CREATING A UNIVERSAL INFORMATION LANGUAGE

OSCAR BENDER-STONE

Abstract. Welkin is a formalized information language. We introduce its use cases and rigorously define its syntax and semantics. From there, we introduce the bootstrap, making Welkin completely self-contained. [TODO: determine how to phrase soundness and incompleteness. Should we include these?].

Contents

1. Introduction

[TODO[MEDIUM]: explain emphasis on information in thesis. Knowledge is important to mention, but **why** information as the focus?]

[TODO: complete general discussion on IM and KM in the first paragraph. The discussions into specific fields is too dense for this intro! Also, emphasize what we do compared to other solutions. Doing this *instead* of specific fields will help!

Current outline: - RDF: lacks named graphs, so reification is difficult. Reification is **built into** Welkin String literals are also reified, and unit IDs and strings are the same, with the latter enabling any character ID more naturally.]

Information Management (IM) is an open area of research as a result of the depth and breadth of disciplines.

In addition to domain specific proposals, there are approaches for general IM which still fail to resolve all issues. One prominent example is Burgin's General Theory of Information [Bur09] that comprehensively includes many separate areas for IM, including the complexity-based Algorithmic Information Theory, through a free parameter called an "infological system", which encompasses domain specific terminology and concepts. In contrast to other approaches, Burgin's generalized theory is flexible and enables greater coverage of different kinds of information [Mik23]. Despite this coverage Burgin does not closely tie the free parameter with his formal analysis of Algorithmic Information Theory, making it unclear how to use this in a practical implementation. Burgin's Theory of Information, along with broad proposals, have severe shortcomings, highlighting major obstacles for IM.

This thesis introduces a language to resolve these issues for both IM and KM. I call this language **Welkin**, based on an old German word meaning cloud [Dic25]. The core result of this thesis is proving that Welkin fulfills three goals: it is **universal**, **scalable**, and **standardized**. For details, see Table 1. The core idea is to generalize Burgin's free parameter and enable arbitrary representations in the theory, controlled by a computable system. A representation is represented a triple: a **sign** represents a **referent** within a **context**. This generalizes RDF triples and relations by allowing the context to incorporate scope and be a general, partial computable operator.[1] Moreover, to address queries on the validity of truth, we use a relative notion that includes a context managed by a formal system, whose ideas are independently developed from [McC93] are enhanced with representations. Truth can then be determined on an individual basis, providing flexibility to any discipline. The focus then shifts to the usefulness of representaitons based on a topological notion of how "foldable" a structure is, which we call **coherency**. This approach is inspired by coherentism, a philosophical position that states truth is determined in comparison to other truths. [Bra23]. We incorporate ideas from coherentism to identify which representations identify their corresponding objects, and we define information as an invariant under these coherent representations. We include definitions on a *working* basis as what is most practical, not an epistemological stance that can be further clarified in truth systems. Additionally, we keep the theory as simple as possible to make scalability and standardization straight-forward. Furthermore, we enhance standardization using a finite variant of the language, providing a self-contained, small area of trusted software and hardware needed, or Trusted Computing Base in the programming languages literature [Rus84].

[1]There are similarities with this triple and Peirce's semiotics, or the study of the relationship between a symbol, the object it represents, and the interpreter or interpretation that provides it that meaning [Atk23]. Our notion is different in that contexts general interpretant.

Goal 1	**Universality**	The language must enable any user created parameters, whose symbolic representation is accepted a computable function. Every computable function must be definable in the language.
Goal 2	**Scalability**	The database must appropriately scale to broad representations of information. Local queries must be efficient. Certificates must be available to prove cases where optimal representations have been achieved.
Goal 3	**Standardization**	The language needs a rigorous and formal specification. Moreover, the bootstrap must be formalized, as well as an abstract machine model. The grammar and bootstrap must be fixed to ensure complete forwards and backwards compatibility. Certificates must be reliably checked and rely on a low level of trust, or a small Trusted Computing Base.

TABLE 1. Goals for the Welkin language.

This thesis is organized according to Table 2.

Section 2	**Rationale**	Introduces Welkin at a high level, with guiding examples.
Section 3	**Foundations**	Introduces the core theory behind Welkin, which is the starting point to specify the user-facing language.
Section 4	**Syntax**	Provides the grammar and proof that it is unambiguous.
Section 5	**Semantics**	Explains how ASTs are validated and processed. Develops representations and coherency, and connects these to a working definition of information.
Section 6	**Information Organization**	Develops a Greedy algorithm to locally optimize information. Creates a certificate that demonstrates when a representation is optimal relative to the current information database.
Section 7	**Conclusion**	Concludes with possible applications, particularly in programming languages and broader academic knowledge management.

TABLE 2. Organization for the thesis.

2. RATIONALE

In this section, we justify the design of Welkin.

2.1. **Information Content.**

The main purpose of an information base is to store information and enable user queries based on the available information. To make this idea precise, we dissect the approach taken by Algorithmic Information Theory, specifically through a well known book by Li and Vitányi [LV19]. Their book focuses on the *information content* of a description, which they summarize as follows:

[TODO[SMALL] still resolve how pages should be mentioned!] [TODO[SMALL] determine if this quote is needed. Maybe it does help with readability]

> *We require both an agreed-upon universal description method and an agreed-upon mechanism to produce the object from its alleged description. This would appear*

> *to make the information content of an object depend on whether it is particularly favored by the description method we have selected. By 'favor' we mean to produce short descriptions in terms of bits.*
>
> — [LV19], pages 101-102

They express this idea with an enumeration D from *objects* to strings called *descriptions*. To ensure that their measure is *minimal* and can be mechanized, D is a partial computable function. The authors proceed define the *information content* of a string through Kolmogorov complexity, the size of the smallest description that accepts an object, or in other words, the smallest program that accepts a string. From there, they prove multiple foundational results for Algorithmic Information Theory, including that Kolmogorov complexity is uncomputable. This quantity can be approximated by several means, which is closely involved to compression algorithms.

However, Li and Vitányi's approach does not generally reflect the ways people disseminate and create new information. The term "object" is a vague term that is vastly different between disciplines and can be difficult to model for entities. For example, consider a dynamic biological systems. In an evolving system, what is the "boundary" of the object? Another issue is well known in the literature as the Symbol Grounding Problem, formulated by Harnard [Har90]. As an example, Harnard considers a person expecting to learn Chinese as their first language with *only* a Chinese dictionary. How does the person ground their symbols in concrete meanings? An information base cannot practically store the denotations of a word, such as storing animals, so how *original meaning* is obtained is unclear.

Several solutions have been proposed to resolve the Symbol Grounding Problem. In some cases, it is nebulous what "store" means for certain concepts, though as abstract ideas. The problem even arises when modeled with *solely* formal entities. For instance, Liu [Liu25] models symbol grounding as axioms in a general deductive system, and uses Gödelian-based diagonalization argument to show that the grounding predicate, indicating which symbols are grounded, is undefinable in a single system. In a related work, symbol grounding can be modeled directly through Kolmogorov complexity [Liu25], demonstrating the same impossibility of completely grounding a system of symbols through a fixed grounding set. These results demonstrate that we need a looser approach, less fixated on philosophical condrums and more oriented to *practical* considerations.

2.2. **Units.**

[TODO[SMALL]: maybe clarify how powerful deduction is? That's my point here, that we don't have to check if a property holds *if* we use a theorem instead, or we use different set of conditions to get a certain property.]

Despite the presence of the Symbol Grounding Problem, we emphasize that an information base is a *tool*, which is useful when fully mechanized for *communication*, not to resolve philosophical inquiries on the existence or absence of things or abilities. Information itself is used for *predictions*: a person that translates the sentence "It will rain today" in Chinese to convey a semantic property of the world, that there will be rain. This scales to larger examples, with major theorems providing even more refined or general properties *given* a set of assumptions. Note that this is different form Shanon's seminal work on Information Theory, in which methods are found to convey the *exact* bits of strings in noisy channels. Because communication *itself* does not carry the physical entities, relationships are key to effectively conveying ideas. A recent work bridges this gap with Shannon's work to express meaning through finite models in first-order logic [Liu25], so that two strings are considered equivalent if they are that are provably equivalent as first order sentences are in fact equal, regardless if the strings have distinct bits.

