Advanced Topics in Compilers

NOELLE

Simone Campanoni
simone.campanoni@northwestern.edu

Outline

- NOELLE’s code structure
- Building upon NOELLE
- Developing NOELLE

Software framework: NOELLE

- Git repo: https://github.com/scampanoni/noelle

- You need to use LLVM 9.0.0
  - On hanlon.wot.eecs.northwestern.edu:
    LLVM_HOME= /home/software/llvm-9.0.0
    export PATH=$LLVM_HOME/bin:$PATH ;
    export LD_LIBRARY_PATH=$LLVM_HOME/lib:$LD_LIBRARY_PATH
  - On peroni.cs.northwestern.edu
    source /project/extra/llvm/9.0.0/enable

- Try to compile the framework

  $ git clone https://github.com/scampanoni/noelle
  $ cd noelle
  $ make

Software framework: NOELLE

- Problem:
  - LLVM provides low-level and only code-centric APIs to middle-end passes
  - This makes the design of advanced code analyses and transformations hard

- Solution:
  - NOELLE complements LLVM by providing a dependence-centric (and more expensive, unfortunately) APIs at different granularities to middle-end passes
  - Even advanced code transformations (code parallelization, code vectorization, loop transformations) can be now implemented in a few lines of code (less than 1000!!!)
  - NOELLE's APIs are optional and you can combine them with LLVM's APIs
  - For most NOELLE's APIs:
    - You pay the cost of an API provided by NOELLE when you invoke that API

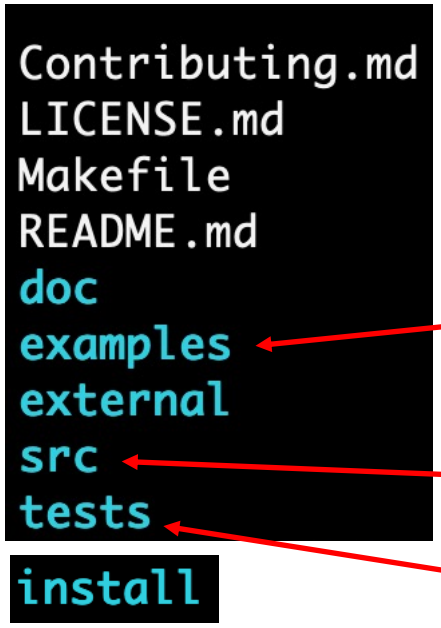
Current limitations of NOELLE

- You can analyze / transform a program, but not a library
  - The existance of main is assumed
  - The whole program is assumed

- The IR code being analyzed/transformed using NOELLE is (at least) normalized using noelle-norm

- You keep track of which abstractions are not longer valid due to changes you have made to the code
  - Suggestion: use all abstractions you need to decide what to do, then do all changes at once
  - Suggestion: you can invoke NOELLE multiple times (learn how to use noelle-fixedpoint)

