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. **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, generalizing equality on terms. 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 2.1.0. 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:**
    - $\vdash KXY = X$
    - $\vdash SXYZ = SXZ$
  - **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 $\vdash Y$.

2.2. **Verifiers.**

3. Formal Reasoning

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

3.1. **Formal Systems.**

To bridge information graphs with formal reasoning, We must first define formal systems generally. Our definition is based on three sources:

- Mendelson [5].
- Cook and Reckhow [6] with “formal proof systems”.
- Strassburger [7].

Definition 3.1.0. A **formal system** is a pair $(\mathcal{F}, \mathcal{R})$ consisting of:

- **formulas** $\mathcal{F}$, a decidable set of binary strings.
- a set of **derivation rules** $\mathcal{R} \subseteq \mathcal{F} \times \mathcal{F}$. We define the **derivation relation** $\Rightarrow_{\mathcal{R}}$ to be the reflexive, transitive closure of $\mathcal{R}$. Furthermore, we require that $\Rightarrow_{\mathcal{R}}$ has a polynomial time verifier $V_{\mathcal{R}}$.

Note that the first condition on $\mathcal{F}$ is redundant: reflexivity in $\mathcal{R}$ ensures that each formula can be recognized in polynomial-time.

Definition 3.1.1. Let $\mathcal{S} \equiv (\mathcal{F}, \mathcal{R})$ be a formal system. A **derivation** or **proof** is a sequence of derivation rules. The **category of proofs** $\mathrm{Proof}(\mathcal{S})$ consists of:

- **Objects:** formulas.
- **Morphisms:** proofs between formulas. Concatenation is defined by concatenating sequences.

Remark 3.1.2. Strassburger [7] advocates to *define* a logic as a category. But this is not immediate for certain logics. For instance, in the sequence calculus, composition of two proofs is not uniquely defined. Our definition approaches this by using an artificial, inefficient representation. Strassburger's work on deep inference addresses this problem, but instead of a generalization, we intrepret it as an *optimization*.

Definition 3.1.3. Let $(\mathcal{F}, \mathcal{R})$ be a formal system, and let $\mathcal{T}$ be a set of formulas. The **deductive closure** of $\mathcal{T}$ is $\mathrm{Th}(\mathcal{T}) = \{\varphi \in \mathcal{F} \mid \exists \psi \in \mathcal{T}.\psi \vdash_{\mathcal{R}} \varphi\}$. We call $\mathcal{T}$ a **theory** if $\mathcal{T} = \mathrm{Th}(\mathcal{T})$. A set of formulas $\mathcal{A}$ serve as **axioms** for a theory $\mathcal{T}$ if $\mathcal{T} = \mathrm{Th}(\mathcal{A})$.

3.2. **Universal Systems.**

We want to study formal systems in general. A key invariant we want to preserve is *faithful representations*, or the notion of "ε-representatoin distance" from José Meseguer [8]. The idea is that a *good* representation of a mathematical object is one which preserves and reflects isomorphism. We can treat this as *soundness* (isomorphic representations have actually isomorphic objects) and *completeness* (isomorphib objects produce isomorphic representations) of the representation, respectively. To make this precise,we introduce **transformations**, which are mappings between formulas of two systems, and then proceed to **morphisms**, which are structure preserving maps.

Definition 3.2.4. Let $(\mathcal{F}_1, \mathcal{R}_1), (\mathcal{F}_2, \mathcal{R}_2)$ be formal systems. Then a **transformation** $f : (\mathcal{F}_1, \mathcal{R}_1) \to (\mathcal{F}_2, \mathcal{R}_2)$ is a pair (F, R), where $F : \mathcal{F}_1 \to \mathcal{F}_2$ is computable and $R \subseteq \mathcal{R}_1 \times \mathcal{R}_2$ is left-total and if $F(\varphi) = \psi$, then $\varphi R \psi$.

Definition 3.2.5. A **morphism** $f \equiv (F, R)$ is a transformation such that $\Rightarrow_R$ is functional. More explicitly, $\varphi \Rightarrow_{\mathcal{R}_1} \psi \Rightarrow F(\varphi) \Rightarrow_{\mathcal{R}_2} F(\psi)$. An **isomorphism** is an invertible morphism whose inverse is also a morphism.

Definition 3.2.6. The **category of formal systems** $\mathbb{F}$ consists of:

- **Objects:** formal systems.
- **Morphisms:** defined in Definition 3.2.6.

Note that this algebraic structure satisfies reflexivity and existence of composites.

Definition 3.2.7. A **sub-category** $\mathbb{F}'$ of $\mathbb{F}$ consists of a subset of objects and a subset of morphisms. A sub-category is **full** if it removes no morphisms. A **framework** is a subcategory equivalent to $\mathbb{F}$ such that the objects form a decidable set.

Frameworks closely relate to the notion of **universal** formal systems.

Definition 3.2.8. A formal system $\mathcal{U} \equiv (\mathcal{F}_{\mathcal{U}}, \mathcal{R}_{\mathcal{U}})$ is **universal** if there is a computable family of injective functions $G = \{G_{\mathcal{S}} : \mathcal{F}_{\mathcal{S}} \to \mathcal{F}_{\mathcal{U}} \mid \mathcal{S} \equiv (\mathcal{F}, \mathcal{R}) \in \mathbb{F}\}$ over all formal systems such that at each fixed system $\mathcal{S}$ and for all formulas $\varphi, \psi \in \mathcal{F}$, $\varphi \Rightarrow_{\mathcal{R}} \psi \Leftrightarrow G_{\mathcal{S}}(\varphi) \Rightarrow_{\mathcal{R}_{\mathcal{U}}} G_{\mathcal{S}}(\psi)$.

Our motivation for defining universal systems is a property called **reflection**, similar to the one outlined in [8]. That is, universal systems *themselves* can be studied in the context of a single universal system. This enables meta-theoretic reasoning.

Theorem 3.2.9. *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. □

Theorem 3.2.10. *Let $\mathcal{U}$ be a unviersal system. Then for every formal system $\mathcal{S}$, $\mathrm{Proof}(\mathcal{S})$ is equivalent to a subcategory of $\mathrm{Proof}(\mathcal{U})$.*

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
5. Mendelson, E.: Introduction to Mathematical Logic. Chapman & Hall/CRC (2009)

6. Cook, S.A., Reckhow, R.A.: The Relative Efficiency of Propositional Proof Systems. Logic, Automata, and Computational Complexity. (1979)
7. Straßburger, L.: What is a Logic, and What is a Proof?. In: Beziau, J.-Y. (ed.) Logica Universalis. pp. 135–145. Birkhäuser Basel, Basel (2005)
8. 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

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