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 even their preference of chalk, dry-erase marker, or neither. I didn't notice in my early mathematical journey; I wanted to absorb a wide range of papers, from number theory to set theory. At the time, I was in secondary education and was around few active researchers. CU changed this experience for me and look forward to collaboration. This diversity is fundamental to research as a whole, even in the many subjects I have not deeply explored, including the sciences, liberal arts, and more.

With research being diverse as is, I started to notice how independent communities can be. For example, I wrote a poster for a cryptography class, concerning an MIT research 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* in cryptography. This does depend on the community, but even a key topic like this can have separate communities. 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* research results be bridged together? Can they be written in *one* place for retrieval, similar to how the internet is the standard for global communication? This is more vague for certain disciplines, such as connecting two distinct areas in philosophy, but we can focus on their *representations* instead: binary strings. Specifically, we can consider *formalized representations*, so sets of strings that are computable, i.e., accepted by a computer. By taking computable to mean the standard notion, Turing computable, our inquiry is now exactly about the proliferation of programming languages. These, too, are extremely distinct and built for different purposes. Translations between these, also known as *transpilation*, is incredibly difficult, as this is generally equivalent to the Halting Problem. Even established classes of *terminating* programs can be difficult to transpile between, especially those used in proof assistants.

After examining many attempts at transpilation, I argue that the fundamental problem is *organization*. [Introduce a relevant example. Maybe between two proof assistants? MAYBE mention deep inference?]

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 [1]. Previously, in the twentith 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. 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 [2]. 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!

1.1. **Background on Information Theory.**

Given the desire to *use* information, how do we define it? Information Theory has established several different major trains of thoughts:

- Shannon + entropy:
  - Main founder of information theory
  - Information = "reduced uncertainty"
  - Primarily probabilistic and based on *communication of bits*. Not enough for *semantics*!
  - Takeaway: information is *used* in communication
- Kolomogorov + complexity:

  - Minimum Description Length (MDL): we should describe objects with the smallest description possible.
  - Kolmogorov complexity = length of *smallest* program accepting a string
  - Practical problem: not computable!
  - Bigger problem: *measures* information, but does not *define it*
  - Takeaway: provides a *computational* lens for information
- Scott domains:
  - Introduced information systems, but in the context of programming language semantics.
  - Problem: provides the axioms and key models, but not clearly tied to the other theories. Also divergence from information in other senses, like ontology (OWL, etc.)
- Ontologies:
  - Frameworks: OWL, Conceptual Graphs, etc.
  - Problem: pretty restricted! Most theories are only first-order, so difficult for certain type theories, e.g., dependently typed.
- Synthesis of information theory + past paragraph
  - We want to explore *information* as a universal framework.
    * Major goal: address information *on* information.
      . Motivation: transfer between different things!
        - Logics: proofs!
        - Models: properties!
        - Solvers: techniques and checkpoints! Reduces computation!
        - And more!
  - But what *is* information?
    * So many examples: hard to know where we should go!
      . Algorithmic ideas.
      . Specific formulations in logics.
      . High level properties.
      . Probabilistic information.
    * *Can* start at an indirect approach to ensure we don't

    miss anything: indirectly define things by how they are *checked*. This checker/verifier MUST be a Turing machine (using a standard notion!).
    * Emphasize: finding a certificate is *not* guaranteed, e.g., finding a proof of a theorem in first order logic.
    * To simplify this: we have *verifiers*/*checkers* that take in binary strings called *certificates*. Correct certificates are accepted by the checker, and rejected otherwise.
    * *Use this definition to justify universality!*

1.2. **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 [3]. 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.

- **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!

1.3. **Organization.**

- Section 2. Foundations: define the meta-theory used + verifiers.
- 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

This section develops two major components for this thesis:

- The base metatheory, which defines binary strings (as binary trees) and proofs for computability. We justify why this theory is reliable, based on work from Artemov [4].
- The definition of a verifier, which is a computable function on binary strings.

2.1. **Computability.**

- From intro: we formally represent something by *how we check it*.
  - Initial idea: use computable functions. *Can* associate a checker to any RE set, even if we restrict these checkers, e.g., to specific linear time functions.
  - BUT, we want to include RE sets
    * Need UTMs! Provides a general appartus to explore *any* RE set
    * Clear Completeness Problem: Halting Problem (decide if x in RE set)
      . Logic in Meta theory: we need *proofs* of this. This is our verifier! Maybe still restrict the verifier suitably for *effective* verification.
  - Introduce set of all RE sets and define UTMs simply. Maybe use lambda terms or meta-theory encoding to simplify this?

  - TODO: focus discussion on abstract rewriting system, *with* computability in mind. Simplifies lots of things. Need to be careful about composition - handle carefully!

Definition 2.1.0. A **formal system** is a pair (X, R) where R is an RE subset of $X \times X$.

