The FRDCSA Project (DRAFT)

Andrew Dougherty
FRDCSA Project
adougher9@gmail.com

Abstract

The Formalized Research Database: Cluster, Study and Apply project (FRDCSA) is an attempt to create a weak AI by collecting and enhancing Free/Libre software. Based on results in Algorithmic Information Theory (AIT), it establishes some constraints that apply to sequences of increasingly powerful programs, namely, that increased program size is ultimately a necessary but insufficient condition for increased capabilities. The easiest method of increasing program size is to collect and package existing Free/Libre software. There are many important software systems which have not already been packaged. Many strategies are used to manage the collection, integration and application of existing software. There are nearly 2000 systems which have been manually collected, in lieu of the automated spidering systems we are developing. However, packaging external code is not sufficient to resolve all encountered problems, so an expansive programme of software development has been implemented creating a library of programs that glue the rest of the functionality together. This consists of over 2400 Perl modules, 440 Prolog files and 600 internal and minor projects, with more expected. Seemingly paradoxically, the creation of more and more projects and software, rather than spreading the project too thin, actually speeds up the completion of major milestones because the added functionality improves the capabilities of the system, creates information fusion, and reduces the distance required to achieve the project goals.

Introduction

This paper documents the motivation for and progress of the Formalized Research Database; Cluster, Study and Apply (FRDCSA) project (Dougherty 2011) towards advancing the capabilities of software systems through development of a system for collecting and redistributing software.

Motivation for Free/Libre Software Development

In order to establish the motivation for the FRDCSA project, we must look at the motivation for Free/Libre software development. There are a few key concepts which work in favor of the present design of the FRDCSA. Due to the unique topological properties of information, it is possible to make copies of software ad infinitum with near-zero additional cost per copy, a.k.a. software has the Zero Marginal Cost property (ZMC). This means that, once developed, anyone can have a copy at no additional (marginal) cost. Another important concept is that of rivalry. Many resources are rivalrous, such as, for instance, an apple. Consumption of an apple prevents others from consuming that apple. However, information by itself is nonrivalrous, consuming the information does not in general prevent others from consuming the information. Free/Libre software is therefore nonrivalrous. Using Free/Libre software does not in general prevent others from using it. This has the same effect - once developed, anyone can have a copy at no additional cost. Lastly, according to the licensing, the entire system of Free/Libre software behaves as one closed system. What you add in one place to the system is practically speaking able to be accessed instantly from another place in the system, and furthermore there is no more or less software available than what is added to the system.

Given that Free/Libre software has the properties of being ZMC, nonrivalrous, and a closed system, it means that it is possible to achieve post-scarcity for software, provided people are able to contribute software to the system. Because software has the ability to solve problems which impede quality of life and to allieviate suffering, owing to the special agency which it possesses, it would seem especially desirable to contribute to this system. As Eben Moglen so poignantly stated: “The great moral question of the twenty-first century is this: if all knowledge, all culture, all art, all useful information can be costlessly given to everyone at the same price that it is given to anyone; if everyone can have everything, anywhere, all the time, why is it ever moral to exclude anyone?” (Moglen 2001)

Additionally, software systems possess a peculiar kind of immortality. Whereas people today have a life expectancy generally under 100 years, there is no predefined life expectancy for software. It may persist, like books, for thousands of years or more. Therefore, whatever acumen it possesses can be continually improved, and proven through regression testing. This means that personal assistants of the future have the ability to retain constantly increasing quantities of knowledge and abilities. Their knowledge does not repeatedly die off with each subsequent generation.

Moreover, extreme rates of technological progression have not been limited to the software field. In tandem, computer hardware has been developing at an accelerated pace. So not only is software able to solve increasingly difficult problems, but it is able to do so faster and faster.

But software post-scarcity would be useless without hardware post-scarcity. Fortunately, computer technology has become commodified to the point that one can purchase brand new a usable desktop computer, the Raspberry Pi Zero W, for as little as $3.14. Although an extreme example, the price of computers generally has fallen drastically, even more so if you count the various embedded devices, like TV boxes.

