Assignment 3: Register allocation for $\mathcal{L}_{Var}$

(Deadline: 06.12.2021 09:00)

Exercise 0: Install Requirements, Testing

Run `pip3.10 install -r requirements.txt` in your cloned repositories directory.

You can test functionality of exercises 1-3 by adding tests to the `register_allocation_test.py` file and executing `pytest` within your cloned repository.

Exercise 1: Liveness Analysis

Given the following rule (cf. Section 3.2):

$$L_{\texttt{before}}(k) = (L_{\texttt{after}}(k) - W(k)) \cup R(k)$$

Implement the `uncover_live` function inside `register_allocation.py`. We recommend creating auxiliary functions to 1) compute the set of locations that appear in an *arg*, 2) compute the locations read by an instruction (the R function), and 3) the locations written by an instruction (the W function). The `callq` instruction should include all of the caller-saved registers in its write-set W because the calling convention says that those registers may be written to during the function call. Likewise, the `callq` instruction should include the appropriate argument-passing registers in its read-set R, depending on the arity of the function being called. (This is why the abstract syntax for `callq` includes the arity.)

Note: The book recommends that this function returns a dict mapping from instructions to sets. A list suffices and is what is used in the test.

Exercise 2: Interference Graph

Given the following rules (cf. Section 3.3):

1. If instruction I_k is a move instruction of the form `movq` s`,` d, then for every $v \in L_{\mathsf{after}}(k)$, if $v \neq d$ and $v \neq s$, add the edge (d, v).
2. For any other instruction I_k, for every $d \in W(k)$ and every $v \in L_{\mathsf{after}}(k)$, if $v \neq d$, add the edge (d, v).

Implement the `build_interference` function inside `register_allocation.py`, which should return the interference graph. Use the implementation of undirected graphs found in `graph.py`.

Exercise 3: Graph Coloring

Given the following algorithm (cf. Section 3.4):

$W \leftarrow$ vertices(G)
while $W \neq \emptyset$ do
    pick a vertex u from W with the highest saturation,
        breaking ties randomly

find the lowest color c that is not in $\{\text{color}[v] \ : \ v \in \text{adjacent}(u)\}$
$\text{color}[u] \leftarrow c$
$W \leftarrow W - \{u\}$

Implement the `color_graph` function inside `register_allocation.py`, which should return a mapping from variables to colors. Also implement the `pre_color` function, which should handle caller-saved and argument-passing registers at function calls. You may use the color/register mapping found inside `register_allocation.py` for precoloring. You may also use the priority queue found in `priority_queue.py`.

Exercise 4: Allocate Registers/Assign Homes

Using `color_graph` and `build_interference`, implement the `allocate_registers` pass in `compiler.py`, which should return a map from variables to registers/the stack. You may use the color/register mapping found inside `register_allocation.py`.

Exercise 5: Update Patch Instructions & Prelude and Conclusion

Update the `patch_instructions` compiler pass to delete trivial moves, i.e. moves where source and destination are the same location (these may appear when two variables share the same register).

Finally, update the `prelude_and_conclusion` pass: Any callee-saved registers used by the register allocator must be saved to the stack in the prelude and restored in the conclusion. The field `used_callee` of the `Compiler` class may be useful.