

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).

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 academia, 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* journal provider. These current estimates, as well as exceedingly large predictions, represents the sheer complexity of digital data.

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, with recent integration with AI systems. The most prominent ones are **ontologies**, particularly OWL, or **knowledge graphs**, e.g., John Sowa’s Conceptual Graphs. More recently, AI systems are being more deeply integrated with databases, providing an easier way for users to query across a large amount of websites or resources at a time. However, most approaches focus on information via a *theory*, connecting data by true relations and facts. The potential for contradictory facts can compromise the reliability of a knowledge base, though there are some promising approaches, including using paraconsistent logics. 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” that face certain limitations:

- **Shannon entropy:** Claude Shannon founded information theory and defined information as the “reduction of uncertainty”, measured in a probabilistic setting. However, this applies to *noisy channels*, such as passing 010 to a receiver (with a probability of success). This is distinct from the *structural* view of information, such as a deterministic total of a data set.
- **Kolmogorov complexity:** Andrey Kolmogorov founded Algorithmic Information Theory, independently connecting Shannon’s work to computability. This is defined on the **Minimum Description Length (MDL)** principle, that the best representation of a string is the smallest one possible. Kolmogorov formalized this idea with Kolmogorov complexity, the size of the smallest Turing machine that accepts a string. While this is more structural in nature

- **Scott domains:** Dana Scott introduced an algebraic structure to represent information and described how information can be consistent with one another. These structures come close to providing a semantic basis, but ultimately focus on the *hierarchies on pieces of information*, and not the *connections between information*.

This thesis proposes to re-examine 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 programming language, called Welkin, originating from the German word *wolke* (cloud) [2]. The following goals resolve the deficiencies observed in the formalisms above:

- **Universality (G1):** we provide a theory to work with *any* partial computable function. Thus, we ensure that checking a certificate is always decidable, and encode appropriately (for set theory, type theories, etc.).
- **Standardization (G2):** the language must be specified by an unambiguous standard and have a reliable **Trusted Computing Base**. To ensure this, the language must have an unambiguous syntax and semantics, as well as a reliable meta-theory.
- **Encoding (G3):** information must be encoded in an optimal yet efficient way (effective linear-time checking). We provide a 64-bit hashing scheme for most implementations, as this ensures enough unique IDs while being efficient on modern hardware.

1.2. Outline.

This thesis is organized linearly, shown in Table 1. Note that this thesis serves as a *high-level*, but precise, guide to the Welkin language. The *precise* specification is the Standard itself, with *minimal* abbreviations and all details explicitly mentioned.

Section	Description
Section 2	Discusses foundations, providing the meta-theory based on Gödel’s System T to more easily encode computability and have a theory equi-consistent to Peano Arithmetic.
Section 3	Defines information and the optimal encoding proven in this thesis. This is optimal w.r.t. to a certain efficiently computable class of encodings.
Section 4	Introduces the LL(1) grammar for Welkin, a graph-based language that naturally encodes lambda terms.
Section 5	Explains the semantics of terms in the theory and finalizes the encoding. This also introduces the 64-bit hashing scheme.
Section 6	Reviews the Welkin Standard and justifies why Welkin is “strong enough” to “encode itself”.
Section 7	Reviews the work in this thesis and provides insights to applications for formal verification and creating custom languages that compile into Welkin.

TABLE 1. Outline of the thesis.

2. FOUNDATIONS

We introduce the base theory needed for this thesis. Our work builds on deep inference, developed by Strassburger [3], and many others. We formally define a formal system and then proceed to show this can be encompassed in a deep inference framework. These sections are closely replicated as steps in the bootstrap (see Section 6).

We will keep this self-contained; additional references will be provided in each subsection. For general notation, we write $:=$ to mean “defined as”.

2.1. Base Notions.

Before continuing, we must introduce some fundamental notions. We introduce **alphabets**, using three columns: the first is the symbol name, in monospace font; the second is the mathematical notation used; and the third is the symbol’s name. See Table 2 for the template. Note that we informally use natural numbers. However, each definition is self-contained. See Remark 2.1.1 for a related discussion. Additionally, sometimes our symbols may be *multiple* tokens; this is addressed in Section 4.

