EC3204: Programming Languages and Compilers (Fall 2023)
Final Exam

100 points in total, 25% of the total score

Date and Time: 12/13, 09:05 – 10:30
Place: Natural Science Building 203 (자연과학동 203호)

Instructor: Sunbeom So

Student ID: _____

Name: _____

* Leave the score table blank

	Min Scores	Max Scores	Your Scores
Problem 1	0	10	
Problem 2	0	10	
Problem 3	0	20	
Problem 4	0	20	
Problem 5	0	10	
Problem 6	-30	30	
Total	0	100	

Problem 1. (10pt) Operational Semantics

Consider the simple imperative language:

$$\begin{array}{rcl} a & \to & n \mid x \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2 \\ b & \to & \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \\ S & \to & x := a \mid \texttt{skip} \mid S_1; S_2 \mid \texttt{if } b\ S_1\ S_2 \mid \texttt{while } b\ S \mid \texttt{do } S \texttt{ while } b \end{array}$$

Assume that we have semantic functions for arithmetic and boolean expressions respectively, which we covered during the class: $\mathcal{A}[\![\ a\ ]\!] : \textbf{State} \to \mathbb{Z}$ and $\mathcal{B}[\![\ b\ ]\!] : \textbf{State} \to \textbf{T}$.

1. (5pt) Complete the big-step operational semantics for the statements (S). No partial points will be given. Add inference rules if necessary.

$$\frac{}{\langle x := a, s\rangle \to s[x \mapsto \mathcal{A}[\![\ a\ ]\!](s)]} \qquad \frac{}{\langle \texttt{skip}, s\rangle \to s} \qquad \frac{\langle S_1, s\rangle \to s' \qquad \langle S_2, s'\rangle \to s''}{\langle S_1; S_2, s\rangle \to s''}$$

$$\frac{\langle S_1, s\rangle \to s'}{\langle \texttt{if } b\ S_1\ S_2, s\rangle \to s'} \text{ if } \mathcal{B}[\![\ b\ ]\!](s) = \texttt{true} \qquad \frac{\langle S_2, s\rangle \to s'}{\langle \texttt{if } b\ S_1\ S_2, s\rangle \to s'} \text{ if } \mathcal{B}[\![\ b\ ]\!](s) = \texttt{false}$$

$$\frac{\langle S, s\rangle \to s' \qquad \langle \texttt{while } b\ S, s'\rangle \to s''}{\langle \texttt{while } b\ S, s\rangle \to s''} \text{ if } \mathcal{B}[\![\ b\ ]\!](s) = \texttt{true} \qquad \frac{}{\langle \texttt{while } b\ S, s\rangle \to s} \text{ if } \mathcal{B}[\![\ b\ ]\!](s) = \texttt{false}$$

$$\frac{\boxed{}}{\langle \texttt{do } S \texttt{ while } b, s\rangle \to \boxed{}}$$

2. (5pt) Complete the small-step operational semantics for the commands (c). No partial points will be given. Add inference rules if necessary.

$$\frac{}{\langle x := a, s\rangle \Rightarrow s[x \mapsto \mathcal{A}[\![\ a\ ]\!](s)]} \qquad \frac{}{\langle \texttt{skip}, s\rangle \Rightarrow s}$$

$$\frac{\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle}{\langle S_1; S_2, s\rangle \Rightarrow \langle S_1'; S_2, s'\rangle} \qquad \frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1; S_2, s\rangle \Rightarrow \langle S_2, s'\rangle}$$

$$\frac{}{\langle \texttt{if } b\ S_1\ S_2, s\rangle \Rightarrow \langle S_1, s\rangle} \text{ if } \mathcal{B}[\![\ b\ ]\!](s) = \texttt{true} \qquad \frac{}{\langle \texttt{if } b\ S_1\ S_2, s\rangle \Rightarrow \langle S_2, s\rangle} \text{ if } \mathcal{B}[\![\ b\ ]\!](s) = \texttt{false}$$

$$\frac{}{\langle \texttt{while } b\ S, s\rangle \Rightarrow \langle \texttt{if } b\ \texttt{(}S\texttt{; while } b\ S\texttt{)}\ \texttt{skip}, s\rangle}$$

$$\frac{}{\langle \texttt{do } S \texttt{ while } b, s\rangle \Rightarrow \boxed{}}$$

Problem 2. (10pt) Denotational Semantics

Consider the following denotational semantics.

