Optimising Compilers: Exercise 3

Questions

1. Show example code which hash-based value numbering can optimise but global value numbering and partial redundancy elimination cannot.
2. Show example code which global value numbering can optimise but hash-based value numbering and partial redundancy elimination cannot.
3. Show example code which partial redundancy elimination can optimise but hash-based or global value numbering cannot.
4. Consider the code below and describe how hash-based value numbering deals with it.

```
int h(int a, int b)
{
        int     c, d;

        if (a != b) {
                c = a * b;
                d = 2;
        } else {
                c = 3;
                d = a * b;
        }

        return c * d + a * b;
}
```

5. Repeat for global value numbering.
6. Repeat for partial redundancy elimination. For PRE: only a very brief description is necessary in ***this*** exercise since the details of SSAPRE are to be discussed tomorrow.

Answers

1. Of these only hash-based value numbering considers expressions with different operators. For example in:

```
int x, y;
void f(int a)
{
        x = 2 * a;
        y = a + a;
}
```

2. PRE does not consider expressions with different variables and hash-based value numbering only has available expressions from ancestors in the dominator tree, and hence is unable to take values coming from ϕ-functions. For example:

```
void f(int a, int b)
{
        int     x;
        int     y;

        x = 1;
        y = 1;

        do {
                x = x + a;
                y = y + a;
                a = a + b;
        } while (a < b);

        return x + y;
}
```

3. Only PRE can deal with partially redundant expressions such as in:

```
void f(int a, int b)
{
        int     c;

        c = a + b;
        if (c > 0)
                a = -b;
        return a + b;
}
```

4. Each of the **then**, **else** and **return** statements will belong to a vertex which is a child of the start vertex in the dominator tree. When the translation to SSA form enters one of these blocks it will not find `a * b` on the stack of expressions and instead push its own instance there. Then before leaving the vertex, it will be popped and not be available for redundancy elimination. The multiplication in the return statement will not be optimised away.

5. Global value numbering will find all three instances of `a * b` to be equivalent but since the multiplication in the return statement is not dominated by any other instance, it will not be optimised away.

6. For PRE on the other hand the multiplication will be optimised away since the Φ inserted in the vertex of the return statement has the expression available from both Φ-operands. In addition since it is *downsafe* and at least one Φ-operand has

the expression as a real occurence, the expression is available and the multiplication in the return statement is optimised away. It would be optimised away with `a * b` in only one of the **then** or **else** clauses or in both.