FORMAL LAND

ROCQ-OF-RUST SYNTAX IMPORT

MARCH 2026

https://formal.land/

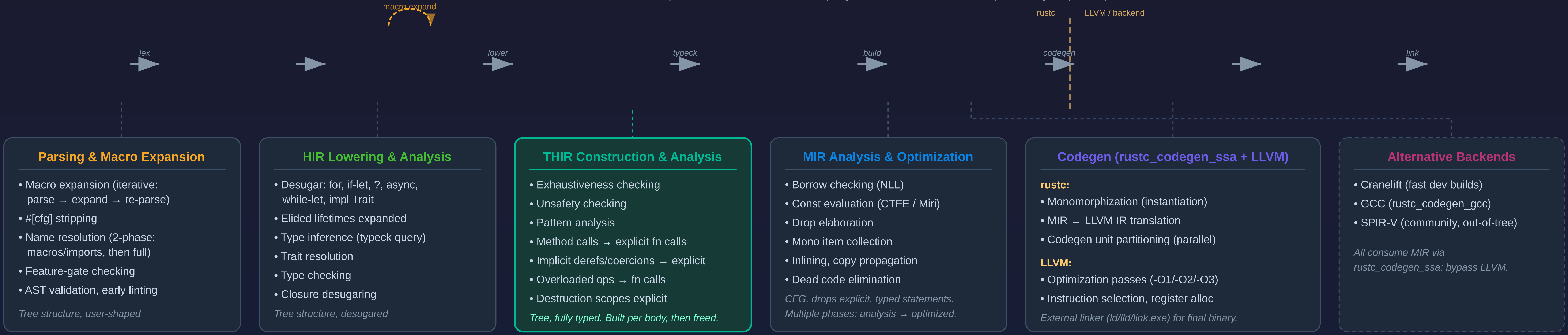
RUST SYNTAX

- Exposed by the Rust **compiler**
- At a **crate** level
- Many **layers** down to the assembly

FORMAL LAND

Rust Compiler (rustc) -- Intermediate Representations Pipeline

Conceptual flow. Actual execution is query-based and demand-driven (not strictly sequential).



Parsing & Macro Expansion

- Macro expansion (iterative: parse → expand → re-parse)
- #[cfg] stripping
- Name resolution (2-phase: macros/imports, then full)
- Feature-gate checking
- AST validation, early linting

Tree structure, user-shaped

HIR Lowering & Analysis

- Desugar: for, if-let, ?, async, while-let, impl Trait
- Elided lifetimes expanded
- Type inference (typeck query)
- Trait resolution
- Type checking
- Closure desugaring

Tree structure, desugared

THIR Construction & Analysis

- Exhaustiveness checking
- Unsafety checking
- Pattern analysis
- Method calls → explicit fn calls
- Implicit derefs/coercions → explicit
- Overloaded ops → fn calls
- Destruction scopes explicit

Tree, fully typed. Built per body, then freed.

MIR Analysis & Optimization

- Borrow checking (NLL)
- Const evaluation (CTFE / Miri)
- Drop elaboration
- Mono item collection
- Inlining, copy propagation
- Dead code elimination

CFG, drops explicit, typed statements.
Multiple phases: analysis → optimized.

Codegen (rustc_codegen_ssa + LLVM)

rustc:
- Monomorphization (instantiation)
- MIR → LLVM IR translation
- Codegen unit partitioning (parallel)

LLVM:
- Optimization passes (-O1/-O2/-O3)
- Instruction selection, register alloc

External linker (ld/lld/link.exe) for final binary.

Alternative Backends

- Cranelift (fast dev builds)
- GCC (rustc_codegen_gcc)
- SPIR-V (community, out-of-tree)

All consume MIR via rustc_codegen_ssa; bypass LLVM.

Why THIR?

THIR bridges HIR → MIR. It carries full type info and makes all implicit operations explicit (coercions, autoderefs, method/operator resolution), simplifying CFG construction.

Built per function body, then freed (arena-allocated).

MIR is the safety backbone

Borrow checking (NLL), move analysis, and initialization checks all run on MIR. Its CFG form enables dataflow analysis that tree-shaped IRs cannot express precisely.

Exists in phases: built → analysis → optimized.

Backend flexibility via rustc_codegen_ssa

MIR is the last rustc-owned IR. The codegen_ssa crate provides a backend-agnostic framework, letting LLVM, Cranelift, and GCC each consume MIR independently.

Codegen units enable parallel compilation.