$$
\begin{array}{rcl}
\mathcal{C}[\![\ c\ ]\!] & : & \textbf{State} \hookrightarrow \textbf{State} \\
\mathcal{C}[\![\ x := a\ ]\!](s) & = & s[x \mapsto \mathcal{A}[\![\ a\ ]\!](s)] \\
\mathcal{C}[\![\ \texttt{skip}\ ]\!] & = & \mathsf{id} \\
\mathcal{C}[\![\ c_1; c_2\ ]\!] & = & \mathcal{C}[\![\ c_2\ ]\!] \circ \mathcal{C}[\![\ c_1\ ]\!] \\
\mathcal{C}[\![\ \texttt{if}\ b\ c_1\ c_2\ ]\!] & = & \mathsf{cond}(\mathcal{B}[\![\ b\ ]\!], \mathcal{C}[\![\ c_1\ ]\!], \mathcal{C}[\![\ c_2\ ]\!]) \\
\mathcal{C}[\![\ \texttt{while}\ b\ c\ ]\!] & = & \mathit{fix} F
\end{array}
$$

where

$$
\mathsf{cond}(f, g, h) = \lambda s. \begin{cases} g(s) & \cdots \text{ if } f(s) = \mathit{true} \\ h(s) & \cdots \text{ if } f(s) = \mathit{false} \end{cases}
$$

1. (5pt) Find the function F for the while-loop below. No partial points will be given.

$$\texttt{while}\ (x < 10)\ x := x + 1$$

2. (5pt) Find $\mathit{fix} F$ of the loop above, i.e., the least solution g that satisfies $F(g) = g$. No partial points will be given.

Problem 3. (20pt) Semantic Analysis

Consider the abstract integer-value domain

$$\widehat{\mathbb{Z}} = \{\texttt{Non-Zero}, \texttt{Zero}, \top\}$$

where `Non-Zero`, `Zero`, and $\top$ represent non-zero integers, zero, and all integers, respectively.

1. (6pt) Define the partial order ($\sqsubseteq$) between abstract values.

$$a \sqsubseteq b \iff \boxed{}$$

2. (9pt) Complete the definition of an abstract semantic function, which over-approximates $\mathcal{A}[\![\ a\ ]\!] : \textbf{State} \to \mathbb{Z}$ (the semantics of arithmetic expressions) that we discussed in class, in terms of the abstract values from $\widehat{\mathbb{Z}}$. The definitions must output the most precise results.

$$\begin{aligned}
\widehat{\mathcal{A}}[\![\ a\ ]\!] &: \widehat{\mathsf{State}} \to \widehat{\mathbb{Z}} \\
\widehat{\mathcal{A}}[\![\ n\ ]\!](\hat{s}) &= \textbf{if } n = 0 \textbf{ then } \texttt{Zero} \textbf{ else } \texttt{Non-Zero} \\
\widehat{\mathcal{A}}[\![\ x\ ]\!](\hat{s}) &= \hat{s}(x)
\end{aligned}$$

$\widehat{\mathcal{A}}[\![\ a_1 + a_2\ ]\!](\hat{s}) =$

`Non-Zero`	
`Zero`	
$\top$	

$\widehat{\mathcal{A}}[\![\ a_1 \star a_2\ ]\!](\hat{s}) =$

`Non-Zero`	
`Zero`	
$\top$	

$\widehat{\mathcal{A}}[\![\ a_1 - a_2\ ]\!](\hat{s}) =$

`Non-Zero`	
`Zero`	
$\top$	

3. (5pt) Explain how we can detect division-by-zero errors using the analysis defined above.

Problem 4. (20pt) Optimization

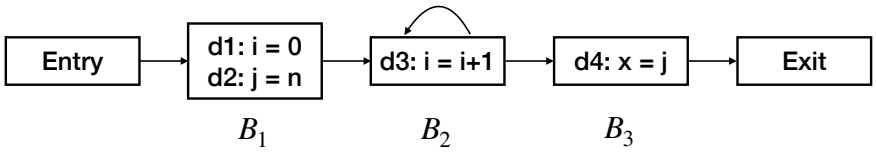
Recall the reaching definition analysis that we learned in class. The reaching definitions are computed by the iterative fixed point algorithm:

$\mathsf{in}(\mathit{Entry}) = \mathsf{out}(\mathit{Exit}) = \emptyset$
For all $i, \mathsf{in}(B_i) = \mathsf{out}(B_i) = \emptyset$
while (changes to any $\mathsf{in}(B_i)$ and $\mathsf{out}(B_i)$ occur) {
  For all i, update
    $\mathsf{in}(B_i) = \bigcup_{P \in \mathsf{pred}(B_i)} \mathsf{out}(P)$
    $\mathsf{out}(B_i) = \mathsf{gen}(B_i) \cup (\mathsf{in}(B_i) - \mathsf{kill}(B_i))$
}

where $\mathsf{gen}(B)$ represents the set of definitions generated at block B, $\mathsf{kill}(B)$ represents the set of definitions killed at block B, and in and out are data-flow equations defined as:

$$\mathsf{in}(B_i) = \bigcup_{P \in \mathsf{pred}(B_i)} \mathsf{out}(P), \qquad \mathsf{out}(B_i) = \mathsf{gen}(B_i) \cup (\mathsf{in}(B_i) - \mathsf{kill}(B_i))$$

Consider the following program.



