Implicit Guarantees of the Computational Complexity of Feasible Programs

Datalogisk institut, Copenhagen University (DIKU)
Master's Thesis

Oleksandr Shturmov
oleks@oleks.info

December 11, 2014.

Chapter 0

Preface

0.1 Audience

The audience of this thesis is anyone interested in the connection of computability and complexity to the theory of programming languages. In particular, what the admittance of particular programming language constructs implies for the complexity of the programs that you can write.

The thesis is directed towards the level of a Computer Science graduate student at the time of writing: The reader is assumed to be familiar with basics of discrete mathematics and the analysis of time and space complexity of algorithms. The reader is also assumed to be familiar with the theory of programming languages, in particular some formal means of specifying the syntax, and preferably also, the semantics of a programming language. The reader is expected to be familiar with Logic in Computer Science.

Part I

Background

Chapter 1

Computability

Notion 1. *A problem is "computable" if it can be solved by transforming a mathematical object over a finite amount of time, without ingenuity.*

Any attempt at a more definite notion of computability seems to arrive at a philosophical impasse, where the notions of "transformation", "mathematical object", and "ingenuity" form a philosophical conundrum. The indefinite notion however, is sufficient to state the following theorem:

Theorem 1. *The class of computable problems is closed under concatenation.*

That is, if a problem P can be solved by solving a computable problem Q, followed by solving a computable problem R, then P itself is computable.

Proof. Since both Q and R are computable, and no transformations are performed, other than to solve the problems Q and R, P itself is computable. $\square$

Thus we arrive at the common notion of an algorithm:

Notion 2. *An "algorithm" is a specification of how a problem can be solved by performing a finite sequence of finite-time transformations of a mathematical object, without ingenuity.*

TODO:

- The above definition is useful for little else - provide some historical perspective on defining computability beyond this notion, and provide a characterization using function algebras (similar to Church) and Turing machines. Draw parallels to type theory an constructivism.
- The classical result that primitive recursive functions are computable. To argue for this, we probably need to argue for a type theoretic approach, in that, what we can construct, we can compute. Function algebras should also be introduced here.
- General recursion (due to Kleene wrt. definition) and it's undecidability (due to Church).

- A different approach to computability: Post and Turing machines. Prove their equivalence to general recursion above.

Chapter 2

Complexity

This may be a good point to mention that, although I have so far been tacitly equating computational difficulty with time and storage requirements, I don't mean to commit myself to either of these measures. It may turn out that some measure related to the physical notion of work will lead to the most satisfactory analysis; or we may ultimately find that no single measure adequately reflects our intuitive concept of difficulty.

— ALAN COBHAM, *Logic, Methodology and Philosophy of Science* (1964)

In practice, the length of computer computations must be restricted, otherwise the cost in time and money would be prohibitive.

— H. E. ROSE, *Subrecursion: functions and hierarchies* (1984)

2.1 Time

2.1.1 Polynomial Time

- Recursive characterization of polytime functions in [Rose (1984)], proving certain claims by [Cobham (1965)]. Both question the relation to the Grzegorczyk hierarchy [Grzegorczyk (1953)].
- Leivant's paper - A Foundational Delineation of Computational Feasibility.
- Bellantoni and Cook paper - A NEW RECURSION-THEORETIC CHARACTERIZATION OF THE POLYTIME FUNCTIONS
- Niel Jones paper.
- Caporaso
- Upper bounds (algorithms) can be produced by expressing the property of interest in one of our languages. Lower bounds proven elsewhere can be used as a proof that the language is expressive enough.

2.1.2 Subpolynomial Time

In what follows, we delineate a hierarchy of complexity classes, strictly under polynomial time. That is, we present a sequence $C_1(n)$, $t(n) = o\left(n^{O(1)}\right)$. For each complexity class C, we present a representative problem P_C. We aim to find problems which are known to be $t(n) = \Theta\left(C\left(n\right)\right)$.

- Some problems, although computable in polynomial time, are still hard to compute in practice (ICALP'2014, Amir Abboud).
- Remind of the definitions of O, Ω, etc.
- For each of the below show that every subsequent class is distinct from the proceeding, and exhibit some "complete" problems for these classes.

$O\left(1\right)$ **— Constant**

$O\left(\alpha\left(n\right)\right)$ **— Inverse Ackermann**

$O\left(\log^*\left(n\right)\right)$ **— Log star**

$O\left(\log\log\left(n\right)\right)$ **— Log-log**

$O\left(\log\left(n\right)\right)$ **— Log**

$O\left(\log\left(n\right)^{O(1)}\right)$ **— Polylog**

$O\left(n^c\right)$**, for** $0 < c < 1$ **— Fractional power**

$O\left(n\right)$ **— Linear time**

$O\left(n\log^*\left(n\right)\right)$ **— n log star**

$O\left(n\log\log\left(n\right)\right)$ **— n log-log**

$O\left(n\log\left(n\right)\right)$ **— n log n** Comparison-based sorting $o\left(n\log n\right)$.

$O\left(n^2\right)$ **— quadratic**

$O\left(n^3\right)$ **— cubic**

$O\left(n^O(1)\right)$ **— polynomial**

2.1.3 Space

Chapter 3

Implicit Characterizations of P

3.1 Bounded Recursion

3.2 Finite Model Theory

3.3 Ramification

3.3.1 Safe Recursion

3.3.2 Tiering

Bibliography

[Cobham (1965)] Alan Cobham, IBM Research Center, Yorktown Heights, New York, USA. *The intrinsic computational difficulty of functions*. 1965. In Proceedings of the 1964 International Congress for Logic, Methodology and Philosophy of Science, pp. 24–30. Edited by Yehoshua Bar-Hillel. Published by North-Holland Publishing Company in Amsterdam, Holland. Printed in Israel, by Jerusalem Academic Press Ltd.

Cited on page 6.

[Grzegorczyk (1953)] Andrzej Grzegorczyk, Institut Matematiczny Polskiej Akademii Nauk. *Some classes of recursive functions*. 1953. Rozprawy Matematyczne: IV. Edited by Karol Borsuk, et al. Published by Polskie Towarzystwo Matematyczne Warszawa in Poland. Printed in Poland, by Wroclawska Drukarnia Naukowa.

Cited on page 6.

[Hofmann (2000)] Martin Hoffmann, Laboratory for Foundations of Computer Science, University of Edinburgh. *Programming languages capturing complexity classes*. 2000. ACM SIGACT News, Vol. 31, No. 1, pp. 31–42. Published by the ACM in New York, USA.

A concise survey of a range approaches to implcit complexity theory between 1965 and 2000.

No citations.

[Sazonov (1987)] V. Yu. Sazonov, Institute of Mathematics, Novosibirsk, USSR. *Bounded set theory and polynomial computability*. 1987. In Proceedings of the 1987 International Conference on Fundamentals of Computation Theory. Lecture Notes in Computer Science, Vol. 278, pp. 391–395. Editied by L. Budach, R.G. Bukharajev, and O.B. Lupanov. Published by Springer-Verlag in Berlin, Germany. Printed by Druckhaus Beltz.

No citations.

[Rose (1984)] H. E. Rose, School of Mathematics, University of Bristol. *Subrecursion: functions and hierarchies*. 1984. Oxford Logix Guides: 9. Typeset by Joshua Associates, Oxford. Published by Clarendon Press, division of Oxford University Press, in New York, USA. ISBN 0-19-853189. Printed in Great Britain, by The Thetford Press Ltd.

Cited on page 6.