Writeup
Group: junhanz vigneshb

Dataflow Analysis

Iterative Framework

The iterative framework is implemented as a C++ abstract class containing true virtual functions for the actual pass to implement. Those virtual functions include determining the domain, the transfer functions and phi special case transfer function, the meet operation, functions indicating it's a forward pass or backward pass, what are the initial value and how to print out one element of the domain. A static transfer function is also kindly provided in the class for the type of transfer function like result=gen|(prev-kill) where the user can use and only need to fill out the gen and kill functions.

The framework first does a swoop on the instructions to get the domain. The elements are stored as Value pointers so that the class isn't required to be a template class and implemented in the .h file. After this the initial value is initialized for every basic block (actually everywhere) according to the virtual initial and boundary functions. Then depending on the virtual function, it keeps doing forward or backward passes using the transfer functions and meet function until nothing changes anymore. Then at the end it prints out each basic block's instruction as well as the elements valid at that place, referring to the given function to printing out each element.

Dataflow Analysis

The dataflow analysis is very simple, one only have to fill out the corresponding virtual functions from the framework and that's it!

Liveness Analysis

The required virtual functions are filled out according to Figure 9.21 of the book. The only thing worth mentioning is that for the PHI instructions, even though they are contained in the entry of the succeeding basic block, it is analyzed in each of the preceding basic block at the exit, so each basic block knows which variables are used.

Available Expressions

The required virtual functions are filled out according to Figure 9.21 of the book. The only thing worth mentioning is that the elements are stored as Value type in the domain, so the transfer function have to first convert it to expressions before comparing.

Loop Invariant Code Motion

	Loop invariant instruction	To preheader?	Justification
S2	y=5	No	Variable assigned elsewhere in loop
S3	q=7	Yes	All requirements fulfilled
S5	h=3	No	Does not dominate exit
S6	x=1	No	Does not dominate exit
S8	h=4	No	Does not dominate all blocks in the loop that used the variable assigned.
S9	x=1	No	
S10	m=y+7	No	Does not dominate exit
S12	y=7	No	Does not dominate exit
S13	r=q+5	No	Does not dominate exit

Lazy Code Motion