TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

EXAMINATIONS – 2015

TRIMESTER 1

SWEN 430

Compiler Engineering

Time Allowed: THREE HOURS

CLOSED BOOK

Permitted materials: No calculators permitted.
Non-electronic Foreign language to English dictionaries are allowed.

Instructions: Answer all questions

You may answer the questions in any order. Make sure you clearly identify the question you are answering.

Question	Topic	Marks
1.	Semantics	30
2.	Type Checking	30
3.	Java Bytecode	30
4.	Dataflow Analysis	30
5.	Machine Code	30
6.	Advanced Topics	30
	Total	180

Question 1. Semantics [30 marks]

Consider the following variant of the λ_W calculus which includes syntax and semantics for simple statements and expressions and where Σ represents a map from variables to values.

$$
\begin{array}{lll}
\texttt{e} ::= & & \textit{expressions:} \\
\quad \mid \texttt{v} & & \textit{constants} \\
\quad \mid \texttt{n} & & \textit{variables} \\
\quad \mid \texttt{e}_1 \ + \ \texttt{e}_2 & & \textit{addition} \\
\\
\texttt{v} ::= & & \textit{values} \\
\quad \mid \texttt{true} \mid \texttt{false} & & \textit{booleans} \\
\quad \mid \ \ldots \ \mid \ -1 \ \mid \ 0 \ \mid \ 1 \ \mid \ \ldots & & \textit{integers}
\end{array}
\qquad
\begin{array}{lll}
\texttt{s} ::= & & \textit{statements:} \\
\quad \mid & \texttt{n} = \texttt{e}\,; & \textit{assignments} \\
\quad \mid & \texttt{return e}\,; & \textit{returns}
\end{array}
$$

$$\frac{\Sigma(\texttt{n}) = \texttt{v}}{\langle \Sigma, \texttt{n} \rangle \longrightarrow \langle \Sigma, \texttt{v} \rangle} \ \text{(R-VAR)} \qquad \frac{}{\langle \Sigma, \texttt{return v}\,; \rangle \longrightarrow \langle \Sigma, \texttt{v} \rangle} \ \text{(R-RETURN1)}$$

$$\frac{\langle \Sigma, \texttt{e}_1 \rangle \longrightarrow \langle \Sigma, \texttt{e}_1' \rangle}{\langle \Sigma, \texttt{e}_1 \ + \ \texttt{e}_2 \rangle \longrightarrow \langle \Sigma, \texttt{e}_1' \ + \ \texttt{e}_2 \rangle} \ \text{(R-ADD1)} \qquad \frac{\langle \Sigma, \texttt{e} \rangle \longrightarrow \langle \Sigma, \texttt{e}' \rangle}{\langle \Sigma, \texttt{return e}\,; \rangle \longrightarrow \langle \Sigma, \texttt{return e}'\,; \rangle} \ \text{(R-RETURN2)}$$

$$\frac{\langle \Sigma, \texttt{e}_2 \rangle \longrightarrow \langle \Sigma, \texttt{e}_2' \rangle}{\langle \Sigma, \texttt{e}_1 \ + \ \texttt{e}_2 \rangle \longrightarrow \langle \Sigma, \texttt{e}_1 \ + \ \texttt{e}_2' \rangle} \ \text{(R-ADD2)} \qquad \frac{\Sigma' = \Sigma[\texttt{n} \mapsto \texttt{v}]}{\langle \Sigma, \texttt{n} = \texttt{v}\,;\ \overline{\texttt{s}} \rangle \longrightarrow \langle \Sigma', \overline{\texttt{s}} \rangle} \ \text{(R-ASSIGN1)}$$

$$\frac{\vdash \texttt{v}_1 \ + \ \texttt{v}_2 = \texttt{v}_3}{\langle \Sigma, \texttt{v}_1 \ + \ \texttt{v}_2 \rangle \longrightarrow \langle \Sigma, \texttt{v}_3 \rangle} \ \text{(R-ADD3)} \qquad \frac{\langle \Sigma, \texttt{e} \rangle \longrightarrow \langle \Sigma, \texttt{e}' \rangle}{\langle \Sigma, \texttt{n} = \texttt{e}\,;\ \overline{\texttt{s}} \rangle \longrightarrow \langle \Sigma, \texttt{n} = \texttt{e}'\,;\ \overline{\texttt{s}} \rangle} \ \text{(R-ASSIGN2)}$$

(a) Using the above rules, evaluate each of the following terms as much as possible. Be sure to show each execution step taken.

