# Compiler Construction: Assignment 4

Fabian Krause

January 11, 2022

# Assignment 4: Compiler for $\mathcal{L}_{If}$

7 passes:

1. Shrink: $\mathcal{L}_{If} \leadsto \mathcal{L}_{If}$
2. Remove Complex Operands: $\mathcal{L}_{If} \leadsto \mathcal{L}_{If}^{mon}$
3. Explicate Control: $\mathcal{L}_{If}^{mon} \leadsto \mathcal{C}_{If}$
4. Select Instructions: $\mathcal{C}_{If} \leadsto x86_{If}^{Var}$
5. Register Allocation: $x86_{If}^{Var} \leadsto x86_{If}$
6. Patch Instructions: $x86_{If} \leadsto x86_{If}$
7. Prelude and Conclusion: $x86_{If} \leadsto x86_{If}$

# 1. Shrink: $\mathcal{L}_{If} \rightsquigarrow \mathcal{L}_{If}$

Simple transformation:

- $e_1$ `and` $e_2 \Rightarrow e_2$ `if` $e_1$ `else False`
- $e_1$ `or` $e_2 \Rightarrow$ `True if` $e_1$ `else` $e_2$

Perhaps confusing order of arguments in `IfExp()` terminal:

- $e_1$ `if` $e_2$ `else` $e_3$
- `IfExp`$(e_2, e_1, e_3)$

# 2. Remove Complex Operands: $\mathcal{L}_{If} \rightsquigarrow \mathcal{L}_{If}^{mon}$

Let expressions! $\texttt{Let}(x, e_1, e_2)$ assigns $e_1$ to $x$, then evaluates $e_2$ which may use $x$.

Used for conditional side-effects:

```
(input_int() + 1) \
    if x > 0 else (42 + input_int())
```

$\Rightarrow$

```
tmp0 = input_int()
tmp1 = input_int()
(tmp0 + 1) \
    if x > 0 else (42 + tmp1)
```

Build `Let` expressions inside out, starting with the last temporary variable.

# 3. Explicate Control: $\mathcal{L}_{If}^{mon} \rightsquigarrow \mathcal{C}_{If}$

Mutual recursive functions:

1. `explicate_effect`: Generate code for a (lone) expression.
2. `explicate_assign`: Generate code for an assignment.
3. `explicate_pred`: Generate code for an `if` expression or statement.
4. `explicate_stmt`: Generate code for statements.

# 4. Select Instructions: $\mathcal{C}_{If} \rightsquigarrow x86_{If}^{Var}$

Cases given in the exercise sheet, straightforward implementation.

# 5. Register Allocation: $x86_{If}^{Var} \rightsquigarrow x86_{If}$

Blocks!

1. Control Flow Graph
2. Arg (`ByteReg`), Read/Write:
   - 2.1 `xorq`
   - 2.2 `cmpq`
   - 2.3 `set`
3. Liveness Analysis (`Jump`, `JumpIf`)
4. Build Interference

# 6./7. Patch Instructions & Prelude and Conclusion: $x86_{If} \rightsquigarrow x86_{If}$

1. Patch Instructions
   1.1 `cmpq`, second argument must not be an immediate
   1.2 `movzbq`
2. Prelude and Conclusion
   2.1 Jump to start after prelude
   2.2 Place conclusion in block named conclusion

Questions?