Given these three architectural properties of Free/Libre software: ZMC, nonrivalry and being a closed system, combined with the low cost of modern computers, the opportunity afforded through the creation of Free/Libre software that can solve significant societal problems is too great to ignore. The ability to service everyone through the creation of a single system compounds and magnifies the motivation to do so.

There is of course one major catch, and that is, the software can become available to anyone at the same price as it available for anyone. Once written, anyone may have it. But it still needs to be written. Funding the development of Free/Libre software has been an issue, just as coordinating contributors.

Motivation for Conglomerating Software

Given the force-multiplication that one achieves through Free/Libre software for the aforementioned reasons, we contend that it is imperative to advance the capabilities of the Free/Libre software as far as practical. The question then becomes how does one achieve this?

The first and major goal of the FRDCSA project, 18 years and running, has been to help to advance the capabilities of Free/Libre software in order to help provide for better security and quality of life for all living creatures. A major assumption, of course, is that FLOSS artificial intelligence, engineered correctly and with unlimited redistribution, satisfices this goal. To avoid polemics, the project is concerned only with implementing a restricted form of weak AI. The approach, motivated by algorithmic information theory and information-theoretic computational complexity of metamathematics, has two prongs - to develop an increasingly complete theorem proving system and library (called Formalized Research Database (FRD)), and to develop an increasingly complete collection of practical software (Cluster, Study and Apply (CSA)).

The Curry-Howard isomorphism establishes a correspondence between programs and proof systems, such that theoretical results applying to each are transferrable to the other. Progress thus moves along what has been described as a dual track of theory and practice.

Thus, the FRD (Formalized Research Database), which corresponds to the theorem proving side of the Curry-Howard Isomorphism, constitutes what is referred to in the post-Gödel era as a relativized or reformed Hilbert's program. The FRD can never be fully completed, but can, by engineering a sequence of iterated Gödelian extensions of logic - each more complete than the previous - decide in the limit all problems of arithmetic (that are not absolutely undecidable). The idea was to the author's knowledge first described in Alan Turing's inaccessible 1939 thesis, that of creating a sequence of logics, each more complete than the previous, based upon the assumption of the existence of increasingly large constructible ordinals (Turing 1939). Another account (Franzén 2004) goes further to suggest that every for all arithmetic propositions there exists some iterated Gödelian extension that decides it, that is, by repeated addition of an axiom stating the consistency of the previous axiomatic system.

The CSA (Cluster, Study and Apply), which corresponding the program side of the Curry-Howard Isomorphism, consists of a set of programs for building and packaging most or all known freely available software systems and datasets. It is characterized by an advanced system of software location, acquisition, analysis, building and packaging, and redistributing.

Together, the FRDCSA attempts to improve the solution space of programming systems by the collection and integration of a wide variety of applicable tools. Engineering the FRDCSA can be likened to creating a tool shop, with the intuition that it is provident to collect and prepare tools for use before problems are encountered, thus sparing the possible failure of a last-minute search for the proper tool under time pressure.

Effectiveness of the Solution Concept

Let us go into more detail about why this solution concept is effective. How does one in general improve the capabilities of software systems? Gödel's second incompleteness theorem effectively rules out a single program as being sufficient for "Artificial Intelligence". There will always be problems that it cannot solve.

Moreover, Chaitin further established that the scope of incompleteness, rather than be limited to a few propositions such as self-consistency, it was a pervasive limitation. He did this using Algorithmic Information Theory (AIT). Basically, he showed that "there are circumstances in which one only gets out of a set of axioms what one puts in, and in which it is possible to reason in the following manner. If a set of theorems constitutes t bits of information, and a set of axioms contains less than t bits of information, then it is impossible to deduce these theorems from these axioms." (Chaitin 1974).

