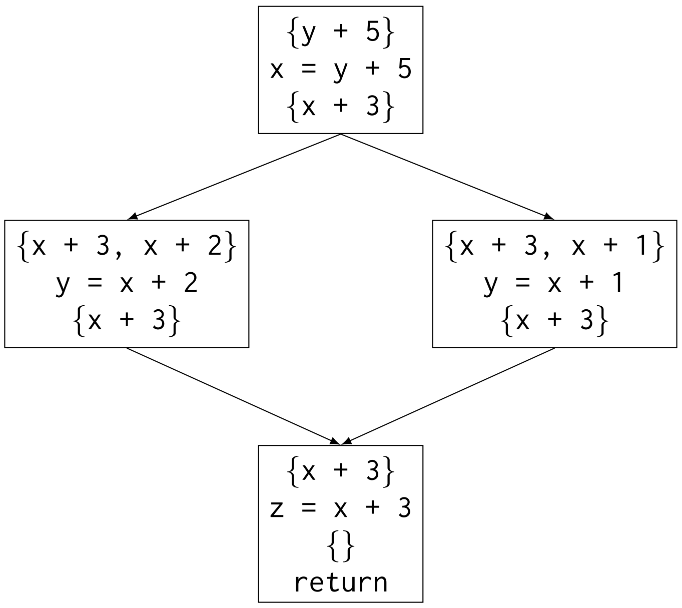
CS164 Programming Languages and Compilers **Spring 2024**

Discussion Worksheet 10: Data-Flow Analysis

1 Very Busy Expressions

One global analysis used for code-hoisting optimizations is very busy expressions. An expression e is very busy at some program point p if e will definitely be evaluated along every path from p to the exit (i.e. the return), without encountering a modification to any variable referenced by e before the expression is evaluated. As an example, the following CFG is annotated at each statement with the set of very busy expressions before and after that statement.



1.1 Is this a forwards or backwards analysis?

Solution: In order to know if an expression is very busy at some statement, we have to consider paths of analysis *after* executing that statement. Thus, information about the expression flows from later instructions to earlier instructions, which makes this a backwards analysis.

1.2 What is the meet operation between multiple paths? In other words, given the sets of very busy expressions of all possible subsequent statements, how do we combine them to obtain the set of very busy expressions for the current statement?

Solution: An expression is very busy iff it is very busy along *all possible* succeeding paths. Therefore, we use the intersection between sets as the meet operation.

1.3 Is this a may or must analysis?

Solution: This is another way to state the answer to the previous subpart. Since each subsequent path "must" have an expression in its set for it to be included in the current statement's set, this is a must analysis.

1.4 Now consider whether one specific expression is very busy at each point in the program. Let $B_{in}(e, s)$ and $B_{out}(e, s)$ denote whether the expression e is very busy right before and after statement s, respectively. Fill out the following the rules for performing the analysis, assuming that e and $e2$ are two different expressions:

(a) B_{in}(e, `var = e`) = __________ Solution: True.

(b) If `e` references `var`, then B_{in}(e, `var = e2`) = __________ Solution: False.

(c) If `e` doesn't reference `var`, then B_{in}(e, `var = e2`) = __________ Solution: B_{out}(e, `var = e2`).

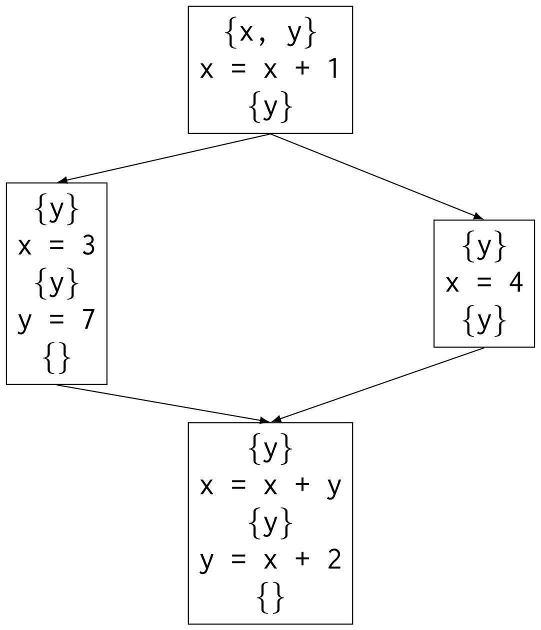
(d) $B_{out}(e, s)$ = __________$(\{B_{in}(e, p) \mid p \text{ is a successor of } s\})$ (fill in with a logical operator)

Solution: AND

2 Possibly Uninitialized Values

In this question, we want to build an analysis which tracks possibly uninitialized values.

We'll start out by considering a variable `x` to be possibly uninitialized at a statement `s` if and only if there is at least one path up to `s` in which `x` is never assigned to. If `x` is assigned to, we say it is initialized, **regardless of what the right-hand side of the assignment contains**. That is, we would like an analysis that computes the following set of dataflow facts at each program point:



We can create a gen/kill dataflow analysis that computes the set of possibly uninitialized values at every point.

