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

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 crytographic schemes and mathematically proved their correctness. I met this researcher at a conference and discovered that *neither* knew about the other! They provided their 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 explored a key tool in program verification to find proofs for a combinatorial problem. According to my advisor, that approach had *never* been considered before. 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 enables a truth checking to be a *flexible extension* to an information system.

Information systems have been extensively studied via *measurements* in Algorithmic Information Theory (AIT). The founding idea of AIT is Minimum Description Length (MDL) principle, that the best definition for an object is the smallest thing that describes it. This is formalized as Kolmogorv complexity of a binary string, the length of the smallest program that computes that string. Being described by a smaller program indicates a string that is easier to describe and thus has less information. But being described by a larger program indicates more *randomness* string and thus more information. A modern figure in this field, Chaitin, describes this relationship as compression. He states this as follows: "A useful theory is a compression of the data; comprehension is compression" [1]. I do not defned the claim that comprehension *is* compression, as that creates similar problems with managing truth. Instead, this is a statement on *representations*, the best we can do with Turing machines as an established notion of computability. [LIST CONTRIBUTIONS OF AIT. WHAT IS GOOD]. [GOOD TRANSITION TO NEXT PARAGRAPH]

Although AIT has major accomplishments with measuring information, this field does not indicate *how* to measure information. [DISCUSS CONNECTION TO ORGANIZATOIN. MAKE THIS A KEY WORD FOR THE WHOLE THESIS!]. The main theorem on this is that any programming language can ressbe used to prove these claims. These translations have not been specifically addressed in the literature, and remain a prolific part in software for actual machines. [DESCRIBE PROBLEMS HERE CONSCISELY.]

Our inquiry can now shift onto a formal representation of information. Formal representations, in general, are sets of strings that are computable, i.e., accepted by a computer. The standard notion of computability is being computed by a Turing machine, and by taking this, we are now *exactly* discussing the problem of interoperability of programming languages. These, too, are extremely distinct and built for different purposes. Semantic preserving and reflecting translations between these, also known as *transpilations*, are incredibly difficult, as they are generally equivalent to the Halting Problem. An alternative approach is to create an Intermediate Representation (IR) that other languages can compile into (*frontends*), and then compiled onto multiple machine architectures (*backends*). The driving standard in industry and research for this purpose is the LLVM compiler project [2]. However, this project faces ongoing

[1]One 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.

challenges, with a brief list including breaking changes and a massive packaging task accross different Operating Systems. One approach that addresses a subset of these problems is MILR [3], another IR to abstract away from the original. Implementing these systems for *general* programs is not the intent of this thesis, and instead, is to *bridge* information *about* programs. Therefore, we are interested in *unifying representations* of these languages.

To demonstrate the difficulty of a unified representation, we provide an example with Python and C Listing 1. [PROVIDE DETAILS ON EXAMPLE] Using LLVM IR would not remove this issue. The fundamental differences in memory management would persist, and while this is possible to some extent, it's not completely seamless. Manual work is generally required. [TODO: what manual work? Why?]

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

```python
def main():
  pass

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

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

Creating bridges beetween formal representations does require a historical change in perspective: *embracing reflection rather than focusing on a single theory*. This train of thought comes from Universal Logic, initiated by Béziau [4]. Previously, during the rapid expansion of foundations in the 20th century, logicians sought the "one true logic", a system to be the basis for all mathematics. Such a system was quickly shown to be impossible by Gödel's incompleteness theorems, with certain results requiring an infinite chain of increasingly more powerful theories. But this was a symptom of a larger problem: translating into the *exact* language of a base logic can be unnatural, just as it is unnatural to represent quotient types in Rocq without an additional theory ontop. To work back in the original logic, a key requirement is *faithfulness*, that isomorphisms in a theory must be reflected, a notion called "ε-representation distance" by Meseguer [5]. However, the researchers surrounding Universal Logic are, too, their own community, and have their own broad defnition of a logic, which is distinct from those in Categorical Logic, Type Theory, and others.

Beyond Universal Logic, a further leap is needed, from the idea a *universal theory* to *universal building blocks*. To support the wide diversity of languages, a spectrum of these building blocks, which we consider to be *information*. This thesis creates a universal information language for this purpose, to express information about *any formal representation*. This includes improving the representation itself!

Our overarching architecture is based on a key idea: separate *cheap queries* from *expensive search*. We develop the entire theory of queries and enable arbitrary extensions to the search prcoedures. This is inspired by and generalizes DPLL(T) [6] for SMT-LIB solvers. In DPLL(T), the goal is to find a proof of a first-order statement into two parts: solve propositional statements in a SAT solver, and solve theory-specific problems with theory solvers.

In addition to this architecture, we establish a small Trusted Computing Base (TCB). In our base logic, uses a a novel technique: Artemov's Logic of Proofs [7]. This establishes our metatheory, which is equi-consistent to [TODO: find this!] (Section 2). [TODO: talk about prototype! Key!]

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 [8]. 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.
- 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.

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 [7]. [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 $t \in L\left(V_\varphi\right)$.*

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 [9] 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 $\mathbf{Info}$ 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* $\mathcal{G}(\mathcal{S}) = \left(\text{Image}(G_{\mathcal{S}}), \mathcal{R}_{\mathcal{U}} \cap \text{Image}(G_{\mathcal{S}})^2\right)$*. 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 sub-category. 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. □

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. Lattner, C., Adve, V.: LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In: Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization. p. 75. IEEE Computer Society, Palo Alto, California (2004)
3. Lattner, C., Amini, M., Bondhugula, U., Cohen, A., Davis, A., Pienaar, J., Riddle, R., Shpeisman, T., Vasilache, N., Zinenko, O.: MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In: 2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). pp. 2–14 (2021)
4. Béziau, J.-Y.: Universal Logic: Evolution of a Project. Logica Universalis. 12, 1–8 (2018). https://doi.org/10.1007/s11787-018-0194-7
5. Meseguer, J.: Twenty years of rewriting logic. The Journal of Logic and Algebraic Programming. 81, 721–781 (2012). https://doi.org/https://doi.org/10.1016/j.jlap.2012.06.003
6. Ganzinger, H., Hagen, G., Nieuwenhuis, R., Oliveras, A., Tinelli, C.: DPLL(T): Fast Decision Procedures. In: Alur, R. and Peled, D.A. (eds.) Computer Aided Verification. pp. 175–188. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
7. Artemov, S.N.: Explicit Provability and Constructive Semantics. Bulletin of Symbolic Logic. 7, 1–36 (2001). https://doi.org/10.2307/2687821
8. Dictionary, O.E.: welkin, n., https://www.oed.com/dictionary/welkin_n
9. 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