Structure

1. Late Binding (Dynamic Dispatch)
2. Virtual Method Tables
3. Context Analysis
4. Synthesis
5. BASE
6. Bonus Task: Type Checking and Type Casts at Runtime
• Virtual methods
  – The actual type of the receiver object, not its base type, determines which method is called
  – From now on, all methods in LOOP are virtual

• Late Binding
  – It is determined only at runtime which (virtual) method is called

• Virtual Method Tables
  – Objects contain hidden references to the virtual method table of their class.
  – In this table, addresses of the methods of the class are stored.
Early or Late Binding?

x: A;
...

x := NEW B.init;
WRITE x.value;
WRITE x.getValue;
## Virtual Method Tables (VMT)

### Objects

<table>
<thead>
<tr>
<th>Address</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object x</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object y</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

### Global Storage

<table>
<thead>
<tr>
<th>Address</th>
<th>VMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A.fn1</td>
</tr>
<tr>
<td>A+1</td>
<td>A.fn2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>VMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>A.fn1</td>
</tr>
<tr>
<td>B+1</td>
<td>B.fn2</td>
</tr>
<tr>
<td>B+2</td>
<td>B.fn3</td>
</tr>
</tbody>
</table>

### Code Example

```plaintext
CLASS A IS
  METHOD fn1 ...
END CLASS

CLASS B EXTENDS A IS
  METHOD fn2 ...
  METHOD fn3 ...
END CLASS

x,y: A;
x := NEW A;
x.fn1;
y := NEW B;
y.fn1;
y.fn2;
y.fn3; | compilation error
```
Context Analysis: Overriding

• If a method overrides another one, their signatures must be identical
  – Otherwise, this would be overloading, which is not supported in LOOP

• What happens if a method overrides a variable, or vice versa?
  – Either, this is forbidden, as it is overloading,
  – Or, it depends whether in the class of the accessing reference, the method or attribute is visible.
Every method is associated with a number
- The index in ist VMT
- A new attribute in class MethodDeclaration

For new methods, numbering starts after the last method number of the base class
- In class Object, numbering starts at 0

In case of overriding, the existing method number is reused
Synthesis: Generating VMTs

• For every class, a VMT must be generated

• Preparation
  – Generate Java array of MethodDeclarations of the actual class
  – Have it filled by the base classes and the actual class
  – Every entry contains the latest overridden method in the most derived class
• Code generation
  – The address of the table is labelled with \texttt{<class>}:  
  – Then generate \texttt{DAT 1, <class>\_<method>} for every method

• Object instances
  – \texttt{NEW} enters the address of the VMT at the address of the object (relative address 0)
  – Attributes start at relative address 1
  – \texttt{ClassDeclaration.HEADER\_SIZE} = 1;
• The address of the object is needed twice:
  – As parameter SELF
  – For determining the address of the VMT
• Not every method call is bound lately ...
Access to attributes and methods of the base class in a method body

**BASE and SELF**

- BASE is the same local variable as SELF, i.e., both lie at the same stack address
- The type of SELF is the actual class, the type of BASE is the base class
- BASE must be an R-value

Method calls via BASE are not bound lately!
Bonus: Type Checking and ...

- `<expr> ISA <class>`
  - Is `<expr>` of the type of `<class>`? (or of one of its subclasses?)

- `<class>(<expr>)`
  - Yields NULL if `<expr>` is not of type `<class>`, and is the identity otherwise

```object
CLASS Main IS
  METHOD main IS
    a : Object;
    b : Main;
    BEGIN
      a := SELF;
      IF a ISA Main THEN
        b := Main(a);
      END IF
    END METHOD
END CLASS
```
Bonus: ...Type Casts at Runtime

• Type is the address of the VMT
• Every VMT has a pointer to the VMT of its base class
  – Address of the base class of *Object* is 0
• *ISA* follows these pointers
• It holds true that
  *NULL ISA Object = TRUE*

CLASS Main IS
  METHOD main IS
    a : Object;
    b : Main;
    BEGIN
      a := SELF;
      IF a ISA Main THEN
        b := Main(a);
      END IF
      END METHOD
END CLASS