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 Peano Arithmetic.

1. INTRODUCTION

Humanity produces a colossal amount of data each year. According to the International Data Corporation, there is currently 163 zettabytes of digital data in the world. Given that the average human can read, on average, 200 words per minute, and approximating a word as 8 bytes, this would require *billions* of years ($7.8675 * 10^9$ minutes). Even in restricted areas, such as academa, the amount of data available cannot be consumed individually. On JSTOR alone, there are over 2800 journals, translating to around 12 *million* articles. Taking the average size of an article to be 5000 words, this amounts to *hundreds* of years ($3 * 10^8$ minutes) for a *single* jounral provider. The sheer scale of this data coincides with the trends in increased complexity, with little further predictions rapidly increasing this total.

In an attempt to tame these large data sets, a key concept called *information* emerged. Within modern databases, this is pronounced in the way data is organized and the relations between them. In modern formalizations, this is best represented by Knowledge Graphs, particulrly OWL and John Sowa's Conceptual Graphs. More recently, AI systems are being more deeply integrated with databases, providing an easier way for users to query accross a large amount of websites or resources at a time. However, most formluations miss on the *underlying structure* behind the data, providing a "shadow" through a model instead of the "thing itself". Additionally, research done in AI aims to provide a good "average"; the amount of fake data produced is a concern, as noted in [1].

Analyzing this problem from a theoretical lens, the natural question arises: *why* is there so much information present? Could it be *compressed* into a smaller form? The leading two theories on the matter provide hard limitations on compression, each with their own notion of "information":

- **Shannon entropy:** Claude Shannon founded information theory and defined informatoin as the "reduction of uncertainty", measured in a probablistic setting.
- **Kolmogorov complexity:** Andrey Kolomogorov founded Algorithmic Information Theory, independently connecting Shannon's work to computability.

However, these theories lack a suitable *semantics*. These were expanded upon in Scott domains, which have been succesful in projects like Prolog. Nevertheless *none* of these theories adequaltey explain what *information* is:

- **Knowledge Graphs** *encode* this structure through First Order Logic, but fall apart through contradictions or fallacious axioms. These can be recitied through logics like Relevant logic, but the emphasis is placed on *truth* rather than *structure*. This falis to provide the notion that information can be false.

- **Shannon entropy** and **Kolmogorov complexity** provide a *measure* of information, not an exact notion of information itself. In the former, this is measured with random strings and focuses on binary strings, whereas the latter gets closer to computation yet studies a *minimum size*, not the properties of a Turnig machine witnessing this bound.
- **Scott domains** come close to providing a semantic basis, but ultimately focus on the *hierarchical structure*, and not the *connections between domains*.

This thesis proposes to rexamine a topic in the literature: the notion of *information*. Using the etymology in Latin, "to form", this thesis develops a formalized programming language to organize information.This provides the missing link to the *semantics* of information, more so than labels within Knowledge Graphs or ones commonly used in AI.

1.1. **Goals.**

The aim of this thesis is to create a formalized programing language, called Welkin, derived from an archaic German word meaning "sky, heavens". The key goals outline the defecincies bserved in the formalisms above:

-
-
-

2. FOUNDATIONS

We introduce the base theory needed for this thesis. This theory embodies a unifying concept for formal systems: computatbility. We capture this through a suitable, simple type system, definable in a single page.

We will keep this self-contained; additional references will be provided in each subsection.

2.1. **Base Notions.**

Before continuing, we must introduce some fundamental recursive definitions.

Definition 2.1.0. Define the **language of binary strings** as $\mathcal{L}_B = \{. \, 0 \, 1 \, w\}$:

- **concatenation** "."
- **zero** 0 and **one** 1.

A **binary string** is defined recursively:

- **Base case:** 0 and 1 are binary strings.
- **Recursive step:** if w is a binary string, then so are $w.0$ and $w.1$.

As notation, we will write $w0$ for $w.0$, and similarly, $w1$ denotes $w.1$.