(i) [2 marks] $\langle \emptyset, \texttt{return 1;} \rangle$

(ii) [2 marks] $\langle \{\texttt{x} \mapsto 1\}, \texttt{return x;} \rangle$

(iii) [2 marks] $\langle \{\texttt{x} \mapsto 1, \texttt{y} \mapsto 2\}, \texttt{x} = \texttt{x} + \texttt{y; return x;} \rangle$

(iv) [2 marks] $\langle \emptyset, \texttt{x} = 1\texttt{; return y;} \rangle$

(b) Consider the following syntax extension for `if` statements:

$$
\begin{array}{lll}
\texttt{s} ::= & & \textit{statements} \\
 & \ldots & \\
\quad \mid & \texttt{if(e)}\ \{\overline{\texttt{s}}\} & \textit{conditionals}
\end{array}
$$

(i) [6 marks] Provide appropriate transition rules to represent the semantics of this statement.

Using your transition rules, evaluate each of the following terms as much as possible. Be sure to show each execution step taken.

(ii) [2 marks] $\langle \{\texttt{x} \mapsto \texttt{true}\}, \texttt{if(x)}\ \{\texttt{y} = 1;\}\ \texttt{return y;} \rangle$

(iii) [2 marks] $\langle \{\texttt{x} \mapsto \texttt{false}\}, \texttt{if(x)}\ \{\texttt{y} = 1;\}\ \texttt{return y;} \rangle$

(c) In the above variant of λ_W, the order of execution for the operands to a binary addition is unspecified.

(i) [4 marks] Briefly, explain what it means for the evaluation order to be unspecified.

(ii) [4 marks] Briefly, discuss how you would modify the above rules to ensure a *left-to-right* evaluation order.

(d) [4 marks] Briefly, outline the difference between a *small-step* and *big-step* presentation of semantics. You should illustrate your discussion with one or more rules presented in both forms.

Question 2. Type Checking [30 marks]

(a) Type checking is a feature present in a number of popular modern languages such as Java and C♯.

(i) [3 marks] What are some of the benefits of type checking?

(ii) [3 marks] What are some of the limitations of type checking?

(iii) [4 marks] Are there any cases where type checking may be bad, annoying, or incorrect? Use examples in your answer where appropriate.

(b) [5 marks] Using the type rules for WHILE (given below) show how `{int x, real y}` is a subtype of `{int | real x, any y}`.

$$\frac{}{\texttt{int} \leq \texttt{real}} \quad \text{(S-Num)} \qquad \frac{}{\texttt{T} \leq \texttt{any}} \quad \text{(S-Any)}$$

$$\frac{}{\texttt{T} \leq \texttt{T}} \quad \text{(S-Reflex)} \qquad \frac{\forall i \in \{0 \ldots |n|\} \;:\; T_i \;\leq\; T'_i}{\{\overline{T}\;\overline{n}\} \;\leq\; \{\overline{T'}\;\overline{n}\}} \quad \text{(S-Record)}$$

$$\frac{\texttt{T}_1 \leq \texttt{T}_3 \quad \texttt{T}_2 \leq \texttt{T}_3}{\texttt{T}_1 \vee \texttt{T}_2 \leq \texttt{T}_3} \quad \text{(S-Union1)} \qquad \frac{\texttt{T}_1 \leq \texttt{T}_2}{\texttt{T}_1 \leq \texttt{T}_2 \vee \texttt{T}_3} \quad \text{(S-Union2)}$$

(c) Briefly describe the following terms:

(i) [2 marks] Type checking is sound.

(ii) [2 marks] Type checking is complete.

(iii) [2 marks] A stuck term.

(iv) [2 marks] Subtyping is reflexive.

(d) [7 marks] Subtyping of functions is contravariant in the parameters, and covariant in results. What does this mean and why does it have to be this way?

Question 3. Java Bytecode [30 marks]

(a) Consider the following method written in Java bytecode:

```
int f(int);
   0: iconst_0
   1: istore_2
   2: iload_2
   3: iload_1
   4: if_icmpge 14
   7: iload_2
   8: iconst_2
   9: iadd
  10: istore_2
  11: goto 2
  14: iload_2
  15: ireturn
```

(i) [3 marks] What is the *maximum stack height* of this method? Be sure to show your working.

(ii) [5 marks] Give Java source code which is equivalent to the above method.

