[Pluto]

scop → prog → prog(T) → .cloog → CloogProgram → C LAST

(when pprinted we see c-syntax for loops & statements)

Since CLooG is meant for generating syntactic code (mainly C code), it cannot be used directly: CLooG generates an internal representation called **CLAST** which is a simple abstract-syntax tree containing only loops, conditions, and statements.

[Polymer]

MLir Preprocessing

1. Wrapping w/ mlir module
2. Spitting out private func e.g @S0()
3. Tag the private func w/ attr {scop.stmt}

further more....

→ Updated MLIR → Convert to OpenScop() from Affine MLIR Loop chunck

```
std::unique_ptr<OslScop> scop =
createOpenScopFromFuncOp(f, srcTable);
```

→ SCOP

[iterDomain, Scattering, Access]

Init plutoContext & Reset Context → Options

```
PlutoProg *prog = osl_scop_to_pluto_prog(scop->get(), context);
pluto_schedule_prog(prog);
pluto_populate_scop(scop->get(), prog, context);
```

Output prog(T)

Transformed SCOP

```
mlir::FuncOp g = cast<mlir::FuncOp>(createFuncOpFromOpenScop(
                                        std::move(scop),
                                        m,
                                        dstTable,
                                        rewriter.getContext(),
                                        prog,
                                        dumpClastAfterPlutoStr)
);
/// Init Cloog Options pointer
/// These Cloog Options will be used to generate CLAST representation
 CloogState *state = cloog_state_malloc();
 CloogOptions *options = cloog_options_malloc(state);

/// Populate Cloog Options
 options->openscop = 1; // The input file in the OpenScop format
 options->scop = scop->get(); // Get the raw scop pointer


/// Prepare Input representation(type: Cloog Input) to generate Cloog from scop
 CloogInput *input = cloog_input_from_osl_scop(options->state, scop->get());

/// This function extracts CLooG option values from an OpenScop scop and updates an existing CloogOption structure with those values. If the
options were already set, they are updated without warning.
/// Why?? After generating Cloog input representation, the options need to be updated by the options of Scop
/// Parameters:
/// scop – Input Scop.
/// options – CLooG options to be updated.
 cloog_options_copy_from_osl_scop(scop->get(), options);


/// Check if there is non empty prog
/// Written by Polymer Authors
/// Why?? After updating Cloog input options by scop options, we need to update the Cloog options by pluto options(i.e. options from prog)
 if (prog != nullptr) updateCloogOptionsByPlutoProg(options, prog);

/// Init CloogProgram
 CloogProgram *program = cloog_program_alloc(input->context, input->ud, options);

 /// Generate cloog_program
 program = cloog_program_generate(program, options);

 /// Convert to CLAST
 clast_stmt *rootStmt = cloog_clast_create(program, options);


/// They are updating the CLAST by reading unrolljam & parallel Pluto options passed as MLIR compiler pass
 if (prog != nullptr) transformClastByPlutoProg(rootStmt, prog, options, prog->context->options);
     /// Both of the functions acting as wrapper around pluto functions
     /// i.e. pluto_get_parallel_loops(). Src: polyloop.c
     /// pluto_mark_parallel(). Src: ast_transform.c
     if (plutoOptions->unrolljam) unrollJamClastByPlutoProg(root, prog, cloogOptions, plutoOptions->ufactor);
     if (plutoOptions->parallel) markParallel(root, prog, cloogOptions);

/// Instantiating the object(deserializer) of Importer class
/// Passing MLIR context, MLIR module, symTable, scop, Cloop options
 Importer deserializer(context, module, &symTable, scop.get(), options);

 if (failed(deserializer.processStmtList(rootStmt)))
   return nullptr;
```

processStmtList() crawls through all the different types of CLAST stmts. And generate MLIR

Holding everything

```
clast_stmt_op stmt_root;
clast_stmt_op stmt_ass;   -> a=b+c
clast_stmt_op stmt_user;  -> S0(t1,t2,t3)
clast_stmt_op stmt_block; -> { }
clast_stmt_op stmt_for;   -> for( ) { }
clast_stmt_op stmt_guard; -> if ( ) , else { }
```