Konzepte objektorientierter Programmierung

Prof. Dr. Peter Müller

Software Component Technology

Exercises 12: Mobile code

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Some rules

iconst n : (S, R) → (int.S, R) if |S| < Mstack
ineg : (int.S, R) → (int.S, R)
iadd : (int.int.S, R) → (int.S, R)
iload n : (S, R) → (int.S, R)
  if 0 ≤ n < Mreg and R(n) = int and |S| < Mstack
istore n : (int.S, R) → (S, R{n ← int}) if 0 ≤ n < Mreg
aconst null : (S, R) → (null.S, R) if |S| < Mstack
aload n : (S, R) → (R(n).S, R)
  if 0 ≤ n < Mreg and R(n) <: Object and |S| < Mstack
astore n : (τ.S, R) → (S, R{n ← τ })
  if 0 ≤ n < Mreg and τ <: Object
getfield C.f.τ : (τ' .S, R) → (τ.S, R) if τ' <: C
putfield C.f.τ : (τ1 .τ2 .S, R) → (S, R) if τ1 <: τ and τ2 <: C
invokestatic C.m.σ : (τ'n . . . τ'1 .S, R) → (τ.S, R)
  if σ = τ (τ1 , . . . , τn), τ'i <: τi for i = 1 . . . n, and |τ.S| ≤ Mstack
invokevirtual C.m.σ : (τ'n . . . τ'1 .τ' .S, R) → (τ.S, R)
  if σ = τ (τ1 , . . . , τn), τ' <: C, τ'i <: τi for i = 1 . . . n, |τ.S| ≤ Mstack

Mstack: Max size of stack
Mreg: Max number of registers

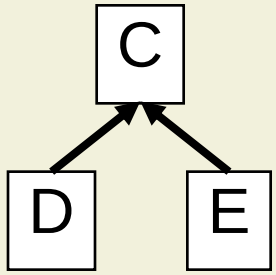
Abstract Interpretation Algorithm

- Abstract interpretation is a fixpoint iteration

```
in( 0 ) := ( [ ] , [ P0,…,Pn,T,…,T ] )
worklist := { i | instri is an instruction of the method }
while worklist ≠ ∅ do
  i := min( worklist )
  remove i from worklist
  out( i ) = apply_rule( instri )
  forall q in successors( i ) do
    in( q ) := pointwise_scs( in( q ),out( i ) )
    if in( q ) has changed then worklist := worklist ∪ { q }
  end
end
```

Inference Example

```
0: aload 0
1: astore 2
2: aload 1
3: goto 1
```



worklist

0	1	2	3

	in	out
0:	([],[D,E,T])	([D],[D,E,T])
1:	([D],[D,E,T]) ([C],[D,E,T]) ([C],[D,E,T])	([],[D,E,D]) ([],[D,E,C])
2:	([],[D,E,D]) ([],[D,E,C])	([E],[D,E,D]) ([E],[D,E,C])
3:	([E],[D,E,D]) ([E],[D,E,C])	([E],[D,E,D]) ([E],[D,E,C])

Type Checking Algorithm

Use and check declared types wherever available

Infer types otherwise

```
foreach basic block of a method body do
  in := types( start )
  foreach { i | instr_i is an instruction of basic block } do
    in := apply_rule( instr_i, in )
    forall q in successors( i ) do
      if types( q ) is declared then
        check that in is assignable to types( q )
        in := types( q )
      end
  end
end
```

