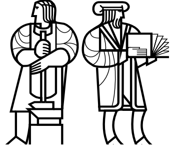


Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.s081 Spring 2018

Test II Solutions

I VMs & Syntax-Directed Translation

In this problem, we will be exploring adding a new feature to the MITScript bytecode specification, namely switch statements. We define the syntax of a switch statement with the following grammar:

$S \rightarrow \texttt{switch}\ (E)\ \{C\}$

$C \rightarrow (\texttt{case}\ V\ \{S\})^*$

$V \rightarrow n$

In evaluating a switch statement, the expression following the `switch` keyword must first be evaluated, sequentially compared to each value that follows a `case` keyword in the order that they appear, before finally evaluating the first block that matches. At the end of each case block, control flow automatically breaks out of the switch statement. An example program utilizing this language construct is written below:

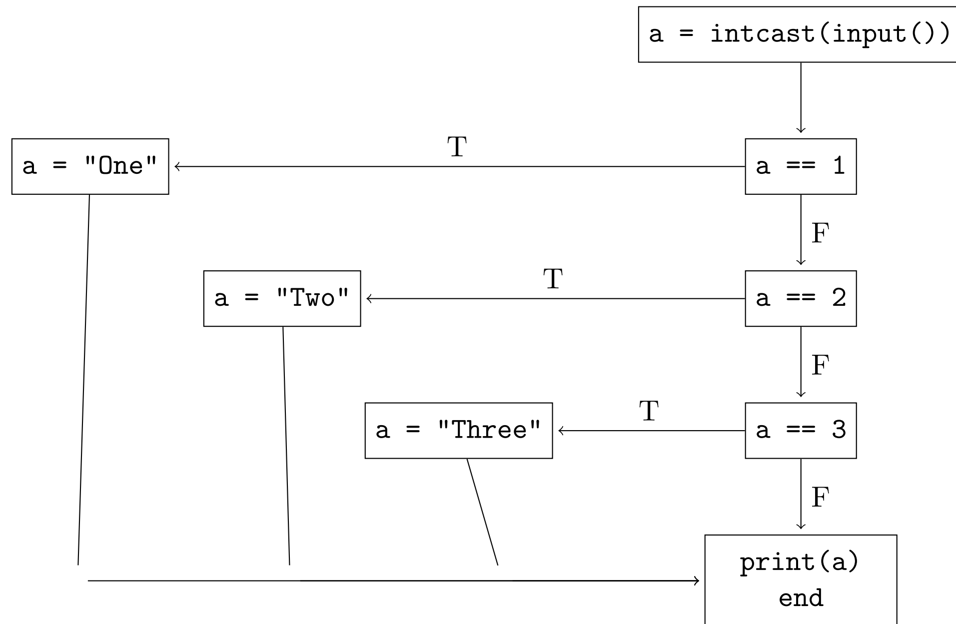
```
a = intcast(input());
switch (a) {
    case 1 {
        a = "One";
    }
    case 2 {
        a = "Two";
    }
    case 3 {
        a = "Three";
    }
}
print(a);
```

As one would expect, this program prints `One` for an input of 1, `Two` for an input of 2, `Three` for an input of 3, and the input itself for all other inputs.

1. [4 points]:

Draw the CFG for the example program on the previous page. Ignore failure cases for `intcast` and `input`, and label edges with T or F when control flow depends on the value of a test at the end of the source basic block.

Solution:



2. [**6 points**]: Fill in the instructions necessary to execute the switch statement from the example in the bytecode below. We have included a table of relevant bytecode instructions.

call m	Call a closure with number of arguments m
eq	Computes an equality between two values
goto i	Go to a new instruction offset
if i	Go to a new instruction offset if the operand evaluates to true
load_const i	Load value of constant at index i of constants
load_global i	Load value of global variable at index i of names
not	Computes the logical negation of a boolean operand
store_global i	Store value into global variable at index i of names

```
constants = [1, 2, 3, "One", "Two", "Three"],
names = [a, print, input, intcast],
instructions = [
    ...
    store_global   0     //stored value of a from intcast(input())
    load_global    0
    load_const     0
    eq
    not
    if             4
    load_const     3
    store_global   0
    goto           16
    load_global    0
    load_const     1
    eq
    not
    if             4
    load_const     4
    store_global   0
    goto           8
    load_global    0
    load_const     2
    eq
    not
    if             3
    load_const     5
    store_global   0
    load_global    1    //print(a)
    load_global    0
    call           1
    pop
    ...
]
```

