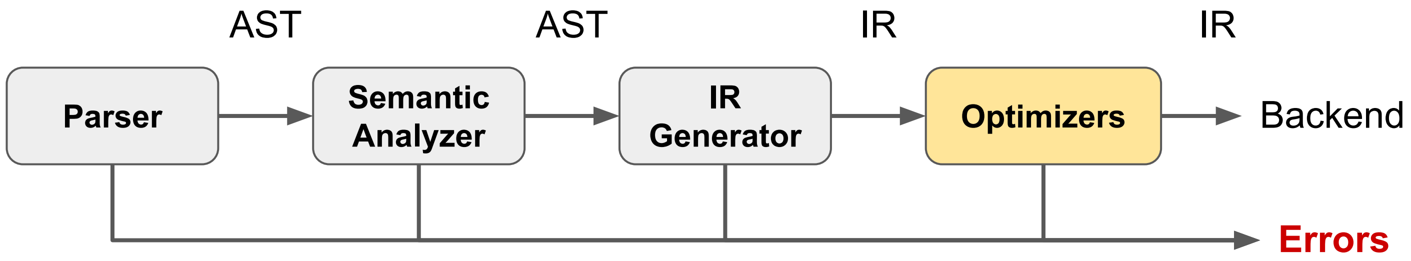
Compiling Techniques

Lecture 13: SSA Optimizations and Rewriting

SSA simplifies Optimization Design



Reminder: Static Single-Assignment (SSA) Form

> *“A program is defined to be in* ***Static Single-Assignment (SSA)*** *form if each variable is a target of exactly one assignment statement in the program text.”*

An important property from this definition is *referential transparency:*

> *“An expression is called referentially transparent if it can be replaced with its corresponding value (and vice-versa) without changing the program's behaviour”.*

A Simple Program: Can we Optimize It?

```
print(42 + 8)
```

⬇

```
%0 = "choco.ir.literal"() <{"value" = 42 : !i32}>
%1 = "choco.ir.literal"() <{"value" = 8 : !i32}>
%2 = """choco.ir.binary_expr"(%0, %1) <{"op" = "+"}>

"choco.ir.call_expr"(%2) <{"func_name" = "print"}>
```

A Simple Program: Can we Optimize It? **Yes!**

```
print(42 + 8)
```

⬇

```
%3 = "choco.ir.literal"() <{"value" = 50 : !i32}>
"choco.ir.call_expr"(%3) <{"func_name" = "print"}>
```

What are the steps needed to optimize this program?

```
print(42 + 8)
```

```
%0 = "choco.ir.literal"() <{"value" = 42 : !i32}>
%1 = "choco.ir.literal"() <{"value" = 8 : !i32}>
%2 = "choco.ir.binary_expr"(%0, %1) <{"op" = "+"}>

"choco.ir.call_expr"(%2) <{"func_name" = "print"}>
```

Step 1: Constant Folding

```
print(42 + 8)
```

⬇

```
%0 = "choco.ir.literal"() <{"value" = 42 : !i32}>
%1 = "choco.ir.literal"() <{"value" = 8 : !i32}>
%2 = "choco.ir.binary_expr"(%0, %1) <{"op" = "+"}>
%3 = "choco.ir.literal"() <{"value" = 50 : !i32}>
"choco.ir.call_expr"(%3) <{"func_name" = "print"}>
```

Step 2: Dead Code Elimination

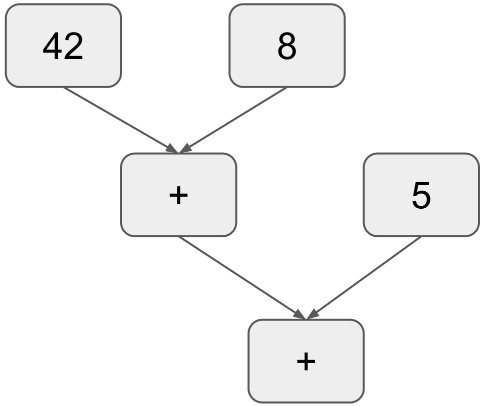
```
print(42 + 8)
```

⬇

```
~~%0 = "choco.ir.literal"() <{"value" = 42 : !i32}>~~
~~%1 = "choco.ir.literal"() <{"value" = 8 : !i32}>~~
~~%2 = "choco.ir.binary_expr"(%0, %1) <{"op" = "+"}>~~
%3 = "choco.ir.literal"() <{"value" = 50 : !i32}>
"choco.ir.call_expr"(%3) <{"func_name" = "print"}>
```

