**Space- and Time- Efficient Memory Layout**

**for Multiple Inheritance**

1. **记号**

a, b, . . . 和 u, v, w, x, y, z表示类

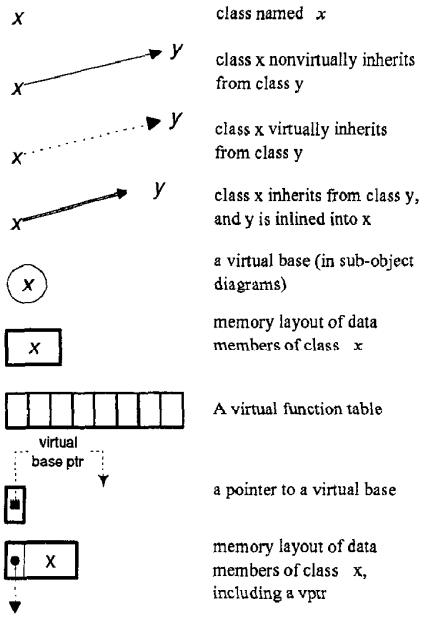
x≤y表示x = y或者x直接或间接继承于y

x<y表示x≤y且x≠y

xy表示y是x的一个直接基类

xy表示y是x的一个直接虚基类

xy表示y是x的一个直接非虚基类



**1．传统存储布局**

**1.1只有非虚继承（例1）**

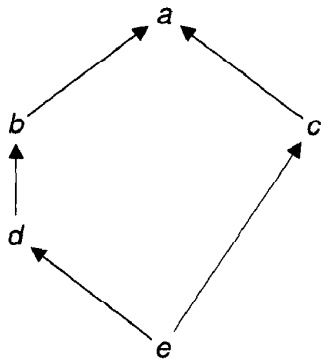


图1 例1的继承图

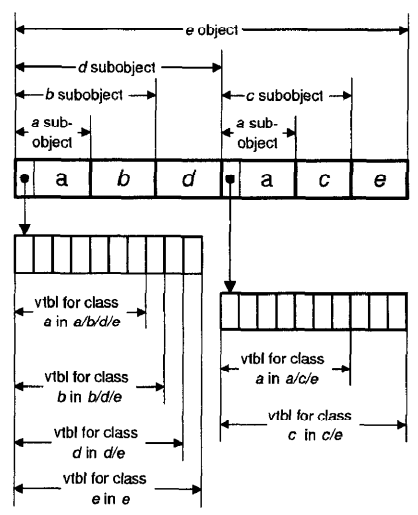


图2 例1的内存布局

内存布局中有2个虚函数表vtbl，2个a的实体，故原继承图等价于下图

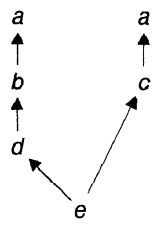


图3 与图1等价的继承图

**1.2含有公共的虚继承（例2）**

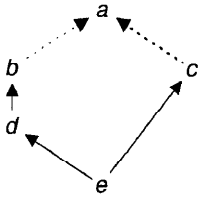


图4 例2的继承图

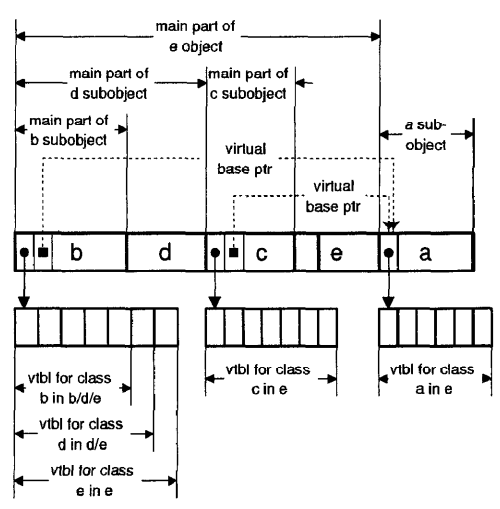


图5 例2的内存布局

还有两个指向虚基类a的指针VBPTR，则有一个是冗余的。对比例1，还多了一个vtbl。

**1.3单虚继承（例3）**



图6 例3的继承图

