Basic MIR

$$
\begin{array}{rcl}
n & \in & \mathbb{Z} \\
x, y & \in & \mathit{Id} \\
f & := & \overline{B : b}; \overline{x : \tau} \\
b & := & \overline{s}; t \\
s & := & x = n \mid x = y \mid x = \mathit{unop}\ y \mid x = y_1\ \mathit{binop}\ y_2 \\
t & := & \mathsf{goto}\ B \mid \mathsf{switch}\ x\ \overline{n : B} \mid \mathsf{return} \\
\epsilon & \in & \mathit{Enum} \\
\tau & := & \mathsf{int} \mid \epsilon
\end{array}
$$

In MIR, a function f is represented with the type `Body`[1], which denotes a control flow graph (CFG) consisting of multiple basic blocks. Each basic block b is represented with the type `BasicBlockData`[2] and consists of multiple statements followed by a single terminator. In a basic block, statements do not have any control flow effect, and only terminators affect the control flow. A basic block is associated with a unique identifier B, represented with the type `BasicBlock`[3]. A statement s has the type `Statement`[4]. While various kinds of statements exist in MIR, we only need to consider assignments. The left-hand side of an assignment is a variable x, and the right-hand side is either an integer n, a variable, a unary operation, or a binary operation. A terminator t has the type `Terminator`[5] and is either a jump to another basic block, a switch, or a return. When returning, the return value is always the value stored in the variable named `_0`. If a function has n parameters, their values are stored in the variables `_1`, `_2`, ..., `_n`. Note that a function declares the type of each variable as well. A type τ is either `int` or a C `enum` type ϵ.

Adding pointers

$$
\begin{array}{rcl}
s & := & \cdots \mid x = \&y \mid x = *y \mid *x = y \\
\tau & := & \cdots \mid *\tau
\end{array}
$$

Adding structs and arrays

$$
\begin{array}{rcl}
F & \in & \mathit{Field} \\
S & \in & \mathit{Struct} \\
s & := & \cdots \mid x.F = y \mid x = y.F \mid x[x'] = y \mid x = y[y'] \\
\tau & := & \cdots \mid S \mid \tau[] \\
p & := & \overline{S\{\overline{F : \tau}\}}; \overline{f}
\end{array}
$$

Adding function calls

$$
\begin{array}{rcl}
g & \in & \overline{\mathit{Function}} \\
p & := & \overline{S\{\overline{F : \tau}\}}; \overline{g : f} \\
t & := & \cdots \mid x = g(\overline{y}); B
\end{array}
$$

Adding function pointers

$$
\begin{array}{rcl}
s & := & \cdots \mid x = g \\
t & := & \cdots \mid x = y(\overline{y'}); B \\
\tau & := & \cdots \mid \overline{\tau} \to \tau'
\end{array}
$$

[1] `https://doc.rust-lang.org/beta/nightly-rustc/rustc_middle/mir/struct.Body.html`
[2] `https://doc.rust-lang.org/beta/nightly-rustc/rustc_middle/mir/struct.BasicBlockData.html`
[3] `https://doc.rust-lang.org/beta/nightly-rustc/rustc_middle/mir/struct.BasicBlock.html`
[4] `https://doc.rust-lang.org/beta/nightly-rustc/rustc_middle/mir/struct.Statement.html`
[5] `https://doc.rust-lang.org/beta/nightly-rustc/rustc_middle/mir/struct.Terminator.html`