1a Consider the following C++ classes:

```
class A{
public:
    int fa1, fa2;
    virtual int g( ){ … }   }

class B: public A{
public:
    A fb1; B* fb2;
    virtual void h( ){ fb2->fa1 = fa2 + fb1.g();} }
```

Give the representation of A and B objects, and the function `void B::h()`.

b Consider the following C++ classes:

```
class C{
public:
    virtual void g(A* x){ … }
    virtual void g(B* x){ … } }

class D: public C{
public:
    B* z;
    virtual void g(B* x){ g(); }
    void g( ){ g(z); }     }
```

Give the representation of C and D objects, and of the functions `void D:: g(B*)` and `void D::g()`.

c Consider the following C++ classes:

```
class A { int fa1, fa2; virtual . . .     };

class B : public virtual A{
    int fb; virtual ...    };

class C : public virtual A{
    int fc; virtual void h(){ fa2 = 100; } };

class D : public virtual A{
    int fd; virtual ... };

class E : public B, public C, public D{
    int fe; virtual ...;
```

Outline (without fully detailing the function part of the virtual tables) the representation of C and E objects. Give the representation of the function `void C::h`, and of `cp=ep` in the context of declarations `E* ep; C* cp`.

For parts a, b and c, use the operators & and * to obtain the address of an entity or the contents of an address.

2 The language L2 is outlined in appendix A, pages 4-5 of this paper.

Consider an extension of L2, with primitives `<< ... >>` and `val`, so that execution of the expression `<<e>>` consists of blocking the execution of the expression `e`, and execution of `<<e>>.val` consists of the execution of `e`.

For example (assuming that L2 also allows for integers, sequential expressions, and local variables):

```
x = 10;
y = << 20/x >>;
x = 5;
y.val;                          // returns 4
```

a Extend the operational semantics of L2 to describe the above.

b Extend the type system of L2, introducing a class `Blocked`, so that `val` may only be applied to blocked expressions.

c Write class definitions for Booleans, which implement the method `ifThenElse` with two blocked arguments, so that if the receiver is true, the first argument is evaluated, and otherwise the second argument is evaluated.

d Write an expression which calculates `x/y` if `x` is greater than `0`, and `x` otherwise. You may assume the existence of a class `Number` with methods `greaterThan` and `divide`, and of `Zero`, a subclass of `Number`.

e For class `Blocked` write a method `repeatUntil` which takes one argument, and repeats execution of the argument until execution of the receiver returns true. (You may want to use `;` to indicate sequential execution, and `new Object` to indicate empty execution).

3 JVML00, a simplified version Java byte code, outlined in appendix B, page 6.

a Which three properties does successful verification guarantee?

b Consider the following piece of JVML00 code, in class `D`, where `A` and `B` are subclasses of `C`:

```
Method void m(A,B)
     limit locals 3
     0    load 1
     2    load 2
     3    store 1
     4    if 2
     5    load 1
```

Give types for local variables and for the stack, so that the code is well-formed.

c Extend the operational semantics and the type system to describe the instruction `goto L`, which unconditionally branches to label `L`.

d Extend the type system (but not the operational semantics) to describe the instruction `getfield A,t,` which loads on the stack the field of type `t`, declared in class `A`, from the object whose address was at the top of the stack, and where that object must belong to class `A` or a subclass of `A`.

4 Consider the ς–calculus, with terms defined by

a, b	$::= x$	*a variable*
	$\mid\ [\ l_i = \varsigma(z_i : A)\ b_i\ ^{i=1..n}\ ]$	an object
	$\mid\ b.l \Leftarrow \varsigma(z{:}\ A)\ a$	method override
	$\mid\ b.l$	method call

and evaluation, for $o \equiv [\ l_i = \varsigma(z_i{:}A)b_i\ ^{i=1..n}\ ]$ (l_i distinct) defined by

$$o.\ l_j \ \rightarrow\ b_j\{\{x_j \leftarrow o\}\} \qquad (j \in 1..n)$$
$$o.\ l_j \Leftarrow \varsigma(y)b \ \rightarrow\ [\ l_j = \varsigma(y)b,\ l_i = \varsigma(z_i{:}\ A)b_i\ ^{i=(1..n)\backslash j}\ ] \qquad (j \in 1..n)$$

