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 metatheory of Goedel's System T (equi-consistent to Peano Arithmetic).

CONTENTS

1. INTRODUCTION

- Engineering Research has empowered humanity for centuries.
  - Provide examples (mechanics, steam engine, medicine, transistors, etc).
  - Cite history of research, with STEM in mind. (Maybe mention liberal arts?).
- Research *communities* are extremely diverse.
  - Examples: many concentrations in CS, Math, etc.
  - Discuss pervasive issues in *bridging* research.
    * Different languages, approaches, journals, even subtly distinct definitions!
    * Maybe cite original example of BSM + UDGs, in which combinatorialists operate separately from SMT-LIB, even though both can support the other.
    * Another example: My Cryptographic professor and a CS researcher that has made a cryptographic tool are not aware of each other!
  - My realization: lots of *potential* for collaboration, BUT can be difficult. Cite Madhusdan P. in my first meeting with him, in which he said that PL communities are separate for a reason.

        * Also cite work on Universal Logic; discuss history of not a *single* logic prevailing, but instead many. Same thing with structures, computational models, and other mathematical objects.
        * Mention Meseguer's work with rewriting logic and how representations can be difficult! Also outline potential missing elements in rewriting logic.
- Develop the idea of a *framework*: shift from a *universal theory* to *universal "building blocks"*
    - Original idea (pre-20th century): there is a *single* theory/object we must mold everything else into. (Draw a picture!)
    - Shaken up by the foundational crises in the 20th century, but also in *every* field in different ways. Physics with quantum mechanics, biology with diversity of organisms, etc.
    - Highlight: communities NEED to keep their notions! This is a *strength* - people have done great things with this philosophy!
    - New idea: *every* formal object is *made* of the same content, but in different ways.
        * This allows communities to shape their own notions *differently*, but this can be compared precisely because they are made of the *same* things. Highlight an analogy with atoms.
        * My aim: show that *information* is precisely this building block, restricted to a computable setting.
- Review past results in Information theory. Possibly expand upon in a separate section.
    - Main theories:
        * 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!
            . Problem: divergence from . Also divergence from information in other senses, like ontology (OWL, etc.)
        * Ontologies:
            . Frameworks: OWL, Conceptual Graphs, etc.
            . Problem: pretty restricted! Might only be first-order.

- 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?
    * We'll exclude less tangible notions, like perception or emotions.
    * Ultimately, even in an informal setting,

    will need to store (tangible) information as a computable string!
    * So many examples: hard to know where we should go!
      . Algorithmic ideas.
      . Specific formulations in logics.
      . High level properties.
      . Probabilistic information.
    * Need to do some detective work - that's part of this thesis!
    * *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!*
- Aim of thesis: create a universal information language to store information in a standardized way.
  - 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!
- Organization (Maybe provide as another table *with* descriptions?)
  - 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 [1].
- 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?
  - 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*.
  - 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. 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.

Definition 2.1.0. An **effective verifier** is a Turing machine that runs in linear time and it accepts an input *must* consume the entire input.

Lemma 2.1.1. *Given an RE set S and recognizer φ, there is an effective verifier V_φ such that $x \in S$ iff there is some trace φ that starts with x with $\varphi x \in S\left(V_\varphi\right)$.*

- Note: this lemma is very close to Kleene representability

For the rest of this thesis, all verifiers mentioned will be effective.

2.2. **Metatheory.**

To formally define computatbility, we require a metatheory $\mathcal{T}$ such that:

1) $\mathcal{T}$ is equivalent to an established theory.
2) $\mathcal{T}$ is reflective: it can prove properties about itself.
3) $\mathcal{T}$ is straightforward to define.
4) $\mathcal{T}$ proves only true properties about computable functions.
5) $\mathcal{T}$ has has efficient proof checking.

The last condition is not strictly necessary, but it does aid in verifying the bootstrap in Section 6.

We could use **Zermelo Frankel Set Theory (ZF)** or **Peano Arithmetic (HA)** directly, as well established first-order theories, but they have two problems. First, defining first-order logic is tedious, specifically free and bound variables. Second, recursively enumerable functions are *encoded* into the theory, rather than being first class citizens. Computable functions are more naturally expressed in type theories, but partial functions are secondary and are awkward to define. By interpreting proofs as programs, under the Curry-Howard correspondence, non-terminating functions translate into proofs of inconsistency. Moreover, in more expressive type theories, like those with dependent types, proof checking has an extreme time complexity.