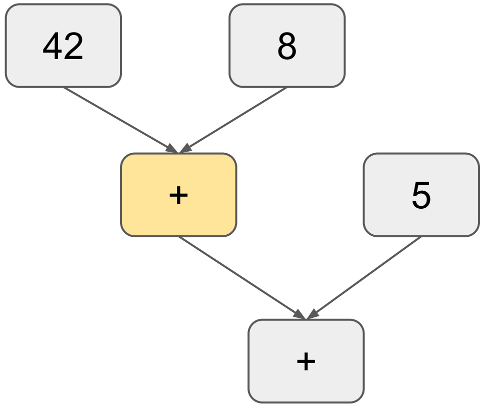
Constant Folding

```
%0 = "choco.ir.literal"() <{"value" = 42 : !i32}>
%1 = "choco.ir.literal"() <{"value" = 8 : !i32}>
%2 = "choco.ir.binary_expr"(%0, %1) <{"op" = "+"}>
%3 = "choco.ir.literal"() <{"value" = 5 : !i32}>
%4 = "choco.ir.binary_expr"(%2, %3) <{"op" = "+"}>
```



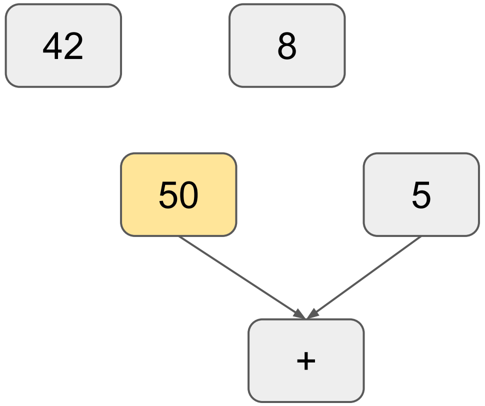
Constant Folding

```
%0 = "choco.ir.literal"() <{"value" = 42 : !i32}>
%1 = "choco.ir.literal"() <{"value" = 8 : !i32}>
%2 = "choco.ir.binary_expr"(%0, %1) <{"op" = "+"}>
%3 = "choco.ir.literal"() <{"value" = 5 : !i32}>
%4 = "choco.ir.binary_expr"(%2, %3) <{"op" = "+"}>
```



1) Find binary_expr where the lhs is a constant literal and the rhs is a constant literal.

Constant Folding

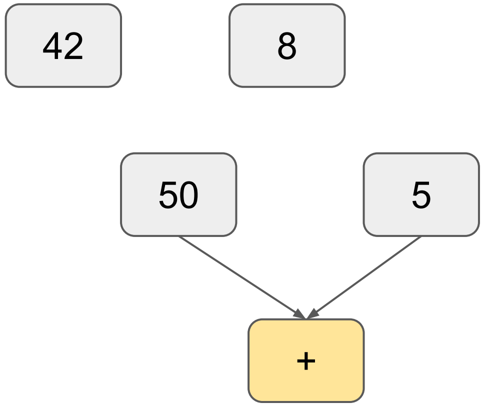


```
%0 = "choco.ir.literal"() <{"value" = 42 : !i32}>
%1 = "choco.ir.literal"() <{"value" = 8 : !i32}>
%2 = "choco.ir.literal"() <{“value” = 50 : !i32}>
%3 = "choco.ir.literal"() <{"value" = 5 : !i32}>
%4 = "choco.ir.binary_expr"(%2, %3) <{"op" = "+"}>
```

1) Find binary_expr where the lhs is a constant literal and the rhs is a constant literal.

2) Compute result and replace binary_expr with literal of value result.