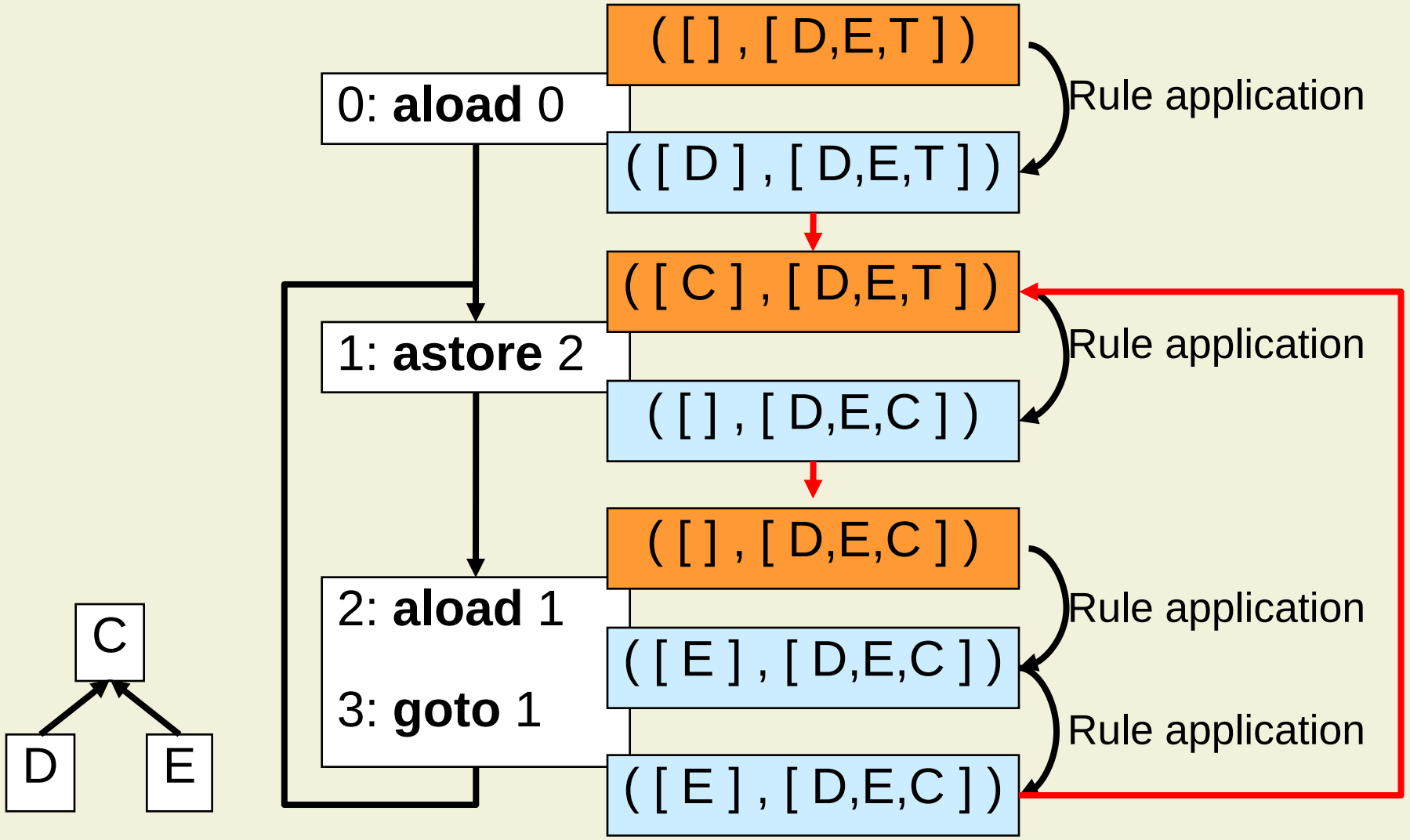
Required types

Check conditions and infer next configuration

Use declared types if $instr_i$ is not a jump

Check declared types

Type Checking Example



Exercise 1 -- A

```
class Example2 {
  void m( Object arg ) {
     Object local;
     local = “Hello”;
     local.concat (“ World!”);
     arg.concat (“Ohh!”);
  } }
```

symbol: method concat (Java.lang.String)
Location: class java.lang.Object
local.concat (“World!”);

The Java compiler uses the declared type of local.

Exercise 1 -- B

```
void m( java.lang.Object ) {
      ldc #2;  // String "Hello";
      astore 2;
      aload 2;  //String local
      ldc #3;  // String "World!";
      invokevirtual String.concat (…)
      aload 1;
      ldc #5;   //String "Ohh!"
      invokevirtual  String.concat (…)
}
```

The bytecode verifier sees a string for local.

