Liveness Analysis

Mia de Leon, Anurag Kashyap, Austin Kao

April 9, 2021

Description

After selecting the instructions, we then worked on analyzing the live registers for each block of our program. This will help our compiler of the Tiger language perform register allocation.

Flow Graph

We created a flow graph that graphs the registers being used and the registers being defined in each block of the program. Each block of the program becomes a node in this graph, with its own unique node ID, a list of the registers used in the block, and a list of the registers defined in the block. We decided to use the functional graph provided to us by Professor Drew Hilton to implement our flow graph.

Liveness Analysis

Using our flow graph, we conducted a liveness analysis to figure out the set of live registers out of the program. We implemented the algorithm found in Appel's book where we initialize these sets to the empty set for each node in the flow graph. Then, we update each set by finding the union of the live-in set for each successor of the node. We also determine the live-in set by finding the union of the set of used registers and the difference between the set of defined registers and the set of registers live-out of the node. These updates continue until the live-out set of each node stops changing.

Interference Graph

After conducting the liveness analysis, we can use the information to generate an interference graph, which determines the registers that cannot be stored in the same register when register allocation is conducted. Like our flow graph, our interference graph uses the functional graph provided to us by Professor Drew Hilton.