Query-based architecture

rustc is demand-driven: analyses are queries that call each other with cached results. This enables incremental compilation -- only re-running queries whose inputs changed. Only lexing/parsing/expansion are sequential.

Data structures: 🟧 AST -- Tree 🟩 HIR -- Tree 🟩 THIR -- Tree (typed, per-body) 🟦 MIR -- CFG (basic blocks + edges) 🟪 LLVM IR -- CFG + SSA form 🟥 Machine Code -- Linear instruction stream

Note: MIR and LLVM IR are both SSA-based CFGs, but MIR retains Rust-level types and safety information while LLVM IR is lower-level and target-aware.

Sources: rustc-dev-guide.rust-lang.org (overview, hir, thir, mir, codegen) -- rust-lang/rustc-dev-guide on GitHub

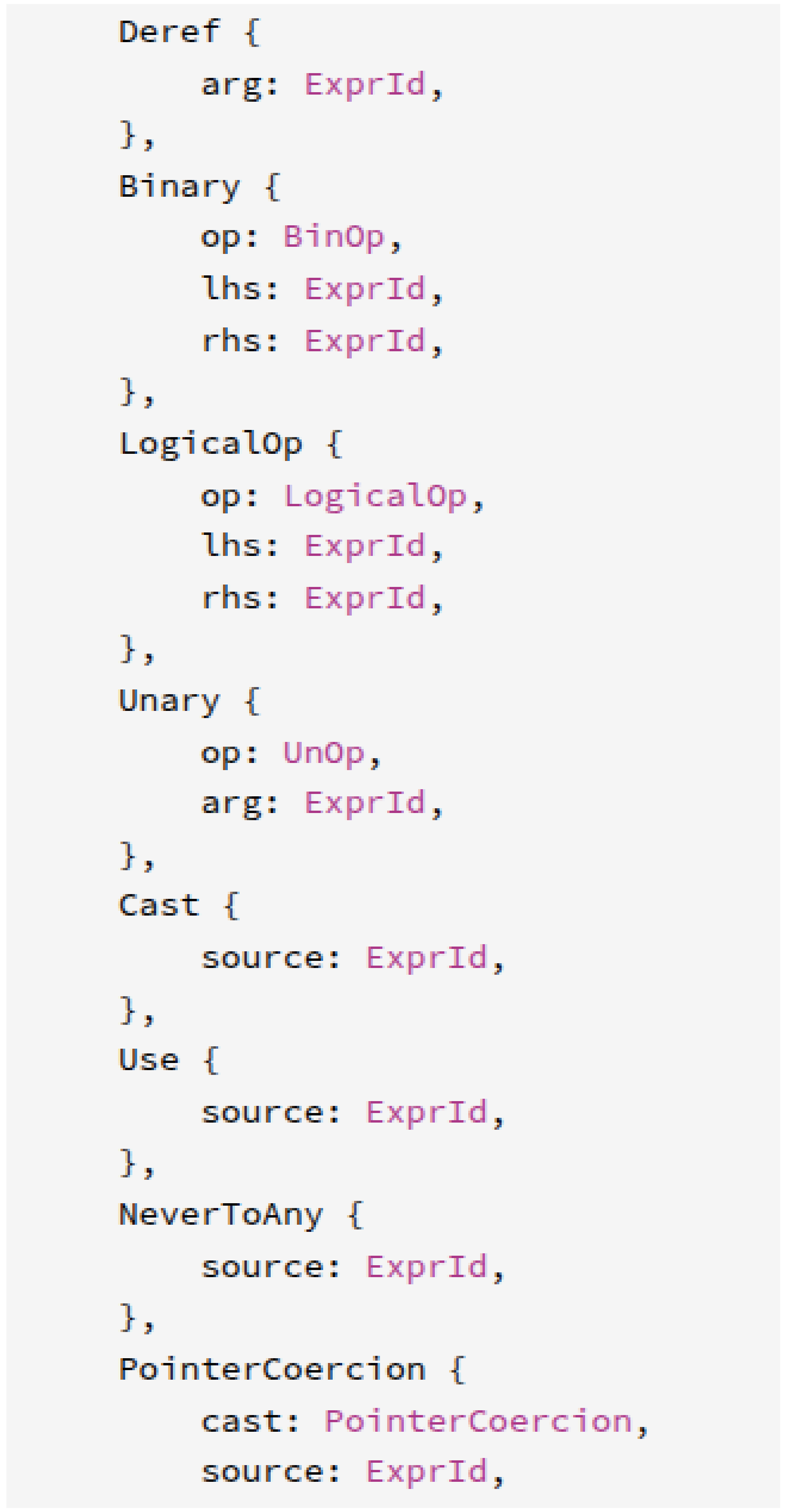
API

- One API to access information
- Cargo integration
- **cargo rocq-of-rust**

FORMAL LAND