[TODO[MEDIUM]: go back and ensure this explanation makes sense!]

Taking inspiration from [Las+24], this thesis completely generalizes their approach using a notion of *handles*. Handles provide a dual relationship between a user and information base through a *representation*, a **sign** that represents a **referent**. The user best determines how to express handles, and in turn, the information base then uses the available presentation to process them. They are defined by how they are *not* restricted; this idea is directly inspired from Fine's idea of arbitrary objects [Fin85], to explain such things as why theorems about triangles can be proven with a single *abstracted* triangles. information base *itself* is not in charge with considering how to store or retrieve certain entities, nor how to communicate effectively to other users. The threshold for "effective" communication is left to the user and tweaked according to their needs. Based on this requiremtn, for information bases to be useful, one must determine the *fidelity* of representations. Handles, by being defined by how they are restricted, are *exactly* described by the consequences of their relationships. This can be interpreted as truth, but we only prove that the language is *expressive enough* to represent any truth management system with a base set of axioms and that can be accepted by some partial computable function.

Our core building block to explain this system is through *units*. A unit is provided by a user-defined enumeration of handles, and units can be broken down, build new units, or act on other units via representations. Our approach is slightly more general than the enumerations defined by Li and Vitányi, see Definition 3.1.6, and they allow *arbitrary* ways to express formal systems, see Section 5. Operationally, units can be used as partial computable functions, but the former are strictly more expressive, due to user-defined handles.

[TODO: make this clear? Can't a unit **be** itself information?]

From the notion of a unit, we practically define information on a unit v as a unit u which *correctly identifies v from any other unit*. We formalize this as an invariant based on an equivalence relation on units, determined by the given set of representations (see Section 5). This notion corresponds to Burgin's idea of information as an *operator* that transforms a system, and is closely to Bateson's famous quote that "information is a difference that makes a difference" [Bat00] and generalizes a propositional rendering of this statement [CITE]. Our practical distinction between information and knowledge is that we *use* information, but users can assert their own notions of these terms by creating restricted contexts. We cthis through Algorithmic Information Theory by showing that the theoretical notion of conditional information content *precisely* coincides with the size of information, counted in bits (up to a constant) [TODO: link to this result. Important! And confirm this result! Currently unchecked.].

[TODO[MEDIUM]: probably provide an example of higher order logic or so? Would be nice! Shows that we don't need *exactly* a thing. But do emphasize that getting *exact* data formats can make information dissemination easier!]

Example 2.2.0. In a scientific experiment, a handle could be an observation or experimental data. The unit is then written as a symbol, say u, and is *implicitly* bound to this meaning. To distinguish from other symbols, say v, the computational content is analyzed.[TODO[MEDIUM]: expand out this example!]

Example 2.2.1. A more looser example is a user written journal for therapy sessions, containing information about daily habits and emotions. While neither of these are stored in the information base, their handles are, via units habit and $\mathrm{emotions}$ in a context $\mathrm{journal}$. Moreover, multiple revisions of the journal can be made with dates or other unique IDs.

There is an important extension required to express *any* partial computable function. With only two components, representations our representation is too restrictive, because we cannot naturally express *conditions* through *conditional representations*. Another issue is that managing two sets of handles is difficult with *solely* unconditional representations. Providing a form of *namespaces*, or a mechanism to distinguish two sets of names, which is crucial for determining and distinguishing available information as well. In addition to naming collisions, we also require a way to provide subject specific knowledge, as Burgin does through infological systems [Bur08]. A key insight in this thesis is showing that expressing conditions is *equivalent* to creating these namespaces: we express this idea as a **context**. This is related to an informal claim made in Meseguer [Mes12], that rewriting logics without conditional rules are strictly less expressive than those with conditions, see Theorem 5.3.4.

[TODO[SMALL]: use better names for entities/businesses/etc]

Example 2.2.2. A business could represent their operations using a unit $\mathrm{business}$ that contains units for their workers and ledgers. This allows another unit, say $\mathrm{business2}$, to contain its *own* label $\mathrm{workers}$ that is separate from the one in $\mathrm{business}$. In addition to separate labels, these contexts can have *distinct* rules, such as those for how business operations are performed.

Moreover, our formal rules are centered around contexts and are related to [McC93] but generalizes the context to be an operator itself (see Section 5).

Briefly, we can characterize units as containing a set of *subunits* (encoded by unique IDs) and a binary relation for *representations*. They objey two rules, which we informally describe now (and postpone nested contexts until Section 5):

- **Internal Transitivity:** if u represents v in context c and v represents x in context c, then u represents x in context c.
- **Lifting:** if u represents v in context c and p represents q in context v, then within context c, p represents q in context u.

2.3. **Base Operations.**

Now, units and information themselves could be expressed in infinitely many languages, with slightly different syntax or semantics. Welkin is carefully designed to be a *minimal* expression of these concepts, with minimal friction to express any other universal information base. These include:

- Intuitive arrow notation for representations, expressed in ASCII as `a - b -> c`. These can be interpreted *as* rewrite rules, depending on the context.

- Traditional braces `{ }` to denote closed definitions of contexts, inspired by the C programming language.
- Paths via dots `.` that is inspired by the Python programming language. Relative paths are denoted with multiple dots `...`, and absolute imports are prefixed with `#`. Subsets of units can be written via `u.{v, x}`, or even `u.{v --> x}` to refer to a subset of representations.
- Imports are done through `@u`, which takes all subunits of `u` and puts them into the current scope. In other words, the implementation *implicitly* adds denotations `v <--> a.v` for each subunit `v` of `u`. This is motivated by the abundance of boolean logic in classical mathematics and computer science, as any finite circuit can be expressed in terms of `and` and `not`. Selecting specific subunits can be done via `@u.{v, x}`.
- Imports can be *negated* via the notation `~@u`, and specific units can be negated through `~u`. This is an uncommon feature of most programming languages, appearing primarily in Haskell and CSS. While potentially opaque, Welkin provides robust definitions to ensure that negated forms can be easily translated into more explicit ones *and* back again. [TODO[SMALL]: provide link, probably to bootstrap or so?]
- Comments *are* strings can be treated as any unit. No comments need to be removed in the files and can *enhance* the study of new subjects.[2] These units can *then* be analyzed by others to promote translations to other human languages, or be studied through the *overaching relationships* in the units.

In general, the minimal restricted keywords is crucial for providing support for other languages. An implementation of Welkin will contain a small section of ASCII encoding for easier standardization, but the rest of the prgoram can be done *entirely* in the user's native language. This is a novel feature in most programming languages, which are either predominantly English or are fine tuned for specific human languages. [TODO: cite source on this about most programming languages being in English, as well as programming languages written in *different* human languages. Would be useful to have.]

2.4. **Bootstrapping.**

There are two important questions for implementing Welkin:

- How do we practically implement the operations?
- How do we check that the rules are used *correctly*?

[TODO[HIGH]: determine how to do this! Still WIP and not backed up yet, and still working on this!.]

For the first issue, the author previously went to manually describing the syntax and semantics, introducing notions from parsing theory and compilers. While this format is traditional for a programming language, this postponed the main rules and made the language specification harder to follow. Moreover, the author sought to explain the Trusted Computing Base, or the amount of axioms or meta-theory required. With the involved manual descriptions, this made the analysis difficult.

[2]Contexts can mimic regular comments, but Welkin aims to be inspectable by users. Encapsulation or private information can be enforced through *rejecting* specific contexts or only accepting certain ones.

