Part 6

Transformations

The Model

- The AST/Model is a critical part

```
print 3 + 4 * 5 + x;
```

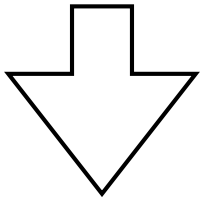
```
PrintStatement(
    Bin('+',
        BinOp('+',
              Integer(3),
              BinOp('*', Integer(4), Integer(5))),
        Location('x')))
```

- Question: Can the model be simplified?

Model Transforms

- Sometimes you can simplify the model

```
PrintStatement(
    Bin('+',
        BinOp('+',
              Integer(3),
              BinOp('*', Integer(4), Integer(5))),
        Location('x')))
```



```
PrintStatement(
    Bin('+', Integer(23), Location('x')))
```

- Example: Constant folding

Algebraic Simplification

- Certain expression patterns can be matched

```
x * 1  -> x
x + 0  -> x
x * 0  -> 0
x * y + z * y -> (x+z)*y
```

- Reuse of common subexpressions

```
(x + y)/a + (x + y)/b
```

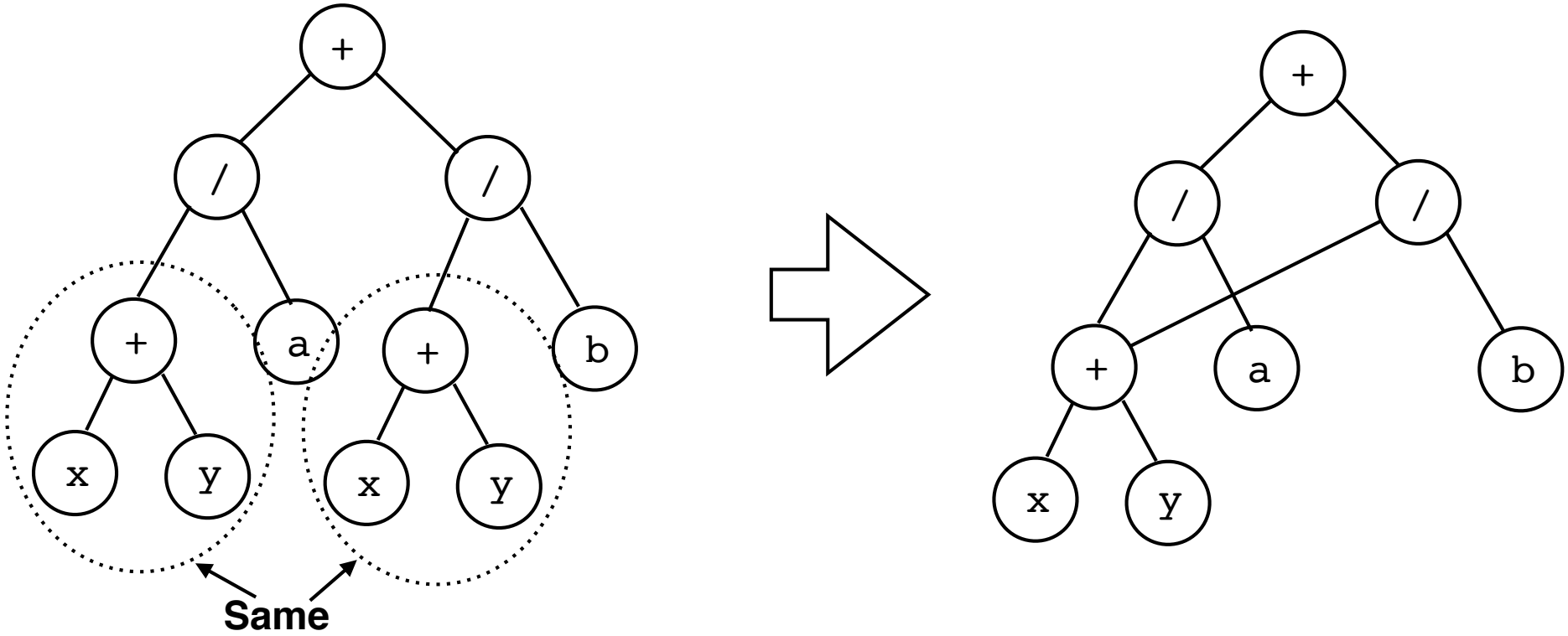
- For example: Can x+y be reused?

```
temp = x + y
temp/a + temp/b
```

DAG Conversion

- Expression trees can sometimes be rewritten as a Directed Acyclic Graph (DAG)

```
(x + y)/a + (x + y)/b
```



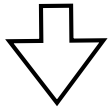
Syntactic Sugar

- Sometimes complicated language features can be expressed in terms of simpler language features (as a kind of rewriting)
- Thought to ponder: What is the most minimal set of language features needed?

Example

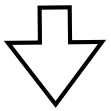
- Unary Operator Conversion

```
UnaryOp('+', operand)
```



```
operand
```

```
UnaryOp('-', operand)
```

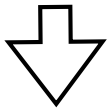


```
BinOp('-', Integer(0), operand)
```

Example

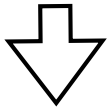
- Short-circuit evaluation

```
x && y
```



```
{ var t = x;  if t { t = y;} t; }
```

```
x || y
```



```
{ var t = x;  if !t { t = y;} t; }
```

Commentary

- Model transformations would usually take place AFTER type checking
- Transforms often render the model as "unrecognizable" compared to original source code (complicates error reporting)
- Critical: Transforms should not change the semantic meaning of the program.

Implementation

- Pattern...

```
def transform(node):
    ...
    return new_node

def transform_binop(node):
    left = transform(node.left)
    right = transform(node.right)
    # Decide what to do ...
    if (isinstance(left, Integer) and
        isinstance(right, Integer)):
        return Integer(
           eval(f'{left.value} {node.op} {right.value}')
        )
    return BinOp(node.op, left, right)
```

Project

- You can optionally try to implement some transforms for Wabbit
- Example: constant folding
- See wabbit/transform.py