1 Inclusion Polymorphism

Consider the last two lines of the example in the following slide:

- In $L_1$, $S$ points to a `Shape` object, but it could just as well have pointed to an object of any one of `Shape`’s subtypes, `Square` and `Circle`.
- If, for example, $S$ had been a `Circle`, the assignment $C := S$ would have been perfectly OK. In $L_2$, however, $S$ is a `Shape` and the assignment $C := S$ is illegal (a `Shape` isn’t a `Circle`).

2 Inclusion Polymorphism

VAR $S : \text{Shape}$; $Q : \text{Square}$; $C : \text{Circle}$;
BEGIN
  $Q := \text{NEW}(\text{Square})$;
  $C := \text{NEW}(\text{Circle})$;
  $S := Q; (* \text{OK} *)$
  $S := C; (* \text{OK} *)$
  $Q := C; (* \text{Compile-time Error} *)$
  $L_1 : S := \text{NEW}(\text{Shape})$
  $L_2 : C := S; (* \text{Run-time Error} *)$
END;

3 Typechecking Rules
TYPE T = CLASS ... END;
    U = T CLASS ... END;
    S = T CLASS ... END;
VAR t,r : T; u : U; s : S;

- A variable of type T may refer to an object of T or one of T’s subtypes.

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Compile-time</th>
<th>Run-Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>t := r;</td>
<td>Legal</td>
<td>Legal</td>
</tr>
<tr>
<td>t := u;</td>
<td>Legal</td>
<td>Legal</td>
</tr>
<tr>
<td>u := t;</td>
<td>Legal</td>
<td>Check</td>
</tr>
<tr>
<td>s := u;</td>
<td>Illegal</td>
<td></td>
</tr>
</tbody>
</table>

4 Run-time Type Checking

- The assignment s := t is compiled into s := NARROW(t, TYPE(s)).

5 Run-time Type Checking...

- The Modula-3 runtime-system has three functions that are used to implement typetests, casts, and the TYPECASE statement

- NARROW takes a template and an object as parameter. It checks that the type of the object is a subtype of the type of the template. If it is not, a run-time error message is generated. Otherwise, NARROW returns the object itself.

1. ISTYPE(S,T : Template) : BOOLEAN;
2. NARROW(Object, Template) : Object;
3. TYPECODE(Object) : CARDINAL;

6 Run-time Checks

- Casts are turned into calls to NARROW, when necessary:

VAR S : Shape; VAR C : Circle;
BEGIN
    S := NEW (Shape); C := S;
END;

VAR S : Shape; VAR C : Circle;
BEGIN
    S := malloc (SIZE(Shape));
    C := NARROW(S, Circle$Template);
END;
7 Implementing ISTYPE

- We follow the object’s template pointer, and immediately (through the templates’ parent pointers) gain access to it’s place in the inheritance hierarchy.

PROCEDURE ISTYPE (S, T : TemplatePtr) : BOOLEAN;
BEGIN
  LOOP
    IF S = T THEN RETURN TRUE; ENDIF;
    S := S^.parent;
    IF S = ROOT THEN RETURN FALSE; ENDIF;
  ENDLOOP
END ISTYPE;

8 Implementing NARROW

- NARROW uses ISTYPE to check if S is a subtype of T. Of so, S is returned. If not, an exception is thrown.

PROCEDURE NARROW(T:TemplatePtr; S:Object):Object;
BEGIN
  IF ISTYPE(S^.template, T) THEN
    RETURN S (* OK *)
  ELSE WRITE "Type error"; HALT;
  ENDIF;
END NARROW;

9 Run-time Checks — Example

TYPE T = CLASS [...];
S = T CLASS [...];
U = T CLASS [...];
V = U CLASS [...];
X = S CLASS [...];
Y = U CLASS [...];
Z = U CLASS [...];

VAR x : X;
10 Run-time Checks — Example...

\[
\begin{align*}
\text{template} & \quad \text{instance variables} \\
T & \quad \text{parent:} \\
S & \quad \text{parent:} \\
X & \quad \text{parent:} \\
Y & \quad \text{parent:} \\
Z & \quad \text{parent:} \\
\end{align*}
\]

11 Run-time Checks — An \(O(1)\) Algorithm

- The time for a type test is proportional to the depth of the inheritance hierarchy. Two algorithms do type tests in constant time:
  
  1. Norman Cohen, “Type-Extension Type Tests can be Performed in Constant Time.”
  
  2. Paul F. Dietz, “Maintaining Order in a Linked List”.

The second is more efficient, but requires the entire type hierarchy to be known. This is a problem in separately compiled languages.

- SRC Modula-3 uses Dietz’ method and builds type hierarchies of separately compiled modules at link-time.

- These algorithms only work for single inheritance.

