A virtual machine for testing compilation/recompilation protocols in multiple inheritance

Julien Pagès

LIRMM, Université de Montpellier & CNRS (France)

5 July 2015

The project

Thesis subject: Study of a virtual machine in multiple inheritance based on perfect hashing

Two main objectives

- A **virtual machine** for an **object-oriented** language in **multiple inheritance**
- Study **compilation/recompilation protocols** in this system
  - key factor to performances
  - but often poorly described in scientific papers

Tools

Reuse

1. A language with selected characteristics : Nit [Privat, 2008]
2. An implementation technique for **multiple inheritance**: **perfect hashing** [Ducournau & Morandat, 2011]
3. Transform the Nit interpreter into a virtual machine

The language

The Nit language

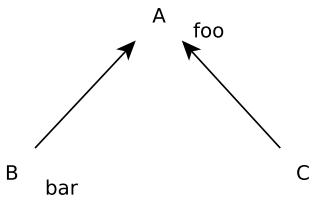
- Full object-oriented: everything is an Object
- Full multiple-inheritance
- Static typing
- Genericity...

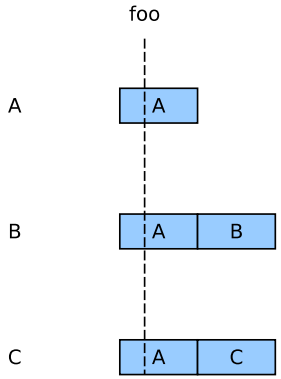
A virtual machine from an interpreter

The existing nit interpreter

- Under the closed-world assumption
- Not optimized at all, thus really simple and reusable code
- Interprets Nit programs from its Abstract Syntax Tree (AST) decorated with the meta-model (names are replaced by objects)

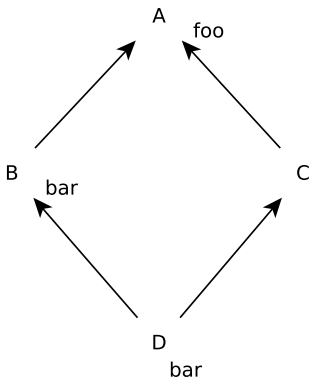
Multiple inheritance and dynamic loading

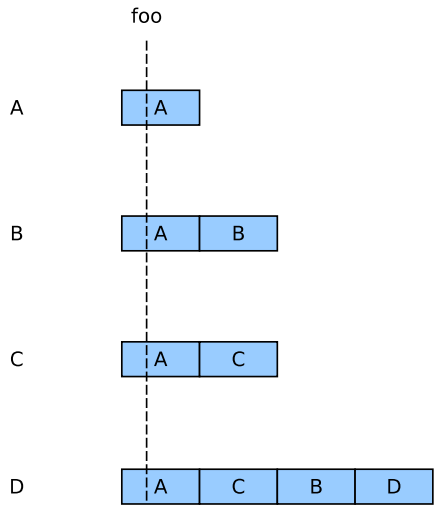




Single-inheritance

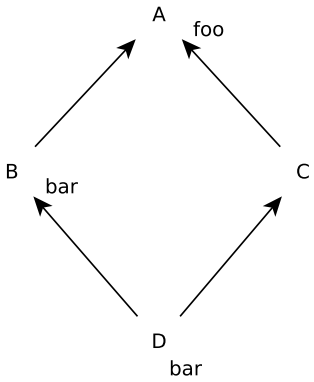
Multiple inheritance and dynamic loading

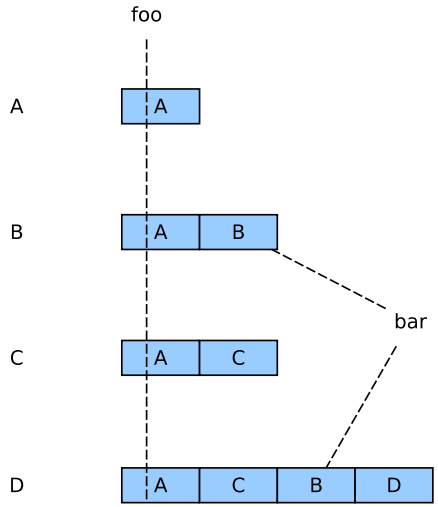




Multiple-inheritance

Multiple inheritance and dynamic loading





Problem: A method (or an attribute) has several positions in virtual tables

Object-mechanism implementations

The three object-mechanisms:

- Method dispatch
- Attribute access
- Subtyping-test

Implementations

Less to most efficient:

- *Perfect hashing* (multiple-inheritance)
- Single inheritance
- Static
- Inlining

The virtual machine

Characteristics

- Dynamic loading: **open-world assumption**
- Lazy compilation
- Gready optimization protocols: current knowledge of the world

Main optimizations

1. Devirtualization
2. Inlining

Problem: Optimizations can lead to de-optimizations: recompilations.

Repair techniques

Problem: Recompilation of a method during its execution.

Hot-repair techniques

- stack-patching (aka OSR)
- code-patching

Still better to avoid them

Avoiding hot-repair techniques

- Guards
- The preexistence property [Detlefs and Agesen, Ecoop'99] is a property of a receiver.

The preexistence property asserts that no recompilation will be needed *during the current activation*.

Definition

A protocol of compilation/recompilation

1. A toolbox:
   - collecting informations: profiling, static analysis
   - optimization techniques: devirtualization, inlining
   - repair techniques: on-stack replacement, code patching, guards, preexistence
2. And algorithms to take decisions

Extension of preexistence

The initial preexistence is based on two rules:

1. the receiver is a parameter
2. the receiver is a private immutable attribute

We propose to extends this property [ICOOOLPS'15]:

- inter-procedural analysis
- preexistence of types

➤ A receiver is now preexistent if its corresponding class is loaded.

Testbed and evaluation

A meta-evaluator benchmark

- The Nit interpreter as a source program
- Run in the nit virtual machine

How collect statistics ?

Statistics on time are not relevant because our system is too slow, two alternatives: *static counters* and *profiling*.

Collected datas

- Number of each implementations
- Number of transitions between implementations
- Number of inlinings

Conclusion

What we have: a virtual machine in multiple inheritance with simulation of compilation/recompilation protocols.

Future study of protocols

- Implement other protocols to compare them
- Study the effect of inlinings wrt preexistence

Perspectives for the virtual machine

- Adding a bytecode format in input of the virtual machine
- Explicit a real intermediate representation

Thank you

If interrested, see you tomorrow at ICOOOLPS.

Example of preexistence

```
fun bar(): B
do
    return new B()
end

fun f(A a)
do
    a.foo(); // a is preexisting
    B b = bar()
    b.foo() // b is preexisting if the class B is loaded
end
```

Simulation of compilation

Computation at first access to a method:

Characteristics

- Local variables numbering
- SSA-Algorithm computation
- Choose the implementation of all object-mechanisms sites


David Detlefs and Ole Agesen.
Inlining of virtual methods.
In *ECOOP'99*, pages 258–277. Springer, 1999.


Roland Ducournau and Floréal Morandat.
Perfect class hashing and numbering for object-oriented implementation.
Software: Practice and Experience, 41(6):661–694, 2011.


Roland Ducournau, Julien Pagès, Jean Privat, and Colin Vidal.
Preexistence revisited.
ICOOOLPS'15 at ECOOP (to appeared), 2015.


Jean Privat.
Nit language.
`http://nitlanguage.org/`, 2008.