NOELLE structure



Examples of LLVM middle-end passes built upon NOELLE

NOELLE's internals

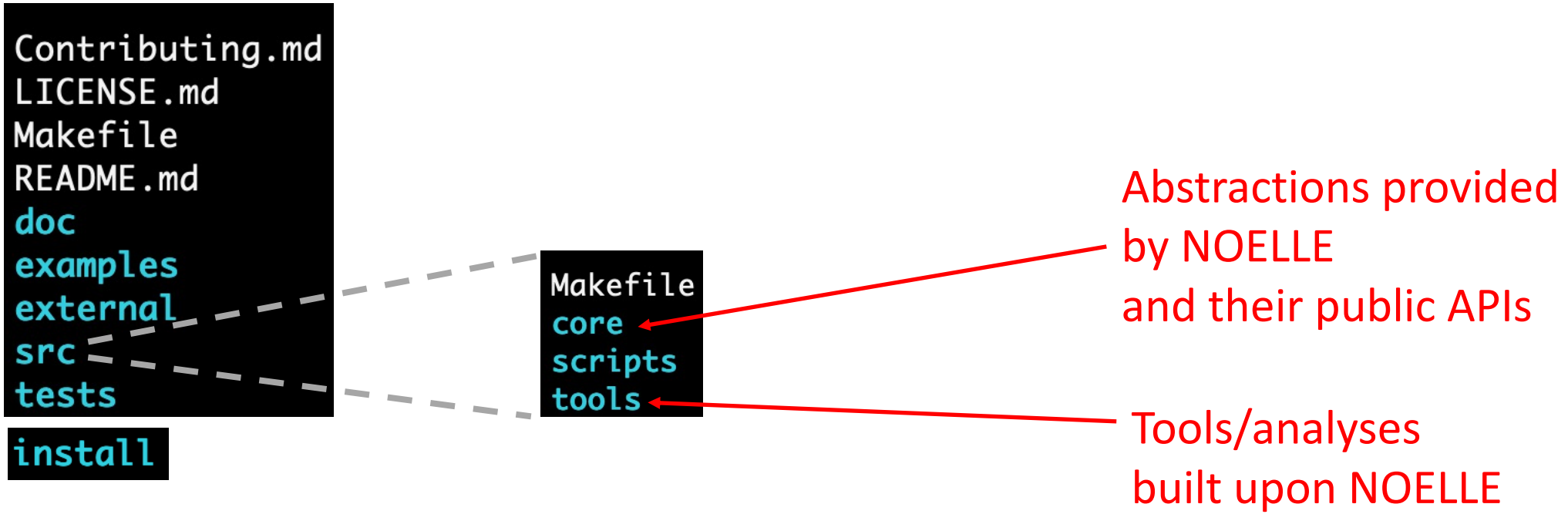
NOELLE's tests

- Unit tests
- Regression tests
- Performance tests

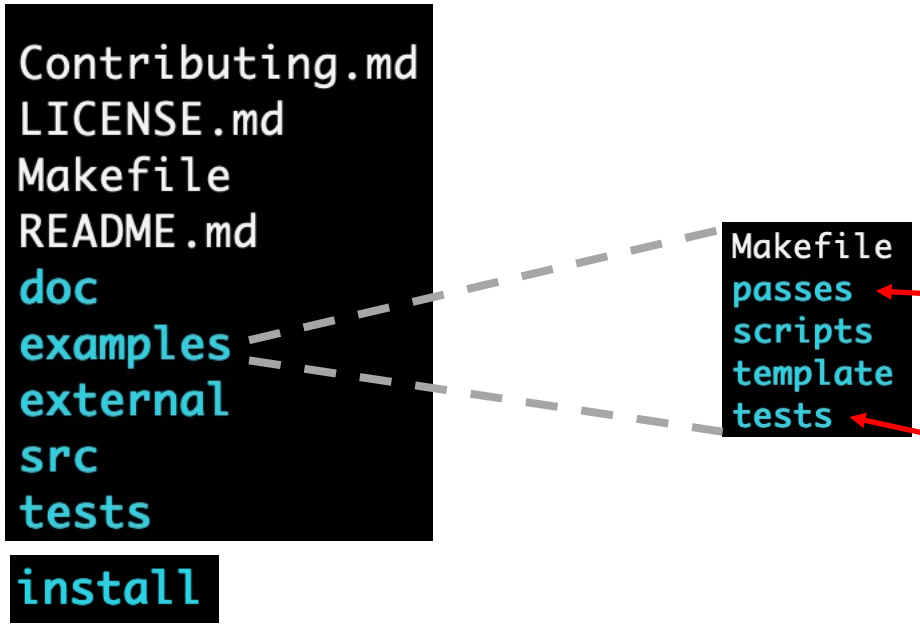
After you compile NOELLE, NOELLE's

- Binaries
- public APIs
- tools

NOELLE structure



NOELLE structure



Outline

- NOELLE's code structure

- Building upon NOELLE

- Developing NOELLE

CatPass.cpp

```
1 #include "llvm/Pass.h"
2 #include "llvm/IR/Function.h"
3 #include "llvm/Support/raw_ostream.h"
4 #include "llvm/IR/LegacyPassManager.h"
5 #include "llvm/Transforms/IPO/PassManagerBuilder.h"
6
7 using namespace llvm;
```

```
 9 namespace {
10   struct CAT : public FunctionPass {
11     static char ID;
12
13     CAT() : FunctionPass(ID) {}
14
15     bool doInitialization (Module &M) override {
16       errs() << "Hel
17       return false;
18     }
19
20     bool runOnFuncti
21       errs() << "Hel
22       return false;
23     }
24
25     void getAnalysis
26       errs() << "Hel
27       AU.setPreserve
28     }
29   };
30 }
```

```
32 // Next there is code to register your pass to "opt"
33 char CAT::ID = 0;
34 static RegisterPass<CAT> X("CAT", "Homework for the CAT class");
35
36 // Next there is code to register your pass to "clang"
37 static CAT * _PassMaker = NULL;
38 static RegisterStandardPasses _RegPass1(PassManagerBuilder::EP_OptimizerLast,
39     [](const PassManagerBuilder&, legacy::PassManagerBase& PM) {
40         if(!_PassMaker){ PM.add(_PassMaker = new CAT());}}); // ** for -Ox
41 static RegisterStandardPasses _RegPass2(PassManagerBuilder::EP_EnabledOnOptLevel0,
42     [](const PassManagerBuilder&, legacy::PassManagerBase& PM) {
43         if(!_PassMaker){ PM.add(_PassMaker = new CAT()); }}); // ** for -O0
44
```

