Modeling C++ inheritance and dynamic semantics using a C++ virtual machine

Евгений Зуев
Марк Есаян
Руслан Гильванов

8 мая 2024
Университет Иннополис

Just to begin with…

«В основе проекта любого ОО-языка должна лежать объектно-ориентированная виртуальная машина»

- Проф. Н. Шилов

(цитирую по памяти)

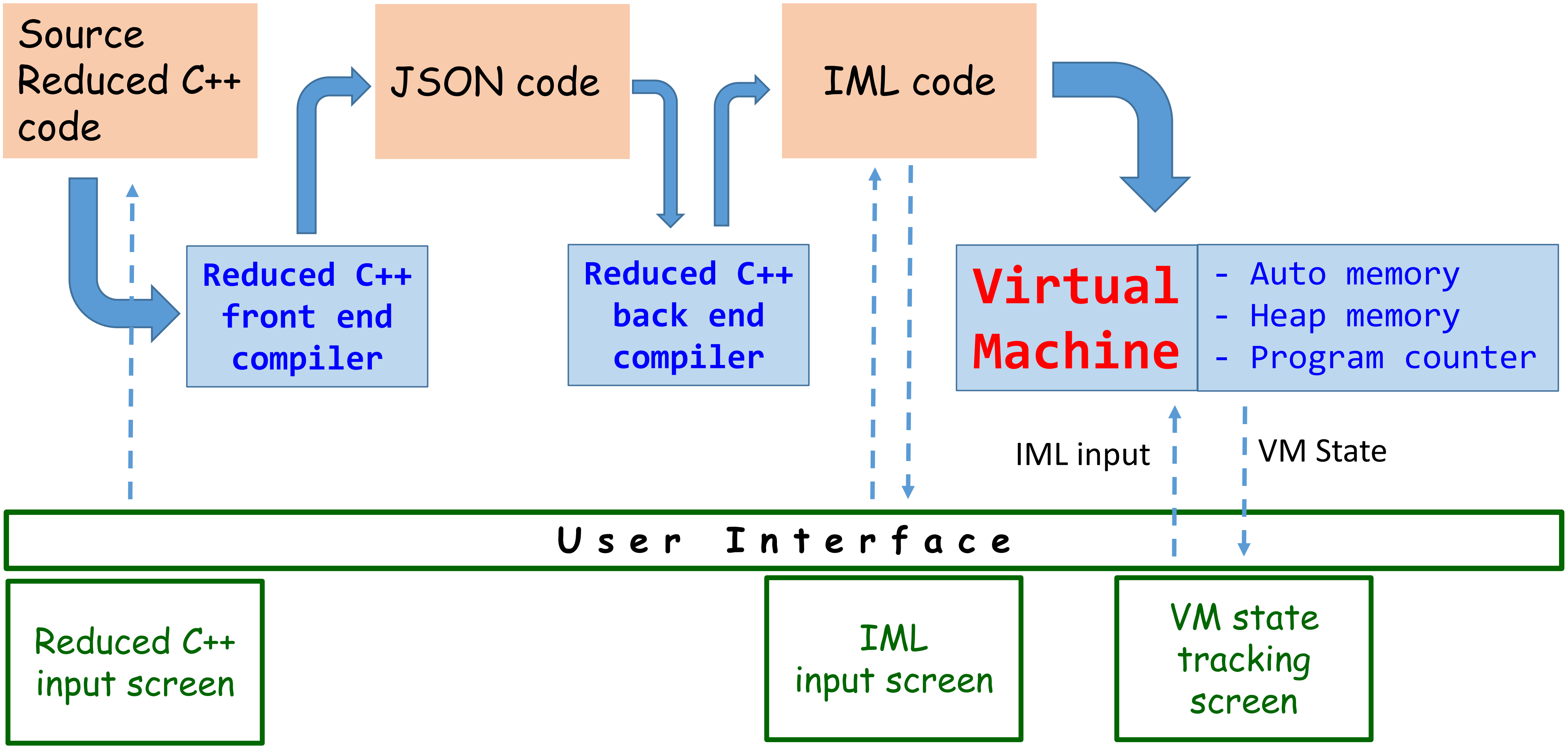
Why C++ Virtual Machine?

