TD12 : Compiling OOLanguages - Tuesday, December 14

Exercise 1 Consider the following Java program :

```
class A {
    int a = 0;
    int x() { return a; }
};

class B extends A {
    int b = 2;
    int e = 4;
    int y() { a = a - b + e; return a; }
};

class C extends A {
    int c = 3;
    int e = 5;
    int x() { return a + e; }
    int z() { e = a * c; return a; }
};

class Test {
     public static void main (String[] args) {
         A p = new B();
         B r = new B();
         C s = new C();
         A t = new A();
         p.x(); r.x(); s.x(); t.x();
     }
};
```

Answer the following questions :

Question 1 The program instantiates three classes of objects (A, B, and C). Show class hierarchy and the object layout.

Solution : The class hierarchy is presented in figure 1.

The object layout is presented in figure 2. □

Question 2 Can the `e` fields in classes `B` and `C` be placed at different offsets ?

Solution : Yes, as there is no inheritance relation between the B and C classes. □

Question 3 How does the compiler deal with polymorphism in this case ?

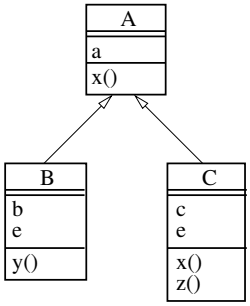

FIGURE 1 – class hierarchy

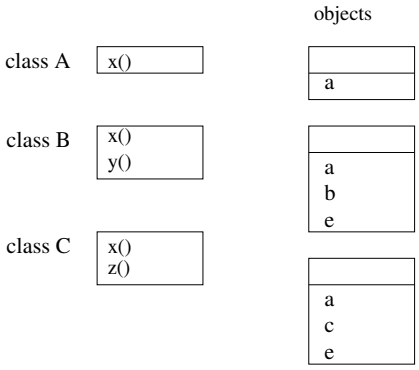

FIGURE 2 – object layout

Solution : Polymorphism is the ability of an object to disguise as another one. In our example this will be the ability of objects of classes B and C to disguise as objects of class A.

```
A p = new B();
```

There are major implications when managing polymorphism in code generation. Each object of class B has two attributes `b` and `e` and inherits the `a` attribute from class A. In order to disguise as an A class object, `p` in the above example needs to place the attributes inherited from class A first in the class instance record (CIR). Therefore, the instantiation would like this : `a`, `c`, `e` in the memory. The `c` and `e` attributes would not be accessible directly, for example `p.c`, but do exist in the memory and are accessible via a cast.

On the other hand, the order of the `c` and `e` variables can be any, but they must stay beyond the variable `a`, inherited from A, so that the code already compiled implying class A does not need changing.

□

Question 4 How does the compiler deal with virtual methods ? How many different methods (i.e., assembly-level procedures) will be generated by the compiler for the above program ? What are the names of these methods ? Give their names in the "assemblerized" form classname methodname.

Solution : In an object oriented language, methods that can be overridden are called virtual. In our example, one overridden method is `x()`. If we execute the below code, because the method `x()` is called from an A class object, the first instinct would be to call the corresponding body associated to this method belonging to class A. However this would be wrong.

```
   A p = new C();
p.x();
```

Overridden methods are enforced even if the object has been cast into the parent class. What we want in this case is to call the `x()` method associated to class B.

This is done by using the technique called virtual method table (VMT). The technique consists in generating one fixed size table for each class, containing pointers to the virtual methods defined by that class. The VMT for our classes are presented in detail in Figure 3.

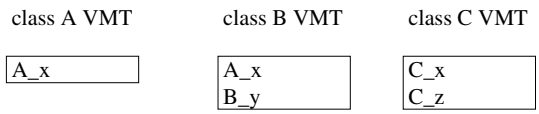

FIGURE 3 – VMTs for the classes defined in the above program

The class instance record (CIR) contains a first entry which points to the VMT corresponding the object actual class. □

Question 5 Java permits a program to explicitly cast an object into another one. But there is a complication : the Java language requires a `ClassCastException` to be thrown, when the cast is not possible. What can the compiler do to allow for this possibility ?