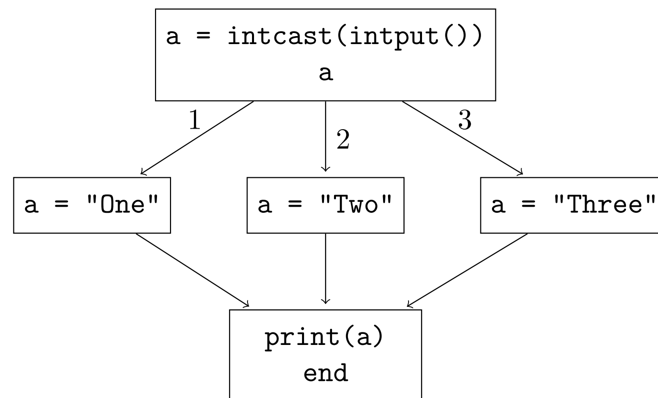
The set of bytecode instructions made available by the virtual machine can effect application performance. For example, with the current instruction set, a switch statement with n case blocks could require up to n comparison checks before executing the appropriate program block. However, where we could optimize this bytecode is when the values in our case statements are part of contiguous set of integers.

Specifically, our current intermediate representation only supports conditional branches that check the result (True or False) of a boolean operation. However, we can instead enable our intermediate representation to branch based on the value of an integer at the top of the stack. This change enables a basic block in a CFG to have more than two children. These edges could can be labelled with integers as well as boolean values now.

3. [4 points]:

Draw the CFG for the example program where there are at most 2 jumps between the basic block for the beginning of the switch and the end basic block for the switch. Be sure to clearly label edges. Again, ignore failure cases for `intcast` and `input`. Furthermore, for the purposes of this problem you can assume that `input()` will return a value from the set {1,2,3}.

Solution:



Finally, we want to edit our bytecode to reflect this new CFG. We introduce the `jmp_ind` instruction, which looks at the the last value pushed onto the stack, and offsets the instruction pointer by that value (1 is the next instruction, 0 would cause an infinite loop).

4. [8 points]: Fill in the remaining instructions necessary to execute the switch statement so that the number of dynamically taken branches for a path in your solution should not change if the programmer were to add another case to the switch statement: **Solution:**

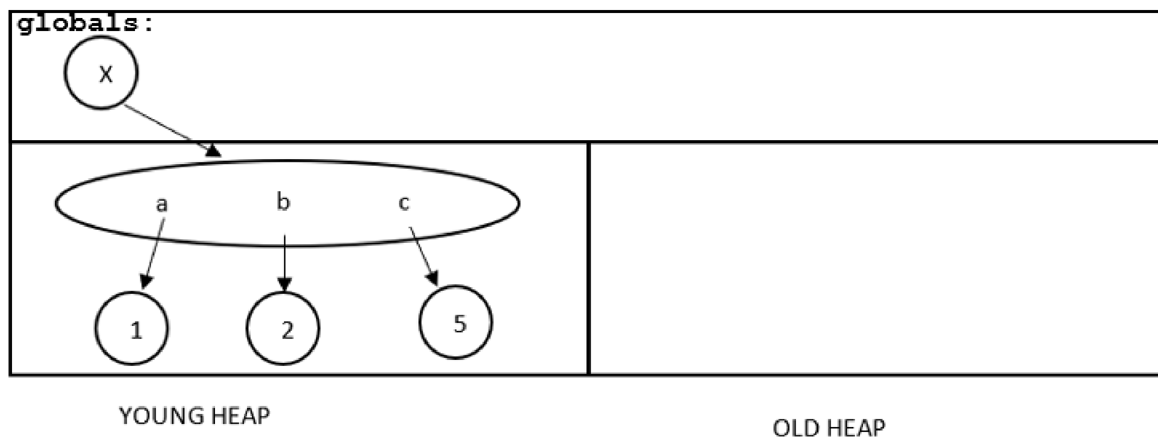
```
constants = [1, 2, 3, "One", "Two!", "Three!"],
names = [a, print, input, intcast],
instructions = [
    ...
    load_global    0     //push value of a onto stack
    jmp_ind
    goto           3
    goto           5
    goto           7
    load_const     3
    store_global   0
    goto           7
    load_const     4
    store_global   0
    goto           4
    load_const     5
    store_global   0
    goto           1
    load_global    1    //print(a)
    load_global    0
    call           1
    pop
    ...
]
```

II Garbage Collection

In this problem, you will perform Generational Garbage Collection for the following program. Assume that the generational garbage collector is run on the young heap after every line. Furthermore, assume objects are promoted to the old heap during the $\mathbf{4}^{th}$ iteration of garbage collection.

```
1: x = {a: 1; b: 2; c: 5;};
2: y = x.b;
3: z = 6;
4: x.a = 3;
5: x.b = y;
```

