RETHINKING PATTERN MATCHING: FLOWCHARTS AND PEG PARSING IN A RISC-V C COMPILER

TAO LIU

Rustlab 2026
Italiy



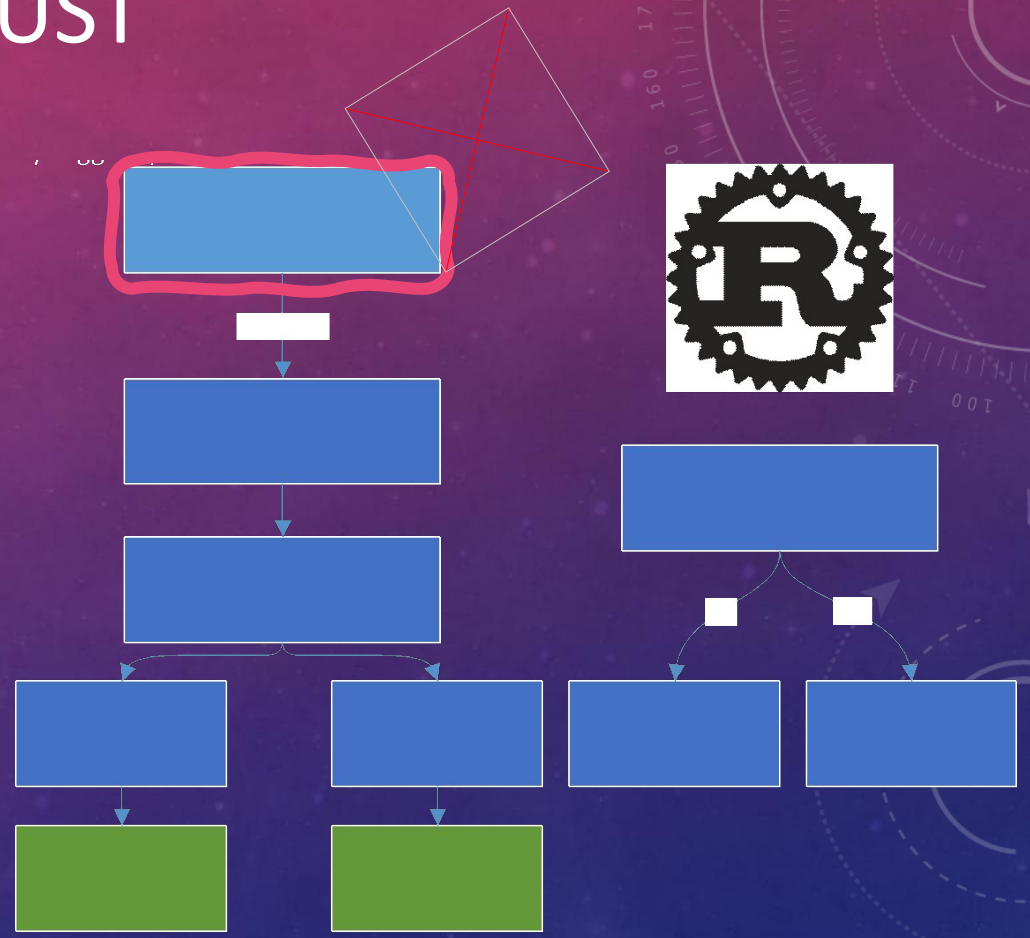
WHO AM I

- Ex-F# core team member in Redmond, WA
- F# for C# Developers by Microsoft Press
- RiscV and RiscV toolchain engineer using Rust after day work
- Make difficult things easy including compilers