	Symbol	Notation	Name
$\mathcal{A} :=$	s0	s_0	symbol zero
	s1	s_1	symbol one
		\dots	
	sn	s_n	symbol n

TABLE 2. Template for an alphabet A .

Definition 2.1.0. The **binary digits (bits)** are given by:

	Symbol	Notation	Name
Bit :=	0	0	zero
	1	1	one

TABLE 3. The symbols used in bits.

Recursive definitions are given in the form of a **judgement** (Figure 1), consisting of **premises** on top and a **conclusion** on the bottom.

$$\frac{P}{C} J$$

FIGURE 1. Template for a judgement.

Definition 2.1.1. The **language of words** \mathcal{L}_W is provided in Table 4. A **word** $w \in W$ is given by the judgements in Definition 2.1.2.

	Symbol	Notation	Name
$\mathcal{L}_W :=$	Bit	Bit	See Definition 2.1.1
	{}	ε	Empty word
	.	.	Concatenation
	=	=	Equality
	!=	\neq	Inequality

TABLE 4. Language of words

$$\frac{}{\varepsilon \in W} \text{ Empty } \frac{w \in W}{w.0 \in W} \text{ Zero } \frac{w \in W}{w.1 \in W} \text{ One}$$

FIGURE 2. Recursive definition of words.

Remark 2.1.2. 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 phenomenon related to the notion of “Kripkenstein” [4]. However, we *can* provide the template and is more suitable as a definition, and we presume these definitions are completely contained (i.e., binary strings are defined by a finite combination of *only* the rules above). On the other hand, proof checking will be done in an ultra-finitistic setting and is addressed in Section 6.

Natively, our encoding uses binary. But to simplify this notation, we introduce short-hands using two other number systems.

3. INFORMATION SYSTEMS

We now define the universal framework for analyzing information. Our approach describes information as a *relation*. We show that this framework encompasses both ontology and formal systems.

From there, we analyze the notion of *compressing* information in an effectively realizable way. We establish two key constructions and generalize them as *information transformations*. The main result in this section is showing that the best *effectively realizable compression scheme* can be obtained by appealing to Tarski’s fixed-point theorem on the lattice of these transformations.

3.1. Information.

Our motto is this: *information is a relation*. We will consider several examples.

Example 3.1.0. Alice tells Bob that “I have a cat”. This is a relation that relates Alice to some cat. In contrast, “a cat” is not a relation, and therefore not information.

Example 3.1.1. A statement like $2 + 2 = 4$ is information. But it is also *true* information, or *knowledge*. We allow $2 + 2 = 5$ to be information as well, because it asserts *some* relation between $2 + 2$ and 5. A non-example is simply the number 2 or a random binary string 0b010001. Neither of these are relations because they are missing an *explicit connection*. Note that these relations can be unary, such as $2 = 2$.

Example 3.1.2. Information can include quantifiers: “Joe has at least one egg.” This asserts the existence of *some* egg. Similarly, we can . However, we want to include more general notions. For instance, we could have quantifiers on *surfaces*: “This ball is red everywhere”. This is not builtin directly as a first-order quantifier. Additionally, we would like to allow for modal sentences, like “There is necessarily one marble in the bag” or “There is possibly one marble in the basket”. We will consider these generalized connectives into our definition (see Definition 3.1.7).

However, we quickly run into philosophical blockades when we want to *use* information.

Experiment 3.1.3. Suppose a person describes their feelings through a painting. Does this painting *convey* information? Perhaps we can infer some emotions, such as feelings of sadness in a rainy scene or happiness in a cheerful one. How exactly do we *use* or even *store* this information? Is this data *subjectively* information, requiring a person as an observer?

We avoid these ideas by focusing on *formal* information. This can be rigorously defined into two key components: a **hierarchy** and a set of **connections**. Our notion is based on **bigraphs**, a data structure created by Robin Milner [5].