THIR EXPRESSIONS

https://doc.rust-lang.org/beta/nightly-rustc/rustc_middle/thir/enum.ExprKind.html



```
    Deref {
        arg: ExprId,
    },
    Binary {
        op: BinOp,
        lhs: ExprId,
        rhs: ExprId,
    },
    LogicalOp {
        op: LogicalOp,
        lhs: ExprId,
        rhs: ExprId,
    },
    Unary {
        op: UnOp,
        arg: ExprId,
    },
    Cast {
        source: ExprId,
    },
    Use {
        source: ExprId,
    },
    NeverToAny {
        source: ExprId,
    },
    PointerCoercion {
        cast: PointerCoercion,
        source: ExprId,
```

FORMAL LAND

ALSO

- Statements
- Types
- Traits, ...

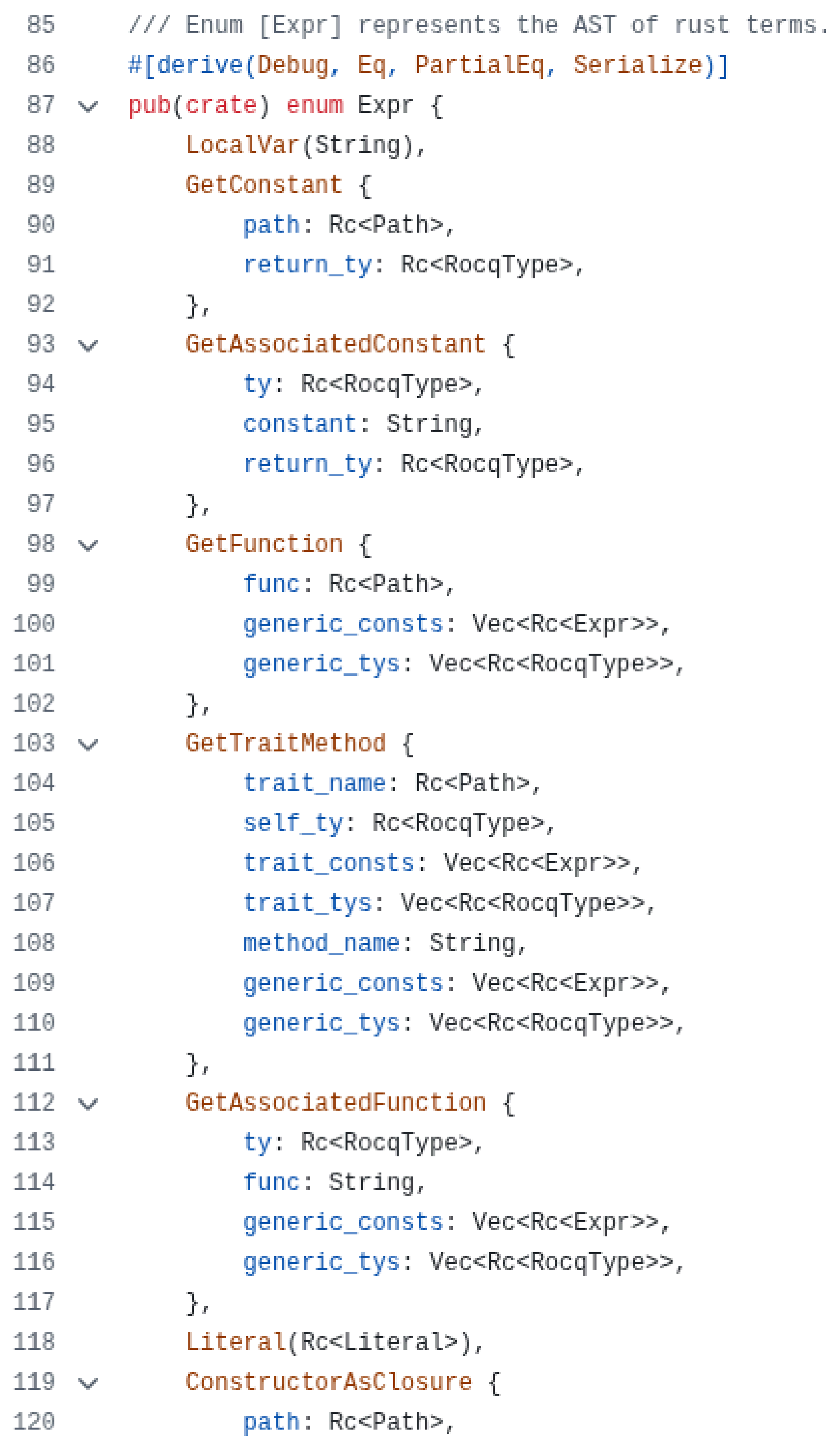
FORMAL LAND

IN ROCQ-OF-RUST

- Around 7,000 lines of Rust
- THIR → **Internal AST**
- Internal AST → **Printing AST**
- Printing AST → **string output**

