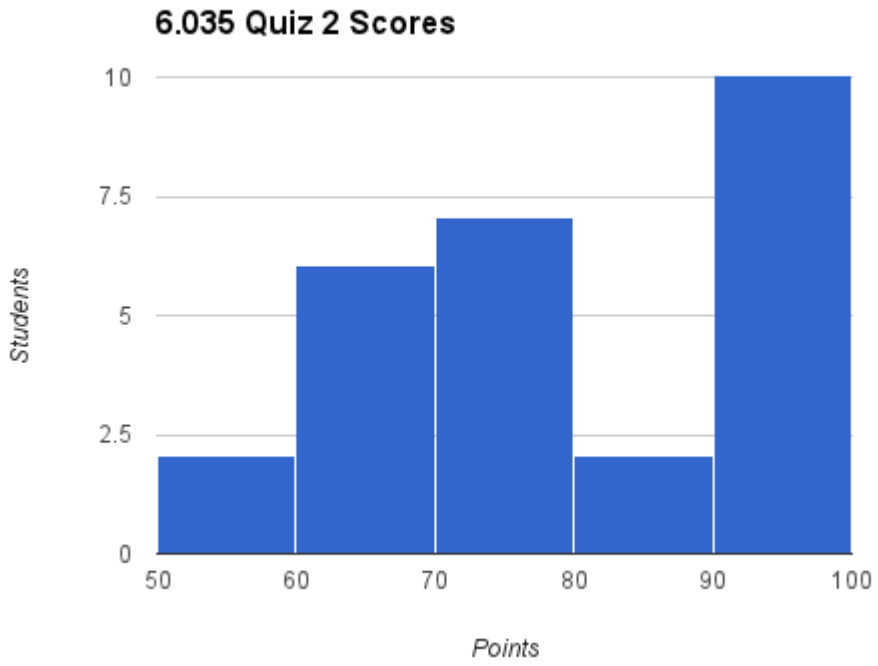
Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.035 Fall 2014

Test II Solutions



Average: 79.4 Median: 77 StDev: 13.5

I Assembly Code Generation

In this problem, you will generate relevant pieces of code for the following program:

```
void main()
{
  int x, y;
  // x = ... ;

  x = xysquare (x,y);  // Loc.1
  // ...
}
```

```
int xysquare(int x, int y)
{
  x = x + y;
  return x * x;
}
```

Write your assembly in AT&T syntax (src then dest). You should only use the instructions described in the table below. Assume Linux ABI calling convention. Remember that the first argument is passed in register `rdi` and the second in `rsi`.

Finally, remember the simple x86_64 addressing modes: `%rax` references register rax, `(%rax)` references memory at the address in rax, and `100(%rax)` references memory at 100 bytes + the address in rax. **Only one dereference** (e.g. `(%rax)`) **is allowed per instruction.**

x86_64 instructions to use:

`enter $n, $0`	Adjust stack for n bytes of local storage
`mov a, b`	Move value of a into destination b
`add a, b`	Add value of a to value in b; store result in b
`mul a, b`	Multiplies values of a and b; store result in b
`call sym`	Call function `sym`
`leave`	Undo effects of `enter`
`ret`	Return from function call

1. [6 points]: Write the assembly code *only* for the function `xysquare`:

Solution:

```
xysquare:
  enter $0, $0
  add %rsi, %rdi
  mov %rdi, %rax
  mul %rax, %rax
  leave
  ret
```