12 Run-time Checks — Alg. II (b)

1. Build the inheritance tree.

2. Perform a preorder traversal and assign preorder numbers to each node.

3. Similarly, assign postorder numbers to each node.

4. Store \(T\)'s pre- and postorder numbers in \(T\)'s template.

**In the Compiler (or Linker):**

```pascal
PROCEDURE ISTYPE ( 
    S, T : TemplatePtr) : BOOLEAN;
BEGIN
    RETURN (T.pre ≤ S.pre) AND (T.post ≥ S.post);
END ISTYPE;
```

**In the Runtime System:**
13 Run-time Checks – Alg. II (c)

TYPE

\[
\begin{align*}
T &= \text{CLASS} \ldots; \\
S &= T \text{CLASS} \ldots; \\
U &= T \text{CLASS} \ldots; \\
V &= U \text{CLASS} \ldots; \\
X &= S \text{CLASS} \ldots; \\
Y &= U \text{CLASS} \ldots; \\
Z &= U \text{CLASS} \ldots; \\
\end{align*}
\]

\[
\begin{array}{c|c|c}
\text{GROUP} & \text{pre} & \text{post} \\
T & 1 & 7 \\
S & 2 & 4 \\
U & 4 & 6 \\
V & 5 & 3 \\
X & 5 & 1 \\
Y & 6 & 4 \\
Z & 7 & 5 \\
\end{array}
\]

\[
\begin{align*}
\sqrt{\text{ISTYPE}(Y,U)} & \quad U\text{.pre} \leq Y\text{.pre} \quad U\text{.post} \geq Y\text{.post} \\
\text{ISTYPE}(Z,S) & \quad S\text{.pre} \leq Z\text{.pre} \quad S\text{.post} \geq Z\text{.post} \\
\sqrt{\text{ISTYPE}(Z,T)} & \quad T\text{.pre} \leq Z\text{.pre} \quad T\text{.post} \geq Z\text{.post} \\
\end{align*}
\]

14 Run-time Checks – Alg. II (d)

- Consider \( U \):
  1. \( U \)'s pre-number is \( \leq \) all its children’s pre numbers.
  2. \( U \)'s post-number is \( \geq \) all its children’s post numbers.

\([U\text{.pre},U\text{.post}] \text{ “covers” } (\text{in the sense that } U\text{.pre} \leq \text{pre} \text{ and } U\text{.post} \geq \text{post}) \text{ the } [\text{pre,post}] \text{ of all its children.}

- \( S \) is not a subtype of \( U \) since \([U\text{.pre},U\text{.post}] \) does not cover \([S\text{.pre},S\text{.post}] \) \((S\text{.post} \leq U\text{.post} \text{ but } S\text{.pre} \not\geq U\text{.pre})\).

15 Inlining Methods

- Consider a method invocation \( m.P() \). The actual procedure called will depend on the run-time type of \( m \).

- If more than one method can be invoked at a particular call site, we have to inline all possible methods. The appropriate code is selected by branching on the type of \( m \).

- To improve on method inlining we would like to find out when a call \( m.P() \) can call exactly one method.
16 Inlining Methods...

17 Inlining Methods — Example

TYPE T = CLASS [f : T][
  METHOD M (); BEGIN END M;
];
TYPE S = CLASS EXTENDS T [
  METHOD N (); BEGIN END N;
  METHOD M (); BEGIN END M;
];
VAR x : T; y : S;
BEGIN
  x.M();
  y.M();
END;

18 Type Hierarchy Analysis

- For each type $T$ and method $M$ in $T$, find the set $S_{T,M}$ of method overrides of $M$ in the inheritance hierarchy tree rooted in $T$.
- If $x$ is of type $T$, $S_{T,M}$ contains the methods that can be called by $x.M()$.
- We can improve on type hierarchy analysis by using a variant of the Reaching Definitions data flow analysis.

19 Type Hierarchy Analysis...

TYPE T = CLASS [][
  METHOD M (); BEGIN END M;];
TYPE S = CLASS EXTENDS T [][
  METHOD N (); BEGIN END N;
  METHOD M (); BEGIN END M;];
VAR x : T; y : S;
BEGIN
  x.M(); $\Leftarrow S_{T,M} = \{T,M,S.M\}$
  y.M(); $\Leftarrow S_{S,M} = \{S,M\}$
END;
20  Readings and References

- Read Scott: 529–551, 554–561, 564–573
- For information on constructing layouts for multiple inheritance, see
  - William Pugh and Grant Weddell: “Two-directional record layout for multiple inheritance.”
- The time for a type test is proportional to the depth of the inheritance hierarchy. Many algorithms do type tests in constant time:
  1. Norman Cohen, “Type-Extension Type Tests can be Performed in Constant Time.”
  2. Paul F. Dietz, “Maintaining Order in a Linked List.”

21  Confused Student Email

What happens when both a class and its subclass have an instance variable with the same name?

- The subclass gets both variables. You can get at both of them, directly or by casting. Here’s an example in Java:

```java
class C1 {int a;}
class C2 extends C1 {double a;}
class C {
    static public void main(String[] arg) {
        C1 x = new C1(); C2 y = new C2();
        x.a = 5; y.a = 5.5;
        ((C1)y).a = 5;
    }
}
```