(iii) [5 marks] Rewrite the above method to an equivalent which uses as few bytecodes as possible.

(b) Consider the following method written in Java:

```
int f(int x, int y) {
  if(x >= 0 && x < y) {
    return 0;
  } else {
    return 1;
} }
```

(i) [5 marks] Translate the above method into Java bytecode.

(ii) [4 marks] Using the above example to illustrate, briefly discuss what is meant by the term *short circuiting*.

(c) For each of the following bytecode functions, briefly discuss one way in which it could or would fail bytecode verification.

(i) [2 marks]

```
int f()
   0:  iconst_1
   1:  iconst_2
   2:  fadd
   3:  freturn
```

(ii) [2 marks]

```
int f()
   1:  ineg
   2:  ireturn
```

(iii) [2 marks]

```
int f(int,int)
   0:  iload_1
   1:  iload_2
   2:  ifeq 10
   5:  ireturn
```

(iv) [2 marks]

```
int f(int,int)
   0: invokestatic Test.g(): int
   3: pop
   3: iconst_1
   3: ireturn
```

Question 4. Dataflow Analysis [30 marks]

(a) Deadcode analysis is carried out by the Java compiler.

(i) [5 marks] What is deadcode?

(ii) [5 marks] Describe an algorithm to identify deadcode in a control flow graph.

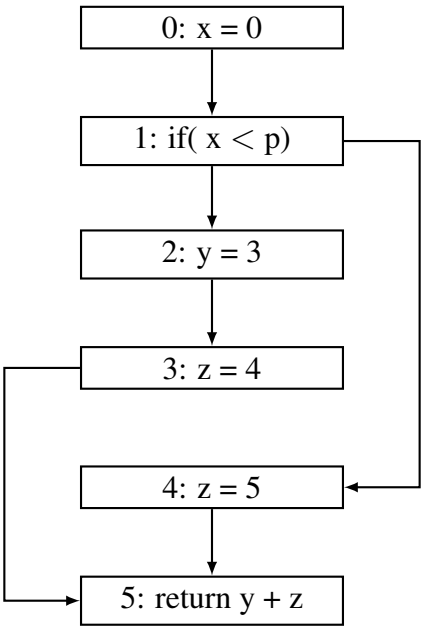
(b) Definite assignment analysis is carried out by the Java compiler.

(i) [2 marks] What is definite assignment?

(ii) [3 marks] Definite assignment is not carried out for all variables in a Java program. Explain which variables it is carried out for, and why it is not done for the others.

(iii) [5 marks] Definite assignment requires four sets. They are: DEF_{IN}, DEF_{OUT}, $USES$, and DEF_{AT}. Explain what each of these sets is for.

(c) Consider the following control flow graph. Assume that p is an argument to the method.



(i) [5 marks] Calculate the definite assignment flow sets for the above control flow graph.

(ii) [5 marks] Explain whether the preceding method is correct with respect to definite assignment?

Question 5. Machine Code [30 marks]

Consider the following function written in `x86_64` assembly language:

```
1  sum:
2          pushq   %rbp
3          movq    %rsp, %rbp
4          movq    %rdi, -24(%rbp)
5          movq    %rsi, -32(%rbp)
6          movq    $0, -8(%rbp)
7          movq    $0, -16(%rbp)
8          jmp     .L2
9  .L3:
10         movq    -16(%rbp), %rax
11         leaq    0(,%rax,8), %rdx
12         movq    -24(%rbp), %rax
13         addq    %rdx, %rax
14         movq    (%rax), %rax
15         addq    %rax, -8(%rbp)
16         addq    $1, -16(%rbp)
17 .L2:
18         movq    -16(%rbp), %rax
19         cmpq    -32(%rbp), %rax
20         jl      .L3
21         movq    -8(%rbp), %rax
22         popq    %rbp
23         ret
```

NOTE: the Appendix on page 11 provides an overview of x86_64 machine instructions for reference.