To resolve these issues, they are combined into approach analogous to dependent type theory in the sense of "proofs as programs", but in a simpler way. We call this process "bootstrapping", analogous to bootstrapping a programming language from another through successive iterations, with each iteration building upon previous ones to define new language features. For Welkin, these steps are:

1) Formalize the base rules on units. This step is very similar to defining the SK-combinator calculus and can be similarly be implemented in a general purpose programming language.
2) Define the syntax through an *invertible syntax description*. This idea, pioneered by Rendel and Ostermann [RO10], provides a combinator approach to define how concrete syntax, that is user written, is converted into abstract syntax, an abstracted form, *and back*. Along with the syntax, the encoding is given *itself* through a representation, namely a table provided in this thesis.
3) Define the rules for path resolution and global IDs. These use a similar result to syntax descriptions and are significantly easier to read. [NOTE: emphasis on WIP here! Not yet developed in thesis !]

[TODO[HIGH]: determine if there are any missing steps! Important! Remove once this is finished.]

3. Foundations

3.1. **Rules.**

[TODO[SMALL]: emphasize that the word slate is important here! Connects to arbitrary objects!]

Units begin as *blank slates* and may be provided implicit bindings. This is done *after* the definition for ease of use.

[TODO[MEDIUM]: address Kripkenstein. Maybe just leave that as implementation dependent? The sole point is to avoid disagreements or keep things standard. Might depend ont the notion of arbtiary objects anyways and is determined by the active users involved?]

[TODO[SMALL]: address equality of binary words. Want to do this elegantly and quickly!]

Definition 3.1.0. A **binary word** is either the symbol 0 or 1, and if w is a binary word, so are $w.0$ and $w.1$, where . the symbol for **concatenation** (an undefined notion).

We will postpone to associativity to maintain the flow of new concepts.

[TODO[MEDIUM]: define in a compact way what "enumeration over all binary words is." Not sure if this *itself* should be done with symbolic units or is related to them?]

[TODO(SMALL): decide whether to add handles! Want the rest to be simple, so shoud be worth justifying!]

Definition 3.1.1. A *handle* is given by a pair $(\mathrm{UID}, \mathrm{HID})$, where UID is a binary word called a **user ID** and HID is the **handle ID**.

[TODO[SMALL]: define semantics of , vs | in a context.]

[TODO[SMALL]: explain what user provided enumeration means! Emphasis on being "blank slates", in a certain sense, so *assignable*, but not necesarially so. Here we can put custom/implicit meaning, and let this be *opaque*. Can be broken down further, or stand on its own. This represents what one would need to *understand* something!]

Definition 3.1.2. A **unit** is defined recursively as one of:
- A *literal* binary word, denoted by $0\mathrm{b}w$.
- A *handle*, see Definition 3.1.5.
- A representation $a \xrightarrow{c} b$ of units a, b, c, where a is the **sign**, c is the **context**, and b is the **referent**.
- A block, which is defined as either $\{\}$ or, for a block g and unit u, $g + \{u\}$ *or* $g \mid \{u\}$.

Remark 3.1.3. In contrast to the requirement to the beginning of Li and Vitányi (see Section 2), the enumeration need *not* be surjective but only *locally* so. Abstracting away from the implicit meaning, units act as partial computable functions, but the latter is strictly *less* expressive by removing user provided meaning.

Units satisfy the following rules, inspired by rewriting logic [Mes12]. These may be interpreted as inference rules *and* computational rules.

[TODO[SMALL]: provide labels/links.]

[TODO[SMALL]: ensure that when evaluating transitivity, non-determinism is possible!]

Definition 3.1.4.
- **Representation**: apply internal transitivity in each context.
  - **R1. Internal Transitivity**: $a \xrightarrow{c} b$ and $b \xrightarrow{c} d$ imply $a \xrightarrow{c} d$.
  - **R2. Lifting:** $a \xrightarrow{c} b$ and $p \xrightarrow{b} q$ implies $p \xrightarrow{a} q \in c$.
  - **R3. Idempotency:** $g + \{a\} + \{a\} \longleftrightarrow g + \{a\}$.
  - **R4. Commutativity:** $g + \{a\} + \{b\} \longleftrightarrow g + \{b\} + \{a\}$.
  - **R5. Associativity:** $\{a, \{b, c\}\} \longleftrightarrow \{\{a, b\}, c\}$.
  - **R6. Trivial Wrapper:** $\{a\} \longleftrightarrow a$.[3]

Remark 3.1.5. Each of these rules imposes no restrictions on what can be expressed, thanks to the presence of contexts. In fact, contexts are *necessary* for Turing completness, as one must express conditional rules. In the absence of contexts *or* rule **R2**, Definition 3.1.7 reduces to simple graph traversal. Now, while contexts can remove restrictions, these rules are carefully chosen to represent information as that which can be repeated multiple times (per context) and is positionally invariant. This allows us to enable *any* partial computable organization of information and, in particular optimize a given organization, see Section 6.

For universality, we need an important base construction that is definable in the theory: the ability to recurse through all IDs.

From there, we can recurse through all *potential* handles. These are user assigned, and whose interpretation is a free parameter in the theory. In other words, handles are *undefined notions* or entirely user-defined.

[TODO(SMALL): again, handle non-determinism here! Important!]

[3]In a set-theoretic context, the statement $\{a\} = a$ is similar to a "Quine atom" in Quine's New Foundations that includes an anti-foundation axiom [Qui37]. However, note that units are *not* necesarially sets, so the connection may not be applicable in all contexts.

$$\text{bit} \longrightarrow 0 \mid 1$$
$$\text{word} \equiv \{\text{head} \longrightarrow \{\text{bit} \mid \text{empty}\}, \text{next} \longrightarrow \text{word}\}$$

FIGURE 1. Generator for IDs in Welkin.

Moreover, for simplicity, we introduce tuples. A pair is: $\text{Pair} \equiv \{\text{first} \longrightarrow \text{word}, \text{second} \longrightarrow \text{word}\}$. A tuple is a nested pair that is left-associative w.r.t the labels first and second: $\text{Tuple} \equiv \{\text{head} \longrightarrow \text{Pair} \mid \text{word}, \text{next} \longrightarrow \text{Tuple} \mid \text{word}\}$.

Now we can prove the Turing definability of Welkin.

[TODO[SMALL]: make sure assignnments make sense! Do address possible ambiguity in direction of arrows. Can be confusing!]

[TODO[MEDIUM]: double check all parts of proof!]

Theorem 3.1.6. *Any partial computable function is definable by a unit.*

Proof. Define a new context C for this proof, containing Pair and Tpl, as defined above. We claim that that any term of the SKI calculus is definable as units in Welkin. To this end, if we can construct terms M and N, then we can represent the composition MN as a pair $\{\{M \longrightarrow \text{head}, N \longrightarrow \text{next}\} \xrightarrow{C} N'$ for some N' in C, and subsequent compositions MNQ as tuples. Thus, it suffices to show this claim K and S combinators are definable as units.

For the K combinator, consider the following construction: $K \equiv \{x, y, \}$ in C. We must show KAB reduces to A, or, more precisely,

$\{\{\{K \longrightarrow \text{first}, A \longrightarrow \text{second}\} \longrightarrow \text{head}, B \longrightarrow \text{next}\} \xrightarrow{C} N'$

[TODO(SHORT): clarify even more nested scopes from Lifting!]

$\{A \longrightarrow \text{first}, B \longrightarrow \text{second}\} \xrightarrow[?]{K} A$ in C.

$\{A \longrightarrow \text{first}, B \longrightarrow \text{second}\} \longrightarrow A$.

Now, for the S combinator, consider $S \equiv ?$. We must show $SABCD$ reduces to $(xz)(yz)$, i.e.,: □

3.2. Truth Management Systems.

The goal of this section is to completely define the following: *a proof of anything expressable as a RE set*. We will then generalize this to more through handles and mark soundness. Note that, because of Rice's Theorem, this method is inherently incomplete with computable methods.

[TODO[MEDIUM]: finish outline! Write up!]