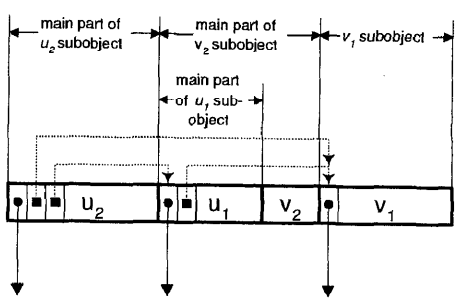


图7 例3的内存布局

在内存布局里u2的主要部分内有2个VBPTR，使得u2访问任何一个虚基类的时间都相同。

但实际上是冗余的，因为v2的主要部分含有指向虚基类v1的指针，因此u2可以通过2跳来访问v1。

**1.4极端例子（例4）**



图8 例4的继承图

a2 含有1个VBPTR，a3含有2个VBPTR，an含有n-1个VBPTR。总共有n(n-1)/2个VBPTR，其中有 (n-1)(n-2)/2个是冗余的。

**1.5其他例子（例5）**

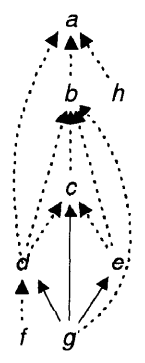
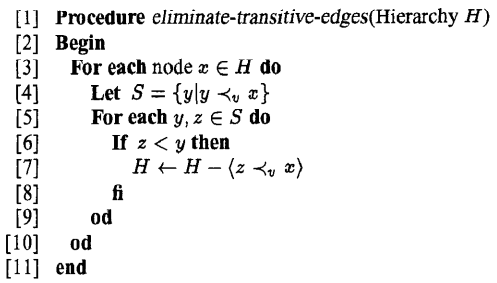


图9 例5的继承图

1. **优化方法**

**2.1 Eliminating** **Transitive Virtual Inheritance**

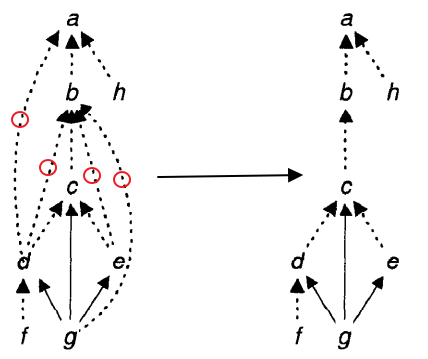
**算法**



**描述**

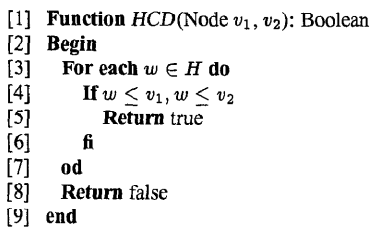
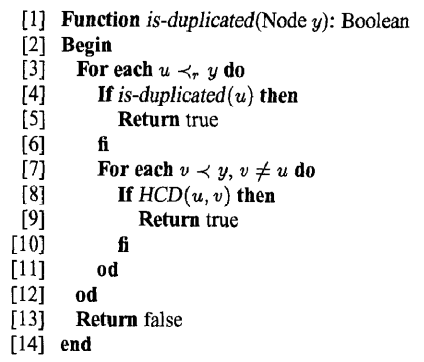
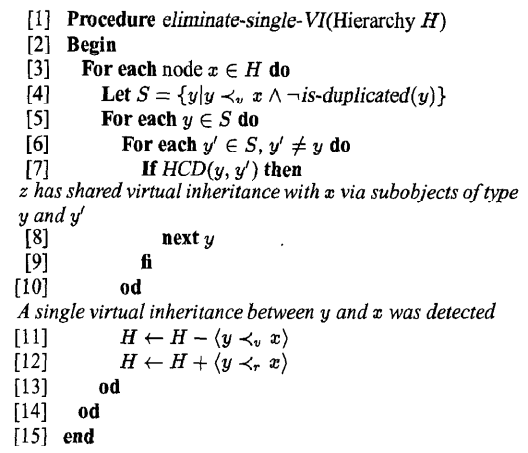
清除传递的虚继承，即如果y，z直接虚继承于x，且z是y的子类，则z的内存布局中只含有一个x，故可以清除掉继承图中”z虚继承于x”相应的边。