Equality on binary strings is defined recursively:

- **Base case:** $0 = 0$ and $1 = 1$, but $0 \neq 1$
- **Recursive step:** let w, w' be binary strings. Then if $w = w'$, then $w.0 = w'.0$ and $w.1 = w'.1$

Remark 2.1.1. The definition for binary strings, as the remaining recursive definitions, serves as a suitable *uniform* abstraction for data. From a physical viewpoint, we cannot *verify* each finite string, a phenomena related to the notion of "Kripkenstein" [2]. However, we *can* provide the template and is more suitable as a definition, and we presume these definitions are completely contained (i.e., no other rules are allowed). On the other hand, proof checking will be done in an ultra-finitistic setting and is addressed in Section 6.

For simplicity, our primary encoding uses binary. We directly use this in the notion of a variable in the next section.

2.2. **System T.**

The choice for foundations go back to the very foundational crisis of the 20th century. During this time, logicians sought a rigorous underpinning of mathematics, responding to uncertainty in the past with the introduction of the real numbers. The predominant system that remains today is ZFC. We explore an alternative well known in computer science: type theory. Starting in 1932, Alonzo Church introduced his original untyped lambda calculus [3]. However, it was quickly shown to be inconsistent, via the Kleene-Rosser paradox, but was fixed in 1936 with a revision [4]. He then restricted it further in 1940 with simple type theory [5], which is the basis today for most proof assistants. For additional context, please consult [6].

The simply typed lambda calculus on its own is too weak for proofs on computability. The solution is to augment this with induction, via Kurt Gödel's System T [7]. and the entire theory can be defined in a single page. We closely follow the presentation from [8]. Our full definitions are provided in Definition 2.2.2 and Definition 2.2.3.

Definition 2.2.2. The **base language of System T** consists of symbols $L_{BT} = \{01 + * \mathbb{N}\,\mathbb{B}\,x : \to \times \lambda \langle\rangle ( )\}$. The **types** of System T are defined recursively:

- **Base case:** These are called **base types**.
  - $\mathbb{N}$ is a type, called the **natural numbers type**.
  - $\mathbb{B}$ is a type, called the **boolean type**.
- **Recursive step:** let σ and τ be types. Then $\sigma \to \tau$ and $\sigma \times \tau$ are types. Moreover, we set $(\sigma) \equiv \sigma$.

The **(complete) language** is $L_{ST} = L_{BT} \cup \mathbf{Var}$, where $\mathbf{Var}$ consists of the **variables**, symbols x_i^τ for each binary string i (the **index**) and type τ. A recursive definition can be adapted from Definition 2.1.1 and the one above.

Definition 2.2.3. The **terms** of System T are defined recursively.

- **Base case:**
  - Each variable x_i^τ is a term of τ.
- **Recursive step:**
  - If u is a term of U and v is a term of V, then $\langle u, v\rangle$ is a term of $U \times V$.
  - Given a term $\langle u, v\rangle$ of $U \times V$, then u is a term of U and v is a term of V.
  - If t has type τ and f has type $T \to U$, then $f(t)$ has type U.
  - For each variable x_i^T of type T, if $f(s)$ has type U, then $\lambda x_i^T . f(x_i^T)$ has type $T \to U$.

Remark 2.2.4. For notational ease of use, we will add several conventions:

- Variables will be denoted with letter names $a, b, ..., z$ with an implicit index.
- We define new notation with parantheses via recursion (with the base case already set above): let σ, τ, ρ be types.
  - We set products to be **left-associative**: $\sigma \times \tau \times \rho \equiv (\sigma \times \tau) \times \rho$.
  - We add **greater precedence** for $\to$ over $\times$: $\sigma \to \tau \times \rho \equiv (\sigma \to \tau) \times \rho$ and $\sigma \times \tau \to \rho \equiv \sigma \times (\tau \to \rho)$.

2.3. **Serial Consistency.**

As mentioned, System T is closely related to Peano Arithmetic. Specifically, Gödel proved that Peano Arithmetic ($\mathbf{PA}$) is equi-consistent to System T. He did this to base the former on an intuisonistic logic via Heyting Arithmetic ($\mathbf{HA}$), further compounded by a theory of functionals. The proof is extremely technical and out of the scope of this thesis; we refer to [7] for details.