After the first line is executed, the state of the heap is depicted below:

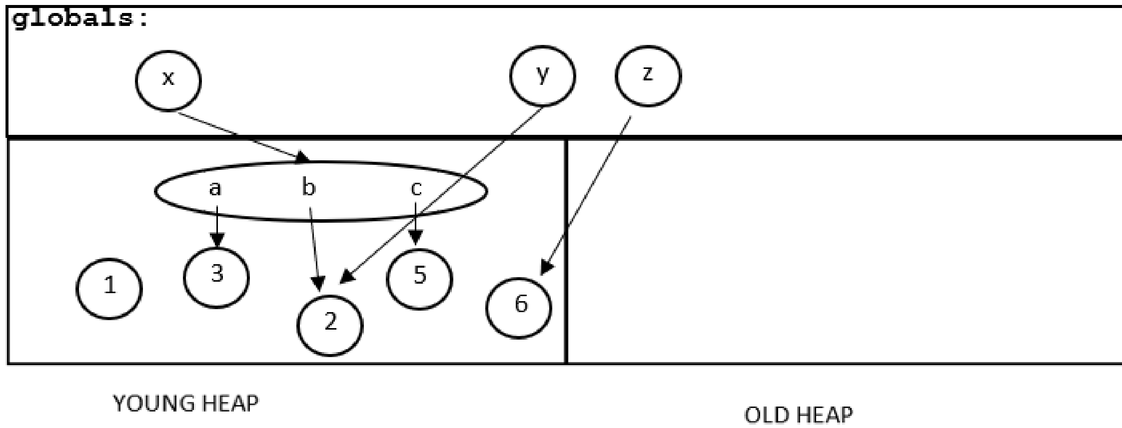


Note that new objects are allocated with new constants and records involving new constants. An assignment of the form $v_1 = v_2$ **does not** allocate a new object, simply points v_1 to the object pointed to by v_2. Also note for the purposes of this problem we are not concerned with how the stack is allocated.

Suppose we paused the program right after line 4 is executed.

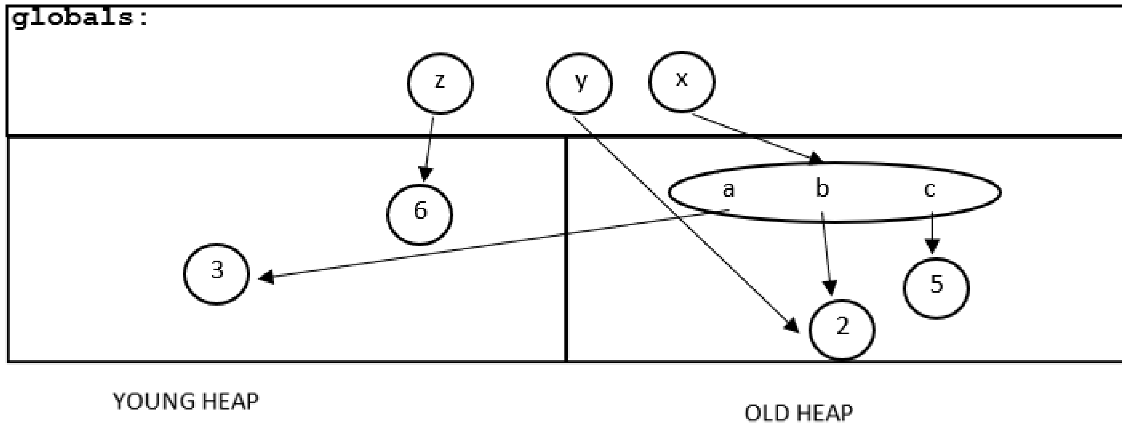
5. [**10 points**]: What is the state of the heap **before** garbage collection takes place?

Solution:



6. [**10 points**]: What is the state of the heap **after** garbage collection takes place?

Solution:



7. [**5 points**]: If at the end of this code snippet we ran the garbage collector on the old heap as well, what objects would remain live on the young/old heaps?

Solution:

The objects representing the record, and the integers 2, 5, 6, and 3.

