Touring Rustc

Declarative Macros

$ whoami

David Wood

@davidtwco

Rust Compiler Team

Programming Languages Group @ Huawei R&D UK

Touring Rustc?

What is?

Declarative Macros

macros rule! `macro_rules!`

Metaprogramming

Code that writes other code

Macros

Pattern specifying how a input should be replaced by an output

Declarative Macros

```
#define DEBUGPRINT(_fmt, ...) fprintf( \
    stderr, \
    "[file %s, line %d]: " _fmt, \
    __FILE__, \
    __LINE__, \
    __VA_ARGS__\
)

int main(int argc, char *argv[]) {
  int x = 3;
  DEBUGPRINT("x is equal to %d\n", x);
}
```

Declarative Macros

```c
int main(int argc, char *argv[]) {
  int x = 3;
  fprintf(
    stderr,
    "[file %s, line: %d]: x is equal to %d\n",
    "main.c",
    3,
    x
  )
}
```

Declarative Macros

```
macro_rules! vec {
    ( $( $x:expr ),* ) => {
        {
            let mut temp_vec = Vec::new();
            $(
                temp_vec.push($x);
            )*
            temp_vec
        }
    };
}

fn main() {
    let xs = vec![1, 2, 3];
}
```

Declarative Macros

```
fn main() {
    let xs = {
        let mut temp_vec = Vec::new();
        temp_vec.push(1);
        temp_vec.push(2);
        temp_vec.push(3);
        temp_vec
    };
}
```

Detour: Regular Expressions

(you should probably know what these are)

Detour: Regular Expressions

Strings describing search patterns.

```
/^[a-z]+:/
```

https://rust-lang.org/

(matches `"https:"`)

Detour: Regular Expressions

Basically every language has some library for regexes..

```
result = re.search(pattern, input)
```

```
let result = Regex::new(pattern)?.find(input)?;
```

Almost every API looks like these - requiring a pattern and a input.

Declarative macros in rustc

What black magic makes them work?

Declarative macros in rustc

```
macro_rules! printer {
    (print $mvar:ident) => {
        println!("{}", $mvar);
    };
    (print twice $mvar:ident) => {
        println!("{}", $mvar);
        println!("{}", $mvar);
    };
}
```

- metavariable
  - `$mvar`
  - compile-time binding to tree of tokens
- tokens
  - `print` (identifier); `(`, `)`, `{`, `}`, `=>` (punctuation); `EOF` (special); etc.
  - single unit of grammar
- don't care about raw bytes/text/line number/column number
  - (except for use in diagnostics)

Declarative macros in rustc

- `parser`
  - Normal Rust parser + token stream to parse
  - "the input"
  - e.g.
    - `print foo`
- `matcher`
  - Representation of the macro pattern
  - "the pattern"
  - e.g.
    - `print $mvar:ident`

```rust
fn parse_tt(
    &mut self,
    parser: &mut Cow<'_, Parser<'_>>,
    matcher: &[MatcherLoc]
) -> ParseResult {
    /* ... */
}
```

(simplified)

Declarative macros in rustc

Parsing invocations can either be a..

- ..success
  - (producing bindings from metavariables to tokens)
- ..failure
  - i.e. wrong metavariable kind
  - (producing a diagnostic)
- ..error
  - i.e. multiple patterns match
  - (producing a diagnostic)

```rust
fn parse_tt(
    &mut self,
    parser: &mut Cow<'_, Parser<'_>>,
    matcher: &[MatcherLoc]
) -> ParseResult {
    /* ... */
}
```

(simplified)

`parse_tt` has a similar signature to regex functions...

...with an input and a pattern...

...is parsing declarative macros is just parsing regexes?

Declarative macros in rustc

Macro parsing is basically the same as a normal regex parser…

..except parsing different kinds of metavariables.

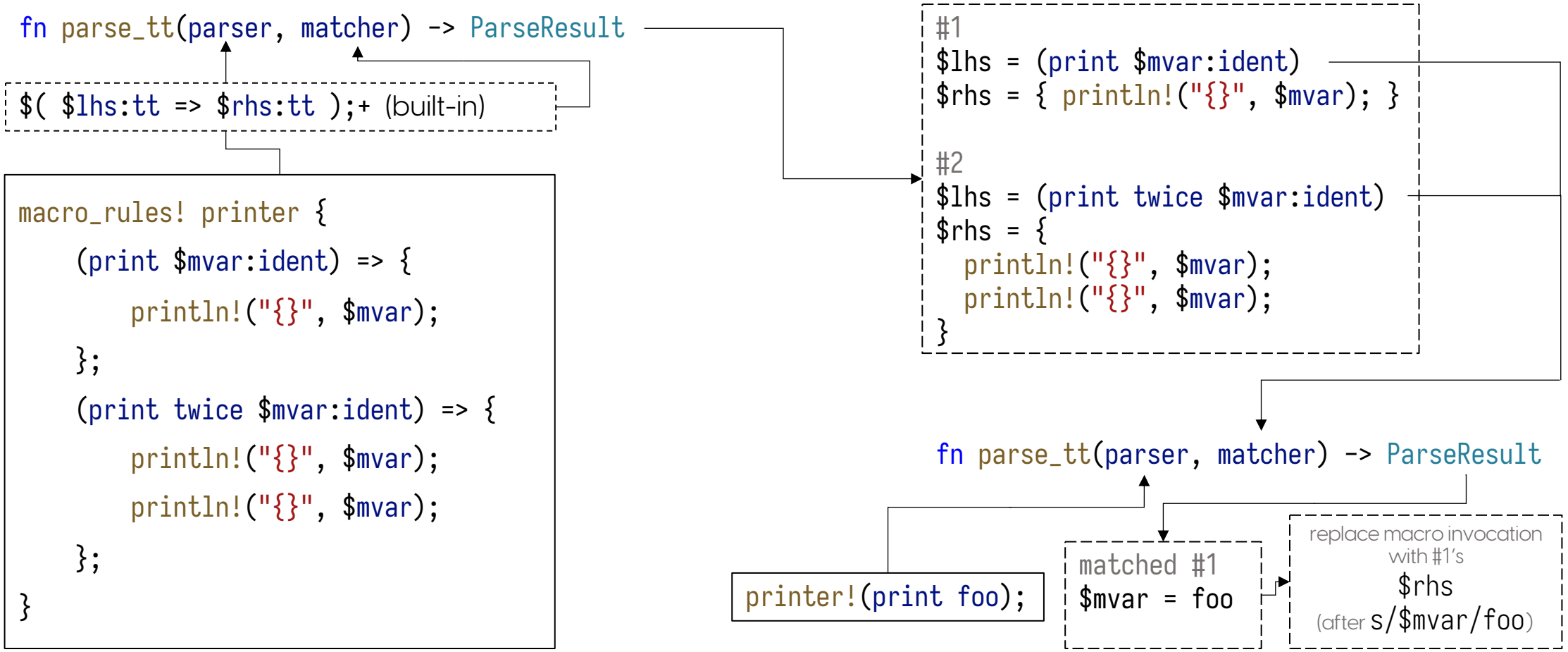
Normal regex parsers have special rules for `\w`, `\d`, `\s`, etc.

Metavariable kinds are similar..

- `$var:expr` calls `Parser::parse_expr`
- `$var:ident` calls `Parser::parse_ident`
- etc.

how do we get the patterns to match against?
what about macro definitions?

Declarative macros in rustc



Thanks!

Feel free to ask questions