Generic classes :

static analysis (Lecture 4)

Baudouin Le Charlier
University of Namur
Belgium

Venise
march 2000
Plan

- Why is static analysis useful?
- What is static analysis?
- Type analysis of generic classes
- Method specialization and Inlining
- Overview of our current and future work
Why is static analysis useful for Java?

- **Optimization**
  - Current Java implementation are inefficient.
  - In particular, **dynamic dispatch** is a major source of inefficiency: a list of pointers towards all possible methods must be maintained at each “call site”. It must be scanned every time the call is activated.
  - **Type analysis** can reduce the cost of dynamic dispatch.
  - **Inlining** (i.e., copying the body of the method at the call site) is possible when a unique target type is detected at the call site.
  - **Dead code elimination** can reduce the size of the object program.
  - **Method specialization** may counter-balance the price to pay for genericity.

- **Verification**
  - null pointer dereferencing
  - cast verification
  - deadlock detection
  - uninitialized variables detection
  - ...
What is static analysis?

- Automatic derivation of program properties that hold for every possible execution.

- Can be done by “executing” the program over a non standard domain of (so-called) abstract values. (Abstract interpretation)

- E.g., in the case of type analysis, abstract values denote sets of types (i.e., the possible types of the actual instances that may arise at run-time).

- The analysis is conservative, i.e., no false result can be “derived” but the results can be inaccurate in some cases. (Unavoidable because of Computability results)

- The analysis must terminate in all cases. Hence, finite abstract domains are used (or other techniques to ensure termination: widening).
Type analysis of generic classes

Our goal is to specialize the generic method `ReadList()` for every concrete class (i.e., `StringList`, `StringL2List`, `StringLstarList`) to obtain more efficient versions by removing the need for dynamic dispatch and by inlining as many methods as possible. There are three steps:

1. Abstract interpretation of the method, assuming it is applied to an instance of `StringList`, `StringL2List`, `StringLstarList` (respectively)

2. Specializing the code for the three versions

3. Inlining.
Abstract interpretation of ReadList() applied to a StringList

public List ReadList()
{
    \this : StringList

    \Since "this" is a StringList, StringList.New() will be executed.
    SharedList l = New(null);
    \l : StringList

    while( MyIO.IsEOS() == false )
    {
        \l : StringList
        l.GetCell();
        SharedList p = New(l);
        \p : StringList
        l = p;
        \l : StringList
    }
    \l : StringList
    return(l);
} \returns a StringList.
Abstract interpretation of `New()` applied to a `StringList`

```java
final protected SharedList New() {
    \this : StringList

    StringList sl = new StringList();
    \sl : StringList

    return (sl);
}
\returns a StringList.
```
Specialization of ReadList() applied to a StringList

```csharp
public List ReadList()
{
    // this : StringList

    // Since "this" is a StringList,
    // StringList.New() will be executed.
    StringList l = New(null);
    // l : StringList

    while( MyIO.IsEOS() == false )
    {
        // l : StringList
        l.GetCell();
        StringList p = New(l);
        // p : StringList
        l = p;
        // l : StringList
    }
    // l : StringList
    return(l);
}
// returns a StringList.
```
Inlining of the specialized methods called by ReadList()

```java
public List ReadList()
{
    StringList l = new StringList();
    l.next = null;
    WriteMsg(); MyIO.GetNewLine();

    while( MyIO.IsEOS() == false )
    {
        l.info = MyIO.GetString();
        StringList p = new StringList();
        p.next = l;
        l = p;
        WriteMsg(); MyIO.GetNewLine();
    }

    return(l);
} \// returns a StringList.
```

There is no dynamic dispatch anymore. Only static method calls remain. Similar and still more impressive results can be obtained for StringL2List and StringLstarList.
Current and future works

- Join project with Prof. Agostino Cortesi (Mestre/Venise), Isabelle Pollet (Namur), and Prof. Pascal Van Hentenryck (Louvain la Neuve (Belgium) and Brown university (USA)).

- We have defined a semantic basis and abstract domains for an accurate abstract interpretation of Java.

- Our ultimate goal is the implementation of a generic platform for the static analysis, the optimization, and the verification of Java programs.

- This could allow us to make lots of money.

- We encourage students from Mestre to join us in this project.