CatPass.cpp

```
#include "llvm/Pass.h"
#include "llvm/IR/Function.h"
#include "llvm/Support/raw_ostream.h"
#include "llvm/IR/LegacyPassManager.h"
#include "llvm/Transforms/IPO/PassManagerBuilder.h"

#include "noelle/core/Noelle.hpp"

using namespace llvm::noelle ;
```

```
struct CAT : public ModulePass {
  static char ID;

  CAT() : ModulePass(ID) {}

  bool doInitialization (Module &M) override {
    return false;
  }

  bool runOnModule (Module &M) override {

    /*
     * Fetch NOELLE
     */
    auto& noelle = getAnalysis<Noelle>();

    /*
     * Use NOELLE
     */
    auto insts = noelle.numberOfProgramInstructions();
    errs() << "The program has " << insts << " instructions\n";

    return false;
  }

  void getAnalysisUsage(AnalysisUsage &AU) const override {
    AU.addRequired<Noelle>();
  }
};
```

Fetch NOELLE

Simple example
of using NOELLE

Declare to LLVM that
your pass depends on NOELLE

Running NOELLE based passes

- noelle-load **rather than** opt

- In 323:
  - opt –load ~/CAT/lib/MYPASS.so –MYPASS A.bc –o B.bc

- Now:
  - noelle-load –load ~/CAT/lib/MYPASS.so –MYPASS A.bc –o B.bc

It will print the invocation to opt with all arguments (in case it will debugging)

```
opt -load /nfs-scratch/simonec/parallelism/parallelization/NOELLEs/2/install/lib/CallGraph.so
    -load /nfs-scratch/simonec/parallelism/parallelization/NOELLEs/2/install/lib/libSvf.so
    ...
   -load /home/simonec/CAT/lib/MYPASS.so -MYPASS A.bc -o B.bc
```

Let’s compile a simple example of code transformation built upon NOELLE

- cd examples/passes

```
callgraph
dfa
dfa2
dfa3
induction_variables
loops
Makefile
pdg
profile
simple
```

- make links ; cd simple

```
CMakeLists.txt -> ../../template/CMakeLists.txt
scripts -> ../../template/scripts
src
```

- ./scripts/run_me.sh
  It will compile and install the pass to ~/CAT
  (like in 323)

```
struct CAT : public ModulePass {
  static char ID;

  CAT() : ModulePass(ID) {}

  bool doInitialization (Module &M) override {
    return false;
  }

  bool runOnModule (Module &M) override {

    /*
     * Fetch NOELLE
     */
    auto& noelle = getAnalysis<Noelle>();

    /*
     * Use NOELLE
     */
    auto insts = noelle.numberOfProgramInstructions();
    errs() << "The program has " << insts << " instructions\n";

    return false;
  }

  void getAnalysisUsage(AnalysisUsage &AU) const override {
    AU.addRequired<Noelle>();
  }
};
```

Let’s run a simple example of code transformation built upon NOELLE

- cd examples/tests

```
0
1
2
3
4
5
6
7
Makefile
scripts
```

- source ../../enable ;
- cd 0 ;
- make -f Makefile_no_profile

```
clang -O1 -Xclang -disable-llvm-passes -emit-llvm -c test.c -o test.bc
llvm-dis test.bc
noelle-norm test.bc -o test_norm.bc
...
noelle-load -load ~/CAT/lib/CAT.so -CAT test_with_metadata.bc -o test_opt.bc
...
The program has 22 instructions
```

```
struct CAT : public ModulePass {
  static char ID;

  CAT() : ModulePass(ID) {}

  bool doInitialization (Module &M) override {
    return false;
  }

  bool runOnModule (Module &M) override {

    /*
     * Fetch NOELLE
     */
    auto& noelle = getAnalysis<Noelle>();

    /*
     * Use NOELLE
     */
    auto insts = noelle.numberOfProgramInstructions();
    errs() << "The program has " << insts << " instructions\n";

    return false;
  }

  void getAnalysisUsage(AnalysisUsage &AU) const override {
    AU.addRequired<Noelle>();
  }
};
```

