SKilLed Communication for Toolchains

Timm Felden
University of Stuttgart, Germany

Abstract—The creation of a program analysis toolchain involves design choices regarding intermediate representations (IRs). Good choices for an IR depend on the analyses performed by a toolchain. In academia, new analyses are developed frequently. Therefore, the best single IR of a research-oriented toolchain does not exist. Thus, we will describe our design of an IR that can be easily adapted to new requirements.

I. BACKGROUND

Successful research projects can outline their dominant programming language. Cooperation with a project using another dominant programming language or usage of a library that is only available for a specific language is only feasible if a language-independent IR is used. An evaluation of serialization technology revealed that any available solution would at least lack one of: independence of programming language, expressiveness, flexibility, maintainability, or resource efficiency.

II. RESEARCH OBJECTIVES

The goal of our research is to find a way of representing an IR for a toolchain, such that the representation is a) not specific to a single research project, b) expressive enough to represent all persisted data structures used in a toolchain, c) general enough to be representable in an acceptable way in most programming languages, d) easy to learn and easy to use, e) change-tolerant, i.e. changes in data structures will only break compiled tools directly operating on exactly those data structures. Further, simultaneous extensions and modifications of unrelated parts of an IR must not influence each other or invalidate data sets unaffected by changes made, f) efficient, meaning that resource consumption can compete with handcrafted IRs. Finally, it should be possible to append to existing files when adding new information only.

1) Methodology: We have created a specification language, SKilL, with the goal of being directly readable by at least 50% of all programmers. This is achieved by an appearance similar to C++ and Java.

The intended usage of SKilL involves an API and implementation generated for a given specification of exchanged data and programming language. That way, the amount of training required for a user is minimal. Furthermore, independent of the used programming language, SKilL is type-safe. In this context type-safe means that if one writes an unsigned 32-bit integer with the value of 42 into a file, it can only be read as the value 42 of the very type. This notion of typesafety provides protection against accidental misinterpretation of data. The type system has been constructed using a formal specification and a minimalistic approach that is expected to be easily representable in most modern programming languages.

Requirements for such a system have been derived by examination of the Bauhaus toolchain, as well as a “what should Bauhaus look like” specification. A binary file format has been used in order to achieve size and performance goals. Binaries store their type information alongside data in order to detect and react to format changes adequately. The published version of the binary file format is an abstraction of problem-specific, hand-crafted solutions – most notably Java Classfiles and LLVM/IR.

2) Status: An initial specification of SKilL [1] has been published already. We expect an updated version to be published this year. The update will contain language extensions enriching the type system. It will provide performance-related improvements discovered in the implementation of back-ends, as well as a cleaner specification of the methodology.

Initial implementations of SKilL include a Scala back-end that proves feasibility of a full implementation of SKilL [2]. Implementations of the core language in Ada and C prove independence of programming language. Furthermore, there are immature implementations of back-ends for Haskell and Java, which may enable joint academic research projects to be carried out. There is a doxygen back-end providing searchable language-independent documentation for a given SKilL specification. Using this test environment, an initial specification of the Bauhaus intermediate language (IML) has been created that consists of 350 type definitions. Thus, we mostly completed tasks a, c, d and e. Tasks b and f will be completed while migrating IML to SKilL, because IML is sufficiently large. The publication of a convincing evaluation showing that we achieved our objectives is planned.

III. EXPECTED CONTRIBUTIONS

We will provide a framework for toolchain-internal communication that is easy to use and understand. Furthermore, the communication is efficient in both size of on-disk representation and runtime. We will prove that the approach scales to a toolchain with at least 200 different user type definitions. We will prove that our approach is sufficiently efficient to deal with IRs obtained from analysing C programs consisting of about 1M LOC. We will provide at least a strong indication that our approach will decrease the development time required for the implementation of new analyses.

REFERENCES
