Morloc: A framework for sharing and integrating functions across programming languages

Overview. Morloc is a new, open source computational platform that has broad application to cyberinfrastructure (and other areas). The project consists of **three components**:

- **a semantic type system** for unifying programming languages
- **a novel compiler** for integrating functions from across languages into high-performance programs
- **a community portal** for curation of a universal library of functions and machine-accessible knowledge about their use

Semantic Type System. The semantic type system is the foundation of our project. It is an approach to unifying programming languages under a single type system. In general, type systems allow humans to encode knowledge about a program. This knowledge can be used by a machine to reason about properties of the program. The knowledge is also beneficial to humans as formal documentation and abstraction from implementation details. If the goal of a type system is to encode usable knowledge, then it seems reasonable to build a type system explicitly based on knowledge engineering. Thus, in Morloc type signatures and entire workflows compile into a list of logical statements (e.g., in Relation Data Format, RDF). This allows machines to reason on the entire program, and to incorporate new knowledge, using the standard semantic toolkit. In this context, typechecking involves checking the logical consistency of the program.

Below is an example of a type signature for the function "sample"

```
sample :: n:Integer, xs:[a] -> ys:[a] where {
   n >= 0
   length(ys) == min(n, length(xs))
}
```

This signature describes a function that returns *n* random elements from the vector *xs* as a new vector *ys*. The vectors, of type *[a]*, contain elements of generic type *a*, which can be anything. The *where* clause introduces a list of constraints. First, at least 0 elements must be sampled. Second, the length of the output vector must be the minimum of *i* and the length of *xs*. These two constraints describe the edge conditions of the function. The compiler can use this knowledge to generate assertion statements, to generate test data, or to perform compile-time analysis of a workflow (e.g., since *sample* can return 0-length lists, a warning would be raised if a function of its output requires lists of length 1 or greater).

The type system can be augmented with domain knowledge to provide rich machine and human accessible documentation, powerful and highly customizable automatic programming, and data validation. The domain knowledge is encoded as relations between types or as annotations of types with special information. Examples of the knowledge that may be encoded include inheritance relationships between types (e.g., WindSpeed inherits from Speed), associations between functions and the functions that benchmark or predict their performance, associations between data types and random generators, and identification of conversion functions.

The Morloc Compiler. The type system is integrated with the compiler, which can build high-performance workflows from multi-lingual compositions of functions. The nodes in a Morloc pipeline are pure, idiomatic functions. The programmer does not have to write a plugin or wrapper; they are responsible only for the pure implementation of the function and specification of its type. Morloc automatically weaves the functions together and generates serialization wrappers around functions that make calls across languages. In the worst case, where all function calls are to foreign languages, the performance of Morloc is equivalent to that of current multi-lingual approaches such as standard UNIX pipelines, workflows platforms (e.g., Galaxy), and Make-like scientific build systems (e.g., snakemake) since they rely on serialization between nodes. The key difference is that in the Morloc ecosystem, the compiler, not the programmer, is responsible for writing the wrappers. Morloc will achieve high-performance scaling by generating Common Workflow Language specifications from Morloc programs. This will allow Morloc programs to be run on local clusters or in the cloud through Arvados.

Community Portal. Our grand vision is to build a universal library wherein computational tools and knowledge about their use are curated by an open community. The Morloc type system and the ontologies of types are intended to act as a common ground for discussing, comparing, and building our collective toolset. Functions written in supported languages may be imported as they are, the Morloc database needs only the raw source code and a Morloc type signature.

Together, these three components create a scientific ecosystem with improvements in the following areas:

- **Usability.** New functions may be uploaded to the Morloc library without having to build and maintain applications or plugins. They may then be immediately tested on new datasets and seamlessly composed into new workflows.
- **Quality.** Morloc's common type system and interoperability handling allows functions to be easily benchmarked, to be compared to similar functions, and to share test frameworks.
- **Discoverability.** Every aspect of the Morloc ecosystem such as functions, workflows, domain knowledge, and other metadata such as benchmark results, user comments, author/license info, may be searched using the elegant SPARQL query language. Importantly, the searched content (at least type signatures and workflows) is *machine-verifiable*, in contrast to informal, user-generated metadata tags.
- **Accessibility.** Functions from across languages are curated and distributed through a common archive.
- **Flexibility.** Like the semantic web, the Morloc library is designed to evolve as new kinds of knowledge are added and support for new kinds of reasoning is implemented. Unlike the semantic web, the Morloc knowledge base is strict and must be internally consistent.

Related Work. There are many scientific platforms that allow scripts or web services to be integrated into workflows (e.g., Galaxy, Taverna and OmicX). Unlike these platforms, Morloc uses *functions* not scripts/services. This key distinction makes the addition of new tools trivial. The existing platforms focus heavily on the non-computational users, and in doing so they sacrifice power for ease of use. In contrast, Morloc is intended to give absolute power and freedom to the programmer (they can write in any language and style they want). This will allow whole libraries to be easily uploaded with only the addition of type signatures. While other platforms do offer resource discovery, Morloc is founded on an inherently searchable model where every feature of every uploaded workflow, and every function in the library, compile into a common, machine-verifiable semantic database.

Current Status. We are in the early stages of developing a working prototype compiler. We designed a working Morloc scripting language that can integrate functions from Bash, R, and Python. It supported higher-order functions, loops, automatic cache handling, and generation of command-line user interfaces. Primitive types were supported but not the full semantic type system. We are now in the early implementation stages of our second prototype, which is re-designed from ground up with the semantic type system as a key focus.

Goals if Funded. If funding is secured the first-tier goals include:

1. Formalization of the semantic type system and design of a core ontology of types
2. Construction of a compiler prototype that can integrate Python, R, Bash, Haskell and C++
3. Extensive testing and refinement of the usability of Morloc through case studies in bioinformatics

Conclusion. If successful, Morloc will create a new programming environment and user community that will allow repurposing of the vast trove of academic "abandonware", create new ideas and opportunities for language-agnostic programming, and stimulate new research and development directions in fields ranging from analytics and informatics to artificial intelligence. Given the current acceleration of machine intelligence, we believe the best and most lasting frameworks will be those that make knowledge and tools available to the machine. This is precisely what we hope to achieve with Morloc through the unification of access to programmatic tools and creation of an elegant system for encoding knowledge about their operation.