2.1 What kind of analysis is this?

(a) forwards may (b) forwards must (c) backwards may (d) backwards must Solution: (a)

2.2 What is the meet operation?

(a) set union (b) complement of set union (c) set difference (d) set intersection Solution: (a)

2.3 What is `Gen(s)`?

(a) {Variables **used** in `s`} (i.e., read from before an assignment)
(b) {Variables **assigned to** in `s`}
(c) {Variables **assigned to** in `s`} - {Variables **used** in `s`}
(d) {Variables **used** in `s`} - {Variables **assigned to** in `s`}
(e) All variables
(f) Empty set

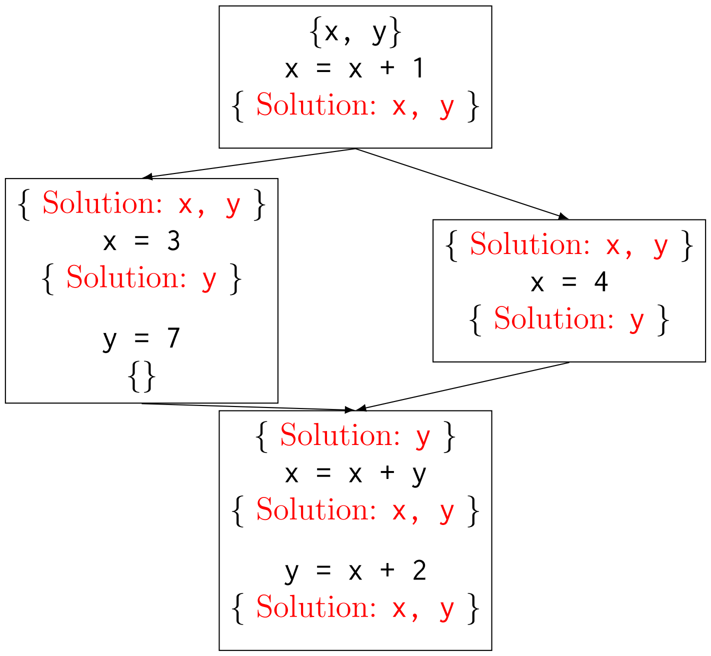
Solution: (f)

2.4 What is `Kill(s)`?

(a) {Variables **used** in `s`} (i.e., read from before an assignment)
(b) {Variables **assigned to** in `s`}
(c) {Variables **assigned to** in `s`} - {Variables **used** in `s`}
(d) {Variables **used** in `s`} - {Variables **assigned to** in `s`}
(e) All variables
(f) Empty set

Solution: (b)

Suppose instead we want to construct an analysis which more precisely computes possibly uninitialized values. In particular, we now say a variable `x` is possibly uninitialized at a statement `s` if there exists a path leading to `s` along which `x` is never assigned to an initialized value. I.e., the statement `x = x` does not initialize `x` if `x` is not already initialized. Based on this new analysis, complete the following dataflow analysis diagram:



2.5 This precise analysis cannot be formulated as a Gen/Kill analysis. What property of the Gen/Kill analysis prevents us from constructing this more precise analysis?

(a) The Gen(s) and Kill(s) functions cannot vary for different types of statements (e.g. cannot distinguish between assignments, if statements).

(b) For a given statement s, the values of Gen(s) and Kill(s) cannot depend on In(s) or Out(s).

(c) The Gen/Kill Analysis must be either forwards or backwards.

(d) Top must be the same for all statements.

(e) Dataflow facts can only grow towards Top.

Solution: (b)

We will now construct a dataflow analysis that performs the precise uninitialized variable analysis described in the previous section. Construct this analysis by completing the following rules.

Let `U(s)` designate the set of variables used in the statement `s`, and `A(s)` designate the variables assigned to in the statement `s`.

2.6 For the predecessor meet rule:

$$\texttt{In(s)} = \sqcap \{\texttt{Out(s')} | \texttt{s'} \in \texttt{pred(s)}\}$$

What is $\sqcap$?

(a) set union (b) complement of set union (c) set difference (d) set intersection Solution: (a)

2.7 If `A(s)` is empty (ie `s` is not an assignment), then `Out(s)` = ________.

Which option completes the blank?

(a) `In(s)` (b) `In(s) - A(s)` (c) `In(s) - U(s)` (d) `In(s)` $\cup$ `A(s)`
(a) `In(s)` $\cup$ `U(s)` (b) `U(s)` (c) `A(s)` (d) $\emptyset$ Solution: (a)

2.8 If `s` is an assignment, and $\forall \texttt{v} \in \texttt{U(s)} : \texttt{v} \notin \texttt{In(s)}$, then `Out(s)` = ________.

Which option completes the blank?

(a) `In(s)` (b) `In(s) - A(s)` (c) `In(s) - U(s)` (d) `In(s)` $\cup$ `A(s)`
(a) `In(s)` $\cup$ `U(s)` (b) `U(s)` (c) `A(s)` (d) $\emptyset$ Solution: (b)

2.9 If `s` is an assignment, and $\exists \texttt{v} \in \texttt{U(s)} : \texttt{v} \in \texttt{In(s)}$, then `Out(s)` = ________.

Which option completes the blank?

(a) `In(s)` (b) `In(s) - A(s)` (c) `In(s) - U(s)` (d) `In(s)` $\cup$ `A(s)`
(a) `In(s)` $\cup$ `U(s)` (b) `U(s)` (c) `A(s)` (d) $\emptyset$ Solution: (d)