- Goal: determine trust of Turing machine T halts on x in *finitely many steps* (so Halting problem)
  - Easy case: T *does* halt on x in finitely many steps. Trusted certificate: trace of Turing Machine.
  - Hard case: T *does not* on x. What can we *tell* is a certifcate

  that is itself expressible as the element of some RE set?
- Analyzing hard case
  - Normal approach by mathematicians: take a trusted theory, e.g., Zermelo-Frankel Set Theory, and see if T not accepting x is provable in such a theory. One still needs to find a proof in this trusted theory, *but* once then, apply modus ponens to get the result. If there happened to be a contradiction, revise beliefs and seek another system. Hasn't happened

with ZF, but in the *traditional* view, this is possible. (We'll get into Artemov below!)
  - Problem: ZFC can't prove *every* case of halting because of Gödel's incompleteness theorem. So more and more powerful theories. These, too, may be trusted, but might end of in an infinite regress, or very unclear with theories like, e.g., Rocq's type theory. No concrete ceiling, *unclear reasons for soundness* (other than being widely assumed).
  - Does provide an idea: *any* such instance of $x \neg T$ that has a finite witness *must* come from some Turing machine (that's what we want). So we want to solve another question: $\exists T'.x \in \mathrm{Halt}(T') \Rightarrow x \in \mathrm{Halt}(T)$. In the case above, T' is a well known formal system.
- Analyzing new problem: $\exists T'.x \in \mathrm{Halt}(\mathrm{T'}) \Rightarrow x \neg \in \mathrm{Halt}(T)$.
  - Ultimate problem: *not* accepting means infinite number of steps.

  So how to connect to a finite thing?
  - Answer: I *think* arbtitrary objects. What we observed above is something

  to do with axiomatic systems. If we *only* allow finite axioms, no quantifiers, we get stuck. We could approximate bit by bit, *but at some point*, it won't be enough for *most* Turing machines.
  - My question: I don't want to be restricted to FOL, as we will address in intro (e.g., quantum logic and needed an extension of FOL to handle!).
  - I think we *precisely* need handles here, because we can *prove*

  that using *only* the places where Turing machines halt isn't enough. **Turing machines alone aren't enough for proof systems, or for *validating* if specs are accurate!**. TODO: definitely need to make a lemma! Essential! (though will need to connect back with arbitrary objects, because technically one *needs* arbitrary objects to even state this!).
- Arbitrary objects + addressing $\exists T'.x \in \mathrm{Halt}(T') \Rightarrow x \neg \in \mathrm{Halt}(T)$
  - So we don't want to accept *any* handle, because that then

  gets into axiomatic systems. We can definitely *analyze* what those systems do and how they behave, yes, but how do we know *for sure* about things about Turing machines? What can we agree on?
  - We need a finite certificate to *represent* an aribtrary

  object, namely being the steps. So we need a *finite way* to determine that T doesnt halt on x

3.3. **Coherency and Information.**

Definition 3.3.7.

Definition 3.3.8.

4. Syntax

We keep this section self-contained with explicit alphabets and recursive definitions. For consistency with Welkin, we write syntax using `type-writer font`. Notationally, we write $a_0, \dots, a_n$ for a finite list of items, and use $a ::= a_1 \mid \dots \mid a_n$ to denote a definition of a in terms of $a_1, \dots, a_n$. For verification purposes, we will incorporate fixed bounds and completely unambiguous notation into Section 3.

4.1. **Words.**

Welkin's main encoding uses binary words, but add notation for decimal and hexadecimal.

```
bit ::= 0 | 1
digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
nibble ::= A | B | C | D | E | F
```

LISTING 1. Binary, decimal, and hexadecimal digits.

A word is a sequence of digits, see Listing 2. We leave concatenation `.` as an undefined notion. We set concatenation to be right-associative, i.e., `(w.w').w'' = w.(w'.w'')`, and abbreviate `w.w'` as `w w'`. For conversions, see Definition 5.1.13.

```
word ::= binary | decimal | hex
binary ::= bit | binary.bit
decimal ::= digit | decimal.digit
hex ::= nibble | hex.nibble
```

LISTING 2. Definition of words.

Equality is defined recursively, shown in Listing 3. Note that leading zeros are allowed.

[TODO[MEDIUM]: determine how to make Listing 3 work well in text **and** Welkin.]

```
0.word = word
0b0.word = word
0x0.word = word
```

LISTING 3. Definition of word equality.

4.2. **Terminals.**

[TODO[SHORT]: determine whether to put these details in appendix. Maybe given the *overarching* grammar, at a high level?] Welkin uses ASCII as its base encoding. The term ASCII is slightly ambiguous, as there are subtly distinct variants, so we formally define US-ASCII as a standard version.[4]

Definition 4.2.0. US-ASCII consists of 256 symbols, listed in Table 3.

To represent general encodings, there is a binary format supported for strings, see Listing 5.

[4]Note that this table *itself* is a representation, which represents glyphs with binary words. The use of these kinds of representations occur frequently in Welkin, see Section 3.

Dec.	Hex.	Glyph	Dec.	Hex.	Glyph	Dec.	Hex.	Glyph	Dec.	Hex.	Glyph
0	00	NUL	32	20	Space	64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

TABLE 3. US-ASCII codes and glyphs.

We denote specific characters through quotes, escaping if necessary. There are several important character classes in Listing 4, denoted through double quotes.

```
PRINTABLE  ::= [0x20-0x7E]
WHITESPACE ::= [0x09, 0x0A, 0x0D, 0x20]
DELIMITER ::= [0x7B, 0x7D, 0x2C, 0x2D, 0x2A, 0x3C, 0x3E, 0x22, 0x27, 0x5C,
0x7D]
```

LISTING 4. Important character classes.

Strings allow escaped single or double quotes, see Listing 5. IDs are special cases of strings that do not require quotes but forbid whitespace and certain characters, see Listing 6.

```
STRING ::= SQ_STRING | DQ_STRING
SQ_STRING ::= "'" (SQ_CHAR | ESCAPE_SQ )* "'"
DQ_STRING ::= "'" (DQ_CHAR | ESCAPE_DQ )* "'"

SQ_CHAR ::= PRINTABLE \ {'}
DQ_CHAR ::= PRINTABLE \ {"}
ESCAPE_SQ ::= "\'" | "\\"
ESCAPE_DQ ::= "\"" | "\\"
```

LISTING 5. Strings.

```
IMPORT ::= "@" ID
ID :: ID_CHAR+
ID_CHAR ::= PRINTABLE / (DELIMITERS + WHITESPACE + "#" + "@" + "'" + "\"")
```

LISTING 6. IDs.

[TODO(MEDIUM): find good way to implement directly with Welkin or discuss.]

4.3. **EBNF Notation and Parse Trees.**

We define our variant of EBNF below:

Definition 4.3.1. An **EBNF** grammar consists of **productions**, which are pairs of the form `r ::= a_1 ... a_n`. On the right-hand side, juxtaposition means concatenation.

- Uppercase names require *no* whitespace between them. Otherwise, whitespace is allowed.
- `a ::= a_1 | ... | a_n` is short-hand for {`a ::= a_i | 1 <= i <= n`}.
- `(a_1)*` means zero or more instances of a_1.
- `(a_1)+` means one or more instances of a_1.
- `(a_1)?` means zero or one instance of a_1.

