Homework 4

Problem 1

a)

First of all, Live variable analysis has property of L, we have finite many program label l and in the property space we define our S as power set of set of program label, meet operator in property space defined as set union.

Secondly, the transfer function of Live Variable Analysis is defined as $LV_{entry}(l) = LV_{exit}(l) \setminus \text{write}(l) \cup \text{read}(l)$, apparently this is $P(D) \to P(D)$ and $LV_{exit}(l) \setminus \text{write}(l)$ is equivalent to $LV_{exit}(l) \cap (LV_{exit}(l) \setminus \text{write}(l))$ and clearly $(LV_{exit}(l) \setminus \text{write}(l)) \subseteq S$ where S is the power set of all program label, therefore the transfer function of Live Variable Analysis statisfies F

b)

For generality, let the operator $\sqcap$ be $\cap$, since $\cap \text{ and } \cup$ share the same rule of associativity. commutativity, distributivity, which for our prove will provide same result

let $l_a, l_b \in L$

$$\begin{aligned} f(l_a \sqcap l_b) &= f(l_a \cap l_b) \\ &= (l_a \cap l_b) \cap K \cup G \\ &= ((l_a \cap K) \cap (l_b \cap K)) \cup G \\ &= ((l_a \cap K) \cup G) \cap ((l_b \cap K) \cup G) \\ &= f(l_a) \cap f(l_b) \end{aligned}$$

This proved distributivity.

c)

Example:

$L = (\mathcal{P}(\mathcal{D}), \cap)$

$\mathcal{F} = \{f : \mathcal{P}(\mathcal{D}) \to \mathcal{P}(\mathcal{D}) | \forall f \in \mathcal{F}, \exists a \notin D \wedge \forall S \in D, f(S) = S + \{a\}\}$

Faint variable analysis is an another example of distributive framework but it nont is bit vector framework.

Problem 2

a)

partially available expressions: An expression is partially available at a program point l, if the expression is available along some path to point l.

$Gen(l_n)$ = { e | expression e is evaluated in basic block n and this evaluation is not followed by a definition of any operand of e}

$Kill(l_n)$ = { e | basic block n contains a definition of an operand of e}

b)

An expression e is anticipable at a program point l, if every path from point l to the program exit contains an evaluation of e which is not preceded by a redefinition of any operand of e.

c)

To remove PRE

1. Identify partial redundancies
2. Identify program points where computations can be inserted
3. Insert expressions
4. Partial redundancies become total redundancies $\Longrightarrow$ Delete them.

Placement Possible Analysis

An expression can be safely inserted at a program point p if it is

1. If it is available at p, then there is no need to insert it at p.
2. If it is anticipable at p then all such occurrence should be inserted to p.

An expression should be inserted to p provided it can be inserted to p along all paths from p to exit.

- safety of Inserting to the exit of a block.

  Should be inserted only if it can be inserted to the entry of all successors.

- safety of Inserting to the entry of a block.

  Should be inserted only if it is upwards exposed

  it can be inserted to its exit and is transparent in the block .

- Desirability of inserting to the entry of a block.

Should be inserted only if:

- it is partially available,
- For each predecessor
  - it is inserted to its exit,
  - is available at its exit.

Partial availability:

$$PA_{entry}(l_0) = \emptyset$$
$$PA_{entry}(l_i) = \bigcup_{l_j \in pred(l_i)} PA_{exit}(l_j)$$
$$PA_{exit}(l_i) = Gen(l_i) \cup (PA_{entry}(l_i) \setminus Kill(l_i))$$

Total availability:

$$TA_{entry}(l_0) = \emptyset$$
$$TA_{entry}(l_i) = \bigcup_{l_j \in pred(l_j)} TA_{exit}(l_j)$$
$$TA_{exit}(l_i) = Gen(l_i) \cup (TA_{entry}(l_i) \setminus Kill(l_i))$$

PRE Data Flow Equation:

$$Entry(l_i) = PA_{entry}(l_i) \cap (AntGen(l_i) \cup (Exit(l_i) \setminus Kill(l_i))) \bigcap_{l_j \in pred(l_i)} (Exit(l_j) \cup (TA_{exit}(l_j)))$$
$$Exit(l_n) = \emptyset$$
$$Exit(l_i) = \bigcap_{l_j \in succ(l_i)} Entry(l_j)$$

d)

insert l:

An expression is inserted at the exit of node n if

- it can be placed at the exit of n, AND
- it is not available at the exit of n, AND
- it cannot be Inserted out of n, OR it is modified in n.

$Insert(n) = Exit(n) \cap (\neg AvOut(n)) \cap (\neg Inn \cup Kill(n))$

delete l:

An expression is redundant in node n if

- it can be placed at the entry of n, AND

- it is upwards exposed in $ode n.

$$Redundant(n) = Entry(n) \cap AntGen(n)$$