a Write out the steps involved in the evaluation of the term *counter.tick.contents* where counter is defined as

$$counter \equiv [\ cont\ = \varsigma(x)0,\ \ tick = \varsigma(y)y.cont \Leftarrow \varsigma(z)y.cont + 1\ ]$$

b Consider the terms

$$tt \equiv [\ if = \varsigma(x)x.then,\ \ then = \varsigma(y)y.then,\ \ else = \varsigma(z)z.else\ ]$$
$$ff \equiv [\ if = \varsigma(x)x.else,\ \ then = \varsigma(y\ \ y.then,\ \ else = \varsigma(z)z.else\ ]$$

which encode in the ς-calculus the meaning of *true* and *false*. Assume further terms *term1*, *term2*, and *booleanTerm,* where *booleanTerm* will evaluate either to *tt* or to *ff*. Encode a conditional expression which will evaluate *term1* if *booleanTerm* returns *tt*, and *term2* otherwise.

c Consider the ςbool-calculus, the following extension of the ς – calculus, with terms defined by

a, b, c	$::=\ x$	a variable
	$\mid\ [l_i = \varsigma(z_i : A)\ b_i\ ^{i=1..n}\ ]$	an object
	$\mid\ b.l \Leftarrow \varsigma(z{:}\ A)$ a	method override
	$\mid\ b.l$	method call
	$\mid$ **true** $\mid$ **false**	
	$\mid$ **if** a **then** b **else** c	

i) Give the necessary additional rules for the operational semantics.

ii) Give a translation from the the ςbool-calculus to the ς-calculus.

iii) Formulate a theorem relating evaluation in the ςbool-calculus and in the ς-calculus.

The three parts carry, respectively, 30%, 20% and 50% of the marks.

Appendix A: Outline of the language L2

Syntax

$$
\begin{array}{lll}
\textit{progr} & ::= & \textit{class}^* \\
\textit{class} & ::= & \mathsf{class}\ c\ \ \mathsf{extds}\ c\ \ \{\ \textit{field}^*\ \textit{meth}^*\ \} \\
\textit{field} & ::= & \textit{type}\ f \\
\textit{meth} & ::= & \textit{type}\ m\ \ (\ \textit{type}\ \mathsf{x}\ )\ \ \{\ e\ \ \} \\
\textit{type} & ::= & \mathsf{bool}\ |\ c \\
e & ::= & \textit{var} := e\ \ |\ \ e\ .m\ (\ e\ )\ \ |\ \ \mathsf{new}\ c\ \ |\ \ \textit{var}\ \ |\ \ \mathsf{this}\ \ |\ \ \mathsf{true}\ \ |\ \ \mathsf{false}\ \ |\ \ \mathsf{null}\ . \\
\textit{var} & ::= & \mathsf{x}\ \ |\ \ e\ .f
\end{array}
$$

Operational Semantics

var, val

$$
\frac{}{\begin{array}{l} \mathsf{v}, \sigma \leadsto_{\mathsf{P}} \mathsf{v}, \sigma \\ \mathsf{x}, \sigma \leadsto_{\mathsf{P}} \sigma(\mathsf{x}), \sigma \\ \mathsf{this}, \sigma \leadsto_{\mathsf{P}} \sigma(\mathsf{this}), \sigma \end{array}}
$$

fld

$$
\frac{\mathsf{e}, \sigma \leadsto_{\mathsf{P}} \iota, \sigma'}{\mathsf{e.f}, \sigma \leadsto_{\mathsf{P}} \sigma'(\iota)(\mathsf{f}), \sigma'}
$$

ass

$$
\frac{\mathsf{e}, \sigma \leadsto_{\mathsf{P}} \mathsf{v}, \sigma'}{\mathsf{x}{:=}\mathsf{e}, \sigma \leadsto_{\mathsf{P}} \mathsf{v}, \sigma'[\mathsf{x} \mapsto \mathsf{v}]}
$$

