

CREATING A UNIVERSAL INFORMATION LANGUAGE

OSCAR BENDER-STONE

ABSTRACT. Welkin is a formalized programming language to store information. We introduce its use cases and rigorously define its syntax and semantics. From there, we introduce the bootstrap, making Welkin completely self-contained under the meta-theory of Goedel's System T (equi-consistent to Peano Arithmetic).

CONTENTS

1. Introduction	1
1.1. Goals	3
1.2. Organization	3
2. Foundations	4
2.1. Logic of Proofs	4
2.2. Computability	4
2.3. Verifiers	4
3. Information Organization	5
4. Impossible Classes	5
5. Efficient Querying	5
6. The Welkin Language	5
6.1. Syntax	5
6.2. Semantics	5
7. Bootstrap	5
8. Conclusion	5
References	5

1. INTRODUCTION

Undergraduates are taught many things in lecture, books, and papers, but there is one unspoken truth: researchers have extremely *diverse* communities. Each community has their own approaches, their own conferences, and their own formatting. They even have a distinct preference to chalk, dry erase marker, or neither. For a long time, I didn't notice this aspect about research. I started my mathematics journey in secondary education, independently reading papers from set theory, logic, and more. I was fortunate to have vast archives available on the internet, but my ability to *collaborate* in research was limited. This quickly changed in college[TODO: maybe make this a stronger sentence?]. Everywhere I went, I saw the multitude of research groups, from those mentioned in lecture to the experiences I worked in to the faculty panels in the COSMOS math club. I worked in several research groups and found how rich and fulfilling each was. This fundamentally changed my perception of research as a whole. I learned that diversity *drives* research, including in the sciences, liberal arts, and many more subjects I have hardly explored.

Because of how diverse research is, these communities are extremely independent and build separate repositories of knowledge. For example, I wrote a poster for a cryptography class, concerning an MIT researcher who created programs of common cryptographic schemes and mathematically proved their correctness. I met this researcher at a conference and discovered that *neither* knew about the other! They provided their

TODO: Create a stronger connection to the fact that the papers I read were ONLINE. This is crucial!

TODO: clean up transitions with examples

own contributions, but I initially assumed that *everyone* contributing to cryptography work on common projects. As another example, at my time in the Budapest Semesters in Mathematics, I adopted a key tool in program verification to find proofs for a combinatorial problem. According to my advisor, that approach had *never* been considered before in his group. The boundaries between these communities isn't so clear, and it seems to take years to begin to *remotely* find them.

The separation of these communities raises a key question: *can* knowledge across disciplines be bridged together?

In addition to this [FIRST CLASS OF PROBLEMS NAME], another major hurdle is truth management. [DISCUSS Problems with truth + corrections from papers don't propagate!] What can be done is addressing *information*, the storage of the *asserted* facts themselves, regardless of truth. As one example, suppose a scientist claims, "X is true about Y". One could debate the veracity of that claim, but what we can say is, "This scientist claims, 'X is true about Y'". Even if we doubt that, we could do: "This claim can be formulated: 'This scientist claims 'X is true about Y''". By using these justifications, stating that a claim is expressible, the *syntactic expression* of the claim can be separated from its *semantic truth value*.¹ I will make this more rigorous in later sections, but this means we can build knowledge bases on top of information systems using flexible extensions.

Information has been extensively studied through *measurements* in Algorithmic Information Theory (AIT). The founding idea of AIT is the Minimum Description Length (MDL) principle, that the best definition for an object is the smallest description that describes it. To formalize this idea, Kolmogorov defined a description as a *program*, and the Kolmogorov complexity of a string as the length of the *smallest program* that computes that string (see [1]). This program is defined via a Turing-computable programming language, and there is a different constant factor depending on the language, but AIT focuses on the asymptotic complexity. A cornerstone of this framework is providing the reason underlying cause of Gödel's incompleteness theorems, Turing's halting problem, Tarski's undefinability of truth, and more: *not all information can be compressed into a finite description*. This view was articulated by Chaitin on information compression. He defined Ω as the probability that a random Turing machine will halt and proved that it cannot be compressed computably. This result is a major theme in AIT, to address the limits of computation.