Exercise 1 -- C

```
class Example2 {
  void m( Object arg ) {
     Object local;
     local = "Hello";
     local.concat (" World!");
     arg.concat ("Ohh!");
  } }
```

symbol: method concat (Java.lang.String)
Location: class java.lang.Object
arg.concat ("Ohh!");

The Java compiler uses the declared type of arg.

Exercise 1 -- D

```
void m( java.lang.Object ) {
      ldc #2;  // String "Hello";
      astore 2;
      aload 2;
      ldc #3;  // String "World!";
      invokevirtual String.concat (…)
      aload 1; //Object arg
      ldc #5;   //String "Ohh!"
      invokevirtual  String.concat(…)
}
```

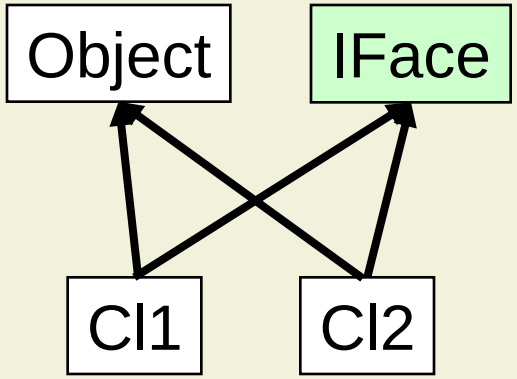
ERROR: The bytecode verifier uses the declared type of arg.

Exercise 2

```
interface IFace {
  void m ();
}

class Cl1 implements IFace{
  public void m() {
  System.out.println("Cl1.m"); }
}

class Cl2 implements IFace{
  public void m() {
  System.out.println("Cl2.m"); }
}
```



Exercise 2

```
public class Test1 {
  public static void main( String[] args ) {
     xxx(true);
     xxx(false);
  }

  public static void xxx( boolean param ) {
     IFace iface = null;

     if( param )  {  iface = new Cl1();}
     else         {  iface = new Cl2();}
     iface.m();
  }
}
```

Exercise 2

```
public class Test1 {
  public static void main( String[] args ) {
     xxx(true);
     xxx(false);
  }

  public static void xxx( boolean param ) {
     IFace iface = null;

     if( param )  {  iface = new Cl1();}
     else         {  iface = new Cl2();}
     iface.m();
  }
}
```

Exercise 2 -- A

```
void xxx(boolean);
   0: aconst null
   1: astore 1
   2: iload 0
   3: ifeq 17
   6: new class Cl1
   9: dup
  10: invokespecial
      Cl1.<init>
  13: astore 1
  14: goto 25
```

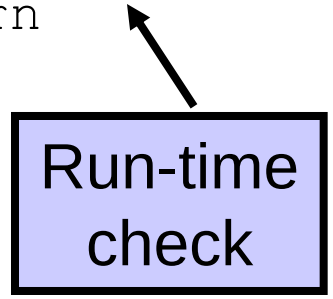
```
  17: new class Cl2
  20: dup
  21: invokespecial
      Cl2.<init>
  24: astore 1
  25: aload 1
  26: invokeinterface
      Iface.m 1
  31: return
```

SCS determines `Object` as type

Exercise 2 -- B

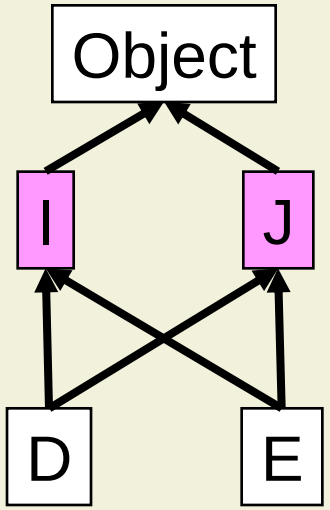
```
void xxx(boolean);
   0: aconst null
   1: astore 1
   2: iload 0
   3: ifeq 17
   6: new class Cl1
   9: dup
  10: invokespecial
      Cl1.<init>
  13: astore 1
  14: goto 25
```

```
  17: new class Cl2
  20: dup
  21: invokespecial
      Cl2.<init>
  24: astore 1
  25: aload 1
  26: invokeinterface
      Iface.m 1
  31: return
```