Solution :

```
   A p = new C();
C nc = (C) p;
```

A cast like the one presented above can be easily solved by looking at the CIR first field of the `p` variable which points to the VMT of the actual object. If the cast VMT address is equal to the VMT address in the CIR, it's we allow the cast, otherwise we trow an exception.

However, if `p` would be a subclass of `C`, casting it to `C` should still be allowed. However, in this case the cast VMT address is different from the CIR VMT address. What we need to do is add another field VMT which points to the parent VMT. In this case, we need to go through the linked list until either we find a matching VMT address, or we get to the `Object` VMT, moment when we should throw the exception. □

Question 6 Draw the memory content at the end of main. Show the pointer links between the pointer variables (p, r, s, t), objects, dispatch tables, and procedures. Note : Your picture should have four kinds of nodes (pointer variables, objects, dispatch tables, and procedures) and one kind of edge (denoting points-to relationship). The content of each pointer in the picture should be depicted as an edge to the target of the pointer.

Solution : Solution is given in figure 4

□

Question 7 Give the sequence of assembler instructions that implements the dynamic dispatch call t.x(). Assume that the value of the variable t is stored in register R0. Assume that the VMTs and the CIRs are laid out as you defined above. Comment your code fragment : what does each offset mean ? what does each register contain ?

Solution :

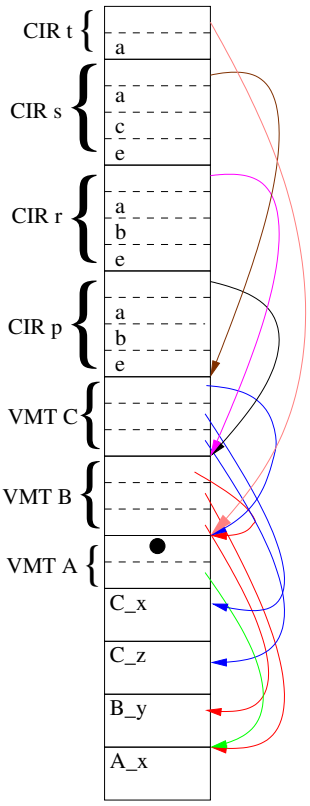

FIGURE 4 – Memory after execution

```
//push arguments to stack; no arguments => no pushing
load R1 [R0]        //the address of the VMT
load R2 [R1+offset] //load the memory address where the method code begins
move RA PC+1        // we return to the next instruction
move PC R2          // we move the program counter to the begining of the method
                    // we use the convention that the address of the object t is passed in R0
```

□

Exercise 2 Data Locality

Consider the code given below for computing a 2D Jacobi stencil :

```
for(int t=0; t<=T; t++)
    for (int i=1; i<=N-1; i++)
        for (int j=1; j<=N-1; j++)
            a[t][i][j] = 1/5*(a[t-1][i-1][j] +
                              a[t-1][i][j]   +
                              a[t-1][i][j+1] +
                              a[t-1][i+1][j] +
                              a[t-1][i][j-1]);
```

Question 8 Convert the above stencil code so to improve data locality using the method seen in course

Solution :

Data locality is improved by reducing the dependence distance

The dependence vectors are :

$$\begin{aligned}\mathcal{D} = \{\vec{d_1} &= (1,-1,0),\\ \vec{d_2} &= (1,0,0),\\ \vec{d_3} &= (1,0,1),\\ \vec{d_4} &= (1,1,0),\\ \vec{d_5} &= (1,0,-1)\}.\end{aligned}$$

First step consists in computing a set of hyperplanes (described by the normal vectors) which don't intersect the dependencies, therefore a hyperplane for which the normal vectors are perpendicular on the dependency vectors.

