Register Allocation via Graph Colouring

Lukáš Kuklínek

`xkukli01@stud.fit.vutbr.cz`

Fakulta Informačních Technologií

Vysoké Učení Technické, Brno

Motivation

- **Registers**: instantaneous access
- **Caches**: a few clock cycles latency
- **Main memory**: hundreds of clock cycles latency

Motivation

- **Registers**: instantaneous access
- **Caches**: a few clock cycles latency
- **Main memory**: hundreds of clock cycles latency

- Supply of registers is limited (architecture-specific)
- Compiler has to work out the assignment of variables to registers
  - Including intermediate code temporaries
  - Leftovers are stored in main memory

Register allocation

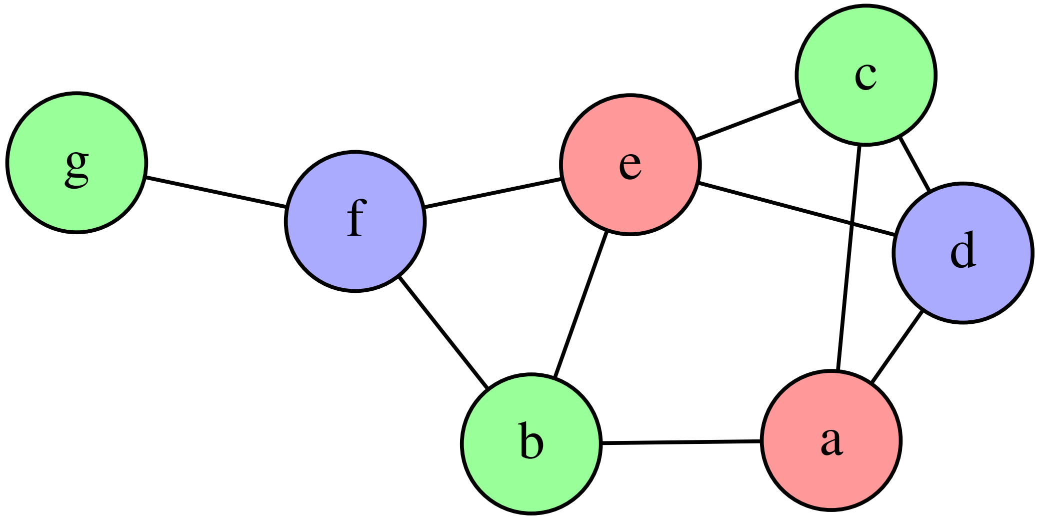
Process overview:

1. Parse Source Code
2. Build Intermediate Representation
3. Build Control Flow Graph
4. Perform Liveness Analysis
5. Build Variable Interference Graph
6. Assign Registers

Output of each phase is the input to the next one.

k-Graph Colouring

- Given an undirected graph $G = (V, E)$
- Given a set of colours C ($|C| = k$)
- Find a mapping $f : V \to C$
- Such that $\forall(u, v) \in E \ : \ f(u) \neq f(v)$



Register Allocation correspondence

Register Allocation	Graph Colouring
Registers	Colours
Variables	Vertices
Variable Interferences	Edges

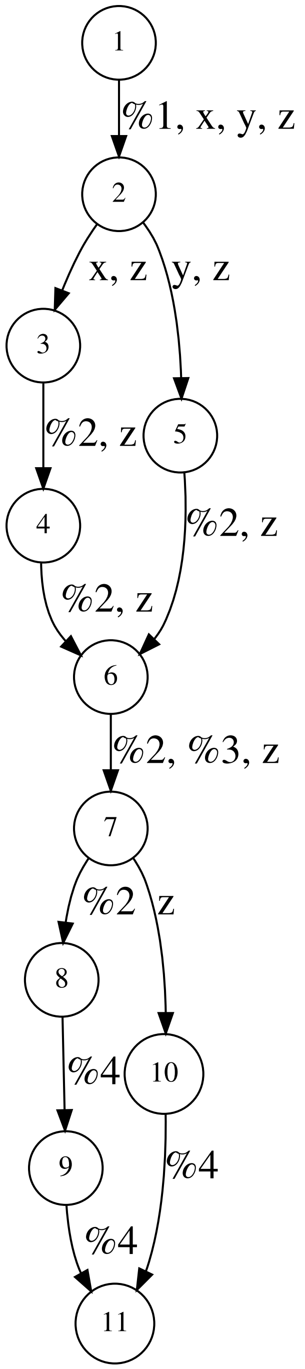
- Two variables *interfere* if they are both live at any point in the program. Such a pair of variables cannot share a single register.
- Interfering variables have an edge between the corresponding nodes, thus the nodes are not assigned the same colour.