Proof of Solution Concept To demonstate this, let TR be the set of all total-recursive functions, and let PR be the set of all partial-recursive functions. Let S be an undecidable logic, and let L_S be the class of formulae in the language of S. Let G be a partial-recursive bijection from formulae in L_S to integers, i.e. $G \in PR \wedge G : L_S \mapsto N$, and let $Th(S)$ be the set of true formula in L, i.e. $\phi : S \models \phi$. Let $GTh(S, G)$ be the Gödel numbers of valid formula in S, i.e. $G(\phi) : \phi \in Th(S)$. Let $Enum(p \in TR, C)$ be the set of integers which are enumerated by the function p, and let $Len(p)$ be the number of symbols in the definition of p. Let the Kolmogorov complexity, $K(p \in TR, C)$, be the size of the smallest recursive function that enumerates the same integer sequence as p, i.e. $\mu.len(q)(q \in TR \wedge Enum(q, U) = Enum(p, C))$.

Definition 0.1. We say p is larger than q iff $K(p, C) > K(q, C)$, written $Larger(p, q)$.

Further, let $D(p, S, L_S, G, C)$ be $G(\phi) : (\phi \in L_S \wedge G(\phi) \in Enum(p, C) \oplus G(\neg\phi) \in Enum(p, C))$.

Definition 0.2. We say p is stronger than q iff $D(q, S, L_S, G, C) \cap GTh(S, G) \subset D(p, S, L_S, G, C) \cap GTh(S, G))$, written $Stronger(p, q)$.

Theorem 1. *For any program p, there exists another program q which is both larger (Kolmogorov-complexity wise) and stronger (proof-theoretically), i.e.* $\forall p \in TR(\exists q \in TR(Larger(q, p) \wedge Stronger(q, p)))$

Proof. By construction: the theory S is undecidable, therefore no program exists which enumerates $GTh(S)$. Hence there exists some $\phi \in Th(S)$ such that p does not enumerate it. Then there must exist a program q which enumerates $\{\phi\}$ and $Enum(p, C)$, since the union of two enumerable sets is also enumerable. Hence $Stronger(q, p)$. $Enum(p, C)$ can further be padded by some finite subset of $GTh(S)$ if necessary to ensure $Larger(q, p)$. $\square$

Theorem 2. *It is not the case that for any program p, there exists another program q which is stronger (proof-theoretically) but* not *larger (Kolmogorov-complexity wise), i.e.* $\neg\forall p \in TR(\exists q \in TR(\neg Larger(q, p) \wedge Stronger(q, p)))$

Proof. For any $p \in TR$, there are only finitely many programs of length less than or equal to $K(p, C)$. Let L be the set of the finitely many distinct sets enumerated by programs of length less than or equal to $K(p, C)$. $\exists l \in L(\neg\exists m \in L(l \subset m))$. In other words, l is a maximal set w.r.t. set inclusion. Let p' be a smallest program that enumerates l. $K(p, C) = K(p', C)$. Therefore, no program exists of less than or equal in length to $K(p', C)$ that enumerates a proper superset of l, i.e. $\neg\exists q \in TR(Stronger(q, p') \vee \neg Larger(q, p'))$. $\square$

This shows that a necessary but insufficient condition for increased abilities is that program lengths of any infinite sequence of increasingly powerful programs is bounded below by some monotonic increasing function. Unfortunately, this holds true for *any* infinite sequence of programs. Proving stronger constraints is one aim of the project.

Refutation of Static Seed AI I will mention in passing the concept of seed AI. The concept of seed AI has often been used to argue against the FRDCSA thesis, suggesting that it is unnecessary to collect software, and so therefore it deserves mentioning why seed AI is not sufficient to achieve an endlessly self-improving AI. Seed AI is the idea that there is some program which rewrites itself such that it is able to improves its problem-solving abilities, and it does so ad infinitum, continually rewriting itself and solving more problems. Let static seed AI denote a program without external input:

Theorem 3. *Static seed AI, the idea that a program can rewrite itself to solve an ever increasing set of problems, is a transitive closure violation.*

Proof. Suppose that a static seed AI, program p, cannot decide ϕ, but p then rewrites itself into q which decides ϕ. Then, program p has decided ϕ, contradiction. $\square$

Combined with Chaitin's result it is clear that there is an information-theoretic limitation preventing this concept from succeeding.

Let dynamic seed AI denote a total-recursive function with external input. For instance, a web spider which reads books and papers. The author is not aware of any results regarding dynamic seed AI, but has a vague intuition that a relativized version of the same transitive closure violation exists. However, it is important to rule this out.