Our solution to these issues is to build on Feferman's framework on explicit mathematics [2]. His work builds on two key ideas. First, separating partial functions from proofs is useful. Second, presenting a theory of computable functions is simpler with combinatory logic, which was specifically developed to remove involved calculations with variables. This is easier still using illative combinatory logic, which has useful logical constants (see [3]).

We will build on Feferman's framework and present *both* levels entirely with combinators. The main component of this section is proving that this theory is equivalent to **Heyting Arithmetic (HA)**. Additionally, we build on Artemov's Logic of Proofs [4] for quantification, augmenting equality on terms with proof certificates. This enables us to discuss programs *and* their proofs in the same logic, while being simpler than dependent types. Finally, we present our system with Hilbert proof system, which favors many axims and few rules of inference presents the logic with many axioms and few rules of inference. This enables the system to avoid contexts, which pose similar challenges as variables.

Definition 2.2.2. We define **Illative Combinatory Arithmetic (ICA)** as follows.

- The **language** $\mathcal{L}_{\mathrm{ICA}}$ consists of:
  - **Constants**: $\bot \mid 0 \mid \mathrm{nat}$
  - **Consequence Relation**: $\vdash$
  - **Equality**: $=$
  - **Base Combinators**: $\mathrm{K} \mid \mathrm{S}$
  - **Auxilary Combinators**: $\mathrm{I} \mid \mathrm{id} \mid \mathrm{B} \mid \mathrm{C} \mid \mathrm{swap} \mid \mathrm{bop}$
  - **Pairing**: $\mathrm{pair} \mid \mathrm{fst} \mid \mathrm{snd}$
  - **Connectives**: $\mathrm{if} \mid \mathrm{join} \mid \mathrm{meet} \mid \mathrm{Imp} \mid \mathrm{A}$
- **Terms** are defined recursively:
- We add useful notation, where X, Y, F, G are terms:
  - $\mathrm{id} \equiv I \equiv SKK$
  - $\mathrm{swap} \equiv C \equiv SBBS$.
  - $F(GX) = BFGX$, where $B \equiv S(KS)K$.
  - **Phoenix**: $\mathrm{bop} \equiv B(BS)B$
  - $X \to Y \equiv \mathrm{Imp}\, X\, Y$.
  - $X \to_A Y \equiv S(B(X \to Y))$.
  - $(X, Y) \equiv \mathrm{pair}\, X\, Y$.
  - $(X) \equiv X$.
- Two sets of axioms called **computational** and **logical**.
  - **Propositional Logic**:
    - $\vdash X \to (Y \to Z)$
    - $\vdash (X \to (Y \to Z)) \to ((X \to Y) \to (Y \to Z))$
  - **Base Combinators:**
    - $\vdash KXY = X$
    - $\vdash SXYZ = XZ(YZ)$
  - **Equality:**
    - **Symmetry:** $\vdash X = Y \to Y = X$
    - **Transitivity:** $\vdash X = Y \to ((Y = Z) \to X = Z)$
    - **Congruence:** $\vdash X = Y \to ZX = ZY$
    - **Application:** $\vdash X = Y \to XZ = YZ$
  - **Quantifiers:**
    - $\vdash A(K(AX)) \to_A X$
    - $\vdash X \to A(KX)$
- One rule of inference called **Modus Ponens**: $\vdash X$ and $\vdash X \to Y$ implies $\vdash Y$.

2.3. **Verifiers.**

3. Information Systems

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

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 $(\bigcup D_i, \bigcup I_i)$. A **transformation** on $\mathcal{S}$ is an information system $(\bigcup D_i, I')$ 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. 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
2. Feferman, S.: A language and axioms for explicit mathematics. In: Crossley, J.N. (ed.) Algebra and Logic. pp. 87–139. Springer Berlin Heidelberg, Berlin, Heidelberg (1975)
3. Czajka, Ł.: A Semantic Approach to Illative Combinatory Logic. Presented at the (2011)
4. 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