例5执行过程eliminate-transitive-edges的结果如下



* 1. **Edge Devirtualization**

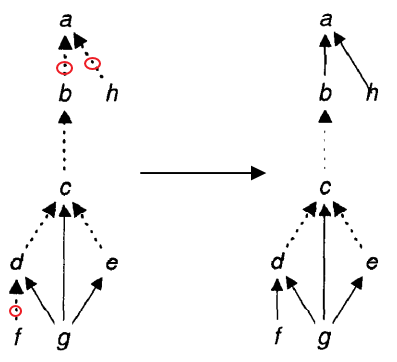
**算法**



**描述**

边去虚化，即考虑直接虚继承于x且在其子类中不重复出现的类构成的集合S，对集合S中的每一个类y，如果y与S中的任何一个其余类都没有公共的子类，那么y可以改为直接非虚继承于x。

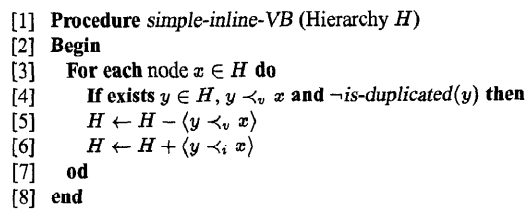
例5执行完过程eliminate-transitive-edges后再执行eliminate-single-VI的结果如下



另外，例4的所有虚继承都变成了非虚继承，此时不需要VBPTR且只需要一个vtbl

* 1. **Inlining Virtual Bases**

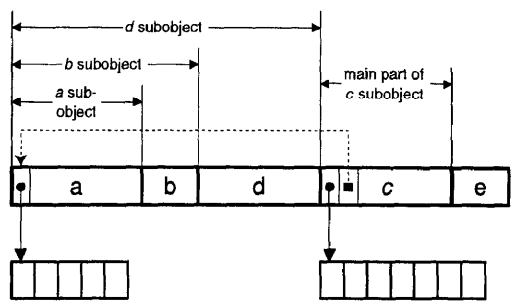
**算法**



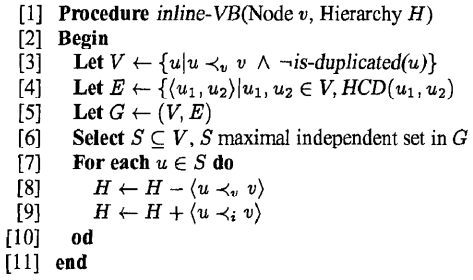
**描述**

内联虚基类，即对每一个类x，若存在y，其直接虚继承于x且在其子类中不重复出现，则改为y内联继承于x，此时x是y的直接虚基类且x内联于y。

例2执行完simple-inline-VB后的内存布局如下，少了一个VBPTR和vtbl。而且访问a也更快了。



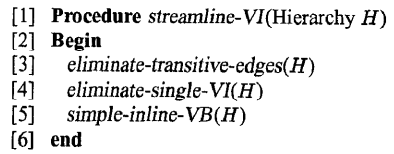
**算法**

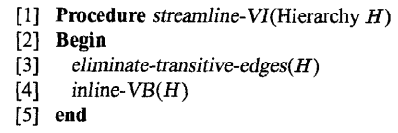


**描述**

更强力的一个版本是inline-VB：考虑直接虚继承于v且在其子类中不重复出现的类构成的集合V，依据是否有公共子类将V划分成多个等价类，每个等价类中选出一个类y，将其改为y内联继承于x。而simple-inline-VB只是从V中选出了一个。

**2.4 总过程**





Inline-VB会内联化单虚继承的边等价于边去虚化，因此不需要执行过程eliminate-single-VI。

1. **引用**

[1] Joseph(Yossi) Gil, Peter F. Sweeney. *Space-andTime-Efficient Memory Layout for Multiple Inheritance*. 1999