CS 6241 : Compiler Optimizations
Homework II
Points : 100
Due : March 9th 2018, 11:55 pm

Important policies:

1. Georgia Tech honor code will be strictly enforced.
2. Give enough details but don't write essays – be crisp and precise. Write solutions using pseudo-code algorithms and notations. Please follow suggested page limits to each question.
3. Make and write suitable assumptions where necessary highlighting them.

Question I [20 points] [1.5 pages]

Consider the problem of detecting available expressions for CSE. (a) Develop and reason about the safety condition for the analysis considering the optimization of common subexpression elimination in terms of actual vs found by the analysis. (b) In the presence of the pointers, revise the data-flow framework for Available Expressions for all different cases of points-to sets as discussed in class. Using the revised framework, show how you meet the safety conditions developed in (a). Illustrate via an example. (c) Modify the available expression analysis to find available expressions on demand inside a given basic block B – this is called demand driven analysis mainly used for doing CSE only within loops. Note that here you will not perform whole CFG analysis but will only focus on detecting availability of only those expressions which are generated inside loop basic blocks for the purposes of redundancy elimination.

Question II [30 points] [2 pages]

A very busy expression e is defined as the one that is anticipatable at a program point p ie, there is an evaluation of e on every path that begins at p before the end of the function or redefinition of its operands. It is proposed to hoist e at a program point p so that it eliminates original expressions *maximally* leading to code size reduction.

(a) Show an example which shows when is it legal to hoist very busy expressions up and when is it not. Also show the case of *maximal hoisting* when hoisting is legal.

(b) Assuming that the Anticipatable_IN[B] and Anticipatable_OUT[B] sets are already computed, write a condition for determining legal hoisting points for a set of Very Busy Expressions. Also write a condition for determining maximal hoisting (for maximally covering very busy expressions) – write an algorithm that uses both of these conditions to maximally and legally hoist very busy expressions. The algorithm must reject illegal hoistings.

(c) Show on the example CFG, how maximal hoisting is performed and how the algorithm rejects illegal hoistings.

Question III [25 points] [1 pages]

Dataflow analysis can be used to detect unsound programs that have uninitialized variables. An uninitialized variable is the one which is used at least one program point before it is defined. Since garbage values can reside in uninitialized variables, it is desirable to detect them. *An uninitialized variable* can propagate such values to other variables' definitions that are derived from them and this can continue transitively. We call such a set of definitions as *unsound*. Implement an analysis pass to detect and warn about *all uninitialized variables* and *unsound definitions* in a program. Turn off LLVM analyses and optimizations and identify the right place in the IR (non SSA) to insert such an analysis. Note: A sound definition can kill the unsound one. Assume that no prior analysis is done (no reaching definition analysis done as well). *You will cast this problem in a manner similar to reaching definition and solve it from scratch without relying on the results of any other analysis*. Please write the data-flow algorithm in your report and document results on a given benchmark. You will turn in both the code and the results.

Question IV [25 points] [1 pages]

On the given CFG shown below (Figure 1), perform PRE showing all the steps.
Does PRE framework discussed in the class (Drechler and Knoop's papers) completely remove partial redundancy? Show an example where partial redundancy is not completely removed.

Figure is on the next page.

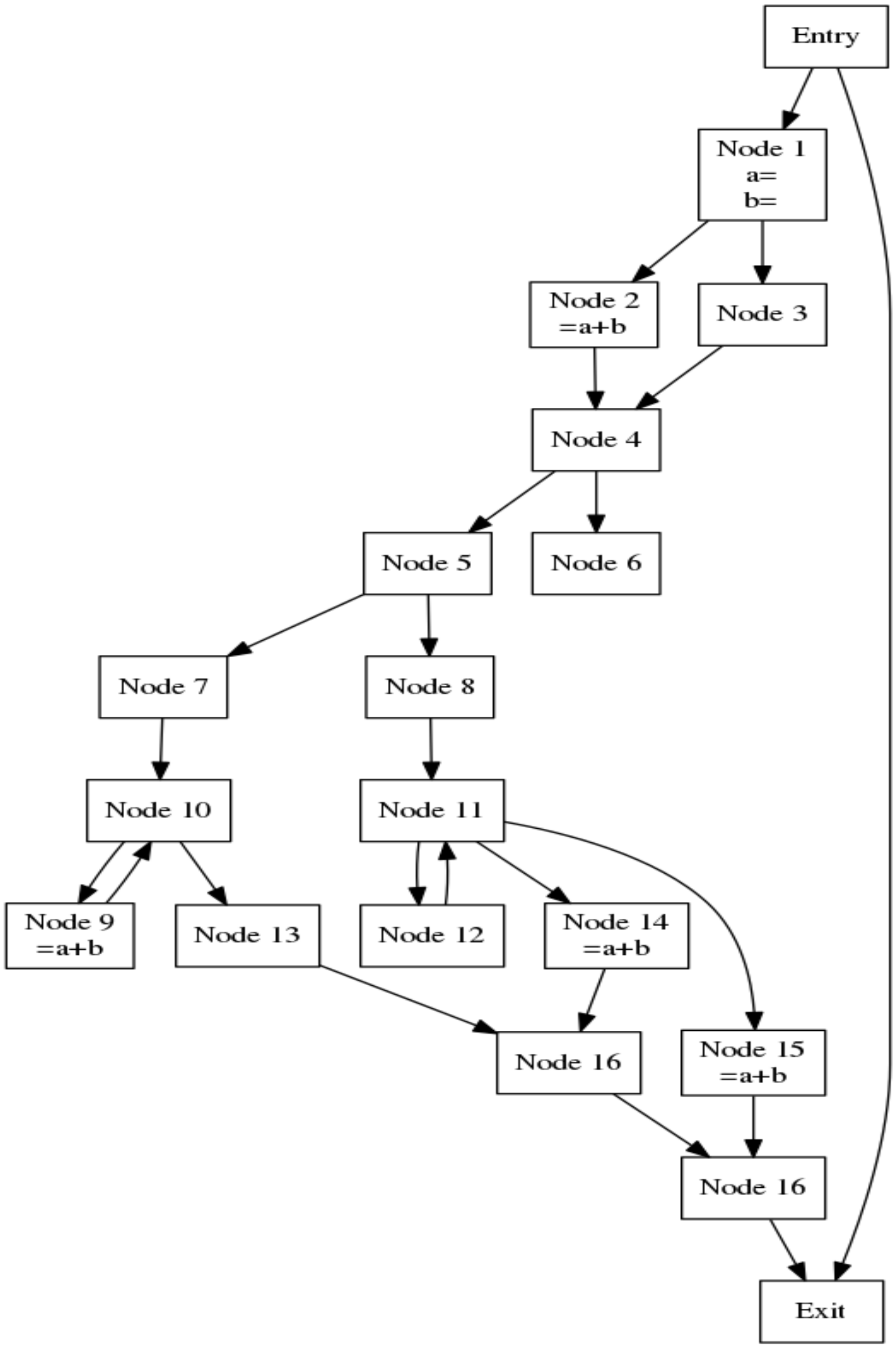

Figure 1. CFG for performing PRE.