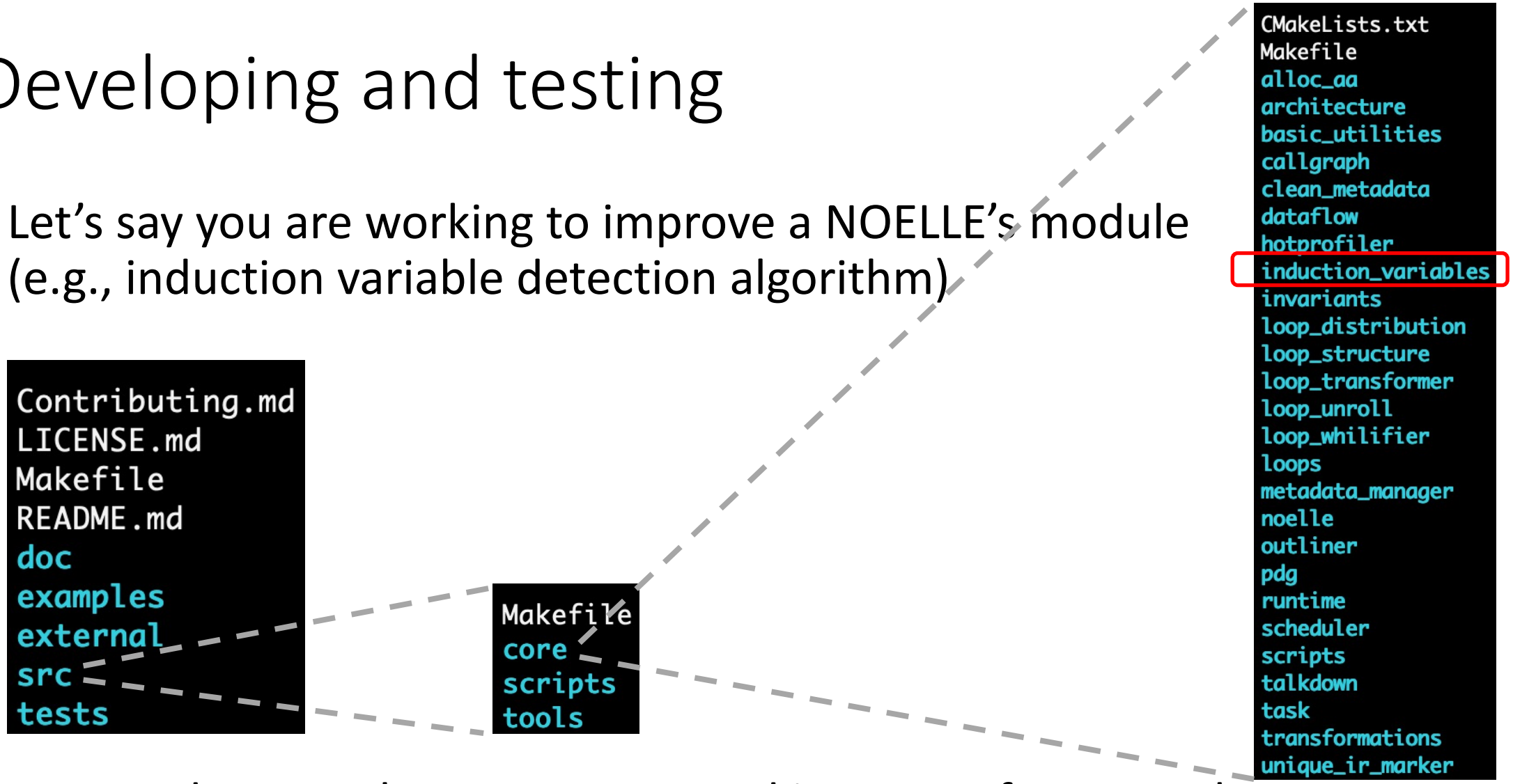
You have to normalize the code before invoking NOELLE