Sample function

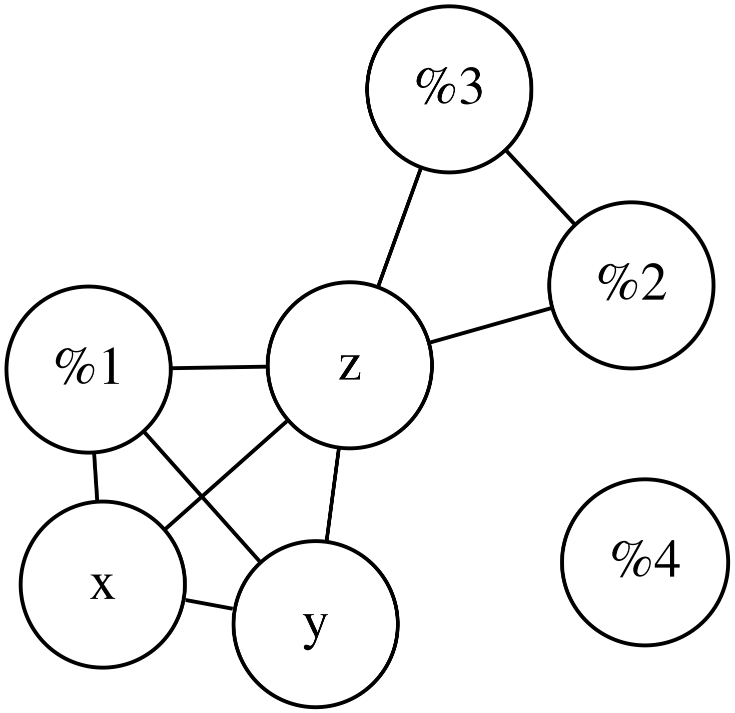
```
inline int max(int a, int b)
{
    return (a > b ? a : b);
}

int greatest(int x, int y, int z)
{
    // max calls will be inlined here
    return max(max(x, y), z);
}
```

Function IR

```
0  greatest(x, y, z):
1              %1 <- gt x y
2              cjmp %1 -> then1 / else1
3     then1:   %2 <- mov x
4              jmp -> end1
5     else1:   %2 <- mov y
6     end1:    %3 <- gt %2 z
7              cjmp %3 -> then2 / else2
8     then2:   %4 <- mov %2
9              jmp endif2
10    else2:   %4 <- mov z
11    end2:    ret %4
```

