COL729 Midterm Exam
Compiler Optimization
Sem II, 2018-19

Answer all 3 questions Max. Marks: 25

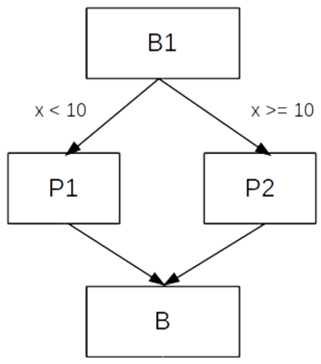
1. Consider the reaching-definitions analysis, which computes all the definitions of the program that *may* reach a program point (as discussed in class).

a. Give the data-flow analysis formulation to compute reaching definitions at each program point. E.g., what is the set of values (V), transfer functions (F), etc. [3]

Reaching Definitions:

Domain	: Powerset(set of all definitions in the program)
Transfer function	: out[b] = Gen[b] U (in[b] - Kill[b]) Gen[b] -- set of definitions in basic block b Kill[b] -- set of definitions killed by b i.e. the definitions reaching b for variables defined by b
Direction	: Forward
Meet operator	: union
Boundary condition	: out[entry] = empty
Initialization condition	: empty (Φ)

b. . The data-flow analysis formulation does not take advantage of the semantics of conditionals. Suppose we find at the end of a basic block a test such as if (x < 10) goto .. . How could we use our understanding of what the test x < 10 means to improve our knowledge of reaching definitions? Remember, "improve" here means that we eliminate certain reaching definitions that really cannot ever reach a certain program point. Just specify what aspects of the formulation described above need to change, and how. [5]



If we have a constant definition of x at B1, then depending on the value of x one of the branches P1 or P2 can be marked as unreachable and its corresponding definitions will not be propagated.

We can thus change the transfer function F(in[b]) to

```
If definitions in in[b] doesn't satisfy the edge condition then
    out[b] = empty
else
    out[b] = Gen[b] U (in[b] - Kill[b])
```

Alternatively, we could also change the meet operator in the analysis from union to

```
out[P1] ∧ out[P2] =
        If P1 is unreachable then out[P2]
        If P2 is unreachable then out[P1]
        If both P1 and P2 are unreachable then empty
        If both P1 and P2 are reachable then out[P1] U out[P2]
```

2. a. Consider the “needed” expressions analysis in the partial-redundancy elimination (lazy code motion) algorithm. Are the transfer functions for this analysis monotone? [1] Are they distributive? [1.5] Prove both statements. [2.5]

Transfer function of “needed” expressions is

$F_B(x) = E_{use} \cup (x - E_{kill})$,

Where Euse is the set of expressions used by the instructions in B,

Ekill is the set of expressions *killed* by the definitions in B

<u>Yes, the transfer function is monotonic</u>

Condition for monotone: $(\forall x,y)\ x \leq y \Rightarrow f(x) \leq f(y)$

In this case,

$x \leq y \Rightarrow E_{use} \cup (x - E_{kill}) \leq E_{use} \cup (y - E_{kill})$

Clearly, the set $(x - E_{kill}) \subseteq (y - E_{kill})$, if $x \leq y$

<u>Yes, the transfer function is distributive</u>

Condition for distributive: $(\forall x,y)\ f(x \wedge y) = f(x) \wedge f(y)$, where $\wedge$ is meet operator

In this case,

$f(x \wedge y) = E_{use} \cup ((x \cap y) - E_{kill})$

$f(x) \wedge f(y) = (E_{use} \cup (x - E_{kill})) \cap (E_{use} \cup (y - E_{kill}))$

$= E_{use} \cup ((x - E_{kill}) \cap (y - E_{kill}))$

$= E_{use} \cup ((x \cap y) - E_{kill})$

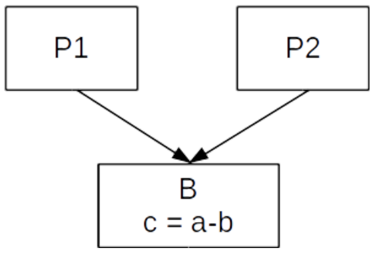
b. Give one example of a data-flow analysis where the transfer functions are not distributive. Provide an example to show that for such an analysis, the maximum fixed point solution (MFP) is different from the Meet-over-paths (MOP) solution. Your example should be different from the standard constant-propagation example discussed in class. [5]

Data-flow analysis: range analysis for variables

Example:

Values V = { [a,b] : a ≤ b }

Meet operator or ([a,b], [c,d]) = [min(a,c) , max(b,d)]



P1: a ∈ [0, 20], b ∈ [5, 10]

P2: a ∈ [50, 70], b ∈ [55, 60]

B : instruction is c = a-b

<u>MFP solution:</u>

Meet for a is

$\wedge$([0, 20], [50, 70]) = [0, 70]

Meet for b is

$\wedge$([5, 10], [55, 60]) = [5, 60]

Range for c is

[0-60, 70-5] = [-60, 65]

<u>MOP solution:</u>

Range of c for P1 -> B is

[0-10, 20-5] = [-10, 15]