Handling Multiple Subtyping

- With multiple subtyping, **several smallest common supertypes** may exist
- JDK solution
  - Ignore interfaces
  - Treat all interface types as Object
  - Works because of single inheritance of classes
- Problem
  - **invokeinterface** I.m cannot check whether target object implements I
  - Runtime check is necessary



Exercise 2 -- C

```
void xxx(boolean);
   0: aconst null
   1: astore 1
   2: iload 0
   3: ifeq 17
   6: new class Cl1
   9: dup
  10: invokespecial
      Cl1.<init>
  13: astore 1
  14: goto 25
```

```
  17: new class Cl2
  20: dup
  21: invokespecial
      Cl2.<init>
  24: astore 1
  25: aload 1
  26: invokevirtual
      Iface.m 1
  31: return
```

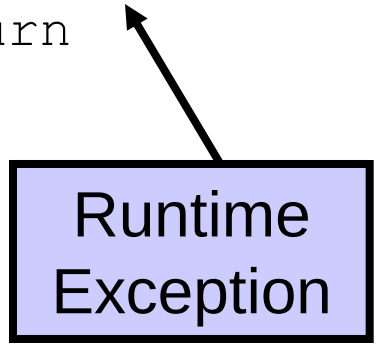
SCS determines `IFace` as type

Static check

Exercise 2 -- D

```
void xxx(boolean);
   0: aconst null
   1: astore 1
   2: iload 0
   3: ifeq 17
   6: new class Cl1
   9: dup
  10: invokespecial
      Cl1.<init>
  13: astore 1
  14: goto 25
```

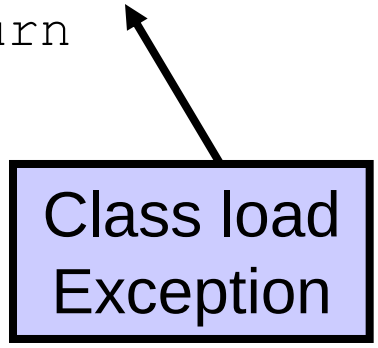
```
  17: new class Cl2
  20: dup
  21: invokespecial
      Cl2.<init>
  24: astore 1
  25: aload 1
  26: invokeinterface
      Iface.funny 1
  31: return
```



Exercise 2 -- E

```
void xxx(boolean);
   0: aconst null
   1: astore 1
   2: iload 0
   3: ifeq 17
   6: new class Cl1
   9: dup
  10: invokespecial
      Cl1.<init>
  13: astore 1
  14: goto 25
```

```
  17: new class Cl2
  20: dup
  21: invokespecial
Cl2.<init>
  24: astore 1
  25: aload 1
  26: invokevirtual
      Iface.funny 1
  31: return
```



Exercise 3 -- Program 1

```
int v = 5;
v = this;
```

Incompatible types

```
1: iconst 5
   //([int],[C,T])
2: istore 1
   //([],[C,int])
3: aload 0
   // ([C],[C,int])
4: astore 1
   // ([], [C,C])
```

Valid transition for **astore**

astore n : (τ.S, R) → (S, R{n ← τ })
if 0 ≤ n < Mreg and τ <: Object

Exercise 3 -- Program 2

```
int v = 5;
v = this;
v = v + 1;
```

Incompatible types

```
1: iconst 5
   //([int],[C,T])
2: istore 1
   //([],[C,int])
3: aload 0
   // ([C],[C,int])
4: astore 1
   // ([], [C,C])
5: iload 1
   // STUCK
6: iconst 1
7: iadd
8: istore 1
```

