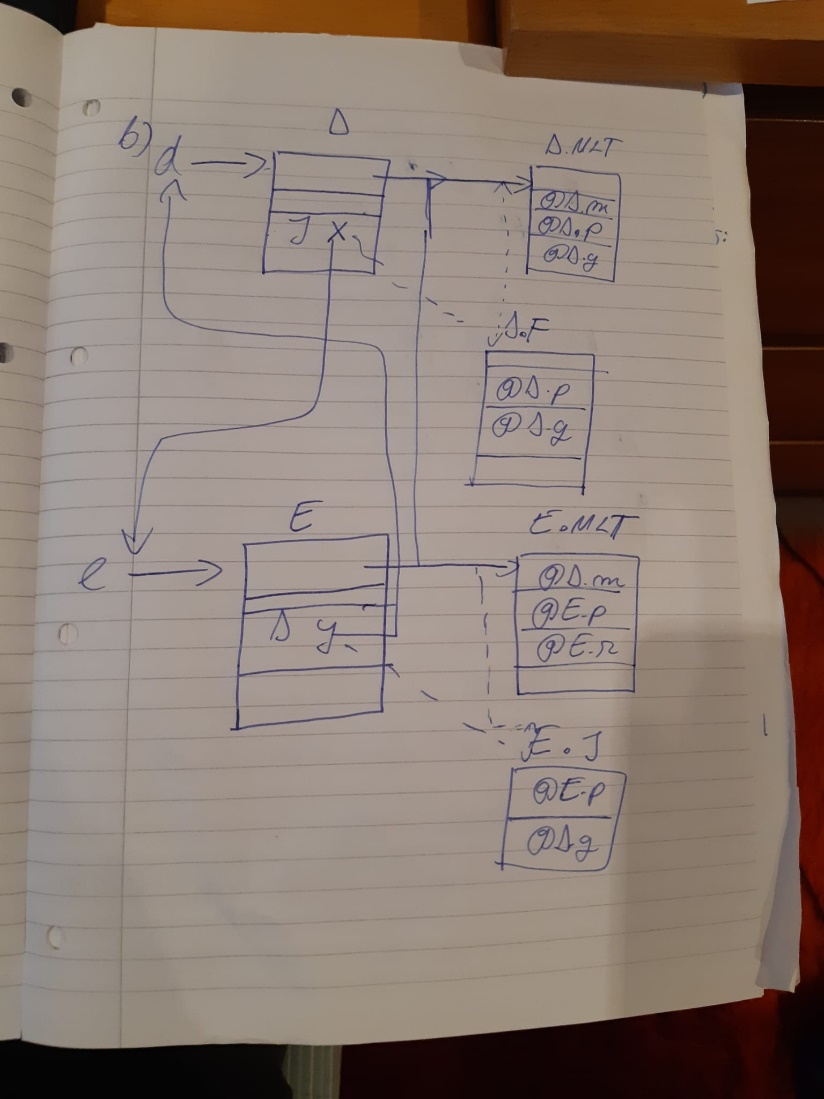
1a) In semantic analysis tutorial

1b)



Ugly diagram but you get the idea. I listed the interface part next to the MLT for clarity when you’re downcasting to the interface in the example

1c)

Assumption (to make it easier): Classes are stored in memory like this, objects also have this at the top of their memory representation:

#address --- superclass address or 0 if none

#address+4 – MLT address

(assume object address is in R1, class address is in R2, return is in R0)

check\_instance\_of:

CMP #0 [R2] ; if superclass/class address is null we return false

BEQ ret\_false

CMP [R1 #4] [R2 #4] ; if MLT points to the same thing we have the same class/interface

BEQ ret\_true

MOV [R2] R2 ; # move R2’s superclass address into register R2, redo

BL check\_instance\_of

ret\_true:

MOV #1 R0

BL exit

ret\_false:

MOV #0 R0

BL exit

exit

…..

2ai)

A list of consecutive ‘A’s that is of length 5n+1 where n>=0

ii)

A list of all numbers between 0-15 in binary except for the number 6

iii)

A string of at least 0 ‘A’s and ‘B’s where you will always have an even number of ‘A’s and ‘B’s

iv)

All non-prime numbers greater than 3

b)

For each distinct pair of alternatives (a, b) of a rule A, First(a) and First(b) are disjoint

For every rule A, if First(a) contains epsilon then First(a) and Follow(a) are disjoint

FIRST: a set of tokens(a) that could start a derivation of a

FOLLOW: a set of all tokens that could follow rule Y in the form of A->XYZ

ci)

Stays the same except for the left factorisation of Seq into:

Seq -> Expr Other

Other -> ‘,’ Seq | epsilon

ii)

FIRST (Expr) = FIRST (Operand) + FIRST (List) = {num, id} + {[} = {num, id, [}

FIRST (List) = {[}

FIRST (Seq) = FIRST (Expr) = {num, id, [}

FIRST (Rest) = {‘,’} + {ε} = {‘,’, ε}

FIRST (Operand) = {num, id}

FOLLOW (Expr) = FIRST (Rest) + FOLLOW(Rest) + {$} = {‘,’, ], $}

FOLLOW (List) = FOLLOW (Expr) + {$} = {‘,’, ], $}

FOLLOW (Seq) = {]} + FOLLOW (Rest) = { ] }

FOLLOW (Rest) = FOLLOW (Seq) = { ] }

FOLLOW (Operand) = FOLLOW (Expr) = {‘,’, ], $}

iii)

Expr → Operand | List FIRST (Operand) and FIRST (List) are disjoint

Rest → ‘,’ Seq | ε FIRST (‘,’ Seq) and FIRST (ε) are disjoint

FIRST (Rest) and FOLLOW (Rest) are disjoint

Operand → num | id FIRST (num) and FIRST (id) are disjoint

iv) not examinable

3a)

transExp (Const i) (r:rs) rx = [Mov (ImmNum i) r]

transExp (Var s) (r:rs) rx = [Mov rx r]

transExp (Minus e1 e2) (r:r2:rs) rx

= transExp e1 (r:r2:rs) rx

++ transExp e2 (r2:rs) rx

++ [Sub (Reg r) (Reg r2)]

bi)

transBExp (LessThan e1 e2) (r:r2:rs) rx label

= transExp e1 (r:r2:rs) rx

++ transExp e2 (r2:rs) rx

++ [Cmp r2 r]

++ [Bge label]

ii)

transExp (IfThenElse bexp e1 e2) (r:r2:rs) rx

= transBExp bexp (r:r2:rs) rx trueLabel

++ transExp e2 (r:r2:rs) rx

++ [Bra falseLabel]

++ [Define trueLabel]

++ transExp e1 (r:r2:rs) rx

++ [Define falseLabel]

Where

trueLabel = some unused label

falseLabel = some unused label

ci)

4ai) LiveIn(4) = {P0, P2, P3, P4}

ii) LiveOut(7) = {P0, P1, P2, P3, P4}

iii) ReachIn(4) = {1, 5, 6, 7, 9}

iv) Only 4 and 5

b)

i) defs(n) != uses(n)

ii)-iii) – not in 2021-2022 curriculum

iv) add p0, p0, #1

add P1, P0, #400

add P2, P0, #4

v) Could do something like this:

mov #0 P0

bra L1

L2:

add P1, P0, #A

mov #0, (P1)

add P0, P0, #404

L1:

cmp P0, #40400

blt L2

(Could maybe initilaize P0 as #A also and change the cmp but you’d still have to write an add so)