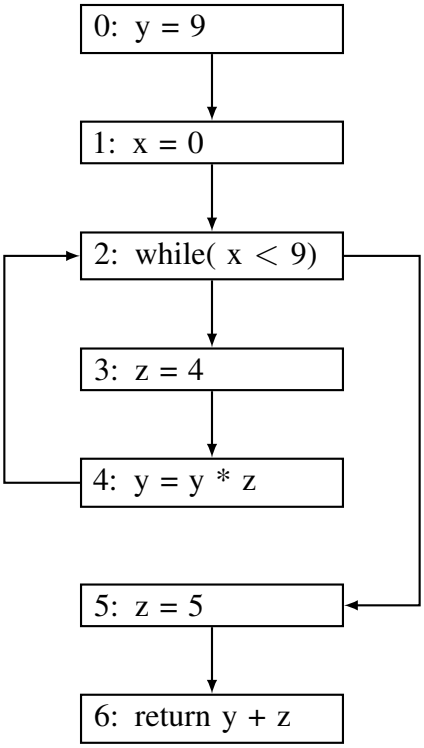
(a) [5 marks] Function parameters are normally passed *on the stack* or *in registers*. How are parameters passed in the above function? Justify your answer.

(b) [5 marks] Consider line 20 in the function. The instruction on this line is a conditional jump. However, there is nothing on this line which tells it which values to compare. Explain how jump instructions know which values they are comparing and how they know which value is bigger, or if they are the same.

(c) [5 marks] Translate the above method into WHILE.

(d) [5 marks] Suggest one optimisation that you could apply to this function, and explain why you believe it will make the machine code more efficient.

(e) Consider the following control flow graph. You can assume that there are no arguments to this method.



(i) [5 marks] Identify the live ranges for the variables x, y, z in the above control flow graph. You can use $ for the parameters being passed into the method.

(ii) [5 marks] Draw the interference graph for x, y, and z.

Question 6. Advanced Topics [30 marks]

(a) A *call graph* provides a static representation of the calling structure for a program.

(i) [5 marks] Draw a call graph for the following Java program:

```
public class Test {
  public static int inc(int x) { return x+1; }

  public static int doubleInc(int x) { return inc(inc(2)); }

  public static int add(int x, int y) {
    for(int i=0;i!=x;++i) { y = inc(y); }
    return y;
  }

  public static void main(String[] args) {
    doubleInc(5);
    add(5,10);
} }
```

(ii) [5 marks] Briefly, describe how you would implement a program to construct a call graph from a set of Java **`class`** files. You may assume all method calls are **`static`**, as illustrated in the following Java bytecode (which represents the `main` method above):

```
public static void main(java.lang.String[]);
     0: iconst_5
     1: invokestatic  doubleInc(int) : int
     4: pop
     5: iconst_5
     6: bipush        10
     8: invokestatic  add(int,int) : int
    11: pop
    12: return
```

(iii) [5 marks] Briefly, discuss why *inheritance* and *method overloading* make the problem of constructing a call graph more complicated.

(iv) [5 marks] Briefly, discuss how *Class Hierarchy Analysis* could be used to improve your program for constructing call graphs.

(b) Garbage collection is a technique used for memory management, for example, in the Java Virtual Machine.

(i) [5 marks] Briefly, outline one approach to implementing a garbage collector.

(ii) [5 marks] Briefly, outline one common challenge faced by a garbage collector.

* * * * * * * * * * * * * * *

Appendix: Overview of x86_64 Machine Instructions

`movq $c, %rax`	Assign constant `c` to `rax` register
`movq %rax, %rdi`	Assign register `rax` to `rdi` register
`addq $c, %rax`	Add constant `c` to `rax` register
`addq %rax, %rbx`	Add `rax` register to `rbx` register
`subq $c, %rax`	Substract constant `c` from `rax` register
`subq %rax, %rbx`	Subtract `rax` register from `rbx` register
`cmpq $0, %rdx`	Compare constant `0` register against `rdx` register
`cmpq %rax, %rdx`	Compare `rax` register against `rdx` register
`movq %rax, (%rbx)`	Assign `rax` register to dword at address `rbx`
`movq (%rbx),%rax`	Assign `rax` register from dword at address `rbx`
`movq 4(%rsp),%rax`	Assign `rax` register from dword at address `rsp+4`
`movq %rdx, (%rsi,%rbx,4)`	Assign `rdx` register to dword at address `rsi+4*rbx`
`pushq %rax`	Push `rax` register onto stack
`pushq %c`	Push constant `c` onto stack
`popq %rdi`	Pop qword off stack and assign to register `rdi`
`jz target`	Branch to `target` if zero flag set.
`jnz target`	Branch to `target` if zero flag not set.
`jl target`	Branch to `target` if less than (i.e. sign flag set).
`jle target`	Branch to `target` if less than or equal (i.e. sign or zero flags set).
`ret`	Return from function.