fldAss

$$
\frac{\begin{array}{l} \mathsf{e}, \sigma \leadsto_{\mathsf{P}} \iota, \sigma'' \\ \mathsf{e}', \sigma'' \leadsto_{\mathsf{P}} \mathsf{v}, \sigma''' \\ \sigma'''(\iota)(\mathsf{f}) \neq \mathcal{U}df \\ \sigma' = \sigma'''[\iota \mapsto \sigma'''(\iota)[\mathsf{f} \mapsto \mathsf{v}]] \end{array}}{\mathsf{e.f}{:=}\mathsf{e}', \sigma \leadsto_{\mathsf{P}} \mathsf{v}, \sigma'}
$$

new

$$
\frac{\begin{array}{l} \mathcal{F}s(\mathsf{P}, \mathsf{c}) = \{\mathsf{f}_1, ..., \mathsf{f}_r\} \\ \mathsf{v}_j \text{ initial for } \mathcal{F}(\mathsf{P}, \mathsf{c}, \mathsf{f}_j)\ ,\ \ \mathsf{j} \in 1, ..., \mathsf{r} \\ \iota \text{ is new in } \sigma \end{array}}{\mathsf{new}\ \ \mathsf{c}, \sigma \leadsto_{\mathsf{P}} \iota, \sigma[\iota \mapsto [[\ \mathsf{f}_1 : \mathsf{v}_1, ..., \mathsf{f}_r : \mathsf{v}_r\ ]]^{\mathsf{c}}]}
$$

meth

$$
\frac{\begin{array}{l} \mathsf{e}_0, \sigma \leadsto_{\mathsf{P}} \iota, \sigma_0 \\ \mathsf{e}_1, \sigma_0 \leadsto_{\mathsf{P}} \mathsf{v}_1, \sigma_1 \\ \sigma_1(\iota) = [[\ ...\ ]]^{\mathsf{c}} \\ \mathcal{M}(\mathsf{P}, \mathsf{c}, \mathsf{m}) = \mathsf{t\ m(\ t_1\ x)}\ \ \{\ \mathsf{e}\ \} \\ \sigma' = \sigma_1[\mathsf{this} \mapsto \iota][\mathsf{x} \mapsto \mathsf{v}_1] \\ \mathsf{e}, \sigma' \leadsto_{\mathsf{P}} \mathsf{v}, \sigma'' \end{array}}{\mathsf{e}_0.\mathsf{m}(\mathsf{e}_1), \sigma \leadsto_{\mathsf{P}} \mathsf{v}, \sigma''[\mathsf{this} \mapsto \sigma(\mathsf{this}), \mathsf{x} \mapsto \sigma(\mathsf{x})]}
$$

The Type System

litVarThis

$$\frac{P \vdash \Gamma \diamond}{\begin{array}{l} P, \Gamma \vdash x : \Gamma(x) \\ P, \Gamma \vdash \mathsf{this} : \Gamma(\mathsf{this}) \end{array}}$$

newNull

$$\frac{\begin{array}{l} P \vdash \Gamma \diamond \\ P \vdash c \diamond_c \end{array}}{\begin{array}{l} P, \Gamma \vdash \mathsf{null} : c \\ P, \Gamma \vdash \mathsf{new}\ c : c \end{array}}$$

fld

$$\frac{\begin{array}{l} P, \Gamma \vdash e : c \\ \mathcal{F}(P, c, f) = t \end{array}}{P, \Gamma \vdash e.f : t}$$

ass

$$\frac{\begin{array}{l} P, \Gamma \vdash x : t \\ P, \Gamma \vdash e : t' \\ P \vdash t' \leq t \end{array}}{P, \Gamma \vdash x{:=}e : t'}$$

fldAss

$$\frac{\begin{array}{l} P, \Gamma \vdash e : c \\ P, \Gamma \vdash e' : t \\ \mathcal{F}(P, c, f) = t' \\ P \vdash t \leq t' \end{array}}{P, \Gamma \vdash e.f{:=}e' : t'}$$

methCall

$$\frac{\begin{array}{l} P, \Gamma \vdash e_0 : c \\ P, \Gamma \vdash e_1 : t'_1 \\ \mathcal{M}(P, c, m) = t\ m(\ t_1\ x)\ \ \{\ e\ \} \\ P \vdash t'_1 \leq t_1 \end{array}}{P, \Gamma \vdash e_0.m(\ e_1)\ : t}$$