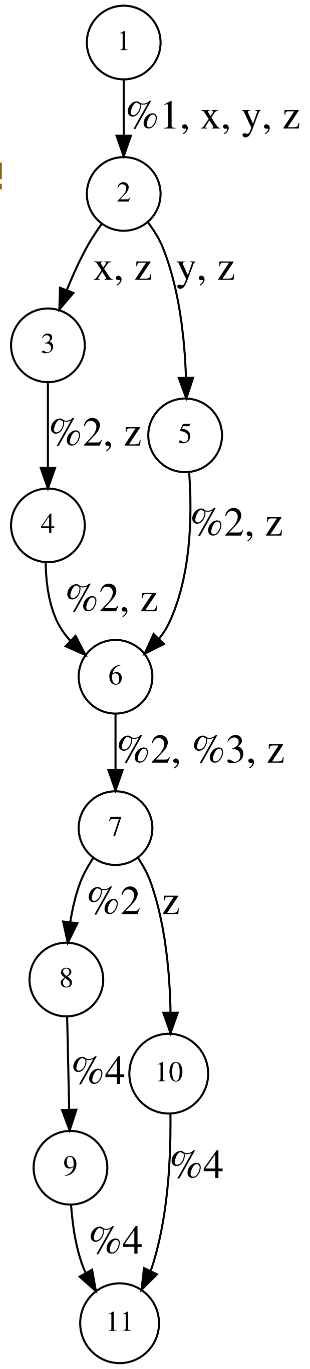


Interference Graph



Step #1

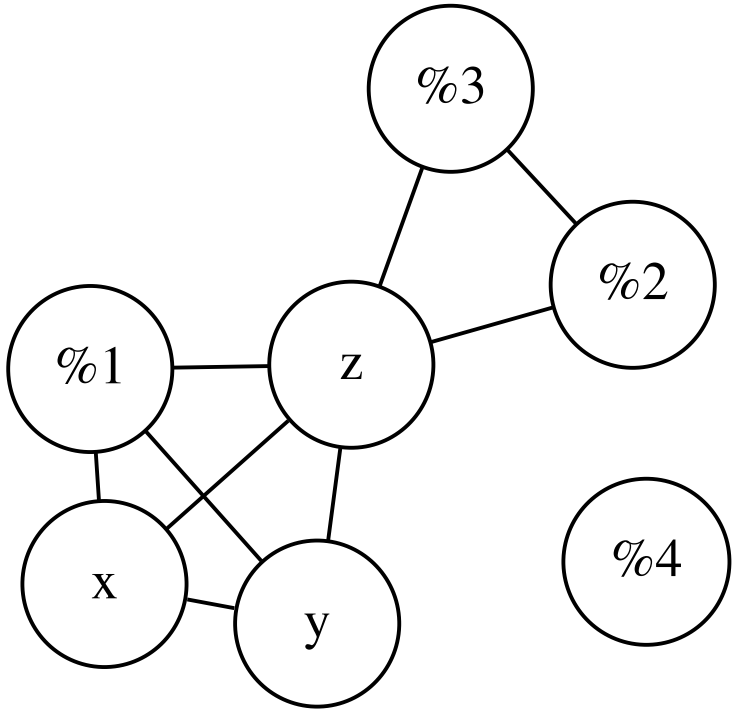
Task: Assign 3 registers (red, green, blue) to these variables



Algorithm overview

- Heuristic: for k-colouring, remove a node with the degree of at most $k-1$
- If the rest of the graph is k-colourable, then the graph with the removed node is also k-colourable.
- If there is no such node, pick a different one, guessing which can be coloured despite having the degree $\geq k$
- Add nodes back in reverse order, assigning colours
- If no colour can be assigned, the variable has to be stored in memory (generate `load`/`store` instruction as appropriate)

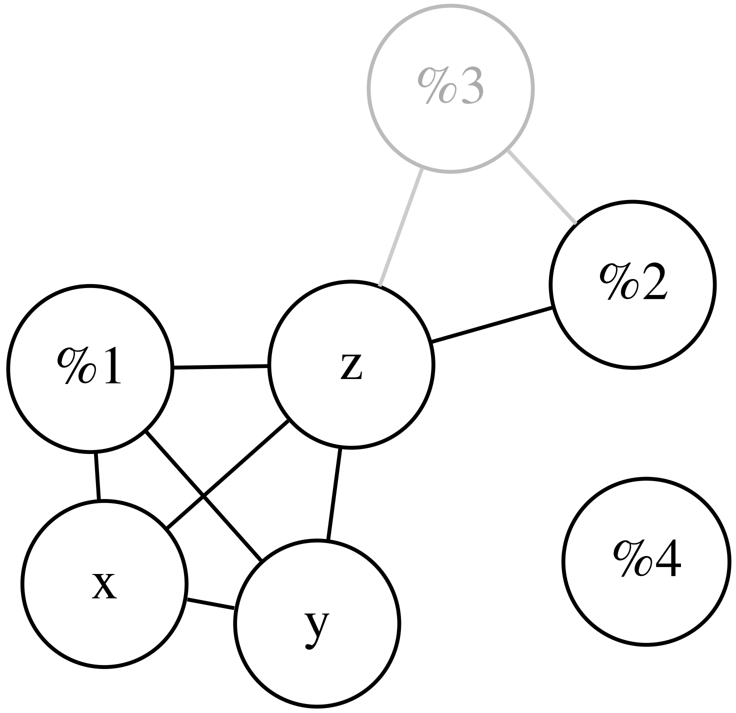
Algorithm Demonstration



Step #1

STACK:

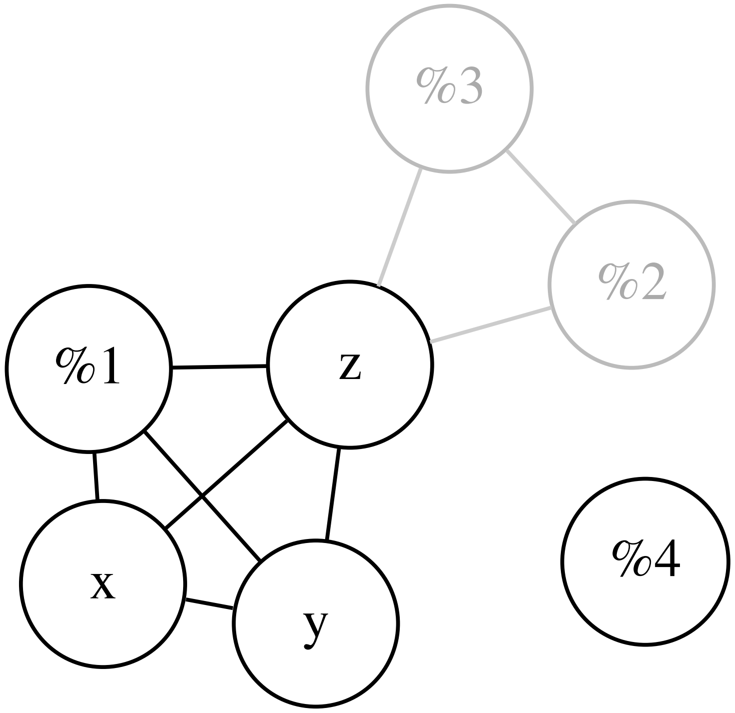
Algorithm Demonstration



Step #2

STACK: %3

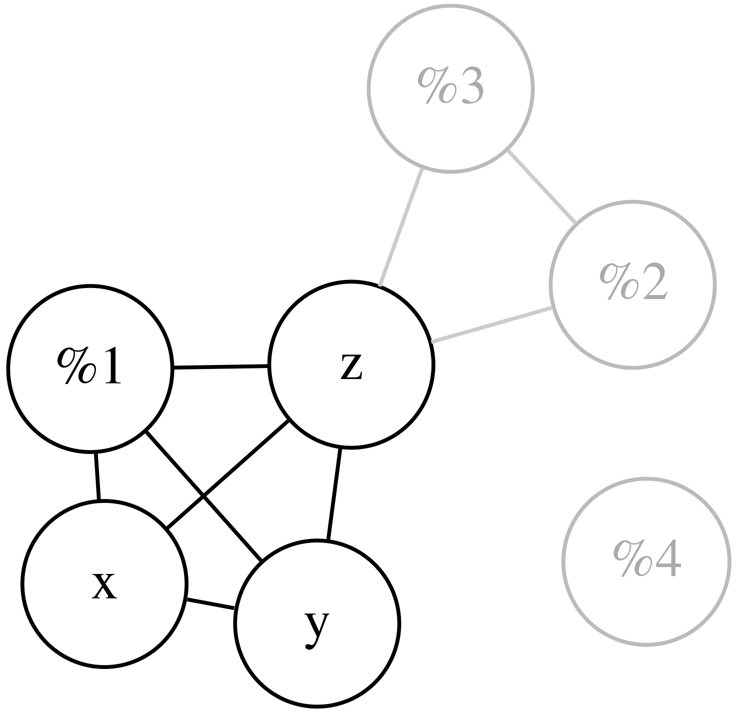
Algorithm Demonstration



Step #3

STACK: %3, %2

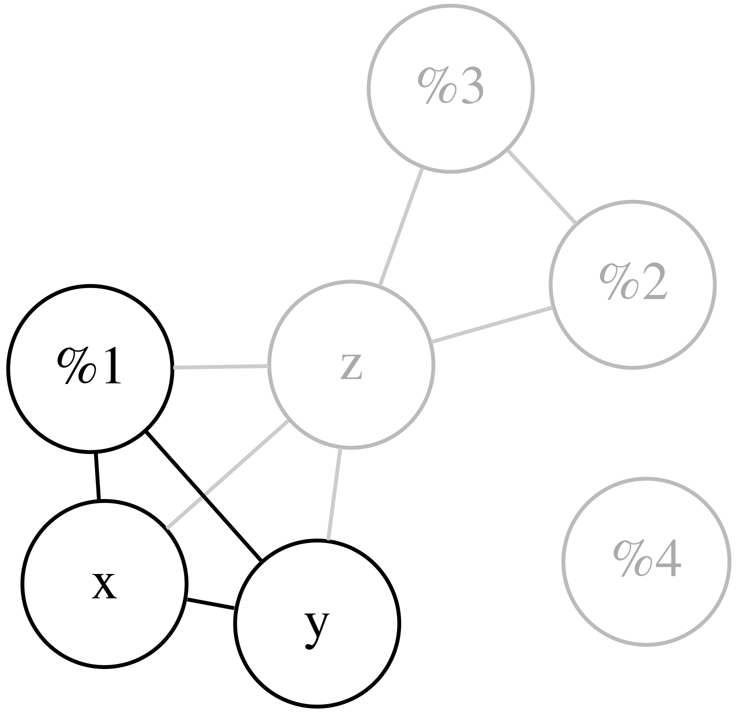
Algorithm Demonstration



Step #4

STACK: %3, %2, %4
Heuristics fails!