$$\begin{aligned}\mathcal{H} = \{\vec{r} = (r_1, r_2, r_3) \in \mathbb{R}^3 | \vec{r}\vec{d_1} &= 0\\ \vec{r}\vec{d_2} &= 0\\ \vec{r}\vec{d_3} &= 0\\ \vec{r}\vec{d_4} &= 0\\ \vec{r}\vec{d_5} &= 0\}\end{aligned}$$

yielding the system :

$$\begin{aligned}r_1 - r_2 &= 0\\ r_1 &= 0\\ r_1 + r_3 &= 0\\ r_1 + r_2 &= 0\\ r_1 - r_3 &= 0\end{aligned}$$

The only solution of this system is $\mathcal{H} = \{(0,0,0)\} \Rightarrow$ no totally independent loops to be pushed towards the exterior of the loops, therefore no fixed rows of the U matrix.

The second step consists in calculating the vectors perpendicular to the previously found hyperplane set (in the direction of the dependences).

$$\mathcal{C} = \{\vec{\phi} = (r_1, r_2, r_3) \in \mathbb{R}^3 | \vec{\phi} \perp H\}$$

with the conditions :

$$\begin{aligned} r_1 - r_2 &\geqslant 0 \\ r_1 &\geqslant 0 \\ r_1 + r_3 &\geqslant 0 \\ r_1 + r_2 &\geqslant 0 \\ r_1 - r_3 &\geqslant 0 \end{aligned}$$

which imply :

$$\begin{aligned} \vec{\phi}\vec{d}_1 &\geqslant 0 \\ \vec{\phi}\vec{d}_2 &\geqslant 0 \\ \vec{\phi}\vec{d}_3 &\geqslant 0 \\ \vec{\phi}\vec{d}_4 &\geqslant 0 \\ \vec{\phi}\vec{d}_5 &\geqslant 0 \end{aligned}$$

As there is no initial vector in $\mathcal{H}$, the only conditions to be maintained are the ones above. These conditions define a cone as shown in Figure 5 :

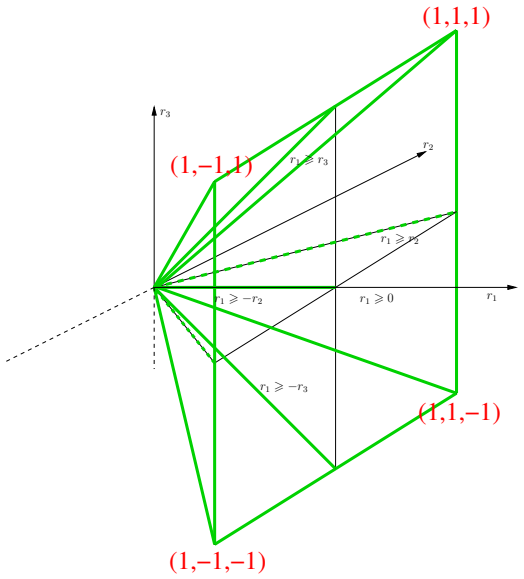

FIGURE 5 – cone defining where the τ vector lives

we choose 3 out of the 4 vectors generating the cone as a basis $(1, -1, 1), (1, 1, 1), (1, -1, -1)$

such that :

$$\mathcal{M} = \begin{pmatrix} 1 & -1 & 1 \\ 1 & 1 & 1 \\ 1 & -1 & -1 \end{pmatrix}.$$

$det(\mathcal{M}) = -4 \Rightarrow \mathcal{M}$ is not unimodular therefore we need to make it unimodular :

We compute the inverse $\mathcal{M}^{-1}$, and we choose $a = 2$ such that $a\mathcal{M}$ has integer coefficients :

$$a\mathcal{M}^{-1} = 2 \begin{pmatrix} 0 & 1/2 & 1/2 \\ -1/2 & 1/2 & 0 \\ 1/2 & 0 & -1/2 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 \\ -1 & 1 & 0 \\ 1 & 0 & -1 \end{pmatrix} = P.$$

The Hermite form of P, computed using `Mupad, hermiteForm(2/n, All)` gives :

$$U = \begin{pmatrix} 1 & -1 & 0 \\ 1 & 0 & 0 \\ 1 & -1 & -1 \end{pmatrix}, \quad H = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix}.$$

