888: LLVM

Week 5 - LLVM testing infrastructure

Tobias Grosser

Advertisement - clang_complete

- C/C++/Objective-C autocompletion
- For vim
- clang based
- Accurate and fast
- `https://github.com/Rip-Rip/clang_complete`

Exercise

LLVM Passes

- Analysis Passes

  `(-domtree, -regions, -basicaa)`
- Transformation Passes
  - Canonicalization Passes

    `(-reg2mem, -indvars, -loop-simplify, -mergereturn)`
  - Optimization Passes

    `(-mem2reg, -tailcallelim, -constprop, -gvn, -instcombine, -instsimplify)`
- Other Passes

  `(-view-cfg, -view-cfg-only, -view-dom, -instnamer, -verify)`

-instsimplify

- Remove redundant instructions
- Cannot create new instructions

$$(X + -1) + 1 \rightarrow X$$

```
define i32 @add1(i32 %x) {
  %l = add i32 %x, -1
  %r = add i32 %l, 1
  ret i32 %r
}
```

to

```
define i32 @add1(i32 %x) {
  ret i32 %x
}
```

-instcombine

- Combine redundant instructions
- Can create new instructions

$$(x \,\&\, z) \,\hat{}\, (y \,\&\, z) \rightarrow (x \,\hat{}\, y) \,\&\, z$$

```
define i32 @test1(i32 %x, i32 %y, i32 %z) {
        %tmp1 = and i32 %z, %x
        %tmp2 = and i32 %z, %y
        %tmp3 = xor i32 %tmp1, %tmp2
        ret i32 %tmp3
}
```

to

```
define i32 @test1(i32 %x, i32 %y, i32 %z) {
        %tmp1 = xor i32 %x, %y
        %tmp2 = and i32 %tmp1, %y
        ret i32 %tmp2
}
```

Regression tests

- Run with `make check`
- Stored in `src-dir/test`
- Each transformation has its own directory

lit.py / llvm-lit

- LLVM integrated tester
- Runs the LLVM and Clang test suite
- Used to run individual tests
- build/llvm-lit

```
~/llvm_build/bin/llvm-lit  ~/llvm_git/test/Analysis/Dominators/
-- Testing: 4 tests, 4 threads --
PASS: LLVM :: Analysis/Dominators/2006-10-02-BreakCritEdges.ll (1 of 5)
FAIL: LLVM :: Analysis/Dominators/2007-07-12-SplitBlock.ll (2 of 5)
XPASS: LLVM :: Analysis/Dominators/2007-07-11-SplitBlock.ll (3 of 5)
XFAIL: LLVM :: Analysis/Dominators/2007-01-14-BreakCritEdges.ll (4 of 5)
UNSUPPORTED: LLVM :: Analysis/Dominators/other.ll (5 of 5)
```

A single test file

- Run line specifies the test command
- %s is replaced with the test file itself
- Test fails if the command has a non-zero return value

```
; RUN: opt -mypass %s
define i32 @test1(i32 %A) {
        %B = xor i32 %A, 12345
        ret i32 %B
}
```

FileCheck / not

- Use 'FileCheck' to check for expected transformations

```
; RUN: opt < %s -instcombine -S | FileCheck %s
define i1 @test1(i32 %A) {
; CHECK: @test1
; CHECK-NEXT: %C = icmp slt i32 %A, 0
        %B = xor i32 %A, 12345
        %C = icmp slt i32 %B, 0
        ret i1 %C
}
```

- Use 'not' to switch return codes

```
; RUN: not opt %s -S
define i1 @test1(i32 %A) {
; Expected failure because of type mismatch
        ret i1 %A
}
```

Our first instcombine improvement

$(A \,\&\, B) \,\hat{}\, (A \mid B) \rightarrow A \,\hat{}\, B$

- Write simple test case
- Find the place where this pattern can be matched
- Implement the simplification
- Generalize transformation + add exhaustive tests

$\rightarrow$ Create a patch and submit it for review.

Exercise

Will be sent out tonight.