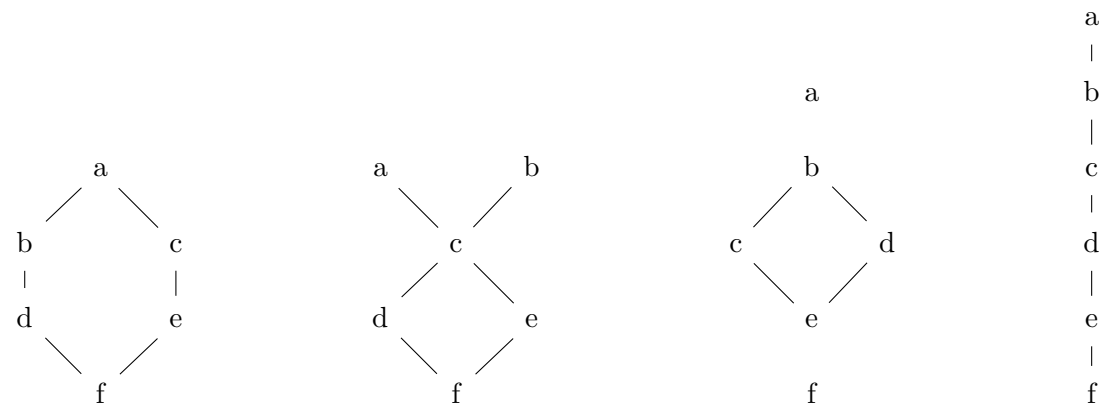
2. [4 points]: Assuming that in the function `main` the variable `x` is located at `%rbp - 8` and the variable `y` is located at `%rbp - 16`, write assembly code for the line in the main function that calls the function `xysquare` (Loc.1).

Solution:

```
  ...
  mov -8(%rbp), %rdi
  mov -16(%rbp), %rsi
  call xysquare
  mov %rax, -8(%rbp)
  ...
```

II Lattices

3. [4 points]: Let the set $P = \{a, b, c, d, e, f\}$. Circle those graphs below that are Hasse diagrams that graphically present a lattice $(P, \leq_i)$. For the graphs that you *do not* circle, give an explanation why.



Solution:

Diagrams 1 and 4 are graphic representations of lattices. The remaining diagrams do not have the bottom element (Diagram 3) or the top element (Diagrams 2 and 3).

For the remaining problems in this section, consider the set of natural numbers $\mathbb{N} = \{1, 2, \ldots\}$ and the partial order $b \leq a$, defined as:

$$b \leq a = \begin{cases} \text{True, if } b \text{ is a divisor of } a \\ \text{False, otherwise.} \end{cases}$$

4. [8 points]: Prove that the pair $(\mathbb{N}, \leq)$ is a lattice.

Solution:

Show that the partial order $\leq$ is reflexive (i.e., $a \leq a$ is true), 2) anti-symmetric (i.e., $a \leq b$ and $b \leq a$ imply $a = b$), and 3) transitive (i.e., $a \leq b$ and $b \leq c$ imply $a \leq c$). These three properties are true for the "divide by" operator $\leq$ defined above.

Alternatively, show that every two elements have a lower bound and an upper bound.

5. [3 points]: Is the lattice $(\mathbb{N}, \leq)$ complete? Explain why or why not.

Solution: It is not complete. The set $\mathbb{N}$ does not have an upper bound.

6. [3 points]: What is the bottom element of the lattice $(\mathbb{N}, \leq)$?

Solution: This is the number 1.

7. [3 points]: What numbers does number 6 cover?

Solution: It covers numbers 2 and 3.

8. [4 points]: What are the greatest lower bound (GLB) and the least upper bound (LUB) for the set $\{2, 3, 5\}$?

Solution: GLB is 1, LUB is 30.

III Reachability Analysis

As you know, reaching definitions is the standard analysis for the constant propagation transformation. In this problem, we will consider the following modified reaching definition dataflow analysis and constant propagation transformation:

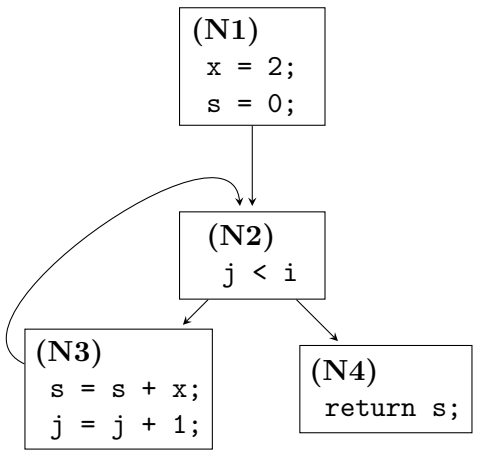
- The transfer functions are the same as for the standard reaching definition analysis.
- The set of the lattice elements is the same as in the standard bitvector reachability analysis.
- The join operator is intersection instead of union.