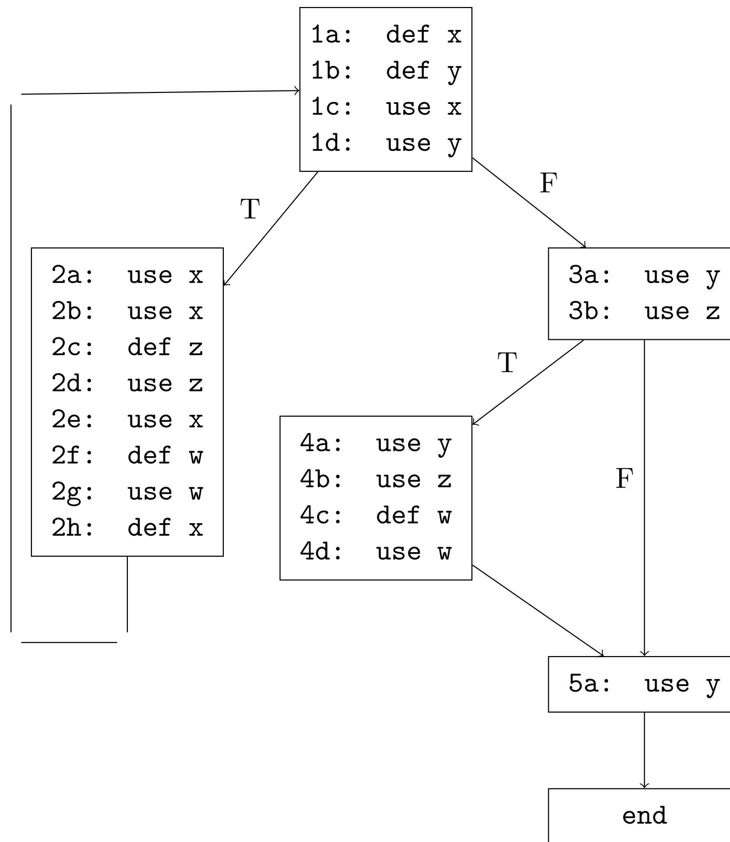
III Register Allocation

In this problem, you will perform register allocation for the following code. Each instruction is labeled with a number. Assume that you do not perform any other optimizations, and none of the variables is subsequently used.

```
 1: x = intcast(input());
 2: y = intcast(input());
 3: while (x < y) {
 4:     z = x + x;
 5:     w = z * x;
 6:     x = 3 + w;
    }
 7: if (y > z) {
 8:    w = y * z;
 9:    print (w);
    }
10: print (y);
```

8. [4 points]:

On the next page, translate this program into a CFG of def and use statements. Be sure to include labels for each statement (block number followed by letter).


1a: def x
1b: def y
1c: use x
1d: use y
T
F
2a: use x
2b: use x
2c: def z
2d: use z
2e: use x
2f: def w
2g: use w
2h: def x
3a: use y
3b: use z
T
F
4a: use y
4b: use z
4c: def w
4d: use w
5a: use y
end

9. [5 points]: Write the set of def-use chains for each variable in the program. Write each def-use chain as a number pair (d, u) where d is the label of an instruction that defines the variable and u is the label of an instruction that uses that definition (include a pair for each use for any given definition). Use labels from your CFG in problem 8.

Solution:

`x`: (1a, 1c) (1a, 2a) (1a, 2b) (1a, 2e) (2h, 2a) (2h, 2b) (2h, 2e)

`y`: (1b, 1d) (1b, 3a) (1b, 4a) (1b, 5a)

`z`: (2c, 2d) (2c, 3b) (2c, 4b)

`w`: (2f, 2h) (4c, 4d)

10. [5 points]: Write the set of webs in the program. Write each web as the set of instructions that belong to the web, in terms of the labels from your CFG in problem 8. We have given you names `w1-w7` for the webs, use only as many names as you need.

Solution:

`w1`: {1a, 1b, 1c, 1d, 2a, 2b, 2c, 2d, 2e}

`w2`: {2h, 2a, 2b, 2c, 2d, 2e}

`w3`: {1b, 1c, 1d, 2a, 2b, 2c, 2d, 2e, 2f, 2g, 2h, 4a, 4b, 4c, 4d, 5a}

`w4`: {2c, 2d, 2e, 2f, 2g, 2h, 3a, 3b, 4a, 4b}

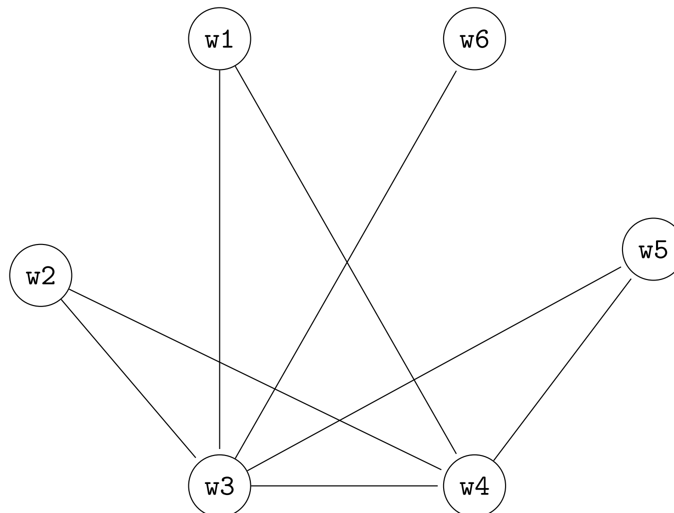
`w5`: {2f, 2g}

`w6`: {4c, 4d}

`w7`:

11. [5 points]: Draw the interference graph for the webs. Each node in the interference graph should represent one web. There should be an edge between two nodes if the two webs interfere. Label each node with the name (`w1-w7`) of the corresponding web.

Solution:



12. [4 points]: Suppose that the architecture we are targeting for compilation has three general purpose registers, `r1`, `r2`, and `r3`.

A. [1 points]: Can we place all the webs in this code in registers (ignore any constraint due to calling convention)?

Solution: Yes

B. [3 points]: If yes, specify a mapping of webs to registers. If no, specify which web you would spill to maximize performance.

Solution:

Assign `w1`, `w2`, `w5`, and `w6` to `r1`.

Assign `w3` to `r2`.

Assign `w4` to `r3`.

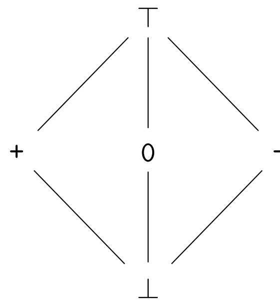
IV Dataflow Analysis

Your task in this problem is to design a dataflow analysis that will determine whether each variable v_1, ..., v_k is positive, negative, or zero at each point in the program, otherwise known as a Sign Analysis. The program itself will be represented as a control flow graph with two kinds of assignment statements:

- $v = c$: sets a variable v to a constant c.
- $v_1 = v_2 + v_3$: sets a variable v_1 to the sum of variables v_2 and v_3.
- $v_1 = v_2$ - v_3: sets a variable v_1 to the difference of variables v_2 and v_3.

13. [8 points]: Design a reasonably precise lattice for this dataflow analysis problem. You should specify the set S of lattice elements and the least upper bound operator $\vee$ over the set of lattice elements. You should define the set of abstract values for a single variable.

Solution:



where

- $\vee$ of $\top$ and any element is $\top$
- $\vee$ of any element and $\bot$ is that element
- $\vee$ of an element with itself is that element
- $\vee$ else is $\top$

14. [4 points]: Consider the following program.

```
a = 0;
b = 0;
c = 0;
if (...) {
  a = -5;
  b = 2;
} else {
  a = -3;
  b = 4;
}
d = a + b;
e = a - c;
f = c - a;
```

What is the analysis result at the end of this program? Specify the mapping of each variable to its abstract value.

Solution: [a $\rightarrow$ -, b $\rightarrow$ +, c $\rightarrow$ 0, d $\rightarrow$ $\top$, e $\rightarrow$ -, f $\rightarrow$ +]

15. [4 points]: What is the most precise transfer function $f_n()$ for a statement n of the form `x = c`? Your transfer function should have a return value that represents the abstract value of `x`.

$f_n() = \texttt{sign(c)}$

16. [6 points]: What is the most precise transfer function $f_n(l_x, l_y)$ for a statement n of the form `z = x + y`? Here l_x and l_y are the abstract values that represent `x` and `y` respectively, and your transfer function should have a return value that represents the abstract value of `z`.

$$f_n(l_x, l_y) = \begin{cases} l_y & l_x = 0 \\ l_x & l_y = 0 \\ l_x \vee l_y & \texttt{else} \end{cases}$$

17. [8 points]: What is the most precise transfer function $f_n(e)$ for a statement n of the form `z = x - y`? Here l_x and l_y are the abstract values that represent `x` and `y` respectively, and your transfer function should have a return value that represents the abstract value of `z`.

$$f_n(l_x, l_y) = \begin{cases} l_x & l_y = 0 \\ l_x & l_y = \bot \\ l_y & l_x = \bot \\ \texttt{+} & l_x = 0 \texttt{ \&\& } l_y = \texttt{-} \\ \texttt{-} & l_x = 0 \texttt{ \&\& } l_y = \texttt{+} \\ \texttt{+} & l_x = + \texttt{ \&\& } l_y = \texttt{-} \\ \texttt{-} & l_x = - \texttt{ \&\& } l_y = \texttt{+} \\ \top & \texttt{else} \end{cases}$$