We verify that the obtained matrix U verifies yields a transformation where the τ vector (normal to the iteration hyperplane) is in the same direction as the dependency vectors :

$$Ud_1 = \begin{pmatrix} 2 \\ 1 \\ 2 \end{pmatrix}, Ud_2 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, Ud_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, Ud_1 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, Ud_1 = \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix} \Rightarrow Ud \geqslant 0 \forall d \Rightarrow \textit{.correct tiling}$$

Code generation :

The inverse of U is :

$$U^{-1} = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 1 & 0 \\ 1 & 0 & -1 \end{pmatrix}$$

we transform the code using :

$$\begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} = U \begin{pmatrix} t \\ i \\ j \end{pmatrix}$$

The initial iteration space :

$$\begin{array}{l} do(t,i,j) \in \mathbf{I} = \{(t,i,j) | 0 \leqslant t \leqslant T, 1 \leqslant i,j \leqslant N-1\} \\ \quad S(t,i,j) \end{array}$$

gets converted into :

$$do(t_1,t_2,t_3) \in \mathbf{UI} = \left\{ (t_1,t_2,t_3) | \begin{array}{l} 0 \leqslant t \leqslant T \\ 1 \leqslant i,j \leqslant N-1 \end{array} \wedge \begin{array}{l} t_1 = t-i \\ t_2 = t \\ t_3 = t-i-j \end{array} \right\}$$

$(t,i,j) = U^{-1}(t_1,t_2,t_3) \Leftrightarrow t = t_2, i = t_2 - t_1, j = t_1 - t_3$

$S(t,i,j)$

Step 1 : fining the bound on t_3

$$\mathbf{UI} = \left\{ (t_1,t_2,t_3) | \begin{array}{l} T \geqslant t_2 \geqslant 0 \\ t_3 + 1 \geqslant t_1 \geqslant t_2 - 1 \\ t_2 - N + 1 \geqslant t_1 \geqslant t_3 - N + 1 \end{array} \right\}$$

which yields :

$t_1 - N + 1 \leqslant t_3 \leqslant t_1 - 1$

producing the code :

$$for(t3 = t1 - N + 1; t3 \leqslant t_2 - 1;)$$

Step 2 : fining the bound on t_2 by eliminating t_3 from the system

$$\{(t_1,t_2) | 0 \leqslant t_2 \leqslant T \wedge t_2 - N + 1 \leqslant t_1 \leqslant t_2 - 1\}$$

which yields :

$$\begin{array}{cc} 0 \leqslant t_2 & t_2 \leqslant T \\ t1 + 1 \leqslant t2 & t_2 \leqslant t_1 + N - 1 \end{array}$$

producing the code :

$$for(t2 = max(0, t_1 + 1); t2 \leqslant min(T, t1 + N - 1);)$$

Step 3 : fining the bound on t_1 by eliminating t_2 from the system

$$\{(t_1) | -N + 1 \leqslant t_1 \leqslant T - 1\}$$

producing the code :

$$for(t1 = -N + 1; t2 \leqslant T - 1;)$$

The full code is :

```
for(int t1=-N+1; t1<=T-1; t1+=TILE_SLICE)
    for (int t2=max(0,t1+1); t2<=min(T,t1+N-1); t2+=TILE_SLICE)
        for (int t3=t1-N+1; t3<=t1-1; t3+=TILE_SLICE)
            for (int tt1=t1; tt1<=min(t1+TILE_SLICE,T-1); tt1++)
                for (int tt2=t2; tt1<=min(t2+TILE_SLICE,min(T,t1+N-1)); tt2++)
                    for (int tt3=t3; tt3<=min(t1+TILE_SLICE,t1-1); tt3++)
                        t=tt2; i=tt2-tt1; j=tt1-tt3;
                        a[t][i][j] = 1/5*(a[t-1][i-1][j] +
```

```
a[t-1][i][j]   +
a[t-1][i][j+1] +
a[t-1][i+1][j] +
a[t-1][i][j-1]);
```

□