Range of c for P2 -> B is

[50-60, 70-55] = [-10, 15]

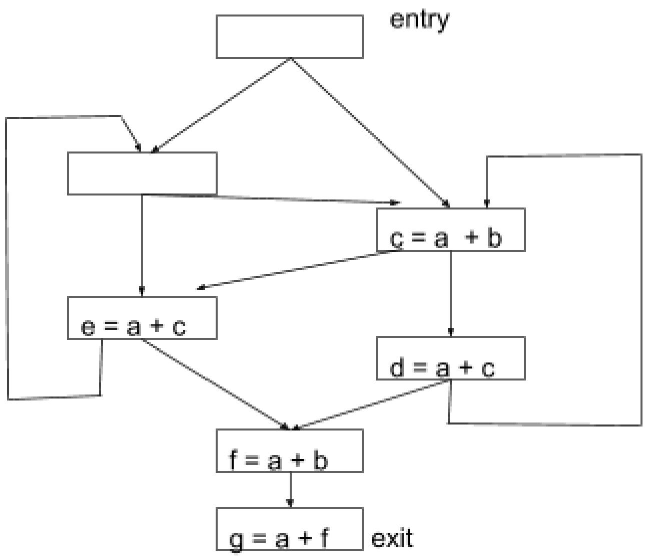
Meet is [-10, 15]

MOP $\subset$ MFP

Other possible analysis could be common-subexpression elimination, Affine expression identification.

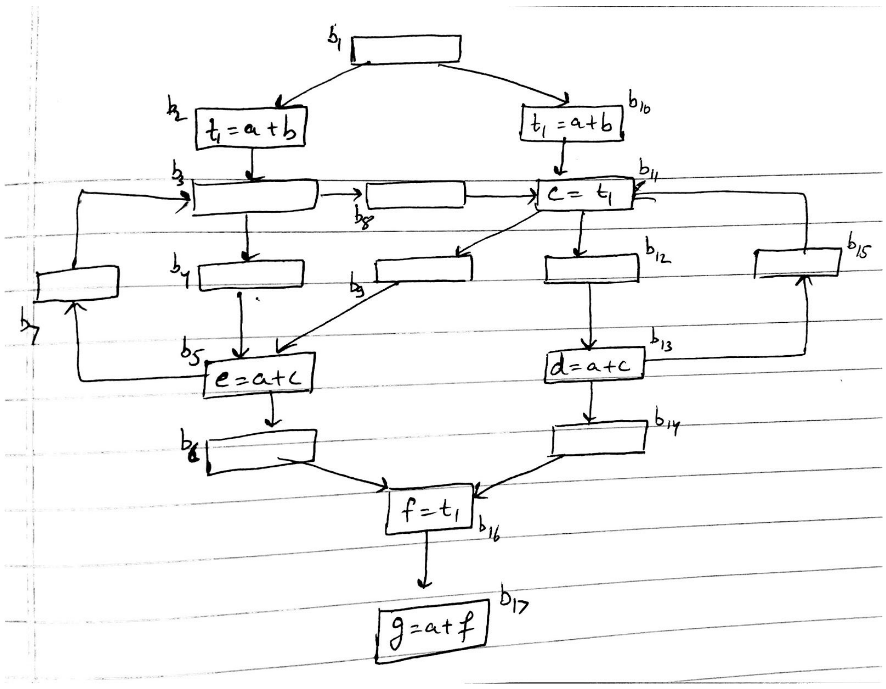
No marks if constant propagation as example analysis

3. Lazy code motion



a. Apply the lazy code motion algorithm described in class to the program above. Show the result of the optimization --- it is not necessary to show any intermediate steps. (Introduce new basic blocks as necessary). [4]

Answer for this program with all critical edges removed is shown below:



Expression a+b is needed in : IN : block 1 to 16 OUT : block 1 to 15
Expression a+c is needed in : IN : 4,5,9,12,13 OUT : 4,9,11,12
Expression a+f is needed in : IN : 17 OUT : 16
Expression a+b is missing in : IN : 1
Expression a+c is missing in : IN : 1,2,3,4,8,9,10,11,12 OUT : 1,2,3,8,10,11
Expression a+f is missing in : IN : 1 to 17 OUT : 1 to 16

Expression a+b is postponed in : IN : 2,10 OUT : 1, 2, 10
Expression a+c is postponed in : IN : 5,13 OUT : 4, 9, 12
Expression a+f is postponed in :

Full marks for correct answer. Partial marking if result is correct according to some passes.

b. Are there redundant operations remaining after the optimization? If not, explain why not. If yes, explain why redundant operations were left even after applying the transformation. [3]

Yes there are some redundancies still left e.g. loop 3-4-5-7-3 will calculate a+c again and again though it is not needed. This is because of edge 11-9-5 which also joins this loop and this path kills expression c and so a+c must be recalculated.

This redundancy can also be reduced if one of the two paths among 3-4-5 or 11-12-13 is replicated, but that is not supported by the lazy-code motion algorithm.

2 marks for identifying redundancies. 1 for explanation.