Constant Folding



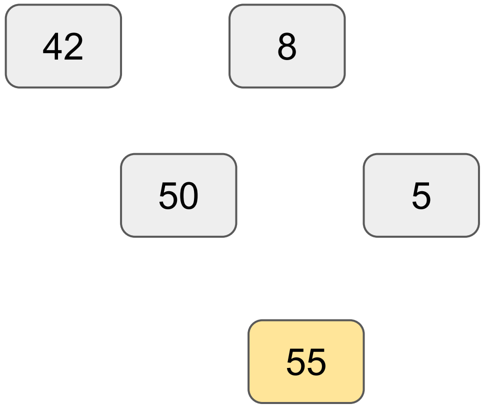
```
%0 = "choco.ir.literal"() <{"value" = 42 : !i32}>
%1 = "choco.ir.literal"() <{"value" = 8 : !i32}>
%2 = "choco.ir.literal"() <{“value” = 50 : !i32}>
%3 = "choco.ir.literal"() <{"value" = 5 : !i32}>
%4 = "choco.ir.binary_expr"(%2, %3) <{"op" = "+"}>
```

1) Find binary_expr where the lhs is a constant literal and the rhs is a constant literal.

2) Compute result and replace binary_expr with literal of value result.

1) Find binary_expr where the lhs is a constant literal and the rhs is a constant literal.

Constant Folding

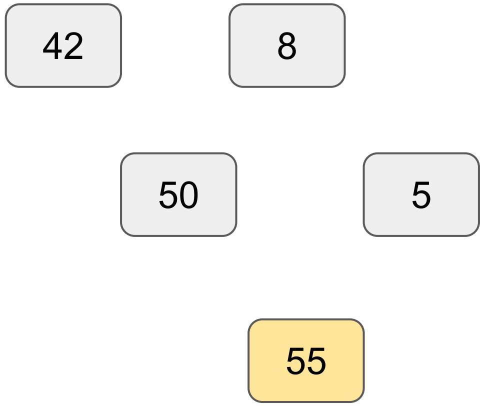


```
%0 = "choco.ir.literal"() <{"value" = 42 : !i32}>
%1 = "choco.ir.literal"() <{"value" = 8 : !i32}>
%2 = "choco.ir.literal"() <{“value” = 50 : !i32}>
%3 = "choco.ir.literal"() <{"value" = 5 : !i32}>
%4 = "choco.ir.literal"() <{“value” = 55 : !i32}>
```

1) Find binary_expr where the lhs is a constant literal and the rhs is a constant literal.
2) Compute result and replace binary_expr with literal of value result.

1) Find binary_expr where the lhs is a constant literal and the rhs is a constant literal.
2) Compute result and replace binary_expr with literal of value result.

Constant Folding



```
%0 = "choco.ir.literal"() <{"value" = 42 : !i32}>
%1 = "choco.ir.literal"() <{"value" = 8 : !i32}>
%2 = "choco.ir.literal"() <{“value” = 50 : !i32}>
%3 = "choco.ir.literal"() <{"value" = 5 : !i32}>
%4 = "choco.ir.literal"() <{“value” = 55 : !i32}>
```

1) Find binary_expr where the lhs is a constant literal and the rhs is a constant literal.

2) Compute result and replace binary_expr with literal of value result.

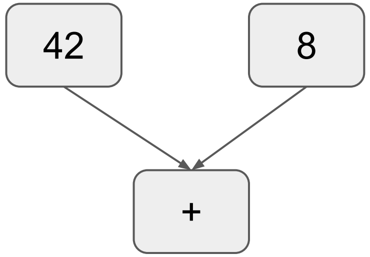
1) Find binary_expr where the lhs is a constant literal and the rhs is a constant literal.

2) Compute result and replace binary_expr with literal of value result.