In addition to addressing these limits, Chaitin's results have profound consequences for the nature of mathematics. He explains:

Mathematics...has infinite complexity, whereas any individual theory would have only finite complexity and could not capture all the richness of the full world of mathematical truth. [2]

Chaitin's claim extends beyond mathematics; the extent of research areas are so vast that the idea of a *single* theory would fail to faithfully reproduce these disciplines. The study

TODO: cite main claims about journals/research communities/etc as needed

TODO: REWORD as needed + merge with discussions on FAIR.

¹One might be worried about a paradox, such as "This claim is expressible: this claim is not expressible." We will avoid this using a clear separation of the overarching metatheory and object theory, with the former being syntactical in nature. To express this separation, we write quotes around the claim itself.

of the areas *themselves* is needed to faithfully represent them. This has been explored in Béziau's field of Universal Logic [3], where the aim is to study *logics* and not a *single* logic. In short, Chaitin's result, and the works in Universal Logic and others, demonstrate that research must be represented *flexibly* as well as faithfully.

The problem with a flexible representation system is precisely *how* to accomplish this. AIT provides asymptotic results on information *measurement*, but does not provide a guide on the fixed representation to use. Chaitin created a LISP variant, designed specifically for the ease of implementation and analysis [4], but this does not address the faithful representations of other languages. Additionally, Universal Logic provides a single definition of a logic, one which can be tedious in exotic logics. Each of these issues underly the importance of *organization* itself, which emerge in the proliferation of general-purpose programming languages.

As a concrete example of organizational challenges, consider a Python program in Listing 1.

```
int main() {
    return 0;
}

def main():
    pass

if __init__ == "__main__":
    pass
```

LISTING 1. Example programs in C (left) and Python (right).

In light of the persistence of organizational issues, another step is required beyond Chaitin's reasoning: we must consider *universal building blocks* themselves. Sticking to Turing machines is sufficient in AIT and studying asymptotics, but the *actual* storage must take place. This is the motivation and driving force of this thesis.

1.1. Goals.

The aim of this thesis is to create a universal information language to standardize *all* formal representations. I call this language **Welkin**, an old German word meaning cloud [5]. This aim will be made more precise in the later sections, where we will formally define a verifier for a programming language.

- **Goal 1:** universality. This language applies to ANY checker. Needs to be extremely flexible towards this goal, so we can ONLY assume, at most, we are getting a TM as an input, or it's somehow programmed.
- **Goal 2:** standardized. Needs to be rigorously and formally specified.
- **Goal 3:** optimal reuse. With respect to some criterion, enable *as much* reuse on information as possible.
- **Goal 4:** efficiency. Checking if we have enough information *given* a database much be efficient!

1.2. Organization.

- Section 2. Foundations: define the meta-theory used + verifiers. Also outline the Trusted Computing Base.
- Section 3. Information Systems: explore information in the context of verifiers. Then synthesize a definition to satisfy Goal 1.
- Section 4. Information Reuse. Develops the optimal informal system (w.r.t to a metric defined in this system) to satisfy Goal 3.

TODO: Explain all claims concretely but concisely, as well as cite other possibilities (e.g., HoTT for logics)! And add anything

TODO: Finish transpilation example! Just show we care about storing the underlying semantics.

- Section 5. Syntax: Go over the simple LL(1) grammar, which is similar to JSON and uses python syntax for modules.
- Section 6. Semantics: Defines information graphs and their correspondence with the optimal informal system in Section 3.
- Section 7. Bootstrap. Fulfill Goal 2 with both the Standard AND the complete bootstrap.
- Section ?. Prototype. Time permitting, develop a prototype to showcase the language, implemented in python with a GUI frontend (Qt) and possibly a hand-made LL(1) parser.
- Section 8. Conclusion. Reviews the work done in the previous sections. Then outlines several possible applications.

2. FOUNDATIONS

Our first step is to define the set of verifiers, a subset of the computable functions. Based on our architecture to separate *query* from *search*, we focus this subset to primitive recursive functions.

