From A-Mir-Formality to Rust

Towards verifying the backend of the Rust compiler using an executable formal specification

Student

Olivier Lischer

Introduction: Rust is a modern programming language that is often used for systems programming where hardware is often is a constraining factor. A common example is embedded systems programming where software is written for microcontrollers, which power, for instance, medical devices. In many safety critical systems such as medical devices, the legislator requires that the technology used has a formal specification that describes how the technology behaves. The Rust language reference currently specifies this in natural language. However, a formal description of the type system is still missing. The a-mir-formality project aims to close this gap. In computer science literature it is not seldom to express type system rules as proof rules. They are precise and concise, but are typically hard to read for the uninitiated. Another problem is, that such rules are typically expressed in informal mathematical syntax. So to ensure that a given program is correct in respect to the given rules, one has to prove it by hand by using the various judgment rules. The a-mir-formality project solves this problem: the type system rules are written in the Rust language, but in the form of judgment rules. When an input program is feed to a-mir-formality, it can conclude whether the program is type correct, and if not which rules are violated. The a-mir-formality project does not work on the Rust programming language itself, but on its own intermediate language which is based on the Rust language. To ensure equality between the formal model and the official Rust compiler, this project implemented a code generation from the a-mir-formality language to Rust language.

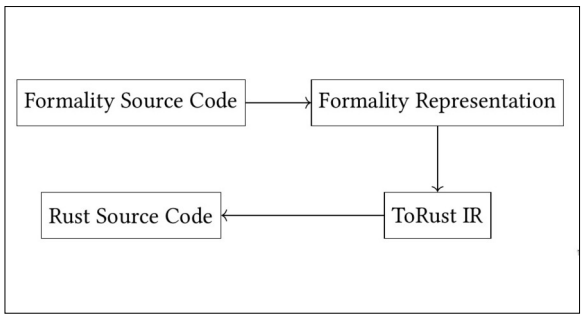
Approach / Technology: In this project, a code generation backend was developed for the a-mir-formality project. For this a new intermediate representation (IR) was introduced that is suitable for generating Rust code from it. The formality representation of a parsed formality source code uses so called de Bruijn indexes to represent generic variables. While these are a good tool to represent variables for code analysis, they cannot directly be used to generated Rust code. Therefore, an approach was developed to transform the indexes to printable types. Further, this project implements an interface for the existing test infrastructure to compile existing a-mir-formality tests to Rust code and check them additionally with the official Rust compiler. If a-mir-formality and compiler come to different conclusions, a discrepancy is detected and the Rust Language Team must decide which project is the right.

Conclusion: This project could implement all the required features, so that all the existing test cases of a-mir-formality can be translated to Rust code and are checked by the official compiler, every time a new pull request is opened in the official GitHub repository. This helps to identify discrepancy between the newly created formal model and the mature Rust compiler.

Rule for deciding the type of Simply Typed Lambda Calculus
Own presentment

```
judgment_fn! {
    pub fn check_type(
        env: Env,
        item: ProgramItem,
    ) => (Ty, Env) {
        debug(env, item)
        (
            (let e = env.extend(LCVar::new(&abs.var, abs.input_ty()).upcast()))
            (check_item(e, &abs.body) => (ty, new_env))
            (if &abs.output_ty() == ty)
            ---("abs has type")
            (check_type(env, ProgramItem::Abstraction(abs)) => (abs.ty(), new_env))
        )
        /* more rules */
    }
}
```

Transformation from the a-mir-formality source code to the Rust source code as implemented in this project
Own presentment



A test case failing because of a discrepancy between a-mir-formality and rustc
Own presentment

```
---- test::basic_where_clauses_pass stdout ----

thread 'test::basic_where_clauses_pass' (102206) panicked at src/test/mod.rs:93:5:
    Checking mycore v0.1.0 (/tmp/formality-Ssl0lG/crates/mycore)
error[E0658]: only lifetime parameters can be used in this context
  --> crates/mycore/src/lib.rs:11:9
   |
11 |     for<T4> u32: A<T4>,
   |         ^^
   |
   = note: see issue #108185 <https://github.com/rust-lang/rust/issues/108185> for more
information
   = help: add `#![feature(non_lifetime_binders)]` to the crate attributes to enable
   = note: this compiler was built on 2025-11-30; consider upgrading it if it is out of
date

For more information about this error, try `rustc --explain E0658`.
error: could not compile `mycore` (lib) due to 1 previous error

note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```


Advisor
Prof. Dr. Farhad D. Mehta

Subject Area
Computer Science

Project Partner
Niko Matsakis - Rust Language Specification team lead