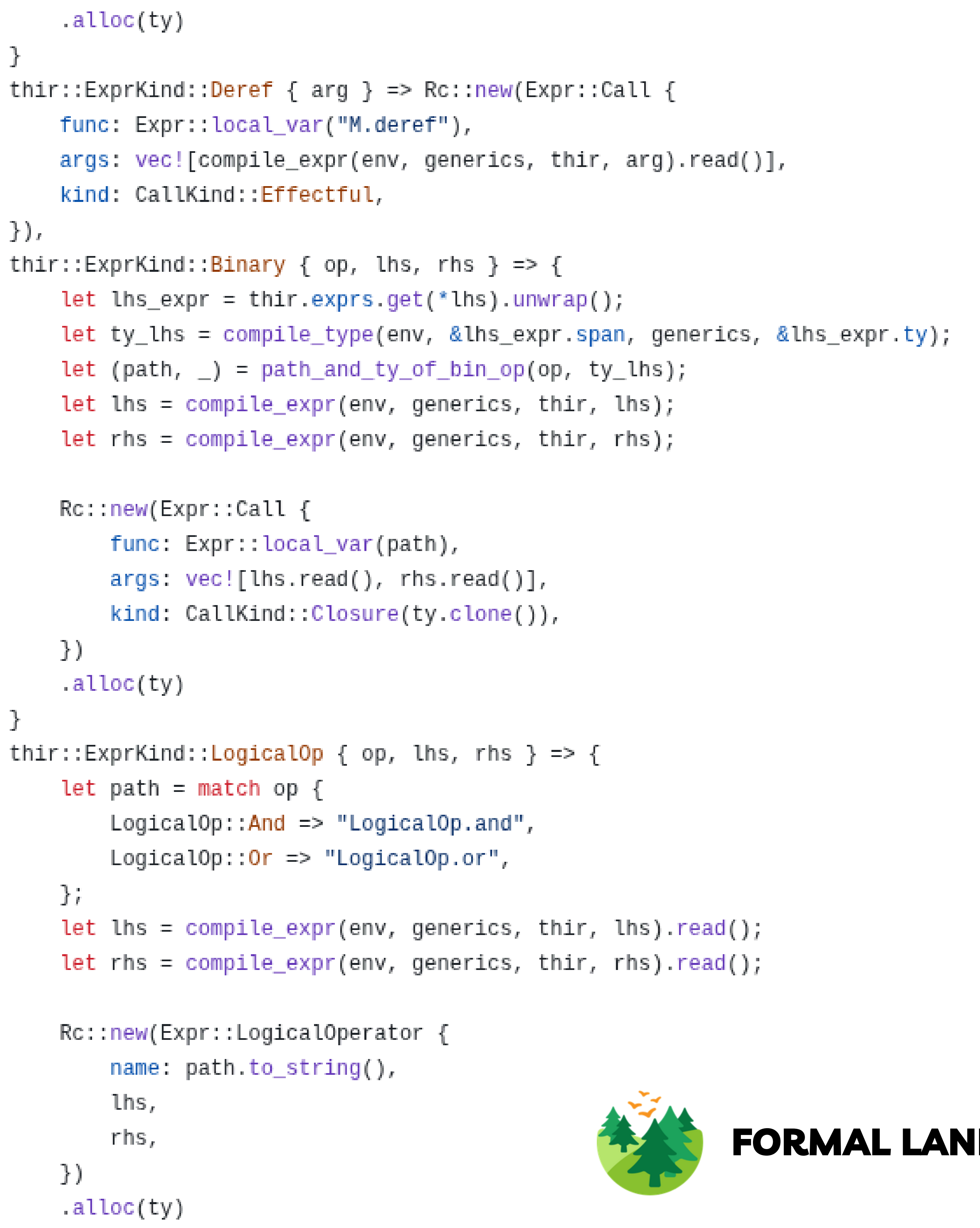
FORMAL LAND

AST



```rust
/// Enum [Expr] represents the AST of rust terms.
#[derive(Debug, Eq, PartialEq, Serialize)]
pub(crate) enum Expr {
    LocalVar(String),
    GetConstant {
        path: Rc<Path>,
        return_ty: Rc<RocqType>,
    },
    GetAssociatedConstant {
        ty: Rc<RocqType>,
        constant: String,
        return_ty: Rc<RocqType>,
    },
    GetFunction {
        func: Rc<Path>,
        generic_consts: Vec<Rc<Expr>>,
        generic_tys: Vec<Rc<RocqType>>,
    },
    GetTraitMethod {
        trait_name: Rc<Path>,
        self_ty: Rc<RocqType>,
        trait_consts: Vec<Rc<Expr>>,
        trait_tys: Vec<Rc<RocqType>>,
        method_name: String,
        generic_consts: Vec<Rc<Expr>>,
        generic_tys: Vec<Rc<RocqType>>,
    },
    GetAssociatedFunction {
        ty: Rc<RocqType>,
        func: String,
        generic_consts: Vec<Rc<Expr>>,
        generic_tys: Vec<Rc<RocqType>>,
    },
    Literal(Rc<Literal>),
    ConstructorAsClosure {
        path: Rc<Path>,
```

THIR → AST



```rust
    .alloc(ty)
}
thir::ExprKind::Deref { arg } => Rc::new(Expr::Call {
    func: Expr::local_var("M.deref"),
    args: vec![compile_expr(env, generics, thir, arg).read()],
    kind: CallKind::Effectful,
}),
thir::ExprKind::Binary { op, lhs, rhs } => {
    let lhs_expr = thir.exprs.get(*lhs).unwrap();
    let ty_lhs = compile_type(env, &lhs_expr.span, generics, &lhs_expr.ty);
    let (path, _) = path_and_ty_of_bin_op(op, ty_lhs);
    let lhs = compile_expr(env, generics, thir, lhs);
    let rhs = compile_expr(env, generics, thir, rhs);

    Rc::new(Expr::Call {
        func: Expr::local_var(path),
        args: vec![lhs.read(), rhs.read()],
        kind: CallKind::Closure(ty.clone()),
    })
    .alloc(ty)
}
thir::ExprKind::LogicalOp { op, lhs, rhs } => {
    let path = match op {
        LogicalOp::And => "LogicalOp.and",
        LogicalOp::Or => "LogicalOp.or",
    };
    let lhs = compile_expr(env, generics, thir, lhs).read();
    let rhs = compile_expr(env, generics, thir, rhs).read();

    Rc::new(Expr::LogicalOperator {
        name: path.to_string(),
        lhs,
        rhs,
    })
    .alloc(ty)
```