COMPILERS FOR RISCV USING RUST

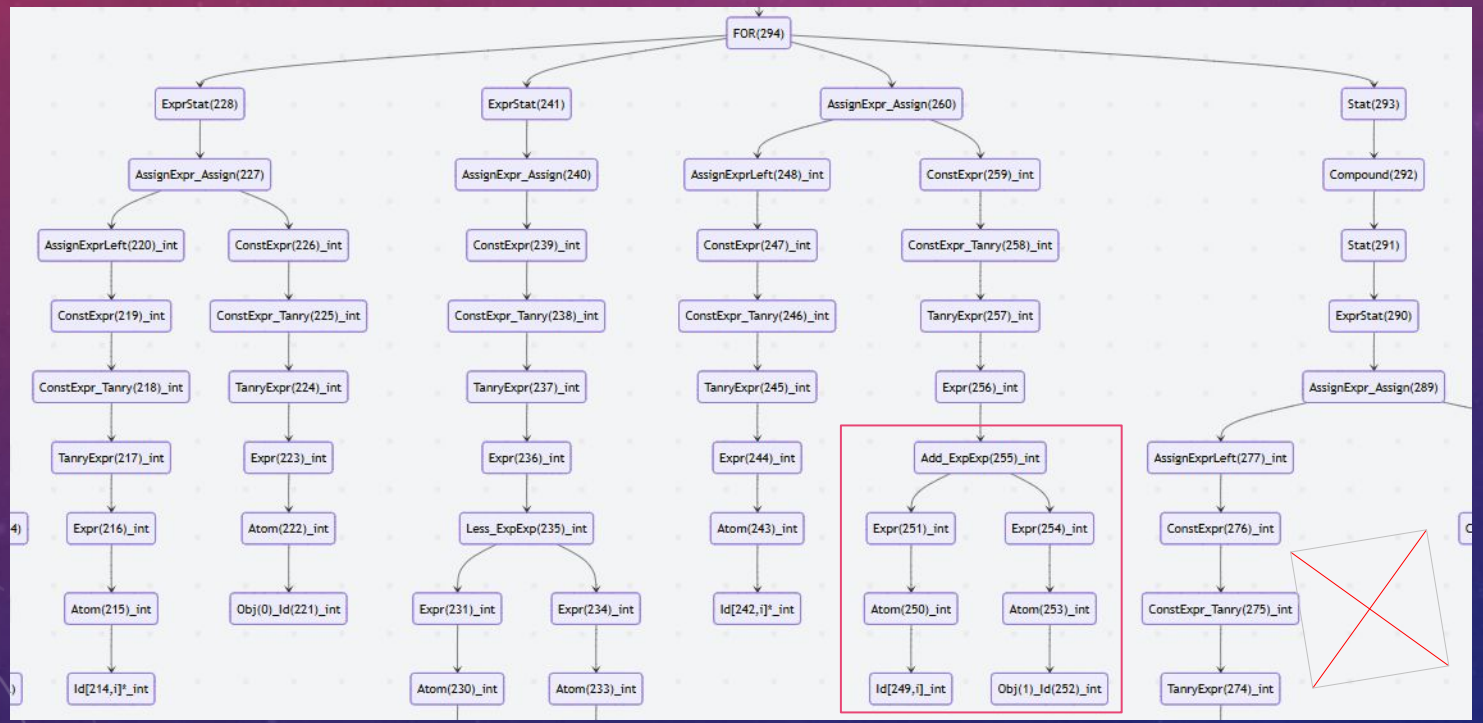
Use Rust to build a user-friendly compiler infrastructure for RiscV from scratch

- Evolving RISC-V: New extension additions becomes common and requires compiler supports
- Development output: Rust has richer language features to enable faster coding compared with C/C++
- Rust: Safer, modular, extensible compiler development.
- Rust is more developer friendly, especially for a new comer into hard core compiler business
    - Rust vs. C++ (LLVM/GCC)



COMPILER

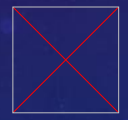
- Compile convert file/string to tree structure (AST) and then lower to instruction/binary item list
- Tree lowering to instruction involves pattern matching, different pattern requires different process



How to describe the pattern?

PATTERN

Define a pattern in mermaid + markdown to find all matching patterns



File can contain multiple diagrams to represent 1+ patterns OR-ed together

iterator var def

this document define a pattern to match the iterator var def definition.

struct name

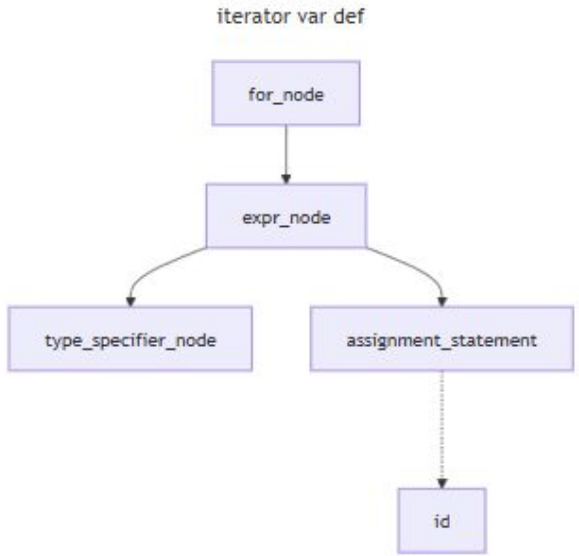
- IterationFor

this is the name of the struct will be visible to invoker.

pattern nodes definition

node_name	definition	is_output
id	is_atom_item_id	y
type_specifier_node	is_typespecifier	y
expr_node	is_expression_statement	y
for_node	is_for_loop	
assignment_statement	is_assignment_expression	

mermaid display



HOW TO MAKE IT HAPPEN?