To explore verifiers, we need a reliable metatheory, establishing our logical TCB. How can we establish what *reliable* means? [TODO: cite that ZFC, or ZFC + inaccessibles, is common in literature.] However, each of these are *specific theories*, and while ZFC + inaccessibles likely suffices for, e.g., formal verification, what about other subjects? We want to make this *as extensible* as possible, akin to an infinite hierarchy of theories via reflection. So we will need a different approach.

The key problem to reliability is, while we can easily verify if a given input is accepted by a verifier, how do we tell when there is *no* such input? We use a novel criterion developed by Artemov's Logic of Proofs [6]. [TODO: bridge this with selector proofs. Also, make sure to explain WHY the arithmetic hierarchy is enough. We JUST want to explore properties of naturals. We can't do beyond. But we should prove this!]

This is where our metatheory come in. [TODO: explore arithmetic hierarchy briefly? How do we know *which* results on computability are trustworthy? This is ESSENTIAL for the TCB!]

2.1. Logic of Proofs.

2.2. Computability.

2.3. Verifiers.

Given a partial computable function φ , let $L(\varphi)$ be the language recognized by φ .

Definition 2.3.0. An **effective verifier** is a Turing machine that runs in linear time and it accepts an input *must* have read the entire input.

In a refined form of Kleene representability, we show that every RE set corresponds to an effective verifier in an important way.

Lemma 2.3.1. *For every RE set S with recognizer φ , there is an effective verifier V_φ such that $x \in S$ iff there is some trace t that starts with x and $t \in L(V_\varphi)$.*

For the rest of this thesis, all verifiers mentioned will be effective. Note that we will return to practical verifiers, those with realistic constants.

3. INFORMATION ORGANIZATION

A natural question arises with universal formal systems: *which* one do we choose? While we have reflection, what is the criterion for the *base* theory? Can this be done? One loose, but natural, metric is this: *a universal system which stores as many “interesting” proofs as possible*. The motivation behind this metric is to enable effective querying of “good” proofs.

We will show that, under a restricted notion of transformation, there is an optimal universal system. This will form the encoding under Section 6.2 and provide a justification for Welkin as this base theory.

4. IMPOSSIBLE CLASSES

The reason to restrict our transformations is two-fold. First, we need to ensure we can *verify* them efficiently. Determining whether a morphism between two formal systems exist can be reduced to the Halting problem, and is therefore not practical for defining an optimal formal system. Second, if we include those transformations that we *can* effectively check, no optimal formal system exists.

Theorem 4.0. *With respect to the class of all computable transformations that can be computably verified, there is no optimal formal system.*

5. EFFICIENT QUERYING

Instead of making proofs most efficient as is, we want to support finding optimal representations. But we want to do this from an efficiently queryable system, which is the most optimal.

6. THE WELKIN LANGUAGE

6.1. Syntax.

6.2. Semantics.

7. BOOTSTRAP

8. CONCLUSION

REFERENCES

1. Li, M., Vitanyi, P.: An Introduction to Kolmogorov Complexity and Its Applications. Springer Publishing Company, Incorporated (2019)
2. Chaitin, G.: The Limits of Reason. Scientific American. 294, 74–81 (2006). <https://doi.org/10.1038/scientificamerican0306-74>
3. Béziau, J.-Y.: Universal Logic: Evolution of a Project. Logica Universalis. 12, 1–8 (2018). <https://doi.org/10.1007/s11787-018-0194-7>
4. Chaitin, G.: Elegant Lisp Programs. (2003). https://doi.org/10.1007/978-1-4471-0015-7_2
5. Dictionary, O.E.: welkin, n., https://www.oed.com/dictionary/welkin_n
6. Artemov, S.N.: Explicit Provability and Constructive Semantics. Bulletin of Symbolic Logic. 7, 1–36 (2001). <https://doi.org/10.2307/2687821>

DEPARTMENT OF MATHEMATICS, UNIVERSITY OF COLORADO AT BOULDER, BOULDER, CO
Email address: oscar-bender-stone@protonmail.com