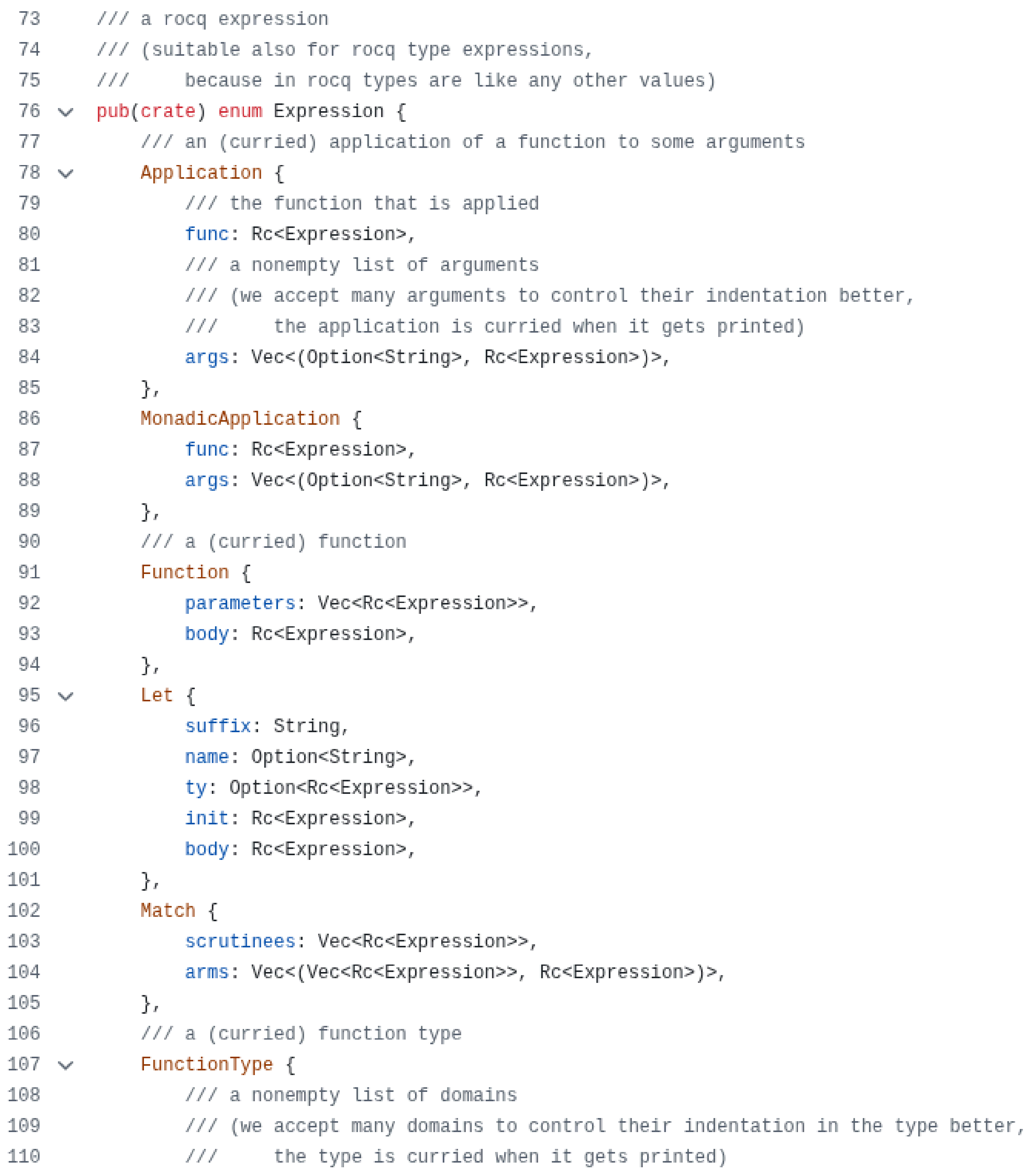
FORMAL LAND

TRANSFORMS

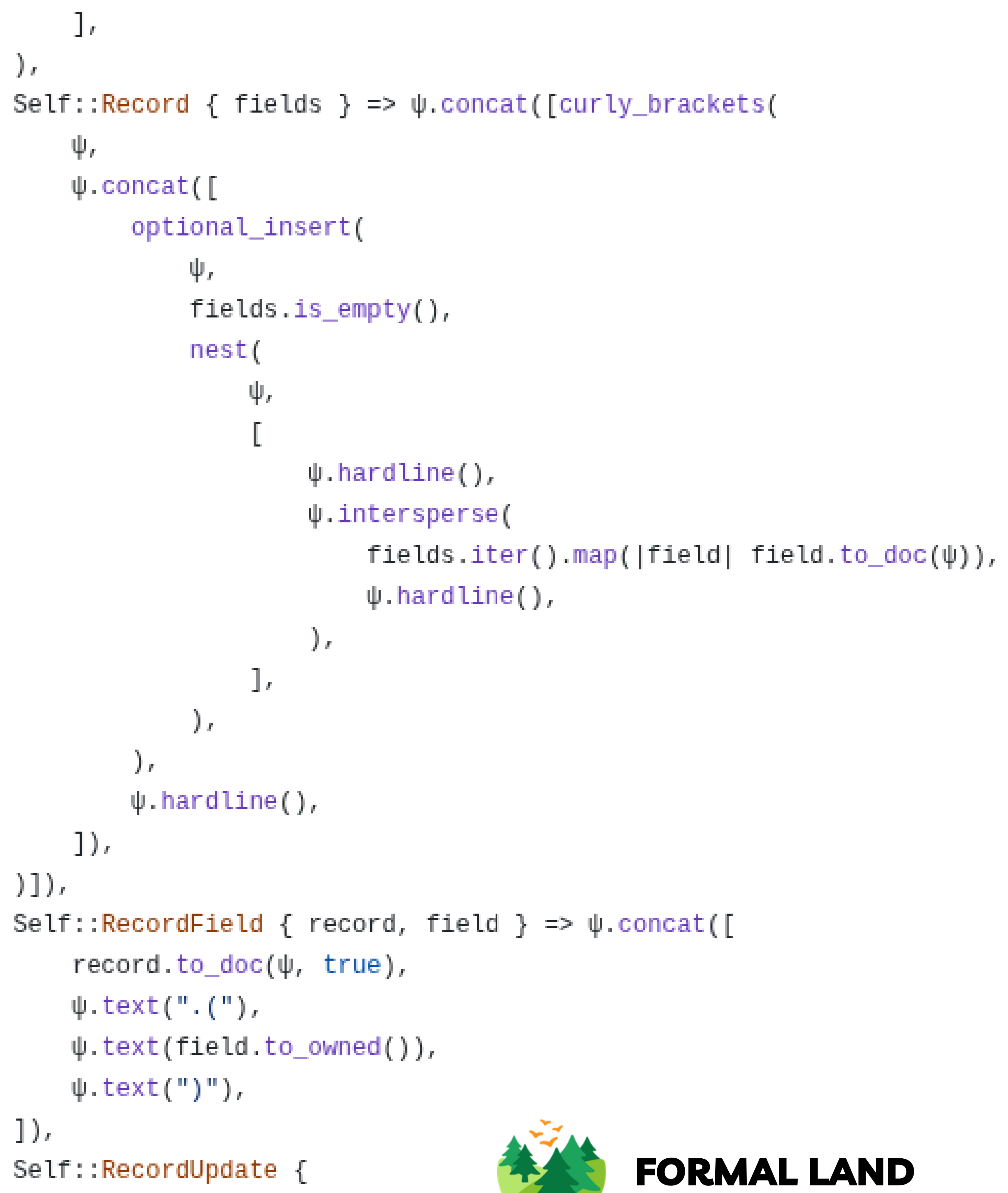
- Most is **one-to-one** + type info
- **Pattern-matching** expansion
- Splitting the crate into **files**
- **Dummy terms** for unknown cases