Outline

- NOELLE's code structure
- Building upon NOELLE
- Developing NOELLE

Developing and testing

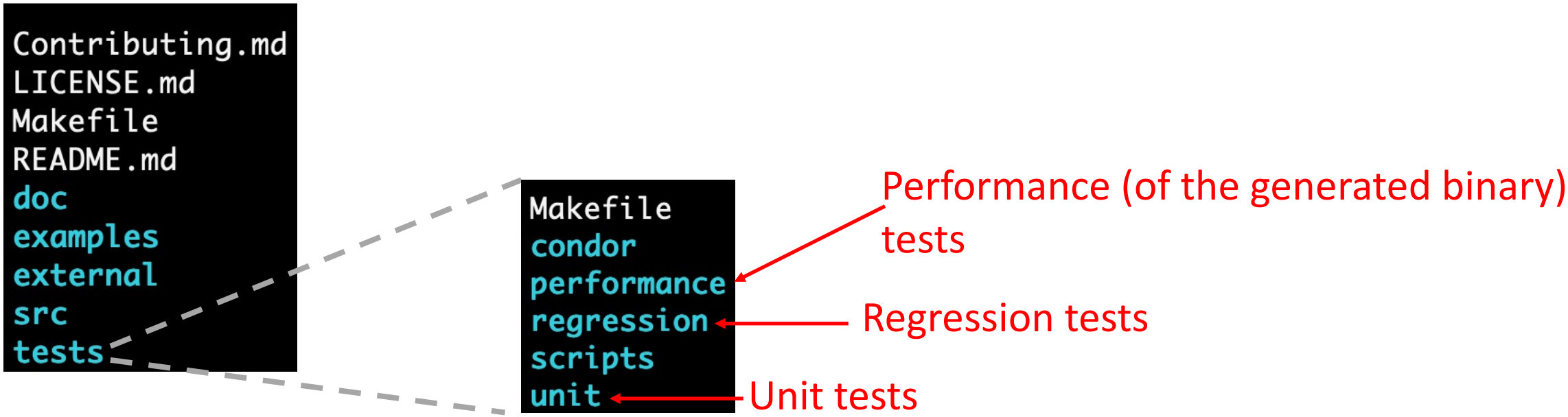
- Let's say you are working to improve a NOELLE's module (e.g., induction variable detection algorithm)



- You need to test the correctness and impacts of your work. …
  - NOELLE can help you do that

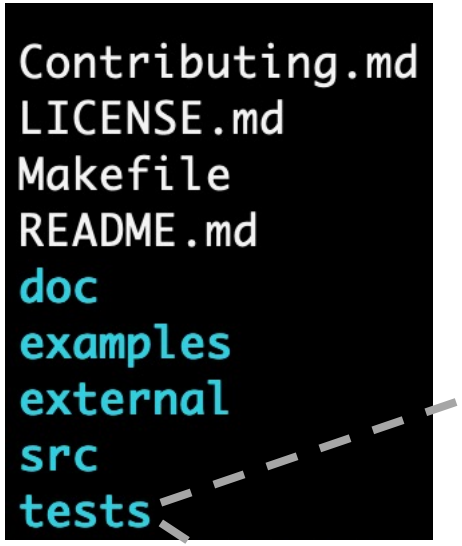
Testing

- NOELLE includes tests for its code transformations (e.g., code parallelization, loop-invariant code motion, etc…)



Testing

- NOELLE includes tests for its code transformations (e.g., code parallelization, loop-invariant code motion, etc…)



cd tests ;

