CS143 Final
Spring 2025

- Please read all instructions (including these) carefully.
- There are 6 questions on the exam, some with multiple parts. You have 180 minutes to work on the exam.
- The exam is open note. You may use laptops, phones and e-readers to read electronic notes, but not for computation or access to the internet for any reason other than to access the class webpage.
- Please write your answers in the space provided on the exam, and clearly mark your solutions. Do not write on the back of exam pages or other pages.
- Solutions will be graded on correctness and clarity. Each problem has a relatively simple and straightforward solution. You may get as few as 0 points for a question if your solution is far more complicated than necessary. Partial solutions will be graded for partial credit.

SUNET ID: _____

NAME: _____

In accordance with both the letter and spirit of the Honor Code, I have neither given nor received assistance on this examination.

SIGNATURE: _____

Problem	Max points	Points
1	20	
2	20	
3	15	
4	15	
5	15	
6	15	
TOTAL	100	

1. Types and Functions

In this problem, we will investigate subtyping rules for first-class functions. Many modern languages combine functional and object-oriented programming styles, making it necessary to have a subtyping principle for function values. We will write $A \to B$ for the type of functions that map an argument of type A to a value of type B (we will only look at functions with one argument).

Recall that the Cool conformance/subtyping relation $A \leq B$ formalizes the intuitive notion that a value of type A can be used anywhere a value of type B is expected. The subtyping facts and rules you give for this problem must obey this principle.

(a) Suppose we have two classes Student and Person satisfying Student $\leq$ Person, then we can form four function types:

$$\begin{aligned}
&\text{Student} \to \text{Student}\\
&\text{Student} \to \text{Person}\\
&\text{Person} \to \text{Student}\\
&\text{Person} \to \text{Person}
\end{aligned}$$

Write down the subtype (<=) relationships that hold between these function types. You do not need to write down the four trivial ones where the two sides are equal, such as (Student $\to$ Student) $\leq$ (Student $\to$ Student).

Answer:

To be equally usable, a function must return a value that is at least as specific as expected, and must accept values at least as general as expected.

$$\begin{aligned}
&(\text{Student} \to \text{Student}) \leq (\text{Student} \to \text{Person})\\
&(\text{Person} \to \text{Student}) \leq (\text{Student} \to \text{Student})\\
&(\text{Person} \to \text{Student}) \leq (\text{Student} \to \text{Person})\\
&(\text{Person} \to \text{Student}) \leq (\text{Person} \to \text{Person})\\
&(\text{Person} \to \text{Person}) \leq (\text{Student} \to \text{Person})
\end{aligned}$$

(b) Suppose we have types A, A', B, B' and the function types $A \to B$ and $A' \to B'$. Write down what we need to know about the relationships between A and A', and B and B' in order to know that $(A \to B) \leq (A' \to B')$.

Answer:

The following relationships must hold:

$$A' \leq A$$
$$B \leq B'$$

To belong to a subtype, a function needs to accept at least as wide of a range of arguments while producing at least as specific results.

2. Code Generation

In this question, we are going to add a new expression to the small expression language from Lecture 12, starting from assembly code.

(a) Describe an expression that would result in the following assembly code, where angle brackets are placeholders for the assembly code of sub-expressions. The `$zero` register holds the value 0.

```
start:
    <assembly code for expr1>
    sw $a0 0($sp)
    addiu $sp $sp -4
    <assembly code for expr2>
    move $t1 $a0
    lw $a0 4($sp)
    addiu $sp $sp 4
    bne $t1 $zero start
```

The `bne` instruction branches to the given label if the two registers are not equal.

Answer:

The expression is a loop that is evaluated at least once and that terminates if `expr2` is zero. We will call this loop construct a do-while loop, although other names are also possible.

Possible CFG for the loop construct: `e := do` e_1 `while` e_2

(b) Write a type rule for the expression, in the style of the type rules in the Cool manual.

Answer:

$$\frac{O, M, C \vdash e_1 : \text{int} \quad O, M, C \vdash e_2 : \text{int}}{O, M, C \vdash \text{do } e_1 \text{ while } e_2 : \text{int}}$$

Other forms are also acceptable, including evaluating E_2 to a bool and evaluating E_1 and the do-while expression to T.

(c) Write the operational semantics rule(s) to describe the runtime behavior of your new expression, in the style of the operational semantic rules in the Cool manual.

