nitigya Grover
((6th Semester)Electronics and Computer Engineering, Dronacharya College of Engineering)
Nitigyagrover01@gmail.com

Introduction

We all studied about what turing machines can do now look at what they cannot do, Although Turing's thesis leads us to believe that there are few limitations to the power of a Turing machine, we have claimed on several occasions that there could rrot uxist anv algorithms for the solution of certain problems. Now we make more explicit what we mean by this claim. Some of the results came about quite simply; if a language is nonrecursive, then by definition there is no membership algorithm for it. If this were all there was to this issue, it would not be very interesting.

Non recursive languages have little practical value. But the problem goes deeper. For example,we have stated(but not yet proved) that there exists no algoritlm to determine whether a context-free grammar is unambiguous. This question is clearly of practical significance in the study of programming languages. We first define the concept of decidability and computability to pin down what we mean when we say that something cannot be done by a Turing machine. We then look at several classical problems of this type, almong then the well-known halting problem for Thring Machines. Form this follow a number of related problems for Turing machines and recursively enumerable languages. After this, we look at some questions relating to context-free languages. Here we find quite a few important problems for which, unfortunately, there are no algorithms.

Some ProblemsThat Cannot Be Solved by Turing Machines

The argument that the power of mechanical computations is limited is not surprising. Intuitively we know that many vague and speculative questions require special insight and reasoning well beyond the capacity of any computer that we can now construct or even plausibly foresee.What is more interesting to computer scientists is that there are questiotns that can be clearly and simply stated, with irrt apparetrt possibility of an algorithmic solution, but which are known to be unsolvable by any computer.

Computability and Decidability

we know that a function ‘g’ on a certain domain is said to be computable if there exists a Turing machine that computes the value of ‘g’ for all arguments in its domain. A function is uncomputable if no such turing machine exists. There may be a Turing machine that can computes the function on the whole domain, but we call the function computable only if there is a, Turing machine that computes the function on the whole of its domain. We see fron this that, when we classify a, function as computable or not computable, we must be clear on what its domain is.

Our concern here will be the somewhat simplified setting where the result of a computation is a, simple “Yes” or “No”. in this case, we talk about a problem being decidable or undecidable. By a problem we will understand a set of related statements, each of which must be either true or false. for example, we consider the statement “For a context free grammer G, the language L(G) is ambigous”. For some G this is true, for others it is false, but clearly we must have one or the other. The problem is to decide whether the statemnets is true for any G we are given. Again, there is a underlying domain, the set of all context-free grammars. We say that a problem is decidable if there exists a, Turing macirine that gives the correct answer for every statement in the domain of the problem.

When we state decidability or undecidability results, we must always know what the domain is, because this may, affect the conclusion. The problem may be decidable on some domain but not on another. Specifically, a single instance of a problem is always decidabale, since the answer is either true or false. In the first case a Turing machine that always answers "true" gives correct answer, while in the second case one that always answers "false" is appropriate. This may seem like a facetious answer but it emphasizes an important point, fhe fact that we do not know what the correct answer is makes no difference, what matters is that there exists some turing machine that does give the correct response.

Refrences:

[1.] E-book : An Introduction toFormal Languages and Automata Third Edition Peter Linz Universityo f Californiaa t Davis.

[2.] www.Google.com

[3.] www.google.com/Theory of Automata and Computaion