If you don't have condor installed in your platform

If you have condor installed in your platform

make

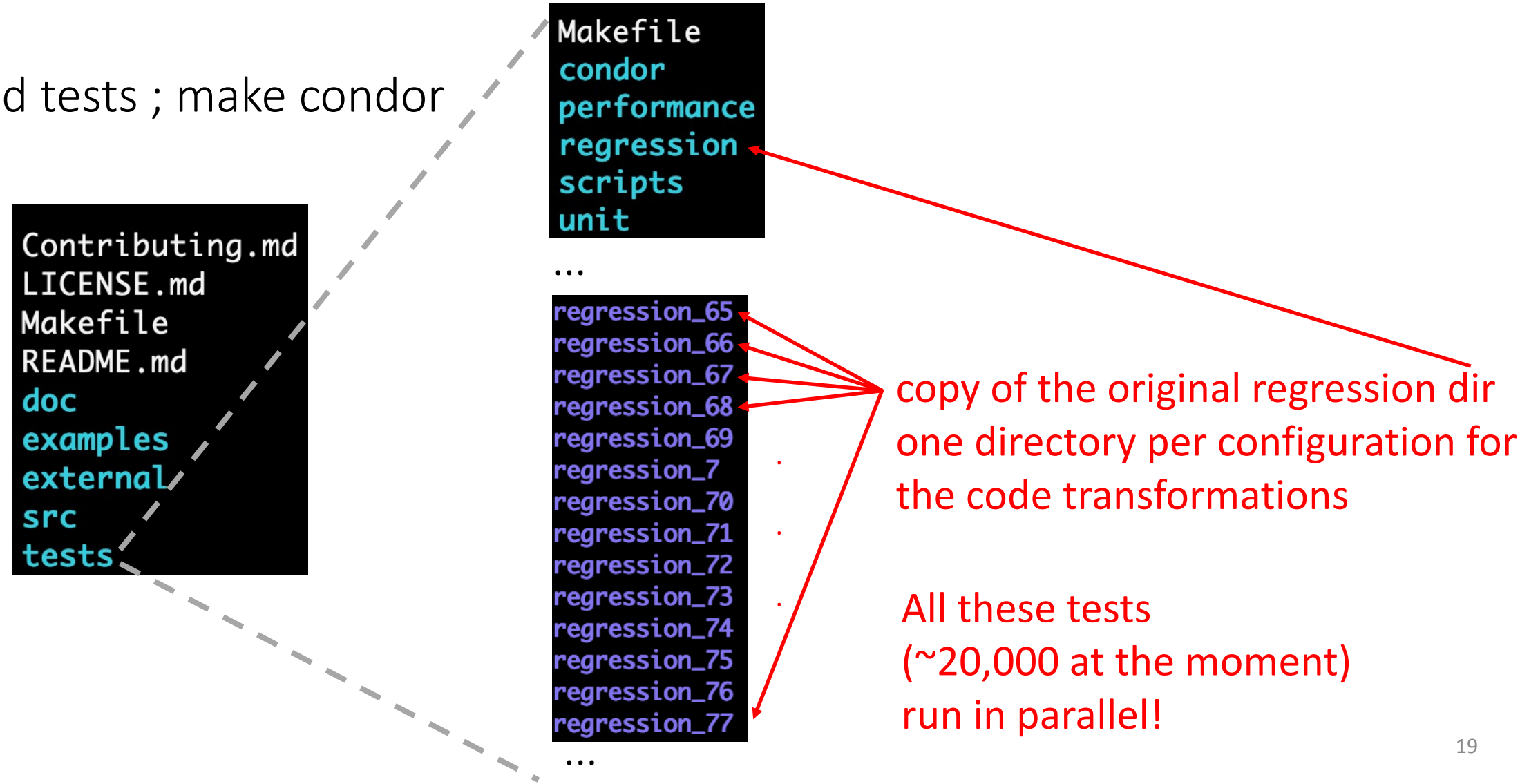
It runs the transformations only using their default configurations (e.g., unroll-factor set to be the default one)

make condor

It generates condor files to run in parallel all transformations with many different configurations (generating more than 20,000 tests that all run in parallel)

Testing with condor

cd tests ; make condor



Testing with condor

cd tests ; make condor

```
Contributing.md
LICENSE.md
Makefile
README.md
doc
examples
external
src
tests
```

```
Makefile
condor
performance
regression
scripts
unit
```

...

```
regression_65
regression_66
regression_67
regression_68
regression_69
regression_7
regression_70
regression_71
regression_72
regression_73
regression_74
regression_75
regression_76
regression_77
```

...

cd tests ; make condor_check

```
$ make condor_check
./scripts/condor_check.sh ;
################################## REGRESSION TESTS:
  Checking the regression test results
    There are 21204 jobs that are still running
    No new tests failed so far
    There are new tests that now pass for all configurations. They are the next ones:

        Chunking
        DSWPIterations_RemovableIntraIterMemEdge
        Exit_call2
        Exit_call3
        IndependentIterations11
        IndependentIterations5
        LICM
        LICM_2
        Multiloops
        Multiloops_list
        ReductionIterationsAnd
        ReductionIterationsOr


################################## UNIT TESTS:
  They are still running


################################## PERFORMANCE TESTS:
  They are still running
```

Testing with condor

cd tests ; make condor

```
Contributing.md
LICENSE.md
Makefile
README.md
doc
examples
external
src
tests
```

```
Makefile
condor
performance
regression
scripts
unit
```

...

```
regression_65
regression_66
regression_67
regression_68
regression_69
regression_7
regression_70
regression_71
regression_72
regression_73
regression_74
regression_75
regression_76
regression_77
```

...