Answer:

Operational semanatics for the false case (the integer 0):

$$\frac{S, E \vdash e_1 : v, S_1 \quad S_1, E \vdash e_2 : 0, S_2}{S, E \vdash \text{do } e_1 \text{ while } e_2 : v, S_2}$$

Operational semanatics for the true case (any integer except 0:

$$\frac{S, E \vdash e_1 : v_1, S_1 \quad S_1, E \vdash e_2 : v_2, S_2 \quad S_2, E \vdash \text{do } e_1 \text{ while } e_2 : v, S_3}{S, E \vdash \text{do } e_1 \text{ while } e_2 : v, S_3}$$

Note: operational semantic rules match on the more specific case, so there is no need to exclude 0 from the values returned by e_2 in the true case.

3. Optimization

(a) The algorithm you learned in lecture for local common subexpression elimination requires the IR to be in single static assignment (SSA) form. Specifically, if two assignments have the same right-hand-side, and the right-hand-side is pure and free of side effects, you can eliminate the later assignment and replace it with the value computed in the earlier assignment. Why is it incorrect to do so if the IR is not in SSA form? Provide an example where it does not work and a brief explanation. Your counter-example should contain no more than five assignments.

Answer:

Consider the following assignment:

```
x = a + b
x = 3
y = a + b
```

Although the first and last statements have the same sub-expression, it is not correct to replace later uses of `y` with uses of `x`. The reason is that the value of `x` is overwritten in the second statement. With SSA form, each variable can only be written to by one statement so such overwriting cannot happen in straightline code.

(b) Give a minimal program for which performing constant folding, copy propagation, and then dead code elimination is better than performing copy propagation, constant folding, and then dead code elimination.

Answer:

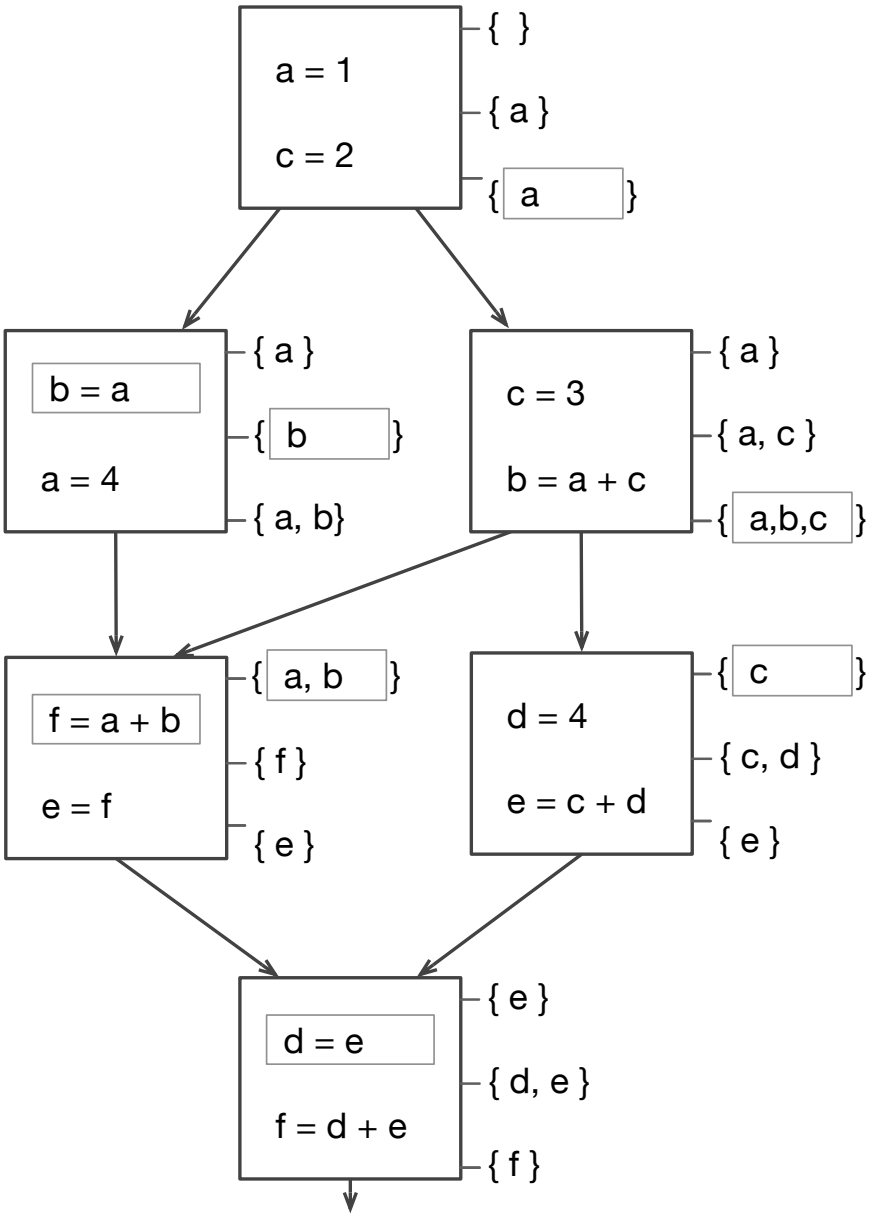
Assume `x` is live at the end of the following statements:

```
y = 1 + 1
x = y + z
```

