# 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 .......... 2
  1.2. Organization .......... 3
2. Foundations .......... 3
  2.1. Logic of Proofs .......... 4
  2.2. Computability .......... 4
  2.3. Verifiers .......... 4
3. Information Systems .......... 4
  3.1. Motivating Examples and Definition .......... 5
  3.2. Constructions and Reflection .......... 6
  3.3. Universality .......... 6
4. Syntax .......... 7
5. Semantics .......... 7
  5.1. Terms .......... 7
6. Bootstrap .......... 7
7. Conclusion .......... 7
References .......... 7

## 1. INTRODUCTION

Research is culiminated through *diverse* communities. At a glance, each community has their own groups, conferences, and communication styles. They even have a distinct preference to chalk, dry erase marker, or neither. But beyond these apparent differences, there are more subtle dimensions, including distinct problems, approaches, and methods. In many cases, communities are specialized to particular topics, with specific terminology and theory. Specialization has promoted progress in multiple fields, including biology, engineering, and the social sciences. In any case, specialized or not, understanding these communities is pivotal to understanding the research they produce, as well as future research directions.

As a result of this diversity, research is fragmented, even between subdisciplines.

Among issues in interdisciplinary research, a key problem is the common storage of knowledge.

To examine the complexity of this knowledge, we can theoretically analyze information through Chaitin's ideas of compression. He explains:

**TODO:** Discuss

**TODO:** Discuss

**TODO:** Rephrase Chaitin's contributions + connection to AIT. Maybe simplify this more?

*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.* [1]

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 of the areas *themselves* is needed to faithfully represent them. This has been explored in Béziau's field of Universal Logic [2], 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.

**TODO:** Explain why we need universal building blocks. A key example is, e.g.,

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 [3], 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.