R(1) != int

iload n : (S, R) → (int.S, R)
if 0 ≤ n < Mreg and R(n) = int and |S| < Mstack

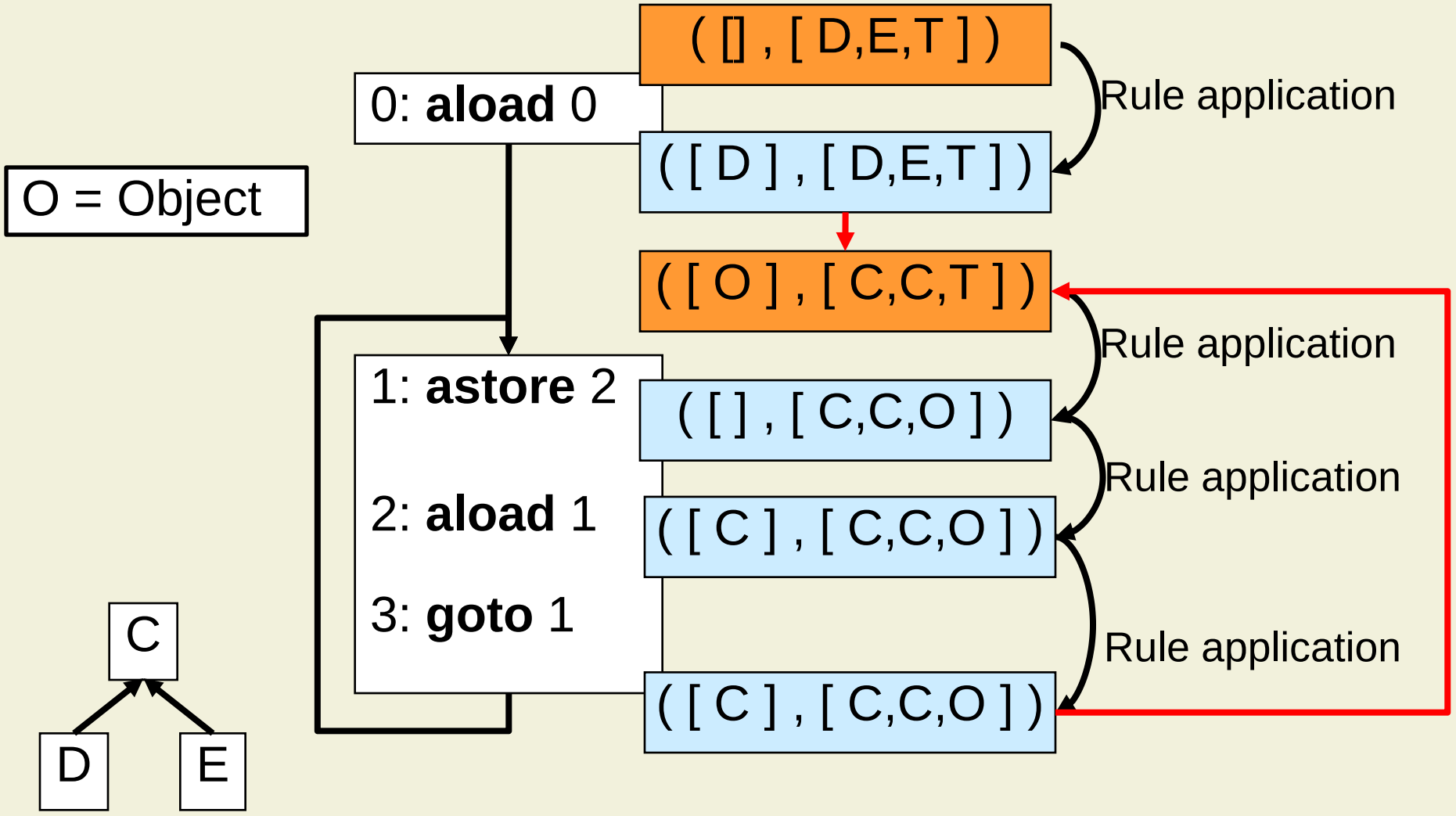
How did you do this?

- The `javap` class file disassembler produces human-readable output, but it can not be assembled again
- For the examples I used the tools `dis` and `ksm` from the kopi compiler suite, available from:

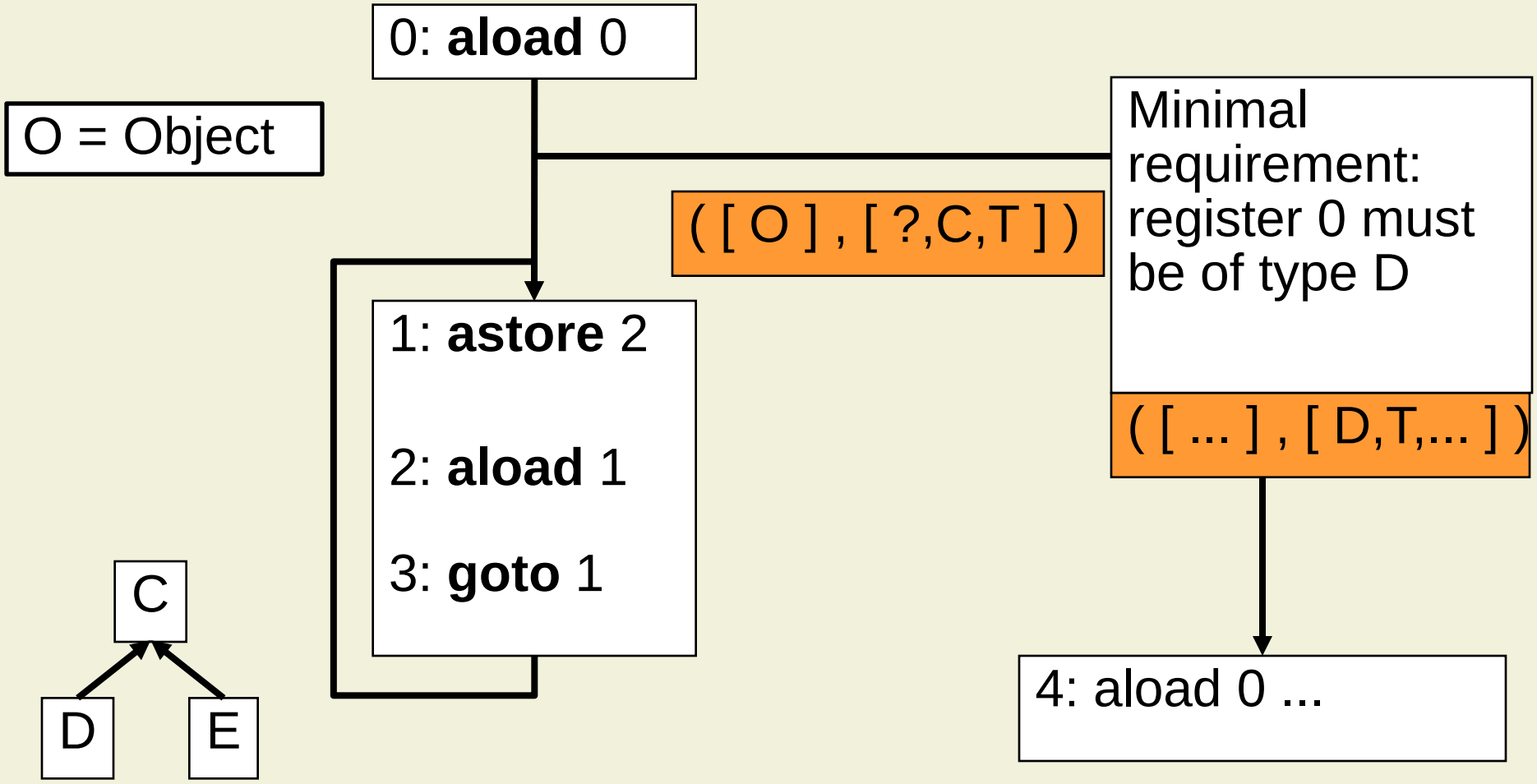
  `http://www.kopiright.com/kopi_projekt.php`

