Stanza Implementation Plan

Patrick S. Li

April 27, 2016

1 Current Status

The silent release version of Stanza3 is ready to be announced. The core library is fairly systematic, the language feature set is stable, and the FFI is full featured. The language can be comfortably used to write applications of up to 30000 lines of code. The FFI allows for easy connection to existing C/C++ libraries and thus with a minor initial investment to write glue bindings, Stanza can be used to write interactive graphical applications.

2 Planned Improvements

The following improvements still have yet to be implemented. Improvements are categorized by the subsystem affected.

Ecosystem

- Stanza requires a port to the Windows operating system.
- Stanza needs an out-of-the-box binding to a cross-platform windowing package. QT seems appropriate but the glue bindings have yet to be written.

Core Library

- The core library requires more polishing, particularly with respect to the files and directories functionality.

Core Language

- The macro system needs to be improved to better support types and name shadowing. The current system behaves poorly when attempting to mix multiple syntax packages. Higher order rules as seen in OMeta are also useful and should be looked into.
- The type inferencer needs to be improved. More of the inference logic needs to be moved into the constraint solver instead of in the brittle generation of custom constraints. If this is done well, then we could progress to a fixed-point solver that is able to compute more return and variable types.
- The type system is missing the ability to specify bounds on type parameters. This is not a big issue for application code, but library datastructures feel restrictive.
- We cannot currently obtain a pointer to a stack allocated value. This is particularly useful for calling C functions that return values by writing to a supplied pointer. Some thought is needed to determine whether it should be allowed. Allowing pointers to stack allocated values implies that the stacks are not relocatable by the garbage collector.

Runtime Features

- Stanza requires better runtime error messages. Types should be printed for cast errors. Object types should be printed out when there are no matching branches, or when there is an ambiguous method. Core functions should not be a part of the stack trace.

Runtime Performance

- We need a generational garbage collector. For programs that manipulate large, long-lived datastructures, the GC performance is not adequate.

Code Quality

- Many loops using the do operating function are not properly inlined. In general, most of them require partial evaluation with abstract objects. For the sake of performance, elide the dynamic-wind when calling do-seq when unnecessary.
- Non-escaping labels need to be analyzed to be transformed into simple jumps, rather than stack creations.
- Redundant boxing and unboxing needs to be eliminated.
- Assembly IR requires support for jumping to a dynamically calculated code label. This is necessary to support efficient switch and dispatch statements.
- Single and multiple-arity type dispatch needs to be improved.
- LoStanza functions need to be inlined for performance.

Compiler Performance

- The register allocator takes up a significant portion of the compile time. The algorithms and datastructures need to be improved. The allocator engine may have to be rewritten in LoStanza.
- The type inference algorithm naively generates many unnecessary constraints. Simple constraints should have their results computed immediately.
- The resolution stage uses an inappropriate datastructure for handling of scopes. Switch to using a table that exploits temporal locality.

Code Improvement

- The input layer needs to be cleaned up and pkg loading needs to be handled properly.
- Pkg files need to refer to externally defined identifiers through an absolute coordinate scheme. This will allow older packages to reference new ones. The assembly IR needs to be updated to eliminate dependence upon specific numerical IDs for types, and allow for caching of assembly code.
- The KForm layer needs to be cleaned up. Currently the logic for insertion of boundary guards are partially split between the type IR and the KForm IR. LoStanza functions are also unable to be inlined currently.

3 Current Implementation Plan

The most obvious impediment for day-to-day programming is the poor compilation performance. This is the most frequently observed issue with Stanza. Faster compilation performance will also improve my own productivity. For these reasons, initial efforts will focus on lowering compilation times.

There are three approaches being considered for improving the compilation times. The first is to improve the quality of the generated code. Since the compiler is bootstrapped, improved code quality will consequently also improve the performance of the compiler. The second is to simply improve the algorithms underlying each compiler phases. This is substantially more straightforward to do than improving code quality but is also less helpful, while improving code quality will improve the performance of all Stanza programs and also of all compilation phases. The last approach is to cache as much of the previous compilation results as possible, and thus avoid having to recompile most of the program. This last approach is conceptually straightforward, but requires a lot of preliminary work to clean up the code base.