4. Register Allocation

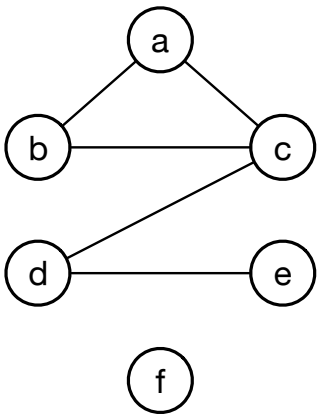
(a) Below is a control-flow graph annotated with liveness information, but someone has spilled coffee on it! Fill in the missing statements and liveness sets. Valid entries for statements are of the form `x = y` or `x = y + z`, where **x,y,z** $\in$ {`a,b,c,d,e,f`}. Make sure there are no dead code.

Answer:



(b) Build the register inference graph for the above control-flow graph.

Answer:



(c) Provide a valid register allocation by listing, for each register, the variables that share it. Use the minimal number of registers.

Answer:

```
r1: a, d
r2: b, e
r3: c, f
```

Other register allocations are possible.

5. Garbage Collection

In the following sub-questions we will explore the relative advantages of the three garbage collection schemes covered in class: mark-and-sweep, stop-and-copy, and reference counting.

(a) Describe a situation where stop-and-copy outperforms mark-and-sweep.

Answer:

A program that allocates a lot of short-lived objects on the heap. This program has a lot of garbage to collect, so stop-and-copy performs better because its garbage collection cost is proportional to the live objects and does not have to traverse the garbage.

(b) Describe a situation where mark-and-sweep outperforms stop-and-copy.

Answer:

A program with a lot of live data on the heap. Since stop-and-copy only has half the space available for live data, it will need to collect garbage more often than mark-and-sweep.

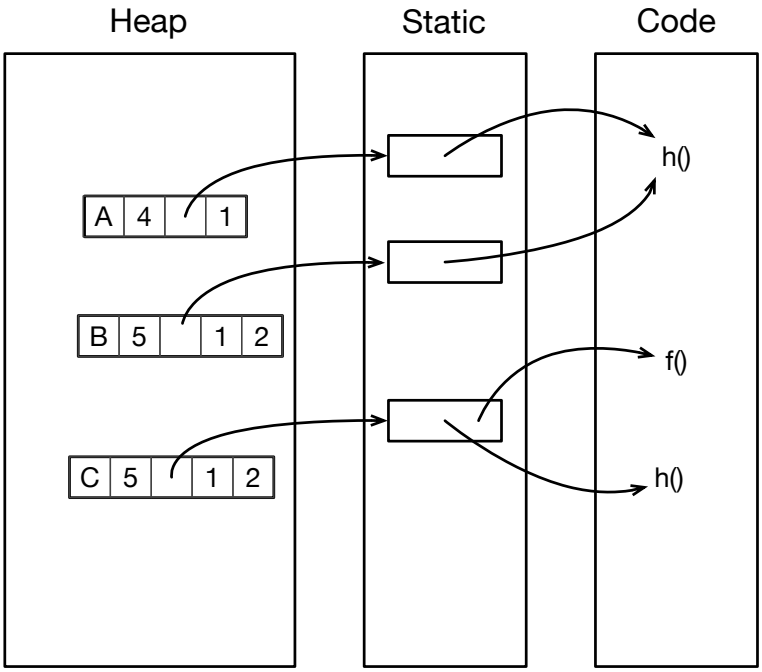
(c) Describe a situation where reference counting outperforms mark-and-sweep.

Answer:

A program where references are seldomly copied, where most objects have long lifetimes, and where garbage collection is invoked several times. Mark-and-sweep must traverse the whole heap during garbage collection, while reference counting only traverses the objects reachable from a dereferenced reference.

6. Runtime Organization

Provide three Cool classes with the minimum total number of features that can result in the following runtime layout:



Answer:

```
class A {
    x: Int <- 1;
    h() : Int {1};
};

class B inherits A {
    y: Int <- 2;
};

class C inherits B {
    f() : Int {1};
    h() : Int {2};
};
```

(blank page for extended answers)