9. [5 points]: Given the same set of elements as in the standard bitvector reachability analysis, and the different joint operator, what should be the partial order for this analysis?

Solution: The partial order is the inverse set inclusion ($\supseteq$).

Example Program. We will first apply our modified reachability analysis on an example program. Figure below presents the CFG of the function `f(int i, int j)`. Each node is annotated with the index **N1** to **N4**.

10. [5 points]: Label all definitions in the CFG and compute the final IN and OUT sets for each node (use the space on the right of the CFG).



Solution:
Let indices in the bitvector be from left to right: (1) `x = 2`, (2) `s = 0`, (3) `s = s + x`, and (4) `j = j + 1`. Then:

IN[N1] = 0000
IN[N2] = 1000
IN[N3] = 1000
IN[N4] = 1000

OUT[N1] = 1100
OUT[N2] = 1000
OUT[N3] = 1011
OUT[N4] = 1000

Transformation. We define the modified constant propagation transformation as follows. For each variable use, check the number of reaching definitions. If there is exactly one reaching definition for that use, and that definition sets the variable to a constant, then replace the variable use with a constant.

11. [3 points]: Can we apply the transformation to the example program? If so, what statements change. If not, explain why.

Solution:

Yes, we can apply the transformation, because **IN**[N3] has the value 1 for the bit that represents the definition `x = 2`. Therefore, we can substitute `s = s + x` with `s = s + 2`.

12. [3 points]: Does the transformed program produce the correct output? (i.e., does the transformation preserve the semantics of this program)

Solution: This transformation doesn't change the semantics of the example program.

13. [5 points]: Is the analysis conservative for this transformation *for all programs*? In other words, does this combination of the analysis and the transformation preserve the semantics of the program? If so, explain why. If not, give a program for which the modified constant propagation transformation changes the semantics of the program.

Solution:

The analysis produces for each node a subset of the definitions that reach that node. It will propagate the definition to a specific node only if this definition is visible along all program paths that reach that node. Since this is the subset of the original reaching definitions, this analysis is conservative analysis for constant propagation.

14. [4 points]: If we change the join operator to union, is the analysis still conservative for the modified constant propagation transformation? If so, explain why. If not, give a program for which the transformation changes the semantics of the program.

Solution:

This analysis is also conservative, because the transformation still requires to have a single reaching definition for each variable. The union operator accounts for the definitions from all program paths; the transformation will proceed if there was only a single definition along all program paths.

IV Precise Sign Analysis

In this question we will build a more precise sign analysis. The purpose of this analysis is to enable the compiler to perform safety checks for calls to the log(x) function.

The analysis will analyze programs with one variable x. The language is defined as a sequence of the statements of this form:

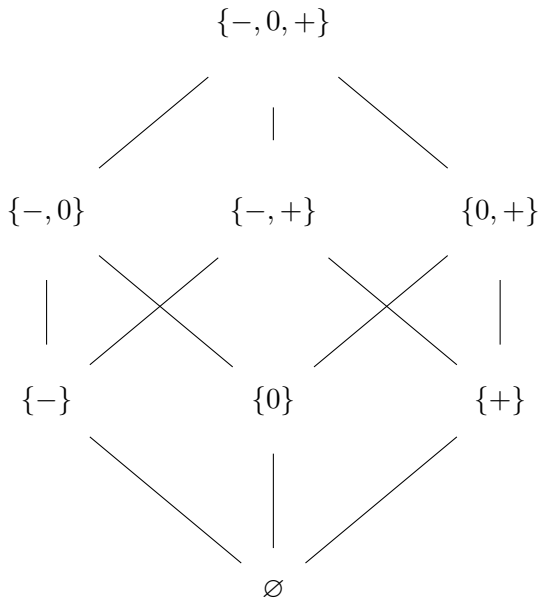
$$
\begin{aligned}
S ::= & \; x = c \\
| & \; x = x + c \\
| & \; \texttt{if } (x == c)\{S_1\} \texttt{ else } \{S_2\};
\end{aligned}
$$

In addition, the very last statement in the program is a call to the `log(x)` function. In the previous definition, each c is a (signed) integer constant.