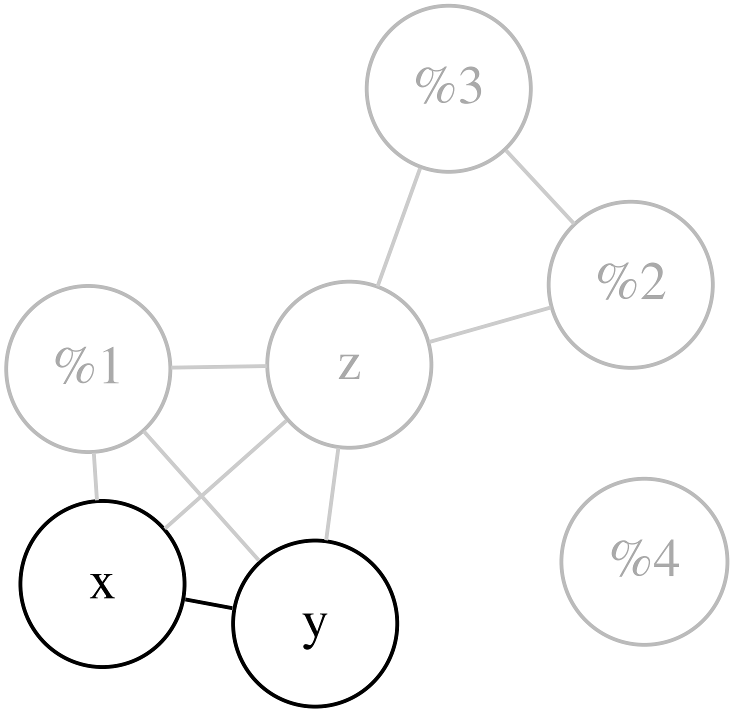
Algorithm Demonstration



Step #5

STACK: %3, %2, %4, z

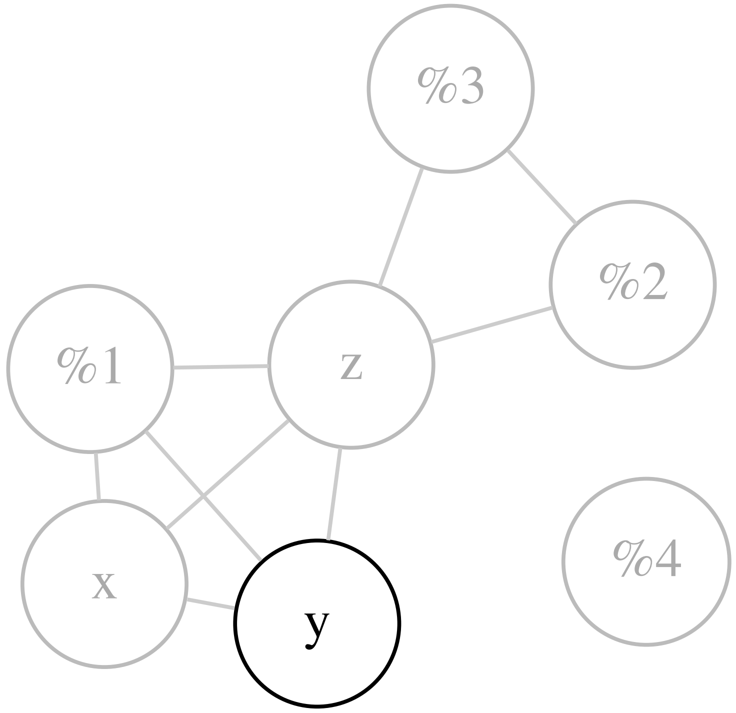
Algorithm Demonstration



Step #6

STACK: %3, %2, %4, z, %1

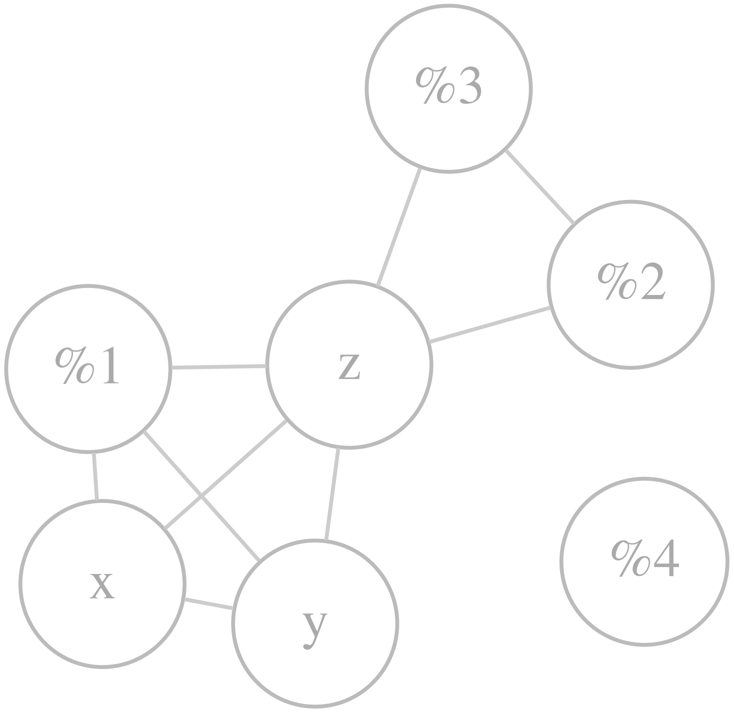
Algorithm Demonstration



Step #7