FORMAL LAND

PRINTING AST → STRING



```rust
/// a rocq expression
/// (suitable also for rocq type expressions,
///     because in rocq types are like any other values)
pub(crate) enum Expression {
    /// an (curried) application of a function to some arguments
    Application {
        /// the function that is applied
        func: Rc<Expression>,
        /// a nonempty list of arguments
        /// (we accept many arguments to control their indentation better,
        ///     the application is curried when it gets printed)
        args: Vec<(Option<String>, Rc<Expression>)>,
    },
    MonadicApplication {
        func: Rc<Expression>,
        args: Vec<(Option<String>, Rc<Expression>)>,
    },
    /// a (curried) function
    Function {
        parameters: Vec<Rc<Expression>>,
        body: Rc<Expression>,
    },
    Let {
        suffix: String,
        name: Option<String>,
        ty: Option<Rc<Expression>>,
        init: Rc<Expression>,
        body: Rc<Expression>,
    },
    Match {
        scrutinees: Vec<Rc<Expression>>,
        arms: Vec<(Vec<Rc<Expression>>, Rc<Expression>)>,
    },
    /// a (curried) function type
    FunctionType {
        /// a nonempty list of domains
        /// (we accept many domains to control their indentation in the type better,
        ///     the type is curried when it gets printed)
```



```rust
    ],
),
Self::Record { fields } => ψ.concat([curly_brackets(
    ψ,
    ψ.concat([
        optional_insert(
            ψ,
            fields.is_empty(),
            nest(
                ψ,
                [
                    ψ.hardline(),
                    ψ.intersperse(
                        fields.iter().map(|field| field.to_doc(ψ)),
                        ψ.hardline(),
                    ),
                ],
            ),
        ),
        ψ.hardline(),
    ]),
)]),
Self::RecordField { record, field } => ψ.concat([
    record.to_doc(ψ, true),
    ψ.text(".("),
    ψ.text(field.to_owned()),
    ψ.text(")"),
]),
Self::RecordUpdate {
```