4.4. **The Welkin Grammar.**

Welkin's grammar is displayed in Listing 7, inspired by a minimal, C-style syntax. Note that when concatenating two terminals, denoted in uppercase, no whitespace between them is allowed, but in any other case, any amount of whitespace is allowed but ignored.

```
start        ::= terms
terms        ::= term ("," term)* ","? | EPS
term         ::= arc | graph | tuple | path
arc          ::= (term ("-" | "<-") term ("-" | "->"))+ term
graph        ::= path? "{" terms "}"
tuple        ::= path? "(" terms ")"

path         ::= MODIFIER? path_segment* unit
path_segment ::= unit | ".*" | "."+
unit         ::= ID | STRING


MODIFIER ::= "#" |  "@" | "~@" | "~"
ID         ::= ID_CHAR+
ID_CHAR    ::= PRINTABLE \ (DELIMITERS | WHITESPACE | "#" | "@" | "~" |
"'" | '"')
DELIMITERS ::= "," | "." | "-" | "<" | ">" | "*" | "(" | ")" | "{" | "}"
STRING     ::= SQ_STRING | DQ_STRING
SQ_STRING  ::= "'" (SQ_CHAR | ESCAPE_SQ )* "'"
DQ_STRING  ::= '"' (DQ_CHAR | ESCAPE_DQ )* '"'
SQ_CHAR    ::= PRINTABLE \ {"'"}
DQ_CHAR    ::= PRINTABLE \ {'"'}
ESCAPE_SQ  ::= "\'" | "\\"
ESCAPE_DQ  ::= '\"' | "\\"
PRINTABLE  ::= [0x20-0x7E]
WHITESPACE ::= [0x09, 0x0A, 0x0D, 0x20]
DELIMITER ::= [0x7B, 0x7D, 0x2C, 0x2D, 0x2A, 0x3C, 0x3E, 0x22, 0x27, 0x5C,
0x7D]
EPS        ::= ""
```

LISTING 7. The grammar for Welkin. The terminals `id` and `string` are defined in Listing 2 and Listing 5, respectively

[TODO(HIGH): determine if this is good as is or needs to be put into bootstrap!]

4.5. **Proof of Unambiguity.**

We now prove that the Welkin language is unambiguous by showing it is LL(1), a rich class of grammars that can be efficiently parsed. For more details, please consult [Aho+06].

Moreover, we define the top of a word in Listing 8.

```
top(word) ::= nil => nil | bit.word => bit
```

LISTING 8. Definition of the top of a word.

Definition 4.5.2. ([RS70]). A grammar is LL(1) iff the following holds: for any terminal w_1 and nonterminal A, there is at most one rule r such that for some w_2, w_3 appearing at the top of A such that,

- $S \Rightarrow \mathrm{top}(w_1) A w_3$
- $A \Rightarrow w_2(p)$
- $\mathrm{top}(w_2 w_3) = w$

Theorem 4.5.3. *There exists some* LL(1) *grammar that accepts the same strings as the Welkin grammar Listing 7. Hence, Welkin's syntax is unambiguous, i.e., every string accepted by the language has exactly one derivation.*

Proof. We use transformations in Table 4 that preserve the language of the original grammar, resulting in Listing 9. For the refactor step by step, see Table 5. We can readily verify that there are no shared prefixes for a single production, see Table 6. Because there are no conflicts, the transformed grammar is LL(1), and hence, the grammar is unambiguous.

Rule ID	Name	Description
T0	**Group Flattening**	Converts Kleene stars `A*` and regex-like lists into right-recursive forms `A' ::= A A' | EPS`.
T1	**Left Refactoring**	Transforms overlapping prefixes `A ::= B C | B D` into `A ::= B (C | D)` to eliminate FIRST set collisions.
T2	**Lexical State Expansion**	Expands complex sequence operators (+, *) into strict right-recursive terminal rules, ensuring contiguous consumption without whitespace interruptions.
T3	**Left-Recursion Removal**	Eliminates immediate left-recursion `A ::= A B | C` by rewriting as `A ::= C A'` and `A' ::= B A' | EPS` to prevent infinite loops.

TABLE 4. Well known transformations on grammars that preserve string acceptance.

```
start      ::= terms
terms      ::= term terms_tail | EPS
terms_tail ::= "," terms | EPS
term       ::= node chain

chain      ::= left_link node right_link
               node chain | EPS

left_link  ::= "-" | "<-"
right_link ::= "-" | "->"

path     ::= path_segment* unit
path_segment ::= MODIFIER? (UNIT | ".*" | "."+)

node       ::= PATH opt_block | block
opt_block  ::= block | EPS
block      ::= "{" terms "}"
             | "(" terms ")"

PATH       ::= MODIFIER PATH_BODY
             | PATH_BODY
PATH_BODY  ::= "." PATH_DOTS
             | UNIT PATH_TAIL
PATH_DOTS  ::= "*" PATH_BODY
             | "." PATH_DOTS
             | UNIT PATH_TAIL
PATH_TAIL  ::= PATH_BODY | EPS

UNIT     ::= IMPORT | ID | STRING
MODIFIER   ::= "#" | "@" | "~@" | "~"
ID         ::= ID_CHAR+
ID_CHAR    ::= PRINTABLE \ (DELIMITERS | WHITESPACE | "#" | "@" | "~" |
"'" | '"')
DELIMITERS ::= "," | "." | "-" | "<" | ">" | "*" | "(" | ")" | "{" | "}"
STRING     ::= SQ_STRING | DQ_STRING
SQ_STRING  ::= "'" (SQ_CHAR | ESCAPE_SQ )* "'"
DQ_STRING  ::= '"' (DQ_CHAR | ESCAPE_DQ )* '"'
SQ_CHAR    ::= PRINTABLE \ {"'"}
DQ_CHAR    ::= PRINTABLE \ {'"'}
ESCAPE_SQ  ::= "\'" | "\\"
ESCAPE_DQ  ::= '\"' | "\\"
EPS        ::= ""
```

LISTING 9. Transformed LL(1) grammar for Welkin, with all terminals defined.

<table>
<tr><th>Original</th><th>Transform</th><th>LL(1)</th></tr>
<tr><td><code>start ::= terms,
terms ::= term ("," term)*
","? | EPS</code></td><td>Transform 1</td><td><code>start   ::= terms
terms   ::= term
terms_tail | EPS
terms_tail ::= "," terms
| EPS</code></td></tr>
<tr><td><code>term ::= arc | graph |
         group | path
arc ::= (term ("-" |
"<-")
          term ("-" | "-
>"))+ term</code></td><td>Transform 4</td><td><code>/* Extracted 'node' to
fix recursion.
   Arcs are strict left/
right link pairs */
term   ::= node
chain

chain   ::= left_link
node right_link
               node
chain | EPS

left_link ::= "-" |
"<-"
right_link ::= "-" | "-
>"</code></td></tr>
<tr><td><code>graph ::= path? "{"
terms "}"
tuple ::= path? "("
terms ")"
path ::= modifier?
path_segment* unit</code></td><td>Transform 2,
Transform 3</td><td><code>/* Left-factor path &
blocks. */
node   ::= PATH
opt_block | block
opt_block ::= block |
EPS
block   ::= "{" terms
"}"
             | "(" terms
")"

/* Expand path +, *
contiguously */
PATH   ::= MODIFIER
PATH_BODY
             | PATH_BODY
PATH_BODY ::= "."
PATH_DOTS
             | UNIT
PATH_TAIL
PATH_DOTS ::= "*"
PATH_BODY
             | "."
PATH_DOTS
             | UNIT
PATH_TAIL
PATH_TAIL ::= PATH_BODY
| EPS</code></td></tr>
</table>

TABLE 5. Refactor of grammar Listing 7 into Listing 9. Entries with - mean that no changes are needed.

