Upstream jank-lang specific patches back to CppInterOp

Name: Iva Fezova

Email: Ivkafezova@gmail.com

Github: https://github.com/Iv-F

Location: Plovdiv, Bulgaria

Mentor: Vasil Vasilev

Project Overview:

This project focuses on upstreaming jank-lang-specific patches to CppInterOp in order to reduce downstream maintenance costs and improve interoperability support within the C++ tooling ecosystem. CppInterOp is a C++ interoperability library built on top of LLVM and Clang that enables programming languages to interact with C++ code using compiler-level information. The jank programming language currently relies on a set of downstream patches to CppInterOp to support its interoperability needs, which are maintained separately from upstream and therefore increase long-term maintenance effort.

By analyzing existing jank-specific modifications, refactoring them into upstream-acceptable components, and contributing them back with proper testing and documentation, the project aims to benefit both the jank language and the broader CppInterOp user base. In addition, the project will follow an incremental and review-oriented approach, prioritizing small and self-contained changes that can be contributed early and independently while remaining compatible with existing LLVM and Clang APIs.

Objectives:

The primary objective of this project is to upstream jank-lang-specific patches into CppInterOp, transforming them from downstream-only modifications into well-designed, maintainable, and generally applicable improvements that align with upstream project standards.

An additional objective of this project is to establish a clear and repeatable process for evaluating downstream patches and determining their upstream suitability. By systematically classifying patches based on size, dependencies, and language-specific assumptions, the project aims to prioritize small, self-contained changes that can be upstreamed with minimal refactoring. This approach is intended to maximize the number of accepted upstream contributions within the available time while also providing a useful framework for future patch evaluation and maintenance.

Implementation Details:

The project will begin by reviewing all jank-lang specific patches applied to CppInterOp to understand what each one does. Patches that cannot be accepted upstream as they are will be simplified and refactored by removing jank-specific assumptions and making the functionality more general when possible. Language-specific behavior will be kept separate through simple interfaces or configuration options. All changes will follow CppInterOp's coding style and design rules and will avoid unnecessary changes to existing behavior. Tests will be added to make sure the changes work correctly and do not break anything. Finally, the cleaned-up patches will be submitted as upstream pull requests, feedback from maintainers will be addressed, and accepted changes will allow the jank-lang project to reduce or remove its downstream patch set.

Implementation work will primarily interact with LLVM and Clang APIs used by CppInterOp, with particular attention paid to API stability and version compatibility. Patches will be submitted incrementally to enable early feedback from upstream maintainers and reduce integration risk. This approach allows the project to make steady progress, adapt to review comments, and increase the likelihood that a larger number of patches can be successfully upstreamed within the available timeframe

Schedule:

Time	Deliverables	Milestones
Week 1	● Full patch inventory (~85 patches) ● Initial classification (easy/medium/hard)	Patch inventory and initial classification complete
Week 2	● Prioritized list of 10–12 easy patches for early upstreaming	First 10 easy patches identified and prioritized
Week 3	● Refactoring plan for first 5 easy patches ● Setup dev/test environment	Refactoring approach and environment ready

Week 4	● Refactor & test first 5 easy patches	5 patches refactored and locally verified
Week 5	● Refactor & test next 5–6 easy patches	Total 10–11 patches refactored and tested
Week 6	● Complete refactoring & testing of ≈12 easy patches total	12 patches fully refactored, tested, and ready for CI
Week 7	● Add regression/unit tests for the 12 refactored patches	Test coverage ensured for first 12 patches
Week 8	● Run CI and fix any failures, ensure all tests pass	CI passes successfully for all refactored patches
Week 9	● Prepare PR descriptions & documentation for upstream submission	Documentation and PRs ready for submission
Week 10	● Submit first 4–5 upstream PRs (easy patches) ● Address initial maintainer feedback	First batch of 4–5 patches submitted upstream
Week 11	● Submit next 4–5 PRs (easy/medium patches) ● Iterate on earlier PRs	~20 patches submitted upstream, initial feedback incorporated
Week 12	● Revise PRs based on maintainer comments ● Refactor / simplify downstream patches	PRs updated with review feedback, downstream patches reduced
Week 13	● Complete upstream submissions ● Achieve ~50% (~42–43) accepted patches ● Remove/simplify corresponding downstream patches	~50% of patches upstreamed, downstream stack significantly reduced