wfClass

$$\frac{\begin{array}{l} \vdash P \diamond_u \\ \mathcal{C}(P, c) = \mathsf{class}\ c\ \mathsf{extds}\ \ c'\ \{\ ...\} \\ \forall f : \mathcal{F}(P, c, f) = t_0 \implies \quad P \vdash t_0 \diamond_t \quad \text{and} \quad \mathcal{F}(P, c', f) = \mathit{Udf} \\ \forall m : \mathcal{MD}(P, c, m) = \mathcal{M}(P, c, m) = t\ m(\ t_1\ x)\ \ \{\ e\ \} \implies \\ \qquad P \vdash t \diamond_t \\ \qquad P \vdash t_1 \diamond_t \\ \qquad P, t_1\ x, c\ \mathsf{this} \vdash e : t' \qquad P \vdash t' \leq t \\ \qquad \mathcal{M}(P, c', m) = \mathit{Udf} \quad \text{or} \quad \mathcal{M}(P, c', m) = t\ m(\ t_1\ x)\ \ \{\ e'\ \} \end{array}}{P \vdash c \diamond}$$

wfProg

$$\frac{\forall c : \ \mathcal{C}(P, c) \neq \mathit{Udf} \quad \implies \quad P \vdash c \diamond}{\vdash P \diamond}$$

Appendix B: Outline of JVML00

Syntax

$$instruction \quad ::= \quad \mathsf{inc} \mid \mathsf{pop} \mid \mathsf{store}\ x \mid \mathsf{load}\ x \mid \mathsf{if}\ L \mid \mathsf{halt}$$

Operational Semantics

$$\frac{\mathsf{P}[pc] = \mathsf{inc}}{\mathsf{P} \vdash pc, f, n : s \rightsquigarrow pc + 1, f, (n + 1) : s} \qquad \frac{\mathsf{P}[pc] = \mathsf{pop}}{\mathsf{P} \vdash pc, f, v : s \rightsquigarrow pc + 1, f, s}$$

$$\frac{\mathsf{P}[pc] = \mathsf{load}\ x}{\mathsf{P} \vdash pc, f, s \rightsquigarrow pc + 1, f[x \mapsto v], s} \qquad \frac{\mathsf{P}[pc] = \mathsf{store}\ x}{\mathsf{P} \vdash pc, f, v : s \rightsquigarrow pc + 1, f, s}$$

$$\frac{\mathsf{P}[pc] = \mathsf{if}\ L}{\mathsf{P} \vdash pc, f, 0 : s \rightsquigarrow pc + 1, f, s} \qquad \frac{\mathsf{P}[pc] = \mathsf{if}\ L, \quad n \neq 0}{\mathsf{P} \vdash pc, f, n : s \rightsquigarrow L, f, s}$$

The Type System

$$\frac{\begin{array}{c}\mathsf{P}[i] = \mathsf{inc} \\ F_{i+1} = F_i \\ S_{i+1} = S_i = int : \alpha \\ i + 1 \in Dom(\mathsf{P})\end{array}}{F, S, i \vdash P} \qquad \frac{\begin{array}{c}\mathsf{P}[i] = \mathsf{if}\ L \\ F_{i+1} = F_i = F_L \\ S_{i+1} = t : S_i = t : S_L \\ i + 1 \in Dom(\mathsf{P}),\ L \in Dom(\mathsf{P})\end{array}}{F, S, i \vdash P}$$

$$\frac{\begin{array}{c}\mathsf{P}[i] = \mathsf{pop} \\ F_{i+1} = F_i \\ S_{i+1} = t : S_i \\ i + 1 \in Dom(\mathsf{P})\end{array}}{F, S, i \vdash P} \qquad \frac{\begin{array}{c}\mathsf{P}[i] = \mathsf{load}\ x \\ x \in Dom(F_i) \\ F_{i+1} = F_i \\ S_{i+1} = F_i[x] : S_i \\ i + 1 \in Dom(\mathsf{P})\end{array}}{F, S, i \vdash P}$$

$$\frac{\begin{array}{c}\mathsf{P}[i] = \mathsf{store}\ x \\ x \in Dom(F_i) \\ F_{i+1} = F_i[x \mapsto t] \\ S_i = t : S_{i+1} \\ i + 1 \in Dom(\mathsf{P})\end{array}}{F, S, i \vdash P} \qquad \frac{\mathsf{P}[i] = \mathsf{halt}}{F, S, i \vdash P}$$