888: LLVM

Week 5 - Instcombine / PatternMatch

Tobias Grosser

Instcombine

- LLVM-IR level peephole optimization
- Run with `opt -instcombine`
- Source code in `lib/Transforms/InstCombine`
- FunctionPass

Instcombine - Architecture

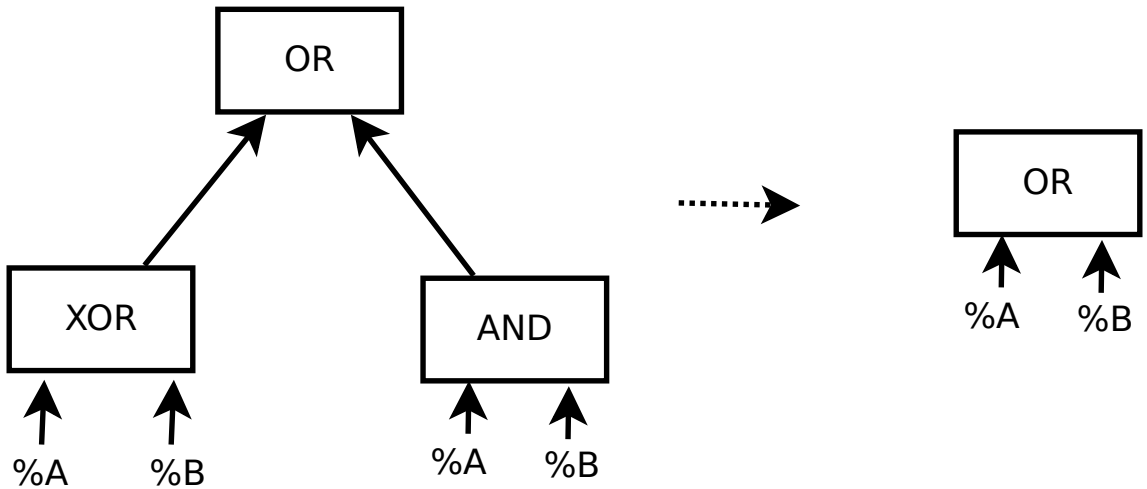
while (optimizations found && not timeout) {

- Search source code for known patterns
- Create simplification for them
- Replace original set of instructions with simplificiation

}

Instcombine - Instruction matching

```
%0 = and i32 %A, %B
%1 = xor i32 %A, %B
%3 = or i32 %0, %1
```



Support/PatternMatch.h

- Match a tree of LLVM instructions
- Capture parts of the instruction tree

```
Value *Exp = ...
Value *X;  ConstantInt *C1;

// Exp == (X | C1)
if (match(Exp, m_Or(m_Value(X), m_ConstantInt(C1))) }
  ... Pattern is matched and variables are bound ...
}
```

Support/PatternMatch.h

- Match and ignore
  - m_Value() - Any value
  - m_ConstantInt() - Any integer constant
  - m_Undef() - Any undefined value
  - m_Zero() - Any zeroinitializer
  - m_One() - Integer of Vector with all elements = 1
  - m_AllOne() - Integer of Vector with all bits = 1
  - m_SignBit() - Integer of Vector with only the signbit set.
  - m_Power2() - Integer of Vector with all elements are a power of two.

Support/PatternMatch.h

- Match and capture
  - m_Value(Value *&V) - Any value
  - m_ConstantInt(ConstantInt *&I) - Any integer constant
  - m_Constant(Constant*&C) - Any constant
- Match and compare
  - m_Specific(Value *V) - Any value

Support/PatternMatch.h

- Match binary operators
  - m_Add(Type *LHS,Type *RHS) - Match an add instruction
  - m_Sub(Type *LHS,Type *RHS) - Match a sub instruction
  - ...
- Match unary operators
  - m_SExt(Type *Operand) - Match an sext instruction
  - m_Neg(Type *Operand) - Match an integer negate
  - ...
- Matchers for control flow

How to create Instructions?

- `include/llvm/InstrTypes.h` defines functions to create instructions
- Two constructors
  - Just create the instruction
  - Create instruction and add it before another instruction
  - Create instruction and add it at the end of a basic block

```
Instruction *I1 = BinaryOperator::CreateOr(A, B);
Instruction *I2 = BinaryOperator::CreateAnd(A, B);
```

IRBuilder

- `include/llvm/Support/IRBuilder.h`
- A helper to automatically
  - Create and insert instructions
  - Get common types
  - Get common constants

```
Value *V = Builder->CreateOr(A, B);
Type *T = Builder->getInt32Ty();
ConstantInt *I = Builder->getInt32(512);
```