STACK: %3, %2, %4, z, %1, x

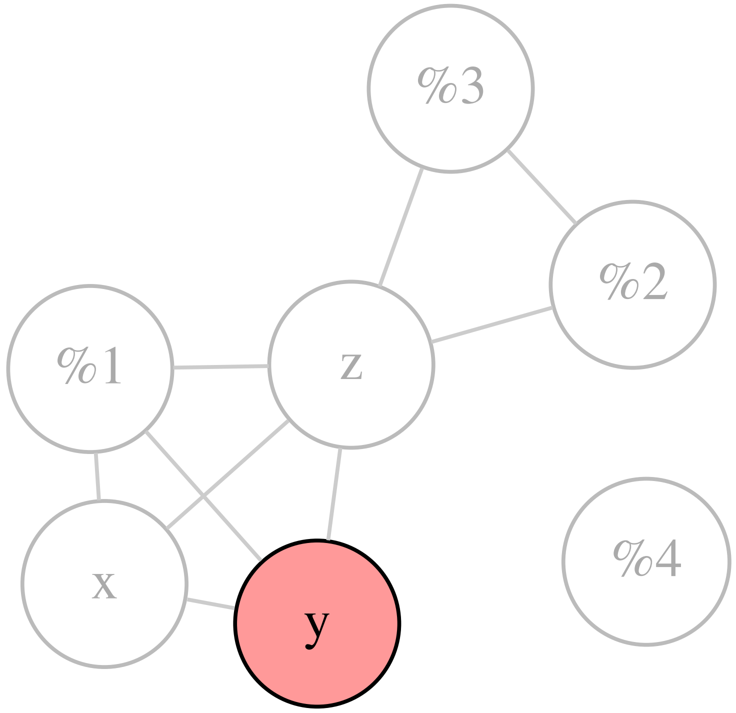
Algorithm Demonstration



Step #8

STACK: %3, %2, %4, z, %1, x, y

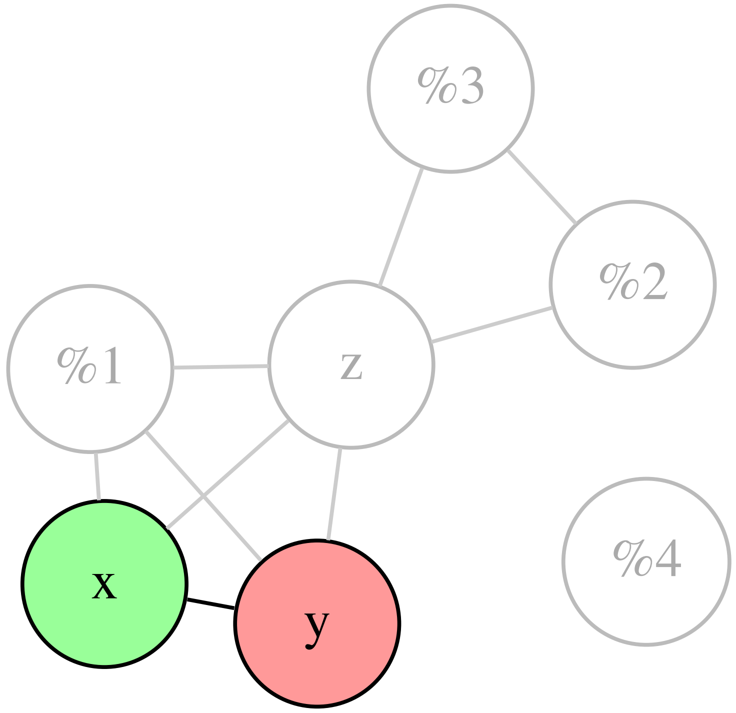
Algorithm Demonstration



Step #9

STACK: %3, %2, %4, z, %1, x

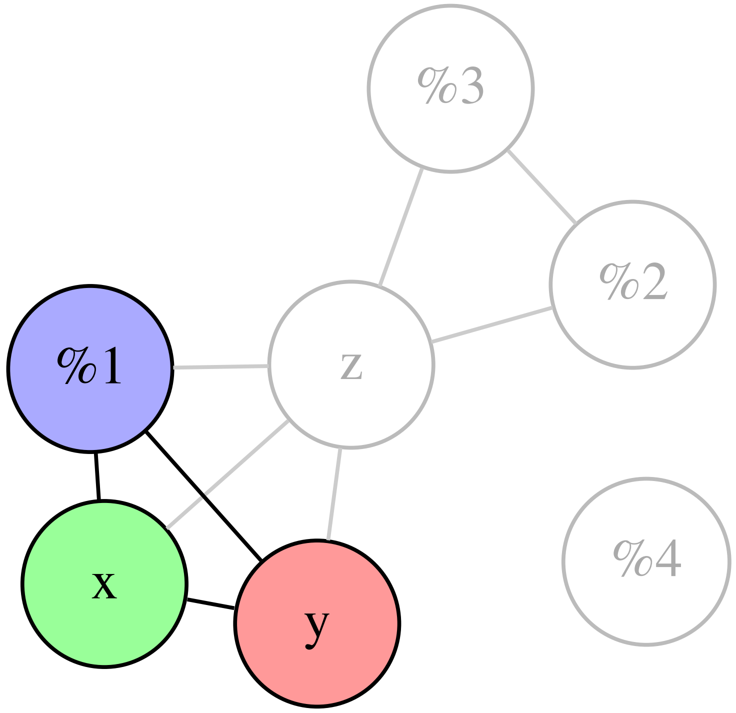
Algorithm Demonstration