- From intro: we formally represent something by *how we check it*.
  - Initial idea: use computable functions. *Can* associate a checker to any RE set, even if we restrict these checkers, e.g., to specific linear time functions.
  - BUT, we want to include RE sets
    * Need UTMs! Provides a general appartus to explore *any* RE set
    * Clear Completeness Problem: Halting Problem (decide if x in RE set)
      . Logic in Meta theory: we need *proofs* of this. This is our verifier! Maybe still restrict the verifier suitably for *effective* verification.
  - Introduce set of all RE sets and define UTMs simply. Maybe use lambda terms or meta-theory encoding to simplify this?
- Problem: too many UTMs!
  - Trivial permutations: relabeling, small changes, etc.
  - How to go from one UTM to another? Lots of "bloat" is possible
- Key inquiry: *how to effectively reuse answers to Halting*?
  - Want to separate *queries* (straightforward) from *search* (hard)!
  - Main solution: represent this as *information*. Show that better systems have *better information compression*. This IS our solution to organization.
- Go back to verifier idea: we'll abstractly assume linear time, BUT for Welkin 64, we can impose specific bounds on *steps*. (Or, provide a demo verifier that can then be improved).
  - Want to use the whole input as well: represents that the *whole* input matters for the query. This limits the inputs themselves: we want a definition of a trace that goes from initial state TO accept. In other words, for THAT specific trace, *each* step is **needed**. For reject, we *want* to do so early if needed.
  - Need to encode this into the meta-theory! Maybe have a further subset to make this easier? Can think of this as an *initial* representation.

2.2. **Verifiers.**

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

Definition 2.2.1. 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 2.2.2. *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.

2.3. **Serial Soundness.**

- Need to established for next section!
  - *How* do we "trust" the output of another TM? Absolutely essential!
  - This *is* our approach to the Trusted Computing Base (TCB), at least the theory
  - Basis: HA
    * Stick to constructive proofs so we have an *actual* witness; closely corresponds to completeness theorem
    * Becuse of Artemov's argument, including explosion is fine; will NOT prove everything, luckily
    * Why not PRA? Because we need to know *what* we assume to ensure the theory *is* self serial-verifying! Want a *clear basis* for *clear reflection*. TODO: define serial-verifying and connect back to mm0 [5]
    * Iterate that this is used in mm0 [5], except with focus on constructive proofs (so technically fewer things shown). We'll get back to this in extensions
  - Extensions: a theory T is reliable iff HA proves "T *constructively* proves that T is serial sound"
    * Constructive is key, so we might forbid proof by contradiction altogether (unless proven first constructively. Need to refine!). This should be inductive in of itself!
    * Only mentioning HA so that we have a good basis. Can think of this as just: T constructively proves that T is serial sound. Our quantification for this general T *comes from* HA
    * From Artemov, can't hope to show serial consistentcy from weaker theories. But because the proof is inherently inductive on combinatorial objects, this makes sense. Using a weaker theory is like using a weaker principle of induction! Not the right shape/ complexity for a larger theory (e.g., ZFC)
    * So from there we *can* work with that theory. BUT, always start with HA as our basis

Lemma 2.3.3. *If T is reliable and T proves that T' constructively proves T' is serial sound, then T' is reliable.*

- Idea of criterion for TCBs seems fine, BUT seems like we need to embed these *as* first order theories. Can we instead translate *into* verifiers entirely?
  - Going back to FOL is an *optimization*. Definitely want to streamline this when we can, but may not be natural! E.g., maybe highlight translations from dependently typed theories into simple ones
  - How do we highlight the logical content of verifiers in an efficient way? IF we use dependent types, we want explicit annotations
    * Also require that we have intuitionistic logic, so essentially we do require on partial computability at *some* rate

  - Exploration on possible definitions:
    * M simulates M' iff if for some surjective computable function $f : L(V_M) \to L(V_{M'})$ that sends traces of M to traces of M'
      . If we took this to be bijective, we get language equivalence, which isn't computable. So we need to state this as a Pi formula instead... but how do we reconcile that with our goal to focus on Σ_1^0 formulas? Roughly we want to say there is a proof of this fact, or posit a machine M'' that searches for a *proof* of such a construction existing
      . Motivation: want to *skip* calculating M'. Maybe M is faster?
- Transition: what is information?
  - Goal: traces = SUP information needed. We can prove this, but just refer.
    * *May* be implied by other information, but we want EXACTLY the trace itself.
    * NOTE: think about multiple traces. May be a SUP in a different sense (or we don't distingush traces for a *specific problem* if they show x in S, BUT it may aid in others)
    * Want to build from this idea, that we want to *enhance* our perception

  of information, NOT just as traces.

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 [6] 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 $(\bigcup\!\!\cdot\, D_i, \bigcup\!\!\cdot\, I_i)$. A **transformation** on $\mathcal{S}$ is an information system $(\bigcup\!\!\cdot\, D_i, I')$ 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 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. Béziau, J.-Y.: Universal Logic: Evolution of a Project. Logica Universalis. 12, 1–8 (2018). https://doi.org/10.1007/s11787-018-0194-7
2. 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
3. Dictionary, O.E.: welkin, n., https://www.oed.com/dictionary/welkin_n
4. Artemov, S.: Serial properties, selector proofs and the provability of consistency. Journal of Logic and Computation. 35, exae34 (2024). https://doi.org/10.1093/logcom/exae034
5. Carneiro, M.: Metamath Zero: Designing a Theorem Prover Prover. Presented at the (2020)
6. 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