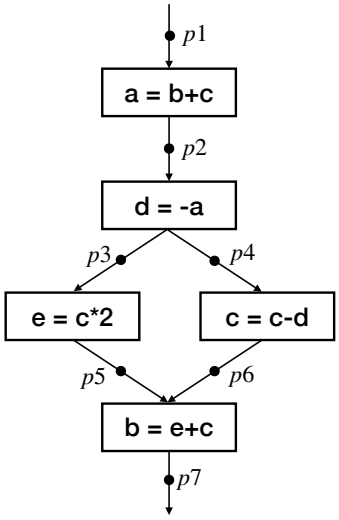
1. (12pt) Compute gen, kill, in, and out of B_1–B_3. For the empty set results, mark "empty".

$\mathsf{gen}(B_1)$	=	{ }	$\mathsf{gen}(B_2)$	=	{ }	$\mathsf{gen}(B_3)$	=	{ }
$\mathsf{kill}(B_1)$	=	{ }	$\mathsf{kill}(B_2)$	=	{ }	$\mathsf{kill}(B_3)$	=	{ }
$\mathsf{in}(B_1)$	=	{ }	$\mathsf{in}(B_2)$	=	{ }	$\mathsf{in}(B_3)$	=	{ }
$\mathsf{out}(B_1)$	=	{ }	$\mathsf{out}(B_2)$	=	{ }	$\mathsf{out}(B_3)$	=	{ }

2. (8pt) In the above data-flow equation for the reaching definition analysis, suppose we changed $\mathsf{in}(B_i) = \bigcup_{P \in \mathsf{pred}(B_i)} \mathsf{out}(P)$ into $\mathsf{in}(B_i) = \bigcap_{P \in \mathsf{pred}(B_i)} \mathsf{out}(P)$. Discuss consequent results from the change in terms of compiler optimizations.

Problem 5. (10pt) Register Allocation

Consider the program below.



1. (5pt) Assuming the live variable set at p_7, denoted $live_{p7}$, is $\{b\}$, list the live variable set for each program point $p2$–$p6$.

$live_{p1} = \{b, c, e\}$, $live_{p2} = \{\quad\}$

$live_{p3} = \{\quad\}$, $live_{p4} = \{\quad\}$

$live_{p5} = \{\quad\}$, $live_{p6} = \{\quad\}$

2. (5pt) Construct a register interference graph, and find a register assignment (i.e., annotate each node with a proper register name) with no more than 4 registers.

Problem 6. (30pt) O/X Questions

You will get 2 points for each correct answer. You will lose 2 points for each wrong answer.

(1) Operational semantics is defined compositionally. (O, X)

(2) Consider the semantic equivalence in big-step operational semantics, denoted $S_1 \equiv S_2$:

$$\langle S_1, s\rangle \to s' \quad \text{if and only if} \quad \langle S_2, s\rangle \to s'$$

Given the two statements below, $S_1 \equiv S_2$. (O, X)

S_1 : `x:=1; y:=2; z:=3;`
S_2 : `x:=1; y:=2; z:=0; if (x!=y) {z:=3} else {z:=4}`

(3) Considering the semantic equivalence in (2) again, $S_1 \equiv S_2$. (O, X)

S_1 : `x:=1; y:=2; z:=3; print (x)`
S_2 : `x:=1; y:=2; print (x)`

(4) A basic block cannot contain instructions where program executions diverge. (O, X)

(5) Assignment instructions with three operands on the right-hand side are called three-address code. (O, X)

(6) There exists an algorithm that can find the exact set of reaching definitions for any program. (O, X)

(7) For some programs containing infinite loops, reaching definition analysis may not terminate. (O, X)

(8) For some programs containing infinite loops, available expressions analysis may not terminate. (O, X)

(9) In reaching definition analysis, the types of the functions in and out are *Block* $\to$ *Definitions*. (O, X)

(10) In reaching definition analysis, $\mathsf{Kill}(d_i)$ does not kill definitions in instructions of subsequent program paths. (O, X)

(11) In reaching definition analysis, $\mathsf{Out}(B_i)$ never shrinks during the fixed point algorithm. (O, X)

(12) We can use reaching definition analysis for constant propagation (i.e., replacing some variables or expressions with known constants). (O, X)

(13) We can use constant propagation analysis to reject programs containing uninitialized variables. (O, X)

(14) Suppose we have an instruction `t = i*3`. Suppose also we have an instruction `t2 = i*3` in some subsequent program path of `t = i*3`. Suppose, finally, `i` is redefined in another subsequent program path of `t = i*3`. In this case, `i*3` cannot be a common subexpression. (Note: instruction specifications described here are incomplete, i.e., there can exist other instructions in each path) (O, X)

(15) In register allocation, there are many-to-one relationships between temporary variables and registers. (O, X)