Graph Colouring

In 1852, while colouring a map of the counties in England, botanist Francis Gutherie noticed that he needed just four colours to colour the map so that no two adjacent counties had the same colour. He then posed the seemingly innocent conjecture: *was this true for any map?*

Convert the map into an interference graph where the counties are represented by nodes and there is an edge between counties that share a common border, then colour the graph so that no two counties that share a common border have the same colour.
Unfortunately this problem is NP-complete

Use graph colouring heuristics to find a solution

Suppose the graph $G$ contains a node $m$ with fewer than $K$ neighbours (where $K$ is the number of available colours)

Let $G'$ be the graph $G - \{m\}$ obtained by removing node $m$ from $G$

If $G'$ can be coloured then so can $G$, for when node $m$ is added to the coloured graph $G'$ its neighbouring nodes will have at most $K - 1$ colours and hence a free colour can always be found for node $m$
Register Allocation

In 1982, 130 years after Francis Guthrie suggested it was always possible to colour a map using just four colours, Gregory Chaitin, a computer scientist working for IBM Research, published the paper “Register Allocation and Spilling via Graph Coloring.”

Chaitin pointed out that global (intra-procedural) register allocation is equivalent to colouring the vertices of a graph with a fixed number of colours, and hence, given a machine with \( K \) registers, the problem is how to assign the registers to the variable names in a program in order to satisfy all the interferences (ie so that no two live variables in a program are stored in the same register at the same time).

For each procedure in a program construct an interference graph \( G = (V, E) \) where:

i. Every vertex \( v \in V \) corresponds to a distinct variable name in the intermediate code representation of a program.

ii. An edge \( (u,v) \in E \) joins two vertices \( u \) and \( v \) if there is a statement in the program where the variable represented by one of them is defined (assigned a value) while the variable represented by the other is alive (holds a value that is needed subsequently) — hence they cannot reside in the same register.

Note: Intermediate code representation assumes the availability of an unlimited (infinite) number of registers.

When a colouring algorithm fails to colour \( G \), then some variable(s) must be spilled (ie stored in a temporary location). Which variable(s)? Where to insert the code?
CS4071 — 2016/17
Advanced Compiler Design Outline

A little basic material not covered in CS3071 including:

• Bottom-up parsing methods
• Intermediate code representations
• Basic code generation methods

The main topics covered in this course will include:

• Advanced code generation methods
• Common sub-expression elimination
• Register allocation and spilling policies
• Optimal code generation for arithmetic expressions
• Global dataflow analysis and optimization techniques
• Static Single Assignment Form

CS4071 — 2015/156
Advanced Compiler Design Outline (continued)

Additionally, subject to time constraints, a selection of topics chosen from the following list will be included:

• Compilation strategies for parallel processors
• Method invocation in object oriented languages
• Debugging issues in optimizing compilers
• Garbage collection techniques
• JIT compilation
• Etc
Live-in: J K
G := mem[J+12]
H := K - 1
F := G * H
E := mem[J+8]
M := mem[J+16]
B := mem[F]
C := E + 8
D := C
K := M + 4
J := B

Live-out: D J K

Live-in: J K
G := mem[J+12]
H := K - 1
F := G * H
E := mem[J+8]
M := mem[J+16]
B := mem[F]
C := E + 8
D := C
K := M + 4
J := B

Live-out: D J K

Live-out:
G J K
G := mem[J+12]
G H J
H := K - 1
F J
F := G * H
E F J
E := mem[J+8]
E F M
M := mem[J+16]
B E M
B := mem[F]
B C M
C := E + 8
B D M
D := C
B D K
K := M + 4
D J K
J := B
C H G F K D J B E

C H G F K D J B E