Optimization Methods

Deadline : 25'th Khordad

1. Convert the following program fragment into an equivalent control flow graph (CFG), taking care to represent the conditional branches and multiple paths of execution.

```
1. x = 5
2. y = 10
3. z = y * 2
4. if (z > x) then
5.     a = z * x
6.     if (a > 50) then
7.         b = a - 5
8.     else
9.         b = 2 * x
10.    endif
11. else
12.    a = z + y
13.    b = a + x
14. endif
15. c = a + b
```

   - Perform a constant propagation optimization on the CFG. Detail each step of the propagation process.
   - Present the optimized CFG and the equivalent code after the optimization. How has this transformation affected the code?
   - Now suppose that the initial values of x and y are not known at compile time, but are instead input by the user at runtime. How would this change affect the applicability of constant propagation in this scenario?
   - Assume that a subsequent optimization pass is capable of removing the conditional checks (i.e., if (z > x) and if (a > 50)). Could additional constant propagation be performed? If so, what would the resulting code look like?

2. Draw activation tree for the following code. Draw the top of the stack and the activation records when $fib(1)$ is called for the first time. You can consider only return values, parameters, control links, and space for local variables; you do not have to consider stored state or temporary or local values not shown in the code sketch.

```
1.int fib(int n){
2.     if(n == 1) return 1;
3.     if(n == 2) return 1;
4.     return fib(n - 1) + fib(n - 2);
5.}
6.int g(int n){
7.     if(n == 1) return 1;
```

```
8.     return fib(n) + g(n - 1);
9.}
10.
11.int main()
12.{
13.     int n = 3;
14.     print(g(n));
15.}
```

3. Consider the following part of a code where only b is referenced outside the code. Apply all local optimization methods and simplify the expressions.

```
b = a**2;
c = b * 2;
e = b * 2;
f = 12;
d = e *  f;
w = e * 12 + e * 4;
z = b - c;
w = w * z;
w = w + 1;
d = d*w;
b = a - d;
```

   - Assume a program with n character is given. Prove that by appropriate application of local methods, one can optimize the code in polynomial time in terms of n where a fixed set of variables are known to be referenced outside.

4. Let's assume that a is only live variable at the end off following code. According to this, answer the following question.

   (a) Draw control flow graph.

   (b) Draw register inference graph (RIG).

   (c) Analyze the RIG from the previous part and determine minimum number of needed register.

$$
\begin{gathered}
\textbf{L0}:\ a = 0 \\
b = 4 \\
c = 10 \\
d = 1 \\
f = 5 \\
h = 1 \\
\textbf{L1}:\ a = a + b \\
a = a + 7 \\
b = a + c \\
c = a * 2 \\
d = d + 1 \\
\text{if } f < b \text{ goto L4} \\
\textbf{L2}:\ d = d + 5 \\
a = a + d \\
\text{goto L5}
\end{gathered}
$$

L3: $c = c * f$
L4: $d = a + f$
if $a > c$ goto L2
L5: $b = b * b$
$a = a + b$
if $b == c$ goto L1

5. **Advanced Constant Propagation, Dead Code Elimination, and Register Allocation in Global Optimization**

   (a) Consider the following segment of pseudo-code that implements a more advanced algorithm:

```
1.  function square(n)
2.      return n * n
3.  end function
4.
5.  x = 5
6.  y = 10
7.  z = y * 2
8.  w = x - y
9.  if (z > x) then
10.     a = square(z * x)
11.     if (a > 50) then
12.         b = a - 5
13.     else
14.         b = 2 * square(x)
15.     endif
16. else
17.     a = square(z + y)
18.     b = a + square(x)
19. endif
20. c = a + b
21. d = x + c
22. e = z + w
23. for i = 1 to y
24.     x = x + i
25. end for
26. y = z * w
27. z = square(x) + y
28. print z
```

   (b) Convert the above program fragment into an equivalent Control Flow Graph (CFG), including the function definition and call.

   (c) Perform constant propagation and dead code elimination optimization on the CFG. Document each step of the propagation and elimination process.

   (d) Assume that the loop from line 23 to 25 cannot be unrolled. How does this affect your constant propagation and dead code elimination optimizations?

   (e) Discuss how you would approach register allocation for the resulting code. Justify your reasoning.

   (f) Suppose that the initial values of ‘x‘ and ‘y‘ are not known at compile time, but are instead input by the user at runtime. How would this change affect the applicability of constant propagation and dead code elimination in this scenario?

   (g) Now suppose that the function ‘square(n)‘ is replaced by a built-in function ‘sqrt(n)‘ that calculates the square root of ‘n‘ and cannot be optimized by constant propagation. How does this change your answer to (b), and what is the new optimized CFG and corresponding pseudo-code?