<table>
<tr><th>Non-Terminal</th><th>Lookahead (a)</th><th>Production Chosen</th></tr>
<tr><td>start</td><td>"#" | "@" | "~@" | "~" | "." | ID | STRING | "{" | "(" | EOF</td><td>terms</td></tr>
<tr><td>terms</td><td>"#" | "@" | "~@" | "~" | "." | ID | STRING | "{" | "("</td><td>term terms_tail</td></tr>
<tr><td></td><td>EOF | "}" | ")"</td><td>EPS</td></tr>
<tr><td>terms_tail</td><td>","</td><td>"," terms</td></tr>
<tr><td></td><td>EOF | "}" | ")"</td><td>EPS</td></tr>
<tr><td>term</td><td>"#" | "@" | "~@" | "~" | "." | ID | STRING | "{" | "("</td><td>node chain</td></tr>
<tr><td>node</td><td>"#" | "@" | "~@" | "~" | "." | ID | STRING</td><td>PATH opt_block</td></tr>
<tr><td></td><td>"{" | "("</td><td>block</td></tr>
<tr><td>opt_block</td><td>"{" | "("</td><td>block</td></tr>
<tr><td></td><td>EOF | "}" | ")" | "," | "-" | "<-" | "->"</td><td>EPS</td></tr>
<tr><td>block</td><td>"{"</td><td>"{" terms "}"</td></tr>
<tr><td></td><td>"("</td><td>"(" terms ")"</td></tr>
<tr><td>chain</td><td>"-" | "<-"</td><td>left_link node right_link node chain</td></tr>
<tr><td></td><td>EOF | "}" | ")" | ","</td><td>EPS</td></tr>
<tr><td>left_link</td><td>"-"</td><td>"-"</td></tr>
<tr><td></td><td>"<-"</td><td>"<-"</td></tr>
<tr><td>right_link</td><td>"-"</td><td>"-"</td></tr>
<tr><td></td><td>"->"</td><td>"->"</td></tr>
<tr><td>PATH</td><td>"#" | "@" | "~@" | "&"</td><td>MODIFIER PATH_BODY</td></tr>
<tr><td></td><td>"." | ID | STRING</td><td>PATH_BODY</td></tr>
<tr><td>PATH_BODY</td><td>"."</td><td>"." PATH_DOTS</td></tr>
<tr><td></td><td>ID | STRING</td><td>UNIT PATH_TAIL</td></tr>
<tr><td>PATH_DOTS</td><td>"*"</td><td>"*" PATH_BODY</td></tr>
<tr><td></td><td>"."</td><td>"." PATH_DOTS</td></tr>
<tr><td></td><td>ID | STRING</td><td>UNIT PATH_TAIL</td></tr>
<tr><td>PATH_TAIL</td><td>"." | ID | STRING</td><td>PATH_BODY</td></tr>
<tr><td></td><td>EOF | "}" | ")" | "," | "-" | "<-" | "->" | "{" | "("</td><td>EPS</td></tr>
<tr><td>MODIFIER</td><td>"#"</td><td>"#"</td></tr>
<tr><td></td><td>"@"</td><td>"@"</td></tr>
<tr><td></td><td>"~@"</td><td>"~@"</td></tr>
<tr><td></td><td>"&"</td><td>"&"</td></tr>
<tr><td>UNIT</td><td>ID</td><td>ID</td></tr>
<tr><td></td><td>STRING</td><td>STRING</td></tr>
</table>

TABLE 6. LL(1) Table for Listing 9

□

5. Semantics

This section describes several phases to transform parse trees into more refined forms called **Internal Representations (IR)**. These phases are:

- Abstract Syntax Trees (ASTs): simplifies the parse tree

and removes punctuation.

- Lexicographic Ordering: Lexicographically orders graphs by names and anonymous graph

content.

- Unique IDs: Assigns IDs to all names and resolves absolute and relative paths.
- Merging: merges units and defines the final scopes.

How ASTs are processed and validated. We postpone information organization to Section 6.

5.1. Abstract Syntax Tree (ASTs).

Given the rationale, we explain how the Abstract Syntax Tree (AST) is processed for the syntax. The AST provides an intermediate step before the final data structure.

Definition 5.1.0. The **Abstract Syntax Tree** is recursively defined from the parse tree of Listing 7 as follows:

- **Terms:** Converted into a list, which is empty if `EPS` is matched.
- **Term:** either a Root, Arc, Graph, Group, or Path, with two additional fields:
  - **Position:** a pair $(\mathrm{Line}, \mathrm{Column})$, where Line is the first

number of newline ("n") characters occurring before the term and Column is the position of this term on the line. Both of these are stored as bytes.

- **Root:** simply stores the corresponding unit.
- **Arc:** This is converted into a list. The first item is $(s_0, c_0 r_0)$, the first triple that occurs in the chain. Then, the remaining triples are added to the list.
  - Left arrows are added as (r_0, c_0, r_0). Edges and double arrows are added as both a left and right arrow.
- **Graph:** The terms are collected into two parts: a list of parts and a list of arcs. Each graph has a name; when no name is provided, it is `""`.
- **Tuple:** The terms are organized recursively, with the base case starting

at `item` and the recursive step at the label `next`. Note that tuples have **closed** definitions and will create copies when accessed or used in an arc.

- **Path:**
  - The number of dots is counted for the relative paths.
  - Star imports are denoted by a special node All.
  - A path is converted into a list of its contents,

  which are pairs containing the relative path number and either Unit or All.
  - The `unit` is added at the end.
- **ID:** converted into strings.
- **String:** Wraps around the contents.
- **Number:** converts decimal and hexadecimal into binary, recursively over words according to Table 7.

The terms in the top-level are put into a Graph node containing a unique, user given ID.

[TODO[SHORT]: determine nice way to merge this with Listing 3!]

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Dec	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bin	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111

TABLE 7. Conversions of digits between different bases.

Definition 5.1.1. An Abstract Syntax Tree is **valid** if the following holds:

- A Root term must exist. Moreover, there must not be conflicting Root term names.
- Relative imports does not exceed the number of available parents.

Remark 5.1.2. An earlier revision of this thesis forbid repetitions of arcs and units. However, this restriction was removed to provide greater flexibility. This will be tracked, see ?.

5.2. **Unified IDs.**

This phase first lexicographically orders the graph by its labels. Anonymous graphs are lexicographically ordered by contents, with arcs treated as triples and lexicographically ordered accordingly. Then, IDs are assigned. The lexicographic ordering ensures the ID is *exactly* the same for two strings that are positionally different. This shows that Welkin is positionally invariant.

5.3. **Unification.**

This phase merges the units into the final data structure.

[TODO[SHORT]: add link to quote!]

Remark 5.3.3. In contrast to the requirement to the beginning of Li and Vitány (see Section 2), the enumeration need *not* be surjective but only *locally* so. Abstracting away from the implicit meaning, units act as partial computable functions, but the latter is strictly *less* expressive by removing user provided meaning.

[TODO: ensure this definition is general enough! We will need to tackle the third rule, having unspecified parameters, more in depth. Does this mean an implementation defined feature? Or does it generalize it?] However, this is only one component: we also must prove we can represent *any* truth management system. This is made possible through contexts. We define a **truth management system** generally as a partial computable function augmented with parameters that denote the truth of base statements or **axioms**. These are intentionally left undefined, in the same vein as **R3**. In fact, by **R3** and Theorem 3.1.1, we obtain the following.

Corollary 5.3.4. *Any computable truth management system can be represented as a*

Note that it is essential to have contexts via **R2**, as shown by the following.

Theorem 5.3.5. *Representations with contexts cannot be expressed with those without.*

Proof. The largest class expressible with unconditional representations are context-free grammars, because... Thus, not all partial computable functions are included, completing the proof. □

5.4. **Queries and Information.**

We set $\left(u \xrightarrow{v} w\right)_c \in x \Leftrightarrow x(u) \xrightarrow{x(v)} x(w)$, where $x(s)$ is the local extension of s in x. We interpret $u \longrightarrow v$ as: the **sign** u represents **referent** v in **context** c. Through Theorem 3.1.1, we will present the following computational interpretation:

$$u \xrightarrow{v} w \text{ iff } \varphi_u(v) \text{ evaluates to } w,$$

where φ_u is the partial computable function given by the ID of u. Note that this is *only* logical equivalence; the former is strictly *more* expressive, due to implicit bindings.

Definition 5.4.6. A unit u is **non-trivial** if it is non-empty and has a non-complete representation graph. A unit u is **coherent relative to a context** u' if $u + u'$, the union of these units, is non-trivial.

Remark 5.4.7. This definition is a natural generalization of consistency in first-order logic. We will frequently rely on this result throughout the thesis.

Definition 5.4.8. Let u, v be units. Then u **contains information** v if for some $s \in v$, $u[s] \neq s$.

Our notion of information helps with one key issue: the general undefinability of non-trivial classes of partial computable functions in formal system. This connects with the absence of a universal *single* formal system that can prove any claim about, e.g., Peano Arithmetic.