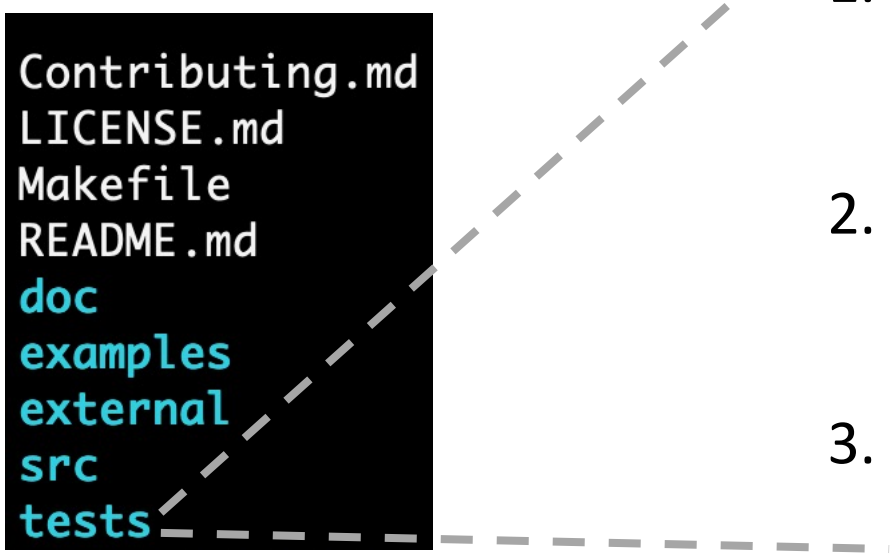
cd tests ; make condor_check

```
$ make condor_check
./scripts/condor_check.sh ;
################################## REGRESSION TESTS:
  Checking the regression test results
    There are 11237 jobs that are still running
    31 new tests failed:
        regression_107/Memory_interprocedural_dependence -noelle-pdg-check -noelle-verbose=3 -noelle-max-cores=2 -noelle-disable-enablers -noelle-disable-inliner -noelle-disable-dead -noelle-parallelizer-force -O1 -Xclang -disable-llvm-passes -O0
```

- Tests that completed successfully get automatically deleted
- Directory of a test that failed is kept (so you can debug it; check compiler_output.txt) and a script to reproduce the fail is automatically generated
- To reproduce the fail:
  - Go to the directory of the test (e.g., cd regression_4/Simple)
  - Run ./run_me.sh

Re-run the tests using condor

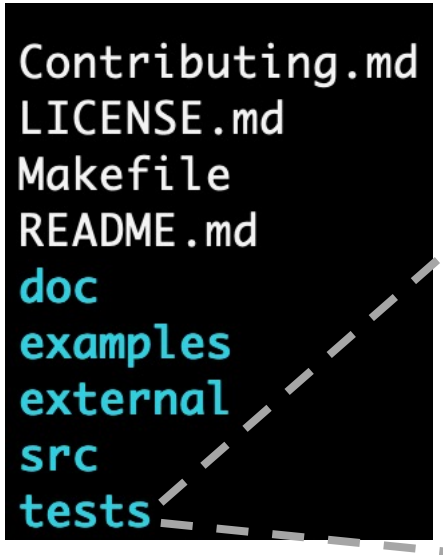
cd tests ;



1. Make sure no tests are still running
   condor_q `whoami`
2. Clean the tests directory
   make clean
3. Run the tests
   make condor

Running a single test without condor

cd tests ; make download



1. Go to the test directory
   (e.g., cd regression/Simple)
2. Clean the directory
   make clean
3. Enable NOELLE binaries in your environment
   source ../../../enable
4. Run the test
   make test_correctness
5. Check the output
   (look at the makefile to understand the scripts)

Always have faith in your ability

Success will come your way eventually

Best of luck!