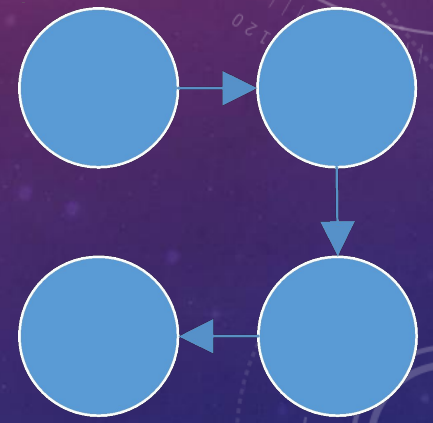
- All the AST node have same type
- Logic programming : ascent!
- Use ascent! to define a graph pattern

```
use ascent::ascent;

ascent! {
    relation edge(i32, i32);
    relation path(i32, i32);

    path(x, y) <-- edge(x, y);
    path(x, z) <-- edge(x, y), path(y, z);
}

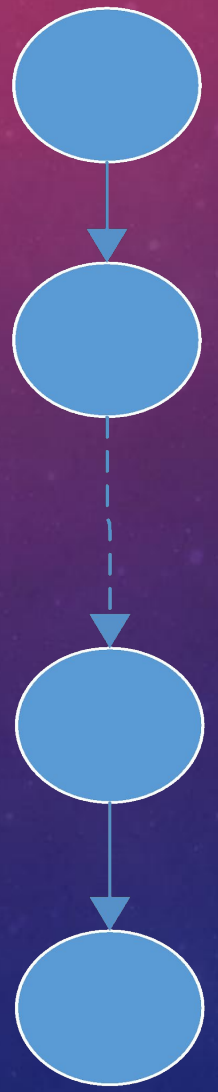
fn main() {
    let mut prog = AscentProgram::default();
    prog.edge = vec![(1, 2), (2, 3), (3, 4)];
    prog.run();
    println!("{:?}", prog.path);
}
```



ANOTHER SAMPLE

```
// rules
single_child_of(child, parent) <--
    node(parent), node(child),
    if parent.is_parent_of(child) && parent.get_children().len() == 1;

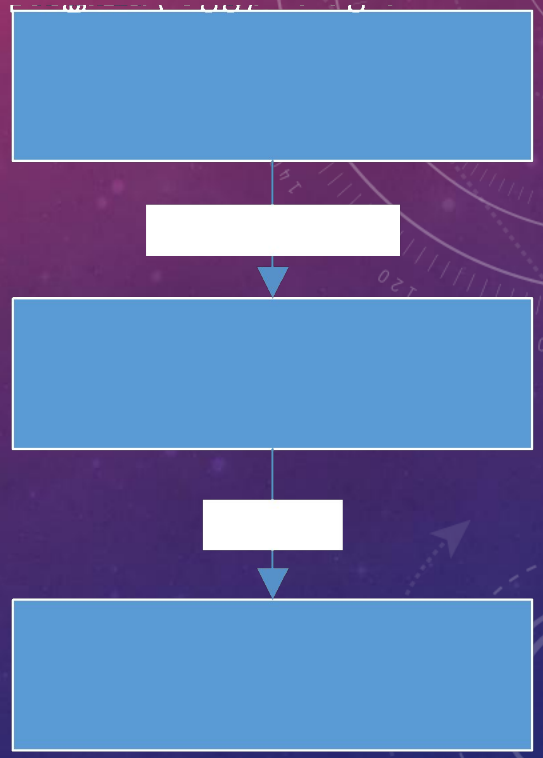
unilinage_of(parent,child) <--
    single_child_of(child, parent);
unilinage_of(parent,grandchild) <--
    single_child_of(child, parent),
    unilinage_of(child, grandchild);
```