Definition 3.1.4. Information is a **bigraph**, a triple (X, T_X, G_X) where:

- $X = V \cup T$ is the **domain**, where V is the set of **variables** and T is the set of **bound terms**, each of which are countable sets of binary strings.
- T_X is the **place graph** or **hierarchy**, a tree with nodes in $\mathcal{P}(X) \cup \{\perp\}$, where the root is a distinguished element $\perp \notin X$.
- $G_X \subseteq \mathcal{P}(X) \times \mathcal{P}(X) \times \mathcal{P}(X)$ is the **link graph**. We write $(A, B, C) \in G_X$ as $A - B \rightarrow C$. In the case where $B = \emptyset$, we simply write $A \rightarrow C$.

Remark 3.1.5. Our notion of bigraph diverges from Milner [5] in several important ways. Firstly, Milner’s theory focuses around modeling non-deterministic operations in programming languages. Briefly, he considers place graphs which are *forests* (with special regions), and allows for “holes” in the link graph. We simplify his definition by using a tree (with a designated root \perp), and incorporating holes instead as patterns. Secondly, Milner’s approach defines an algebra for bigraphs, as well as dynamic semantics (via *bigraphical reactive systems*). This is not immediately natural for our generalized setting; we will return to this issue in Section 3.3.

We interpret the elements of X as *parts*, and think of the tree T_X as defining a *part-whole* relation. A key design of Welkin is to enable *multiple* notions of part-hood and seamlessly work among these. We provide a motivating example one possible construction.

Example 3.1.6.

- **Physical Composition:** Let $X_1 = \{\text{house, wall, floor}\}$ and suppose wall and floor are parts of house. In this case, parthood means *physical composition*.
- **Classification:** Let $X_2 = \{\text{animal, dog, bird}\}$, and let dog and bird be parts of animal. Parthood is treated as a *taxonomy* among things, specifically animals.
- **Hybrid:** Suppose we want to put X_1 and X_2 above into a knowledge base. One way to distinguish between *physical composition* and *taxonomy* is to introduce new links: *makes* and *isa*, respectively. We take $T_3 = T_1 \cup T_2$ and introduce new relations: $\text{wall} \xrightarrow{\text{makes}} \text{floor} \xrightarrow{\text{makes}} \text{house}$ $\text{dog} \xrightarrow{\text{isa}} \text{animal}$, $\text{bird} \xrightarrow{\text{isa}} \text{animal}$, respectively.

3.2. Formal Systems.

We must first define a general class of formal systems. Considering the diversity of formal reasoning, we need a *static* and *dynamic* component. These have interpretations in **logic** and **computability**, outlined in Table 5.

We will focus on the *logical* interpretation in our terminology, but note that this is intentionally *general* to encompass any other interpretation.

Feature	Logic	Computation
Static	Formulas	States
Dynamic	Inference	Transitions

TABLE 5. Features needed in a general formal system.

Given this criterion, we define a formal system as follows, loosely based on the definition provided by Strassburger [6].

Definition 3.2.7. A **formal system** is a pair $\mathcal{S} \equiv (\mathcal{F}, \mathcal{R})$ 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 to be the reflexive, transitive closure of \mathcal{R} . Furthermore, we require that \Rightarrow is computable in polynomial time.

A **sentence** is a closed derivation, and a **theory** is a set of sentences in \mathcal{F} .

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

3.3. Information Transformations.

3.4. Optimal Compression Scheme.

4. SYNTAX

5. SEMANTICS

5.1. Terms.

We expand upon to analyze 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. Dictionary, O.E.: welkin, n., https://www.oed.com/dictionary/welkin_n
3. Strassburger, L.: From Deep Inference to Proof Nets via Cut Elimination. *Journal of Logic and Computation*. 21, 589–624 (2011). <https://doi.org/10.1093/logcom/exp047>
4. Loar, B., Kripke, S.A.: Wittgenstein on Rules and Private Language. *Noûs*. 19, 273 (1985)
5. Milner, R.: Bigrphical Reactive Systems. In: Larsen, K.G. and Nielsen, M. (eds.) *CONCUR 2001 — Concurrency Theory*. pp. 16–35. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
6. 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)

DEPARTMENT OF MATHEMATICS, UNIVERSITY OF COLORADO AT BOULDER, BOULDER, CO
Email address: oscar-bender-stone@protonmail.com