- Great learning experience to look at the bytecode and play around with it

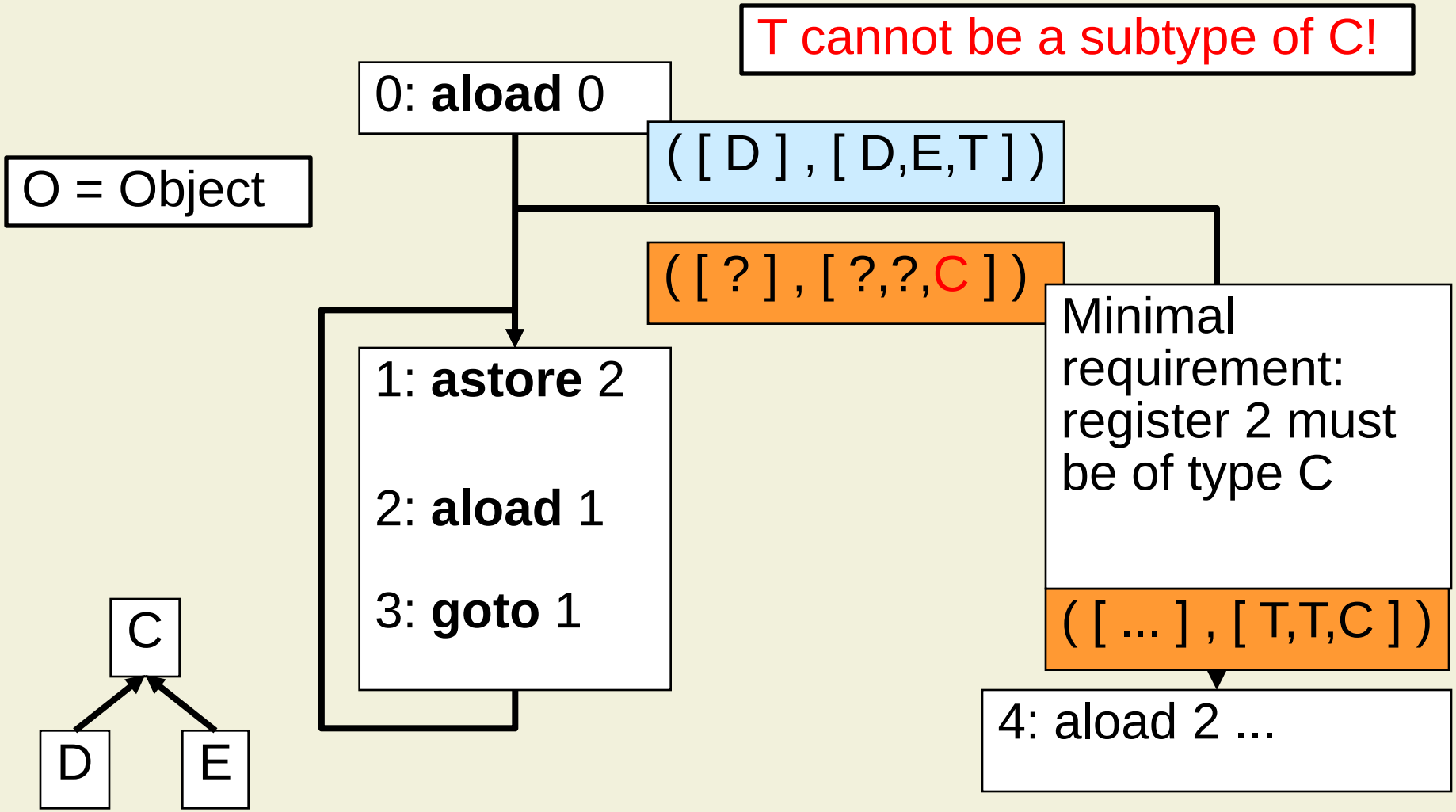
Exercise 4 -- A


([] , [D,E,T])
0: aload 0
([D] , [D,E,T])
Rule application
O = Object
([O] , [C,C,T])
Rule application
1: astore 2
([ ] , [C,C,O])
Rule application
2: aload 1
([C] , [C,C,O])
3: goto 1
Rule application
([C] , [C,C,O])
C
D
E

Exercise 4 -- B


0: aload 0
O = Object
Minimal requirement: register 0 must be of type D
([O] , [?,C,T])
1: astore 2
2: aload 1
3: goto 1
([...] , [D,T,...])
C
D
E
4: aload 0 ...

Exercise 4 -- C


T cannot be a subtype of C!
0: aload 0
([D] , [D,E,T])
O = Object
([?] , [?,?,C])
Minimal requirement: register 2 must be of type C
1: astore 2
2: aload 1
3: goto 1
([...] , [T,T,C])
4: aload 2 ...
C
D
E