Step #10

STACK: %3, %2, %4, z, %1

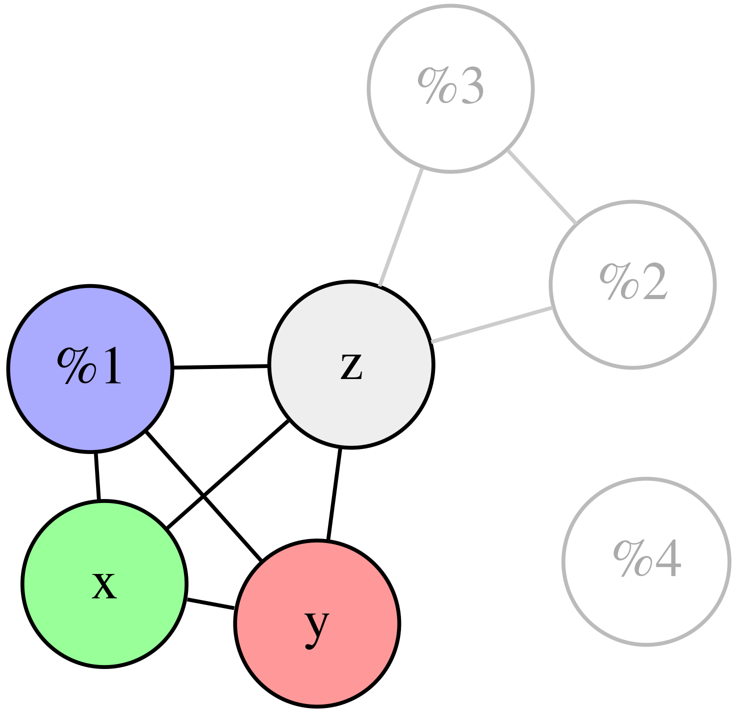
Algorithm Demonstration



Step #11

STACK: %3, %2, %4, z

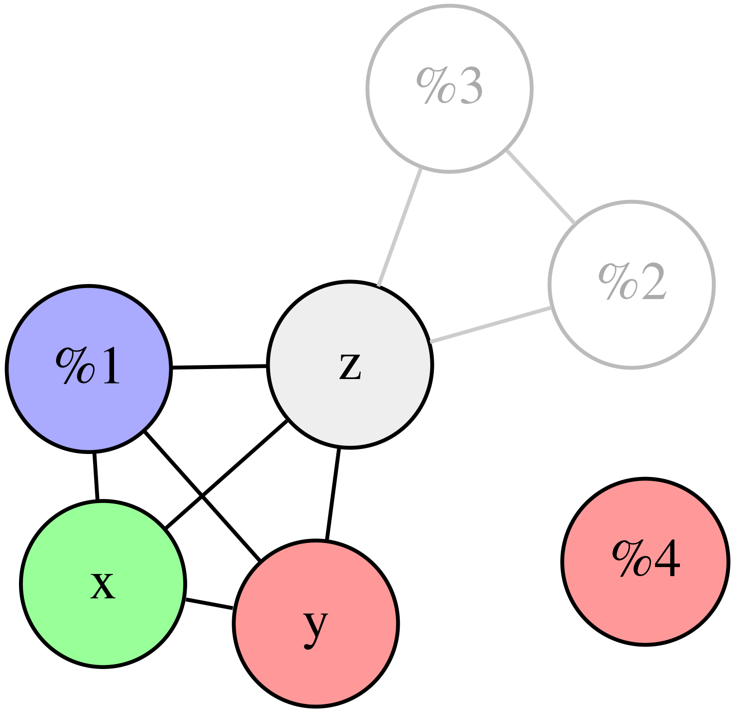
Algorithm Demonstration



Step #12

STACK: %3, %2, %4
Optimistic colouring fails, have to spill variable `z`.