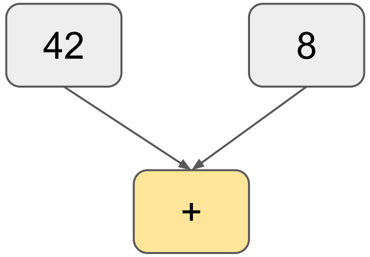
Repeat (if needed)

Dead Code Elimination

```
%0 = "choco.ir.literal"() <{"value" = 42 : !i32}>
%1 = "choco.ir.literal"() <{"value" = 8 : !i32}>
%2 = "choco.ir.binary_expr"(%0, %1) <{"op" = "+"}>
```



Dead Code Elimination



```
%0 = "choco.ir.literal"() <{"value" = 42 : !i32}>
%1 = "choco.ir.literal"() <{"value" = 8 : !i32}>
%2 = "choco.ir.binary_expr"(%0, %1) <{"op" = "+"}>
```

1) Find binary_expr without users

Dead Code Elimination

```
%0 = "choco.ir.literal"() <{"value" = 42 : !i32}>
%1 = "choco.ir.literal"() <{"value" = 8 : !i32}>
~~%2 = "choco.ir.binary_expr"(%0, %1) <{"op" = "+"}>~~
```

1) Find binary_expr without users
2) Delete binary_expr

Dead Code Elimination

```
%0 = "choco.ir.literal"() <{"value" = 42 : !i32}>
%1 = "choco.ir.literal"() <{"value" = 8 : !i32}>
~~%2 = "choco.ir.binary_expr"(%0, %1) <{"op" = "+"}>~~
```

1) Find binary_expr without users
2) Delete binary_expr
3) Find literal without users

Dead Code Elimination

```
%0 = "choco.ir.literal"() <{"value" = 42 : !i32}>
~~%1 = "choco.ir.literal"() <{"value" = 8 : !i32}>~~
~~%2 = "choco.ir.binary_expr"(%0, %1) <{"op" = "+"}>~~
```

1) Find binary_expr without users
2) Delete binary_expr
3) Find literal without users
4) Delete literal

Dead Code Elimination

```
%0 = "choco.ir.literal"() <{"value" = 42 : !i32}>
~~%1 = "choco.ir.literal"() <{"value" = 8 : !i32}>~~
~~%2 = "choco.ir.binary_expr"(%0, %1) <{"op" = "+"}>~~
```

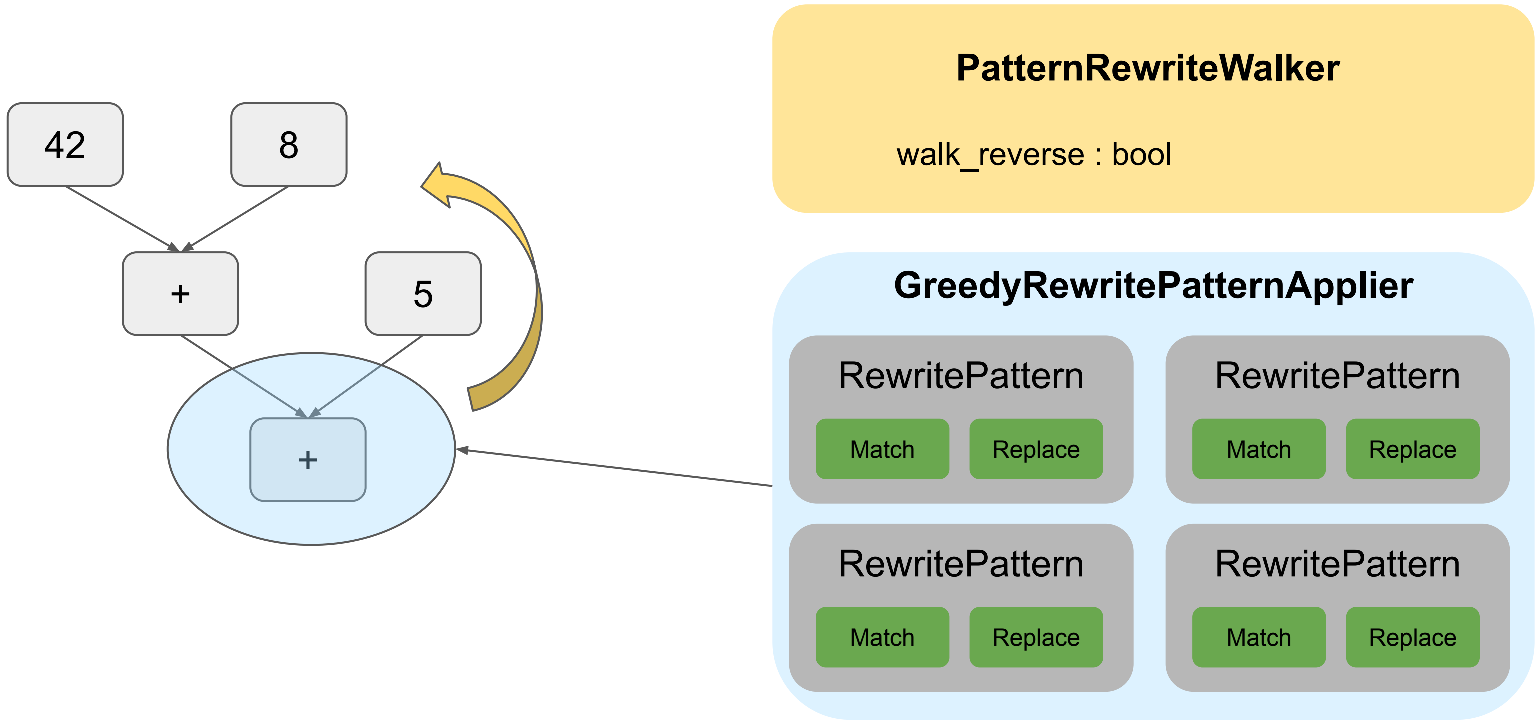
1) Find binary_expr without users
2) Delete binary_expr
3) Find literal without users
4) Delete literal
5) Find literal without users