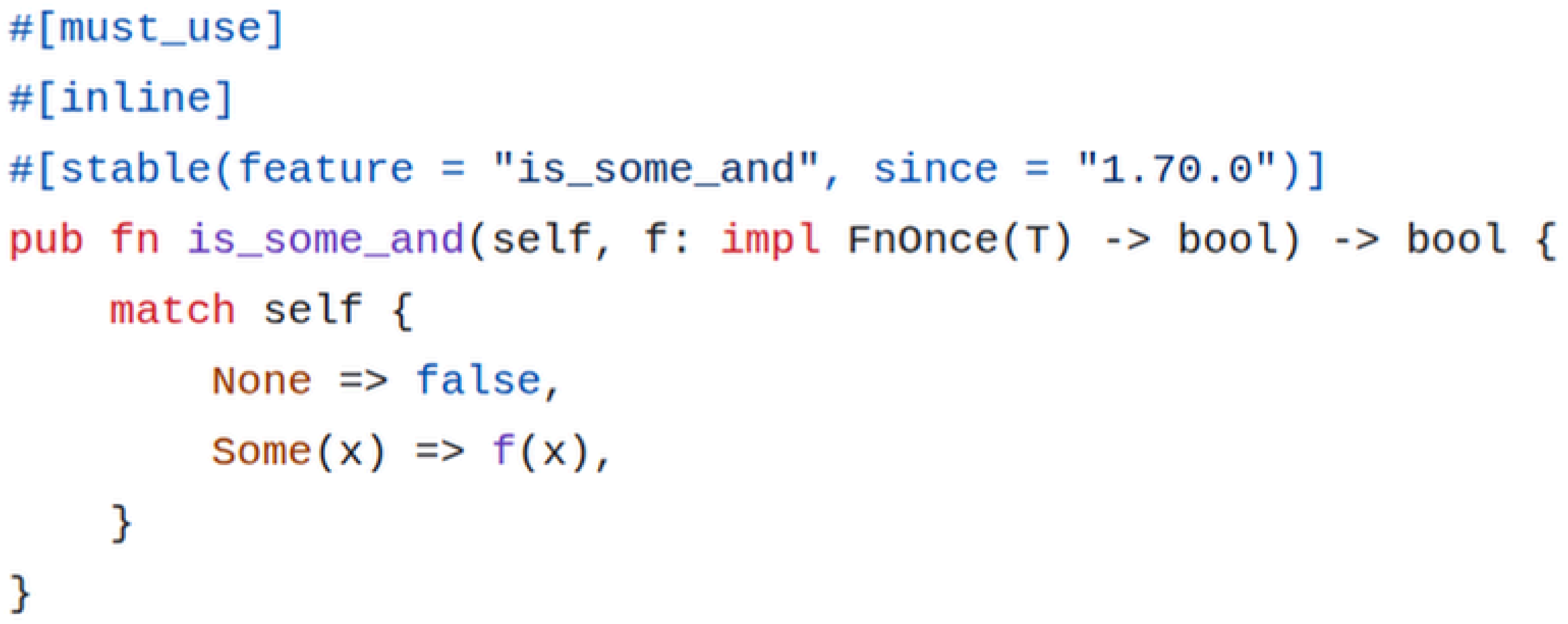
FORMAL LAND

PRETTY-PRINTING

- Using **pretty** crate
- Indentation to be readable

FORMAL LAND

core/option.rs



```rust
#[must_use]
#[inline]
#[stable(feature = "is_some_and", since = "1.70.0")]
pub fn is_some_and(self, f: impl FnOnce(T) -> bool) -> bool {
    match self {
        None => false,
        Some(x) => f(x),
    }
}
```

rocq-of-rust ➡️

core/option.v

```
Definition is_some_and (T : Ty.t) (ε : list Value.t) (τ : list Ty.t) (α : list Value.t) : M :=
 let Self : Ty.t := Self T in
 match ε, τ, α with
 | [], [ impl_FnOnce_T__arrow_bool ], [ self; f ] =>
   ltac:(M.monadic
     (let self := M.alloc (| Ty.apply (Ty.path "core::option::Option") [] [ T ], self |) in
     let f := M.alloc (| impl_FnOnce_T__arrow_bool, f |) in
     M.match_operator (|
       Ty.path "bool",
       self,
       [
         fun γ =>
           ltac:(M.monadic
             (let _ := M.is_struct_tuple (| γ, "core::option::Option::None" |) in
             Value.Bool false));
         fun γ =>
           ltac:(M.monadic
             (let γ0_0 :=
               M.SubPointer.get_struct_tuple_field (| γ, "core::option::Option::Some", 0 |) in
             let x := M.copy (| T, γ0_0 |) in
             M.call_closure (|
               Ty.path "bool",
               M.get_trait_method (|
                 "core::ops::function::FnOnce",
                 impl_FnOnce_T__arrow_bool,
                 [],
                 [ Ty.tuple [ T ] ],
                 "call_once",
                 [],
                 []
               |),
               [ M.read (| f |); Value.Tuple [ M.read (| x |) ] ]
             |)))
       ]
     |)))
 | _, _, _ => M.impossible "wrong number of arguments"
 end.
```

DIFF FRIENDLY

- **Diff in the translation** proportional to **diff in the Rust** source
- Keep ordering, file structure
- Few generated names

FORMAL LAND

NEXT

Definition of **THIR** primitives

in **Rocq**.

FORMAL LAND

LINKS

- https://github.com/formal-land
- https://formal.land/blog

FORMAL LAND

THANKS

FORMAL LAND