To keep track of the sign of variable x, we will use the lattice $(\mathcal{P}(\{-, 0, +\}), \subseteq)$. For example, if x has a non-negative value, then the analysis will represent this as a set $\{0, +\}$. If x has a positive value, the analysis will represent this as a set $\{+\}$.

15. [4 points]: Draw a Hasse diagram for the lattice.

Solution:



Analysis. The safety check compiler pass looks at the sign analysis result at the function call to `log(x)` at the end of the program. For the `log(x)` function it checks if the analysis result indicates that x is positive. If so, the program passes the safety check, otherwise it fails.

We want our analysis to be conservative for this use. In other words, if the program passes the safety check, it must be the case that no execution of the program will ever pass a zero or negative value to `log(x)`. Your job is to implement a conservative sign analysis for this use.

16. [2 points]: Is your proposed analysis forward or backward?

Solution: Forward

17. [2 points]: What is the join operator for the analysis?

Solution: Union

Note: Below, the input v of the transfer function is either the set IN or the set OUT of the node, depending on whether your analysis is forward or backward.

18. [5 points]: What is the transfer function for a statement `x = c`?

Solution: $f_{x=c}(v) = \text{sign}(c)$ where $\text{sign}(c) = \begin{cases} +, & c > 0 \\ 0, & c = 0 \\ -, & c < 0 \end{cases}$

19. [5 points]: What is the transfer function for a statement `x = x + c`?

Solution: $f_{x=x+c}(v) = \bigcup_{y \in v} \text{sign2}(y, c)$, where the function $\text{sign2}(y, c)$ is defined as:

e/c	$-$	0	$+$
$-$	$\{-\}$	$\{-\}$	$\{-, 0, +\}$
0	$\{-\}$	$\{0\}$	$\{+\}$
$+$	$\{-, 0, +\}$	$\{+\}$	$\{+\}$

20. [8 points]: Are these two kinds of transfer functions distributive? Prove or provide a counterexample.

Solution:

The function $f_{x=c}(v)$: trivially distributive since it does not depend on the input v.

The function $f_{x=x+c}(v)$:

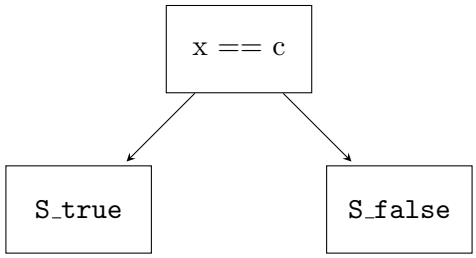
$$\begin{aligned} f(v_1 \cup v_2) = & \bigcup_{y \in v_1 \cup v_2} \text{sign2}(y, c) = \\ & = \left(\bigcup_{y \in v_1} \text{sign2}(y, c) \right) \cup \left(\bigcup_{y \in v_2} \text{sign2}(y, c) \right) = \\ & = f(v_1) \cup f(v_2) \end{aligned}$$

21. [4 points]: Does your analysis always terminate? If so, explain why. If not, give an example program where it does not terminate.

Solution:

The analysis terminates because 1) the lattice is finite and 2) the transfer functions are monotonic. The monotonicity of transfer functions is implied by the functions' distributivity (which is proved in the previous problem).

We can take into consideration the path information to make the analysis more precise when analyzing conditionals (since x is compared with a constant c). Conditional statements get compiled down into nodes with a true and false branch as follows:



22. [5 points]: Give the transfer function $f_{x==c,true}(v)$ for the true branch and $f_{x==c,false}(v)$ for the false branch when analyzing the top node $x == c$:

Solution:

$f_{x==c,true}(v) = \text{sign}(c)$

$$f_{x==c,false}(v) = \begin{cases} \{+,-\}, & \text{if } c = 0 \\ v, & \text{otherwise} \end{cases}$$

23. [5 points]: Give a program that always passes a positive value to log(x) but fails the safety check analysis.

Solution:

```
x = 3;
x = -2;
log(x);
```