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

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

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

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

```python
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**, based on the old German word *welkin* meaning cloud [4]. 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 critierion, 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.
  - Want to cover *why* we want to use our system. For minmality, want to start with the least necessary in RE sets. This is exactly outlined in Peirce's Reduction Thesis, with previous artitifical formalisms proving this resolved in [5].
- Section 4. Information Reuse. Develops the optimal informal system (w.r.t to a metric defined in this system) to satisfy Goal 3.
- 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.

**TODO:** Maybe provide as another table *with* descriptions?

## 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 + inaccessibilies 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 $\varphi$, let $L(\varphi)$ be the language recognized by $\varphi$.

**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 $\varphi$, there is an effective verifier $V_\varphi$ such that $x \in S$ iff there is some trace $t$ that starts with $x$ and $t \in L\left(V_\varphi\right)$.*

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 Systems

Now with our meta-theory in Section 2, we can proceed to discuss information systems.

- Now can study the set of *verifiers*. A univeral verifier is a verifier of a UTM. So we only need *one* such verifier. We encode the TM AND the input in question, and then simulate the whole TM.
  - Same problems as before: *how* do we organize this?
  - Instead of asking what information is broadly (we'll come back to that in the overarching semantics), we want to ask, what is information *for verifiers*?
    * Recall translation from before: we *define* things by how they *are checked*. To information on mathematical objects corresponds to *information on how they are checked*.
      . By how, this includes:
        - *what* things are/are not checked.
        - If it speeds it up/is a slow route.

      . We capture this how by defining information *as* a relation between an input and a language *per* a trace.
      . Current idea in a formal setting: *information are traces*.
        - Broadly treat information *as* a relation. Manifested as traces to include source, target, *and* steps inbetween.
        - Problem: this isn't efficient! Come up with non-trivial examples that, while we discern here, doesn't resolve the problem!
        - Also note: may not be *nice* w.r.t. certain systems. E.g., for sequent calculus, composition may not be guaranteed! So we can say a composite exist if *some* steps inbetween do, but this isn't always nice! Not *isomorphic* to sequent calculus proofs (depending on the system)! TODO: cite Strassburger deep inference paper to cite this issue and his approach.

        - Currently: we treat *proofs* as being traces. Need to resolve in the meta-theory!
- Assuming we have a representation for traces (maybe via lambda terms? Review!),

we can look at the space of *all* these traces.

- HOW do we organize them? They are *base* information, but pretty dense.
    - Inefficient with equivalence modulo labeling
    - So we explore information systems based on these bounds:

      (1) explicit = trace(s) <= faster trace 1 <= faster trace 2 <= … <= implicit = statement of problem (x in S)
        * TODO: create diagram that highlights this! Make the outer circle the problem, Halting instance, and then the center is the set of explicit traces. When we union these, then the an explicit trace in one machine is the implicit one for another - maybe show this in a simple lemma?
        * … ensured by Blum's speedup theorem - we will make this result self-contained + define Blum complexity measures!
        * Consider *statement* of problem to be "most implicit" by fiat:

      even if we use a proof search approach (so this may NOT halt) or use more uncomputable means, that still adds "extra" to the search itself.
        * Note that <= here is NOT necesarially implication; we are more thinking about a rough *measure* of being an *exact* trace of a TM
        * Blum [7] says you can always do better (*in general*).
            . Cite example: palindromes! Easy to discuss on efficiency
        * TODO: show that this corresponds to *shorter traces*, fixing the main measure as the size of the trace. Again, to separate *query* from *search*, we do NOT want to involve any aspect of time, so the main other attribute is the space itself.
    - Our goal: how to organize when we have a **space** of things like (1)

  that are *interconnected*?
    - Old approach (diagram!): show that people had *different* ways to tackle the stuff in-between. That's where most theories like (e.g., dependently typed theories, HOL, etc.)
    - New approach: separate *query* from *search*! Unify the previous approaches by foucsing on optimizing the *left side* that we *can* control effectively

### 3.1. **Motivating Examples and Definition.**

We start with simple informal examples to explore the concept of information:

- Statements about the world.
- Taxonomies.
- Mathematical relations.
- More sophisticated: formal theories.

Each of the previous examples suggests a common definition: *information is a relation*. However, we want to express any formalizable kind of information. A binary relation *can* encode any other computable one, but not without clear reasons or connections.

We add this missing component by using triadic relations instead, building off of semiotics and related schools of thought.

**Definition 3.1.0.** An **information system** is a pair $(D, I)$, where:

- $D$ is the **domain**, a finite, computable set of **data** in $\mathbb{N}$
- $I$ is **(partial) information**, a partially computable subset of $D \times D \times D$. This information is **complete** if $I$ is totally computable.

## 3.2. **Constructions and Reflection.**

Let **Info** be the set of all information systems.

A natural construction to include is a system with *indexed information*, akin to indexed families of sets. But we want to have information between information as well. We can think of a disjoint union of systems as the weakest transformation between systems.

**Definition 3.2.1.** Let $\mathcal{S} = \{(D_i, I_i)\}_{\{i \in \mathbb{N}\}}$ be a family of information systems, indexed by a partial computable function. Then the **sum** of $\mathcal{S}$ is $(\biguplus D_i, \biguplus I_i)$. A **transformation** on $\mathcal{S}$ is an information system $(\biguplus D_i, I')$ such that for each $i \in I, I' \cap (D_i \times D_i \times D_i) = I_i$.

As another construction, we can naturally model formal systems by asserting that $I$ is reflexive and transitive in a general sense, formalized by below.

**Definition 3.2.2.** A **formal system** is an information system such that the information relation is reflexive and "transitive in a general sense" (WIP).

This construction is **information compressing**: we can compuably encode information into a different representation and computably decode it back.

Special systems:

- $(\emptyset, \emptyset)$ is the **null information system**
- $(\mathbb{N}, \mathbb{N} \times \mathbb{N} \times \mathbb{N})$ is the **discrete information system**

**Lemma 3.2.3.** *Let $\mathcal{H} \equiv (\mathbf{Info}, \leq)$, where $(D_1, I_1) \leq (D_2, I_2)$ iff $D_1 \subseteq D_2$ and $I_1 \subseteq I_2$. Then $\mathcal{H}$ forms a Heyting Algebra, with bottom element being the null information system and the top element being the discrete one.*

**Theorem 3.2.4.**

- *The discrete information system cannot represent $\mathcal{H}$.*
- *$\mathcal{H}$ can represent any extension of this structure and therefore induces an idempotent operator.*

TODO: Show that a formal system provides a proof system that is a way to *optimize* the search space of information systems.

## 3.3. **Universality.**

**Theorem 3.3.5.** *Every universal formal system induces a framework $\mathbb{F}'$, as the image of the functor $\mathcal{G} : \mathbb{F} \to \mathbb{F}'$, given by $\mathcal{G}(\mathcal{S}) = \left(\mathrm{Image}(G_{\mathcal{S}}), \mathcal{R}_{\mathcal{U}} \cap \mathrm{Image}(G_{\mathcal{S}})^2\right)$. Conversely, every framework induces a universal formal system.*

*Proof.* We must show that $\mathbb{F}'$ is a framework for $\mathbb{F}$. Clearly this is a computable subcategory. To prove $\mathcal{G}$ is an equivalence, notice that $\mathcal{G}$ is full and faithful as a full sub-category of $\mathbb{F}$. Additionally, $\mathcal{G}$ is essentially surjective precisely by construction. This completes the forwards direction.

Conversely, a univeral framework can be formed from a system by creating a computable encoding of the formulas and rules of a system. The family $G$ can then be defined from an equivalence from $\mathbb{F}$ to $\mathbb{F}'$, which can be easily verified to preserve and reflection derivations. $\square$

## 4. Syntax

## 5. Semantics

### 5.1. **Terms.**

We expand upon to anaylze the ASTs generated from Section 4.

## 6. Bootstrap

## 7. Conclusion

## References

1. Chaitin, G.: The Limits of Reason. Scientific American. 294, 74–81 (2006). https://doi.org/10.1038/scientificamerican0306-74
2. Béziau, J.-Y.: Universal Logic: Evolution of a Project. Logica Universalis. 12, 1–8 (2018). https://doi.org/10.1007/s11787-018-0194-7
3. Chaitin, G.: Elegant Lisp Programs. (2003). https://doi.org/10.1007/978-1-4471-0015-7_2
4. Dictionary, O.E.: welkin, n., https://www.oed.com/dictionary/welkin_n
5. Koshkin, S.: Is Peirce's Reduction Thesis Gerrymandered?. Transactions of the Charles S. Peirce Society: A Quaterly Journal in American Philosohpy. 58, 271–300 (2022). https://doi.org/10.2979/csp.2022.a886447
6. Artemov, S.N.: Explicit Provability and Constructive Semantics. Bulletin of Symbolic Logic. 7, 1–36 (2001). https://doi.org/10.2307/2687821
7. Blum, M.: A Machine-Independent Theory of the Complexity of Recursive Functions. J. Acm. 14, 322–336 (1967). https://doi.org/10.1145/321386.321395

Department of Mathematics, University of Colorado at Boulder, Boulder, CO
*Email address:* oscar-bender-stone@protonmail.com