

Homework 5: Dataflow Analysis

Solution

1. Build control-flow graph for the given code.

```
A: i = 0;
   change=false;
   |
   V
|-----> B: while (i < size) -----|
|           |                         |
|           V                         |
|           C: elem1 = arr[i];         |
|           elem2 = elem1 * elem1 / 2; |
|           if (elem2 != elem1)       |
|           |                         |
|           V                         |
|           D: change = true;         |
|           |                         |
|           V                         |
|           E: cur[i] = elem2; <-----|
|----- i = i + 1;                   |
|                                     |
|           F: if (!change) <-----|
|           | |                       | |
|           | |-----|               |
|           V |                       |
|           G: res = v;                 |
|           return res;                 |
|                                     |
|           H: res = 0; <-----|
|           i = 0;                     |
|           |                         |
|           V                         |
|----->I : while (i < size)-----|
|           |                         |
|           V                         |
|           J: res = res + cur[i];     |
|----- ++i;                         |
|                                     |
|           K: return res; <-----|
```

2. Compute the set of live variables at the exit of each basic block. Note that you should treat each array (e.g., arr and cur) as a single collective variable, and any modification or use of any element of the array would qualify a definition or use of the whole array.

	Kill	UEvar	Lout	Lin	Lout	Lin
A	i,change	\emptyset	v, cur, size, change, i, arr	v, cur, size, arr	v, cur, size, change, i, arr	v, cur, size, arr
B	\emptyset	i, size	v, cur, size, change, i, arr	v, cur, size, change, i, arr	v, cur, size, change, i, arr	v, cur, size, change, i, arr
C	elem1,elem2	i, arr	elem2, i	i, arr	v, cur, size, change, i, arr, elem2	v, cur, size, change, i, arr
D	change	\emptyset	elem2, i	elem2, i	v, cur, size, change, i, arr, elem2	v, cur, size, change, i, arr, elem2
E	\emptyset	elem2, i	\emptyset	elem2, i	v, cur, size, change, i, arr	v, cur, size, change, i, arr, elem2
F	\emptyset	change	v, cur, size	v, cur, size, change	v, cur, size	v, cur, size, change
G	res	v	\emptyset	v	\emptyset	v
H	res, i	\emptyset	res, cur, i, size	cur, size	res, cur, i, size	cur, size
I	\emptyset	i,size	res,cur,i	res, cur, i, size	res, cur, i, size	res, cur, i, size
J	res, i	res,cur,i	\emptyset	res, cur, i	res, cur, i, size	res, cur, i, size
K	\emptyset	res	\emptyset	res	\emptyset	res

3. Convert the CFG into SSA form. Note when constructing SSA form, you can ignore arrays as they can be modified more than once; i.e., they are not part of the SSA form.

Note: phi functions for elem1 and elem2 are not necessary as they are not alive at the entry to B. But no deductions should be made if the redundant phi functions are inserted.

A: i0 = 0;

```

change0=false;
  |
  V
|-----> B: change1=phi(change3,change0);
|         i1 = phi(i2,i0);
|         while (i1 < size) -----|
|         |
|         V
|         C: elem1 = arr[i1];
|            elem2 = elem1 * elem1 / 2;
|            if (elem2 != elem1)
|            |
|            V
|            D: change2 = true;
|            |
|            V
|            E: change3=phi(change2,change1);
|               cur[i1] = elem2;
|-----i2 = i1 + 1;
|
|         F: if (!change1) <-----|
|            |
|            |-----|
|            V
|            G: res0 = v;
|               return res0;
|
|         H: res1 = 0; <-----|
|            i3 = 0;
|            |
|            V
|----->I: res2=phi(res3,res1);
|         i4 = phi(i5,i3);
|         while (i4 < size)-----|
|         |
|         V
|         J: res3 = res2 + cur[i4];
|-----i5=i4+1;
|
|         K: return res2; <-----|

```