Merge Programming Language

utfeight

September 16, 2023

Contents

1 Merge

NOTE: Any of the features are not implemented yet. It's planned to do so.

1.1 TLDR

New approach to programming languages with generic multilang communication system, **allowing you code** with multiple programming languages in a project with a modern automated interface.

1.2 Manifest

Developing a project using multiple programming languages can easily become a complex task. although there are some Simpler solutions to this problem. They generally lack relaibility, performance, stability etc.

This is a common situation that happens to all of us. For example:

> X programming language has a library/framework that I cannot use with Y programming language.

There are some solutions to this problem like:

- Transpilers: Automated X lang to Y lang converters. (just like .png to .jpeg but more complex)
- Rewrite: Rewriting the program in X lang. (e.g fish shell was written in C++, now it is getting rewritten in rust)
- FFI's (Foreign Function Interface)
- ABI's (Application Binary interface)
- Language Communication Protocols (e.g IPC, Shared Memory, Server)
- Wrappers: Custom libraries that wrap around the native language using Language Communication Protocols.

One of the few solution's that I mentioned above are hard to implement in general. Others are:

- Overall Transpilers: Could generate erroneous output (defective transpilers)
- Rewrites: Consumes a lot of time & effort. Because of the human factor.

Because of these strenuous ways of implementing multi-lang interfaces We introdoce you merge-lang

1.2.1 merge-lang

> A generic meta programming language that automates the process of combining programming languages in a project

Merge takes another approach than the other solutions I mentioned.

1. Initial Algorithm
    - Inference (understands the data communication points: more on that in the next section)
    - Constructing the file structure (splitting code to it's pieces by the fewest whispers[1])
    - Compiling
    - Scheduling Execution
    - Runtime

1.2.2 Inference

This is probably the most complex part of merge-lang. (Also the most innovative way) Look at the following example:

```
#[python]
use *;

fn main() {
    python::print("hello, world");

    #[python]
    {
        a = "world"
        print(f"hello {a}")
    }

    python!{
        a = "hello"
        print(f"{a} world!")
        import a from b
        a.hello()

        def python_fn():
            return 5
    };

    let val_py = python!(python_fn());
    let val_py = python::python_fn();

    println!("from rust: {val_py}");
}
```

[1] data transmissions done between programming languages.

1.2.3 File Structure

Writing our own compiler for all languages that we support would be imposible. Because of this merge compiler will split your code into pieces that external languages can understand.

With this tecnique you can integrate any programming language to merge-lang using It's API

1.2.4 Compile

Merge language compiles to rust code, then LLVM IR and ASM.

1.2.5 Scheduling Execution

The dependencies need to evaluate in a way that every language can get the value they need at the runtime in a linear way. (Just like single vs multi-thread apps)

Think this as a major surgery with you (the doctor) and Mr. Clumsy (the nurse)

1. If Mr. Clumsy gives you a cleaver instead of a lancet, the patient'd probably die. So Mr. Clumsy must give you the right tool to do the surgery. But It doesn't end here.
2. If Mr. Clumsy'd give you a lancet one minute later (or before) than the time you need it. The patient'd die again because of haemorrhage. So timing is a must too!

And don't forget that we made a preconception by saying that Mr. Clumsy will give us a thing.

1.2.6 Runtime

It's wanted to see a nice execution sequience between languages that you use.

1.3 Package Manager

merge package manager is pending right now.

2 API

Merge-lang introduces an API to be able to use more and more languages with it.