[TODO: clean up this example. Want to emphasize what is information here, so, e.g., we may say left and right nodes don't have information about each other, in general]

Example 5.4.9. Consider the recursive definition of a binary tree: either it is a null (leaf) node, or it contains two nodes, left and right. We can model this as follows:

- First, create units for each of the notions: `tree {null, left, right}`.

[TODO: add a condition that the left and right trees are distinct, to show this is possible!]

- Next, we write, `tree { nil --> .tree, left..tree, right..tree, {.left, .right} --> .tree}`. Notice that we refer to the *namespace* via a relative path, `.tree`, thereby enabling recursion.
- We can test this out in Welkin with: `my_tree {.tree.left --> {nil --> .tree}, .tree.right {nil --> .tree} }`. This is then coherent with the previous definition.

Are are two important ideas in this example. First, an abstraction can be defined prior to a concrete model. The other way is possible as well, showing how developing representations are flexible in Welkin. Second, the derivations of trees can now be formulated. So we can define descendants and ancestors, and test against the coherency of the tree.

A key technique in managing information and truth through contexts is through the following theorem. FIXME: this is currently a stub! Need to create the **correct** condition. Use this as a starting point:

Theorem 5.4.10. *A unit u contains information about v iff $u + v$ is coherent.*

[TODO: Develop the notion of a query and its relation to information. Ultimately, we want to define information based on how useful it is for querying the database. We want to define a query to be anything we can inquire *about* a database that we we can (partially) computably represent. Information should then follow quickly from there as a *partial* answer. Having *enough direct* information means being able to *fully* solve the query. Moreover, the goal will be to use that this notion of enough is efficient, so checking for this should be efficient, say $O(n)$ or $O(n^2)$. This will likely be based on rewrite rules, in combination with axiom **R3**.]

[TODO: a core part of queries is **indirect** information, or information that may not be directly visible by immediately applying rewrite rules. This relates to my earlier attempts on a universal progress theorem. What I want is to know if query is "well-posed" or *can* be solved by a computable representation. This well-posedness needs to be *defined* in the truth system itself, and part of this may be undecidable. However, most of this should be converted to *finitistic properties of comptuable functions*. This will be my new version of "universal progress", and I may provide an example where introducing *bridging* representations may be effective, such as through mathematics and music (which is *not* necesarially a homomorphism between proof systems.)]

6. Information Organization

The presentation of Welkin's universal expressivity, stated as Theorem 3.1.1, is fixed with one particular representation. Following the analogue of units to practical computable functions, we define **Universal Representation Systems (URS)** as the analogues of Universal Turing Machines, see Definition 6.1.18.

A major problem for scalability is *choosing* a URS. Possibly the use of multiple URSs for different use cases is more optimal, in some sense? The key operation in an information base is *querying*, so this must be as efficient as possible. . This As established in Section 5, bounded queries can be answered in $O(?)$ time. The problem then becomes about optimizing the number of steps. While this is query dependent, and depends on the database, we prove that any of these criterion can be converted to one about *size*. Our proof generalizes Blum's axioms [Blu67] and Kolomogorov complexity [LV19]. While finding the absolute smallest size of a unit that will best optimize a query is impossible, we *can* optimize the database with the available information. Our localized algorithm provides a nice architecture to solve problems: combining bounded queries in the database to confirm the presence of an answer, combined with unbounded searches by some search procedure or heuristics. Note that the search procedure may or may not be computable; what is important is that bounded queries are always efficient. We also provide proof certificates.

6.1. **Universal Systems.**

Note that there are multiple ways to prove Theorem 3.1.1, infinitely in fact. This motivates the following definition.

Definition 6.1.0. A universal representation system (URS) is a unit that can represent any representation.

Theorem 6.1.1. *A unit is a universal representation system if and only if it can represent any partial computable function. Moreover, any universal representation system can represent any universal representation system. In particular, representing itself is called* ***reflection****.*

[TODO: dissue axiomatic systems! Want to emphasize the relevant **process** (per context) is important! That is, the journey to discover new things. ONLY FI the specification is complete in some way (or "finalized"), it is then that axiomatic systems **can** help. Expand this discussion into a paragraph or two.]

The term *universal* is specifically for expressing *representations* symbolically. The free parameter still needs to be included and is an additional feature on top of partial comptuable functions. However, the *management* of these symbols is done entirely with partial computable functions.

The next section discusses the issue of *managing* the infinitely many choices for URSs.

6.2. **Localized Size Compression.**

Instead of making proofs most efficient as is, we want to support finding optimal representations. But we want to do this from an efficiently queryable system, which *is* the most optimal.

7. Conclusion

[TODO: reformat this to write: foundations, syntax, semantics.] This thesis introduced Welkin, a universal, formalized information language. The syntax (Section 4) was defined rigorously with a small EBNF, shown to be accepted by an LL(1) grammar, showing that parsing is unambiguous. The semantics (Section 5) were provided with several passes to convert parse trees into units, which contain both a hierarchical and relational structure for scoping and direct representations, respectively. Units have key properties that enable them to express any partial computable function Theorem 3.1.1, in conjungtion with expressing any truth management system, demonstrates **universality** of the system. This is practically demonstrated by showing that all the major paradigms in Information Management and Knowledge Management are expressible within Welkin. Moreover, it was shown that there is a way to best organize the language given available information Section 6, showing **scalability**. Finally, the bootstrap in Section 3 self-hosts the language within a bounded 64 variant, whose complete Unambiguity (as well as the grammar's prior) establishes **standardization**. Revisions further enhance this by

The remaining sections show several areas for future work. This list is not exhaustive and, by the previous arguments, and can be applied to *any* subject with computable representations (essentially, any human subject).

7.1. **Programming Languages and Formal Verification.**

[TODO: add discussions on supporting interoperability with languages, providing more robust and unified implementations of core libraries, compilers, and operating systems.]

Taking the enable custom hardware implementations for checking and reduce the surface area for attacks on verifying certifications for many applications, including cryptography.

Moreover, the proposed architecture could use an LLM as an oracle.

7.2. **Mathematical and Scientific Knowledge.**

[TODO: discuss more applications of Welkin in depth.]

There are several possible projects to pursue in mathematics and scientific research. For mathematics, there are several existing rojects for storing mathematical information (see [CF09] for more details). Older proposals, including the QED Manifesto [KR16] and the Module system for Mathematical Theories (MMT), aimed to be more general and have seen limited success. More centralized systems, like `mathlib` in the Lean proof assistant [The20], have seen adoption but do not give equal coverage nor are interoperable with other systems. Welkin enables this interopreeable through gradual translations, and with Section 6, one can always determine if there is enough *direct* information to complete a translation. This will help facilitate reusability among major tools, and aid in formal verification (Section 7.1) well.

Along with mathematics, Welkin could provide more rigorous frameworks for the sciences, with are currently scattered with different proposals. One prominent proposal is the Findable Accessible Interoperable Reusable (FAIR) guidelines [Wil+16]. Instead of providing a concrete specification or implementation, FAIR provides best practices for storing scientific information. However, multiple papers have outlined problems with these overarching principles, including missing checks on data quality [Gui+25], missing expressiveness for ethics frameworks [Car+21], and severe ambiguities that affect implementations [Jac+20]. Welkin addresses these by using contexts strategically. Experiments can be compared using revisions, and disagreements between experts can be analyzed using separate contexts. These contexts can then *distinguish* between different theories, and scientists can select the unit with the best or most comprehensive evidence. Metrics for such evidence can be *representable* to a certain point, but at a minimum, they can be more effectively analyzed.

7.3. **Humanities.**

[TODO: significantly expand.]

Information Management in the humanities has few models, including an adaption of FAIR [Har+20] and discipline specific, linked databases in the PARNTHEOS project [Hed+19]. Welkin could assist by providing a space to help standardize this and localize different publication styles and literary theories.

References

[Aho+06] Aho, Alfred V., Lam, Monica S., Sethi, Ravi, and Ullman, Jeffrey D., *Compilers: Principles, Techniques, and Tools (2nd Edition)*, Addison Wesley, 2006.