Theorem 2.3.5. *The consistency of the following theories are equivalent:*

- *System T*
- *Heyting Arithmetic*
- *Peano Arithmetic*

Due to Gödel's second incompletness theorem, none of these theories can prove that *any* of them are consistent. However, a weaker property is *is* provable within these systems and represents a revival of Hilbert's program. This was discovered by Sergei Artemov, outlined in multiple papers [9]. We refer to his latest one but recommend the others.

Theorem 2.3.6. $\mathbf{PA}$ *proves that there exists a primitive recursive function (PRF)* s *such that, given a proof of* D*, verifies that it contains no contradictions. We call* s *a* ***selector*** *and say that* $\mathbf{PA}$ *is* ***serial-consistent****.*

Note that Artemov's condition is distinct from the normal definition of consistency, that there is a *single* proof to demonstrate this consistency. Thus, this does *not* contradict Gödel's incompleteness theorems, and in fact underlies an important principle, closely aligning to Artemov's views:

> ***Finitistic consistency relies on inductively verifying proof-checkers as combinatorial objects.***

This is closely tied to an equivalent notion: Σ_1^0 proofs. This involves the arithmetic hierarchy, but in short, this is the set of *partial computable statements* we wish to verify. With Sergei's theorem, we obtian:

Corollary 2.3.7. $\mathbf{PA}$ *is* ***serial***-Σ_1^0-***sound****, which means that there is a PRF* t *that, given a statement of a* Σ_1^0 *statement in* $\mathbf{PA}$*, provides a* $\mathbf{PA}$*-proof.*

With this, we can *definitively claim* which statements are proven correctly in the realm of computatbility. This is key for the reliablity of the method presented in the remaining sections of this paper.

3. Information Systems

We introduce the bulk of this thesis: providing an optimality criterion for an informatoin system and deriving the best one in terms of explicitly comptuable structures. We first introduce structures, followed by their representations. We then prove that the study to convert betweewn representations is RE-complete, ensuring the theory is sufficiently expressive.

3.1. **Structures.**

In FOL, the usual definition of a structure relies on a tuple of finitely many relations and function symbols on a domain. We simplify the definition; this will be expanded upon in Section 5.

Definition 3.1.0. A **structure** is a **bigraph**, namely a tuple (X, T, G), where

- X is a **domain**, a set of binary strings
- T is a tree on X, the **hierarchy** of X. We make this more precise with the following constructors:
  - We add a symbol $\emptyset \notin X$ called the **root**.
  - There is a **parent function** $p : X \to X \cup \{\emptyset\}$ that is surjective. The preimage $p^{-1}(x)$ is the set of **children of** x. this can be encoded by a binary predicate φ_p such that it is functional,
    $\forall x.\forall y_1.\forall y_2.(p(x, y_1) \wedge p(x, y_2) \to y_1 = y_2)$
    and surjective,
    $\forall x.\exists y.p(y, x)$.
- G is a **hypergraph** on X, encoded by a definable binary predicate φ_G in $\mathbf{PA}$.

Any hypergraph can be used to represent a FOL-structure via a disjoint; the latter can then be considered to be an indexed family of relations.

3.2. **Bases For Turing Machines.**

Let $\mathcal{M}$ be the set of all Turing machines. We examine a suitable lattice-structure on this set provide a semi-lattice as a step towards organizational optimality.

Definition. Let $\mathcal{A} \subseteq \mathcal{M}$. We say $\mathcal{A}$ **spans** $\mathcal{M}$ if there is an explicitly computable, surjective $f : \mathcal{M} \to \mathcal{F}(\mathcal{A})$ such that $\mathcal{A} = \{M \mid f(M) = \{M\}\}$. In this case, we call f an **analyzer** of $\mathcal{A}$.

Analogous to group theory, bases enjoy a computable version of the First Isomorphism Theorem.