GLUE EVERYTHING TOGETHER

- Pest (PEG-based parsing crate) for parsing
- Build.rs generates the rust code from markdown / mermaid
- include! embeds generated code

```
include!(concat!(env!("CARGO_MANIFEST_DIR"), "/src/languages/clang/logic/expr_post_ref.gen.rs"));
```

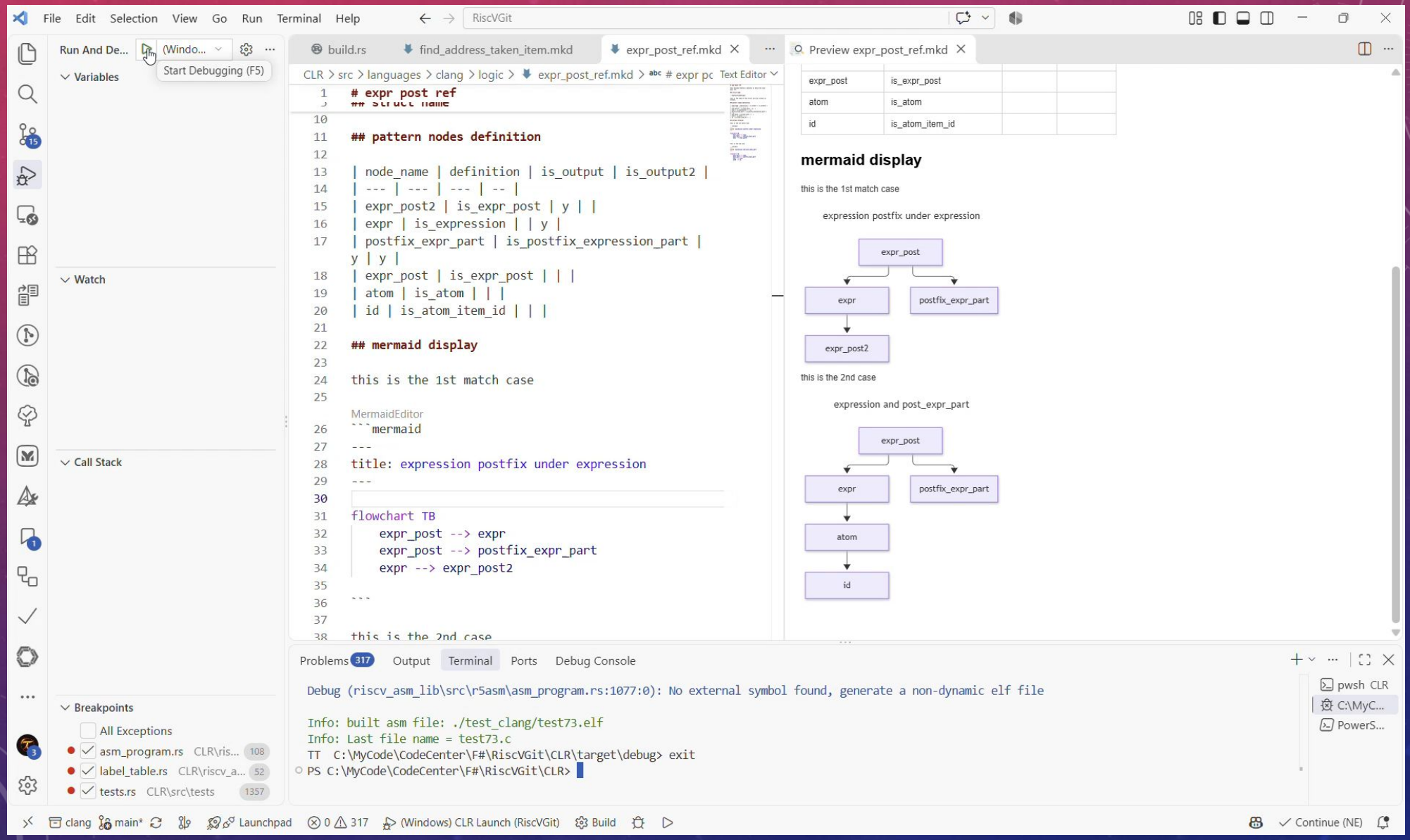


SOME THOUGHTS / TIPS

- Code review friendly for human reviewer
- GPT can be used to check and generate those patterns
- Additional semantic checks can be added to check those pattern (mkd) files
- My experience on using DSL vs. Rust native coding
    - Different folk has different stroke

Open sourced at https://github.com/NomoLynx

DEMO: RUN C TESTS



Q & A