Dead Code Elimination

```
~~%0 = "choco.ir.literal"() <{"value" = 42 : !i32}>~~
~~%1 = "choco.ir.literal"() <{"value" = 8 : !i32}>~~
~~%2 = "choco.ir.binary_expr"(%0, %1) <{"op" = "+"}>~~
```

1) Find binary_expr without users
2) Delete binary_expr
3) Find literal without users
4) Delete literal
5) Find literal without users
6) Delete literal

Pattern Rewriting aka. Peephole Optimizations



Match and Rewrite: Constant Folding

```
@dataclass
class BinaryExprRewriter(RewritePattern):

    @op_type_rewrite_pattern
    def match_and_rewrite(self, expr: BinaryExpr, rewriter: PatternRewriter) -> None:
        if expr.op.data == '+':
            if isinstance(expr.lhs.op, Literal) and
               isinstance(expr.lhs.op, Literal):
                lhs_value = expr.lhs.op.value.parameters[0].data
                rhs_value = expr.rhs.op.value.parameters[0].data
                result_value = lhs_value + rhs_value
                new_constant = Literal.get(result_value)
                rewriter.replace_op(expr, [new_constant])
        Return
```

Apply on Full Module

```
def choco_flat_constant_folding(ctx: MLContext, module: ModuleOp)
        -> ModuleOp:


    walker = PatternRewriteWalker(GreedyRewritePatternApplier([
        BinaryExprRewriter(),
    ]))

    walker.rewrite_module(module)
    return module
```

Match and Rewrite: Dead Code Implementation

```
@dataclass
class LiteralRewriter(RewritePattern):

    @op_type_rewrite_pattern
    def match_and_rewrite(self, literal: Literal, rewriter: PatternRewriter) -> None:
        if len(literal.results[0].uses) == 0:
            rewriter.replace_op(literal, [], [None])
        Return

@dataclass
class BinaryExprRewriter(RewritePattern):

    @op_type_rewrite_pattern
    def match_and_rewrite(self, expr: BinaryExpr, rewriter: PatternRewriter) -> None:
        if len(expr.results[0].uses) == 0:
            rewriter.replace_op(expr, [], [None])

        return
```

Apply on Full Module

```
def choco_dead_code_elimination(ctx: MLContext, module: ModuleOp)
        -> ModuleOp:


    walker = PatternRewriteWalker(GreedyRewritePatternApplier([
        LiteralRewriter(),
        BinaryExprRewriter(),
    ]), walk_reverse=True)

    walker.rewrite_module(module)
    return module
```