Theorem. *Suppose $(\mathcal{A}, f)$ is a basis for $\mathcal{M}$. The following hold:*

- *Let ρ be the function that takes Turing machines to the smallest Turing machine M' such that $f(M) = f(M')$. Then $g = f \circ \rho$ is an analyzer for $\mathcal{A}$. We call g the **canonical analyzer** w.r.t the basis.*
- *The inverse $f^{-1} : \mathcal{F}(\mathcal{A}) \to \ker(f)$ is explicitly computable; we call this a **synthesizer**. In particular, so is g^{-1}, which we call the **canonical synthesizer** (w.r.t. the basis).*
- *Define the following operations on Turing machines:*
  - $M_1 \sqcup M_2 = g^{-1}(f(M_1) \cup f(M_2))$
  - $M_1 \sqcap M_2 = g^{-1}(f(M_1) \cap f(M_2))$

*Then $\sqcup$ and $\sqcap$ are explicitly computable, and $(M_1, \sqcup, \sqcap)$ is a semi-lattice. We call the induced partial order $M_1 \sqsubseteq M_2 \Leftrightarrow M_1 = M_1 \sqcap M_2$ a **part-hood** relation, and the system $(\mathcal{M}, \sqcup, \sqcap)$ the **Mereological System** of the basis.*

Proof ■

3.3. **Mereological Rewrite Systems.**

We generalize a basis to include rewrite components. This will be the starting point for discussing the optimality of the semantics.

Definition 3.3.1. Let (A, f) be a basis on $\mathcal{M}$. The **Mereological Category** $\mathcal{C}(A, f)$ is the largest category closed under explicit transformations on $\mathcal{F}(\mathcal{A})$. In detail, it contains:

- Objects: $\mathcal{F}(\mathcal{A})$.
- Morphisms: $\mathrm{Hom}(A_1, A_2) = \mathcal{E}(g^{-1}(A_1), g^{-1}(A_2))$. If $\mathrm{Hom}(A_1, A_2) \neq \varnothing$, then we write $A_1 \to A_2$.

Definition 3.3.2. Definition. A **progress theorem** is a proposition p of the form $\forall A.\exists D_A.\forall B.D_A(B) \Rightarrow B \to A$ where D_A is a family of explicitly computable unary predicates called the **progress predicate of p**. We say a Mereological Category has **progress p** if it satisfies p. Note that A may be free in D_A. We write $\mathrm{Prog}(\mathcal{C})$ for the set of progress theorems satisfied by $\mathcal{C}$.

Definition 3.3.3. A Mereological Category has **universal progress** if the **Universal Progress Theorem (UPT)** holds: for every $A \in \mathcal{A}$, there is a N such that, for every B with N elements, $B \to A$. This ensures the existence of $m : \mathcal{F}(\mathcal{A}) \to \mathbb{N}$, given by $m(A) = \min\{N \mid \forall B, |B| \geq N.B \to A\}$.

Definition 3.3.4. The category of Mereological Categories with universal progress consists of:

- Objects: Mereological Categories.
- Morphisms: $\sigma : \mathcal{C}(A_1, f_1) \to \mathcal{C}(A_2, f_2)$ are the faithful functors.

Our goal is to find the final object in this meta-category above with universal progress.

4. Syntax

5. Semantics

5.1. **Terms.**

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

6. Bootstrap

7. Conclusion

References

1. Romano, A.: Synthetic geospatial data and fake geography: A case study on the implications of AI-derived data in a data-intensive society. Digital Geography and Society. (2025)
2. Loar, B., Kripke, S.A.: Wittgenstein on Rules and Private Language. Noûs. 19, 273 (1985)
3. Church, A.: A Set of Postulates for the Foundation of Logic. Annals of Mathematics. 33, 346 (1932)

4. Church, A.: An Unsolvable Problem of Elementary Number Theory. American Journal of Mathematics. 58, 345 (1936)
5. Church, A.: A formulation of the simple theory of types. Journal of Symbolic Logic. 5, 56–68 (1940)
6. Laan, T.: The evolution of type theory in logic and mathematics. Presented at the (1997)
7. Gödel, K.: Über Eine Bischer Noch Nicht Benützte Erweiterung Des Finiten Standpunktes. Dialectica. 12, 280–287 (1958)
8. Girard, J.-Y., Taylor, P., Lafont, Y.: Proofs and types. Cambridge University Press, USA (1989)
9. 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

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