[Atk23] Atkin, Albert, Peirce's Theory of Signs, Spring2023 ed., Metaphysics Research Lab, Stanford University, 2023.

[Bat00] Bateson, Gregory, *Steps to an ecology of mind*, University of Chicago Press ed., University of Chicago Press, Chicago, 2000.

[Blu67] Blum, Manuel, A Machine-Independent Theory of the Complexity of Recursive Functions, J. Acm **14** no. 2 (1967) 322–336.

[Bra23] Bradley, F. H., The Principles of Logic, Mind **32** no. 127 (1923) 352–356.

[Bur08] Burgin, Mark, Foundations of Information Theory, 2008.

[Bur09] Burgin, Mark, *Theory of Information*, World Scientific, 2009.

[CF09] Carette, Jacques, and Farmer, William M., A Review of Mathematical Knowledge Management, in *Intelligent Computer Mathematics* (Carette, Jacques, Dixon, Lucas, Coen, Claudio Sacerdoti, and Watt, Stephen M. (eds.), Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 233–246.

[Car+21] Carroll, Stephanie Russo, Herczog, Edit, Hudson, Maui, Russell, Keith, and Stall, Shelley, Operationalizing the CARE and FAIR Principles for Indigenous Data Futures, Scientific Data **8** no. 1 (2021) 108.

[Dic25] Dictionary, Oxford English, welkin, n., Oxford University Press, 2025.

[Fin85] Fine, Kit, *Reasoning with Arbitrary Objects*, Blackwell, New York, NY, USA, 1985.

[Gui+25] Guillen-Aguinaga, Miriam, Aguinaga-Ontoso, Enrique, Guillen-Aguinaga, Laura, Guillen-Grima, Francisco, and Aguinaga-Ontoso, Ines, Data Quality in the Age of AI: A Review of Governance, Ethics, and the FAIR Principles, Data **10** no. 12 (2025).

[Har90] Harnad, Stevan, The Symbol Grounding Problem, Physica D **42** (1990) 335–346.

[Har+20] Harrower, Natalie, Quinn, Mary, Tóth-Czifra, Erzsébet, and others, *Sustainable and FAIR Data Sharing in the Humanities: Recommendations of the ALLEA E-Humanities Working Group*, Berlin, 2020.

[Hed+19] Hedges, Mark, Stuart, David, Tzedopoulos, George, Bassett, Sheena, Garnett, Vicky, Giacomi, Roberta, and Sanesi, Maurizio, *Digital Humanities Foresight: The future impact of digital methods, technologies and infrastructures*, GOEDOC, Dokumenten-und Publikationsserver der Georg-August-Universität Göttingen, 2019.

[Jac+20] Jacobsen, Annika, Miranda Azevedo, Ricardo de, Juty, Nick, Batista, Dominique, Coles, Simon, Cornet, Ronald, Courtot, Mélanie, Crosas, Mercè, Dumontier, Michel, Evelo, Chris T., Goble, Carole, Guizzardi, Giancarlo, Hansen, Karsten Kryger, Hasnain, Ali, Hettne, Kristina, Heringa, Jaap, Hooft, Rob W.W., Imming, Melanie, Jeffery, Keith G., Kaliyaperumal, Rajaram, Kersloot, Martijn G., Kirkpatrick, Christine R., Kuhn, Tobias, Labastida, Ignasi, Magagna, Barbara, McQuilton, Peter, Meyers, Natalie, Montesanti, Annalisa, Reisen, Mirjam van, Rocca-Serra, Philippe, Pergl, Robert, Sansone, Susanna-Assunta, Silva Santos, Luiz Olavo Bonino da, Schneider, Juliane, Strawn, George, Thompson, Mark, Waagmeester, Andra, Weigel, Tobias, Wilkinson, Mark D., Willighagen, Egon L., Wittenburg, Peter, Roos, Marco, Mons, Barend, and Schultes, Erik, FAIR Principles: Interpretations and Implementation Considerations, Data Intelligence **2** nos. 1–2 (2020) 10–29.

[KR16] Kohlhase, Michael, and Rabe, Florian, QED Reloaded: Towards a Pluralistic Formal Library of Mathematical Knowledge, Journal of Formalized Reasoning **9** (2016) 201–234.

[Las+24] Lastras, Luis A., Trager, Barry M., Lenchner, Jonathan, Szpankowski, Wojciech, Wu, Chai Wah, Squillante, Mark S., and Gray, Alexander, Breaking through the classical Shannon entropy limit: A new frontier through logical semantics, arXiv e-prints (2024) arXiv:2501.00612.

[LV19] Li, Ming, and Vitanyi, Paul, *An Introduction to Kolmogorov Complexity and Its Applications*, 4th ed., Springer Publishing Company, Incorporated, 2019.

[Liu25] Liu, Zhangchi, A Unified Formal Theory on the Logical Limits of Symbol Grounding, 2025.

[Liu25] Liu, Zhangchi, An Algorithmic Information-Theoretic Perspective on the Symbol Grounding Problem, 2025.

[The20] The mathlib Community, The lean mathematical library, in *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, Association for Computing Machinery, New Orleans, LA, USA, pp. 367–381.

[McC93] McCarthy, John, Notes on formalizing context, in *Proceedings of the 13th International Joint Conference on Artifical Intelligence - Volume 1*, Morgan Kaufmann Publishers Inc., Chambery, France, pp. 555–560.

[Mes12] Meseguer, José, Twenty years of rewriting logic, The Journal of Logic and Algebraic Programming **81** no. 7 (2012) 721–781.

[Mik23] Mikkilineni, Rao, Mark Burgin's Legacy: The General Theory of Information, the Digital Genome, and the Future of Machine Intelligence, Philosophies **8** no. 6 (2023).

[Qui37] Quine, W. V., New Foundations for Mathematical Logic, The American Mathematical Monthly **44** no. 2 (1937) 70–80.

[RO10] Rendel, Tillmann, and Ostermann, Klaus, Invertible syntax descriptions: unifying parsing and pretty printing, in *Proceedings of the Third ACM Haskell Symposium on Haskell*, Association for Computing Machinery, Baltimore, Maryland, USA, pp. 1–12.

[RS70] Rosenkrantz, D.J., and Stearns, R.E., Properties of deterministic top-down grammars, Information and Control **17** no. 3 (1970) 226–256.

[Rus84] Rushby, John, A Trusted Computing Base for Embedded Systems, in *Proceedings 7th DoD/NBS Computer Security Initiative Conference*, Gaithersburg, MD, pp. 294–311.

[Wil+16] Wilkinson, Mark D., Dumontier, Michel, Aalbersberg, IJsbrand Jan, Appleton, Gabrielle, Axton, Myles, Baak, Arie, Blomberg, Niklas, Boiten, Jan-Willem, Silva Santos, Luiz Olavo Bonino da, Bourne, Philip E., Bouwman, Jildau, Brookes, Anthony J., Clark, Tim, Crosas, Mercè, Dillo, Ingrid, Dumon, Olivier, Edmunds, Scott C., Evelo, Chris T. A., Finkers, Richard, González-Beltrán, Alejandra N., Gray, Alasdair J. G., Groth, Paul, Goble, Carole A., Grethe, Jeffrey S., Heringa, Jaap, Hoen, Peter A. C. 't, Hooft, Rob W. W., Kuhn, Tobias, Kok, Ruben G., Kok, Joost N., Lusher, Scott J., Martone, Maryann E., Mons, Albert, Packer, Abel Laerte, Persson, Bengt, Rocca-Serra, Philippe, Roos, Marco, Schaik, Rene C. van, Sansone, Susanna-Assunta, Schultes, Erik Anthony, Sengstag, Thierry, Slater, Ted, Strawn, George O., Swertz, Morris A., Thompson, Mark, Lei, Johan van der, Mulligen, Erik M. van, Velterop, Jan, Waagmeester, Andra, Wittenburg, Peter, Wolstencroft, Katy, Zhao, Jun, and Mons, Barend, The FAIR Guiding Principles for scientific data management and stewardship, Scientific Data **3** (2016).

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