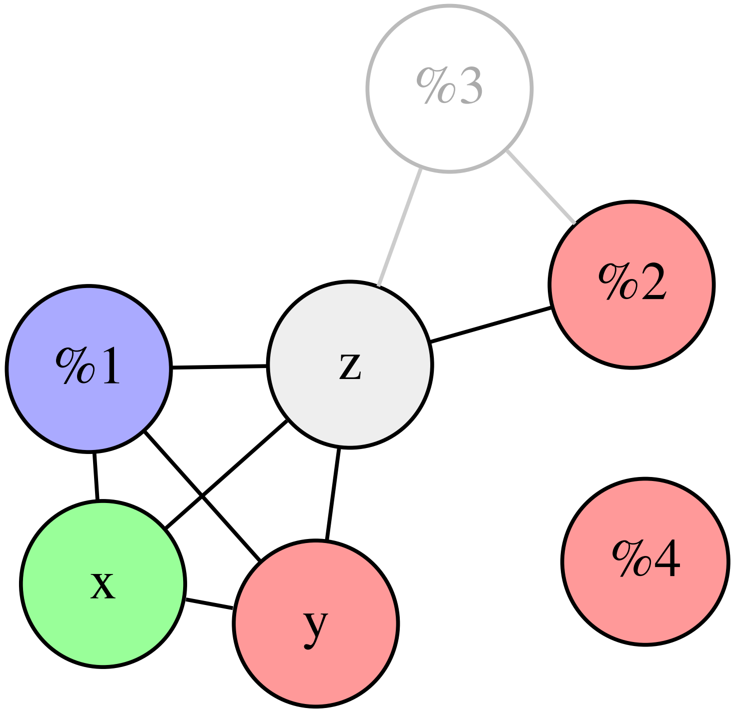
Algorithm Demonstration



Step #13

STACK: %3, %2

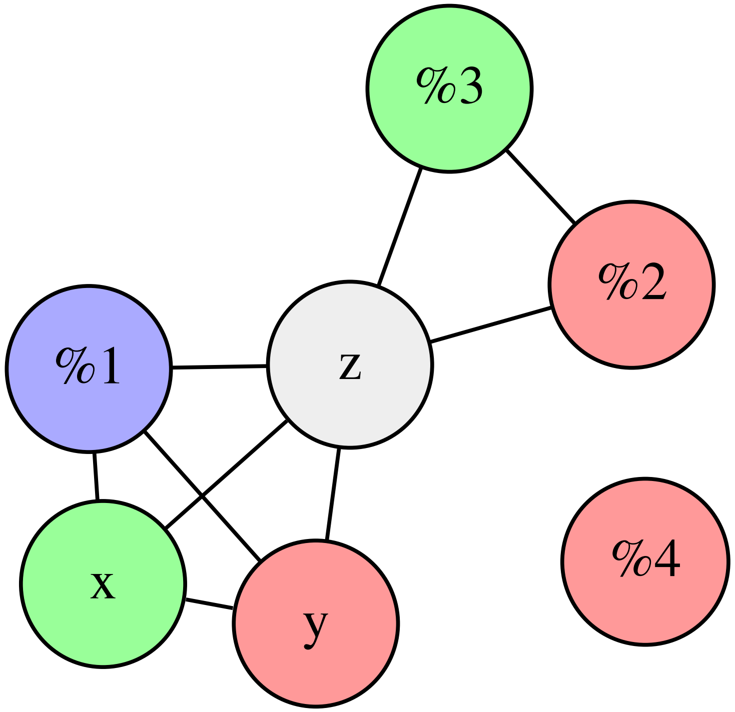
Algorithm Demonstration



Step #14

STACK: %3

Algorithm Demonstration



Step #15

STACK:

Final Remarks & Conclusion

- Both problems, graph colouring and optimal register allocation, are NP-complete, hence the heuristics
- Heuristics on many levels
- Colouring algorithm adjusted for the register allocation use-case: spilling, pre-coloured nodes
- Widely adopted approach (GCC, LLVM)

References:

- Michael Matz. *Design and Implementation of a Graph Coloring Register Allocator for GCC*. 2003.

The End

Thank you!

Questions?