Another related concept which might be confused with static seed AI is that of Gödel Machines. These are systems that rewrite themselves whenever they can prove that their modifications improve the efficiency or expand the current (not potential) proof-theoretic closure of the program. However the theory of Gödel Machines does not suppose that they will exceed their transitive closure.

Another common misconception is that the FRDCSA is impossible due to Gödel's second incompleteness theorem. But it is precisely this theorem which is the cornerstone of the project.

Overview of Collection Projects

ByteLibrary The ByteLibrary project is just the latest incarnation of the central project goal of increasing software complexity through the collection and interrelation of existing software combined with the development of special purpose software to harness the collected software.

ByteLibrary is a website where users may submit what we call metasites. These are sites which contain lists of projects or programs. Many such sites exist on the web. Often they are found on academic research websites. They might contain a long list of the programs that have been implemented.

Someone once asked, "what are you going to do with all these programs you've collected?" The relevant point is that there was some need, some problem, which drove the authors to write their programs. They wrote it because whatever tools they originally possessed did not adequately do the job. Therefore, we would like to identify and record whatever motivation the author had in writing their software. Ideally with a formal definition, but an English gloss can contribute meaningful information to the system, for instance, doing keyword or terminological extraction. We collect the software, and then prepare a wrapper that allows us to export problems to the solvers and return the answer to our system, creating what might be thought of as a blackboard architecture of solver agents. By indexing the software, and the problems it purports to solve, establishing a matchmaker service no doubt powered by additional software we collect and/or write, we can increase our solution space.

The ByteLibrary system takes the metasite list, and requests that curators approve and reject each particular URL as containing information relevant to their curation goals. Sites deemed relevant to at least one curator are then automatically crawled using a focused crawler which searches a few links deep for urls that appear to contain software. The entire list of sites, site summaries, and software URLs is then presented to the user. If there is version information a technique is applied which selects the latest version of the software, since at present our storage space is not large enough to accomodate all versions of software, although that would be a desirable future goal.

The system then proceeds to add the software, datasets and related information that the user selected into a download queue. Software is then downloaded to the storage and is indexed into the system. At some future point it will be packaged for the system. This is because until it is packaged the capabilities remain outside of convenient access for non-expert users who might lack the know-how to properly build and install the software. By having a computer or a human-computer team package the software, we reduce the amount of effort spent on packaging the software from N packages times M users to simply N packages times O packaging systems. This is further reduced by tools like Alien which translate between package formats.

Other Notable FRDCSA Systems

The Free Life Planner The Free Life Planner is the flagship application of the FRDCSA Project. It is designed to provide for personal life planning, especially targeted towards the disadvantaged. More information is available from (Dougherty 2018) and (Dougherty 2011).

Acknowledgments

I am grateful to Justin Coslor for providing several large hard drives to build ByteLibrary. I am further grateful to Jess Balint for financial support of the project, and for software contributions such as the UniLang Java Client which enabled Alexa integration. As well as Douglas Miles who has contributed large swaths of code and invaluable assistance with Prolog for various aspects of the project. Additionally to my fiancee and my future mother in law for their encouragement and support and my mother for her support throughout the years of this project.

I further wish to thank all of the researchers who have or will have provided references and pointers, those who have made progress in this field, and in particular those who have released software, documentation and papers under freely redistributable or preferably FLOSS licenses, such as the authors of OPTIC. Such licenses make the work of an open source systems integrator much easier.

References

Chaitin, G. J. 1974. Information-theoretic limitations of formal systems. *Journal of the ACM* 21:403–424.

Dougherty, A. J. 2011. Temporal planning and inferencing for personal task management with spse2.

Dougherty, A. J. 2018. The free life planner.

Franzén, T. 2004. Transfinite progressions: A second look at completeness. *The Bulletin of Symbol Logic* 10(3).

Moglen, E. 2001. The dot communist manifesto: How culture became property and what we're going to do about it.

Turing, A. 1939. Systems of logic based on ordinals. In *Proc. London Math. Soc.*, 161–228.