- To compare (numerically) semantically equivalent solutions with and without inheritance.
- To compare (numerically) "virtual" and "non-virtual" implementation approaches.
- To evaluate (numerically) costs of virtual mechanism
- To evaluate (numerically) costs of dynamic resolution of the "diamond" problem.

- Nobody has done this before ☺

Oh, really??

Overall system configuration



Reduced C++

- *A representative subset of the original C++ language*
- *Backward compatible with full C++*

- Classes: full & preliminary declarations
- Single, multiple, virtual inheritance
- Data members and member functions
- Virtual, pure virtual member functions
- Function overriding
- Constructors with mem- & ctor-initializers
- **new** operator

- Standalone functions
- Pointer and integer types
- Qualified names and selectors
- Variable declarations with initialization
- Simple assignments

Reduced C++: an Example

Typical C++ class configuration

```
class A {
    int a;
    A(int param) : a(param) {}
    A() {}
};

class B : A {
    int b;
    B(int paramA, int paramB)
        : A(paramA), b(paramB) {}
};

int main() {
    A* a = new B(3, 4);
    exit(0);
}
```

Reduced C++: an Example

Typical C++ virtual function use

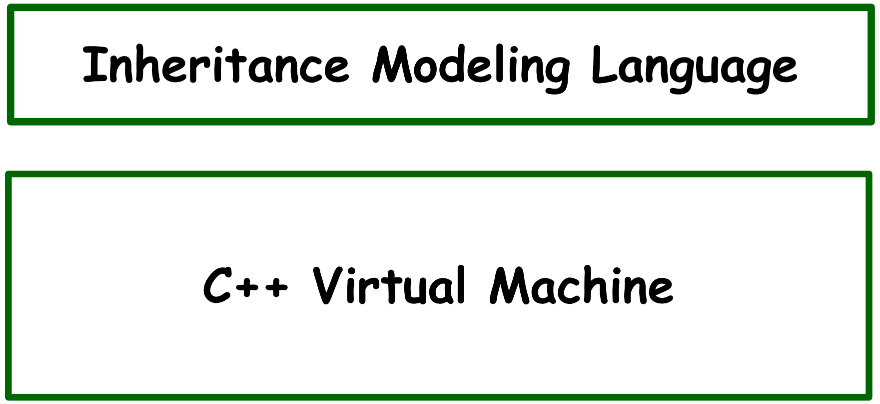
```
class Base {
    int b;
    virtual void f() { }
    Base(int pb) : b(pb) { }
};

class Derived : Base {
    int d;
    void f() { } // overrides Base::f
    Derived(int pb, int pd) : Base(pb), d(pd) { }
};

int main() {
    Base* b = new Derived(3,4);
    b->f();  // Derived::f() is called
}
```

IML: Inheritance Modeling Language

- "Byte code" for a virtual machine
- Stack based, registerless architecture
- Procedural mechanism support
- Supports common constructs and actions as declarations, initializations, assignments
- Supports full polymorphic behavior as defined in C++



- Command line shell for the VM

- Heap management
- Stack management
- Vtable & VTT management
- Dynamic dispatch

Reduced C++: Compilation 1

```
int a;
```

```
ADDRESS_ASSIGN a auto_storage_ptr
ADDRESS_FORWARD auto_storage_ptr 8
```

```
class A {
  int a;
  int b;
  A(int otherA,int otherB) :
    a(otherA), b(otherB) { }
};
```

```
COMMENT base_constructor_A_intintint
ADDRESS_PUT     thisClass auto_storage_ptr
ADDRESS_FORWARD auto_storage_ptr 8
ADDRESS_ASSIGN  otherA auto_storage_ptr
ADDRESS_FORWARD auto_storage_ptr 8
ADDRESS_ASSIGN  otherB auto_storage_ptr
ADDRESS_FORWARD auto_storage_ptr 8
ADDRESS_ASSIGN  fromLine auto_storage_ptr
ADDRESS_FORWARD auto_storage_ptr 8
ASSIGN          thisClass otherA 8
ADDRESS_FORWARD thisClass 8
ASSIGN          thisClass otherB 8
GO fromLine
```

Reduced C++: Compilation 2

```
A a(1,2);
```

```
ADDRESS_ASSIGN  a auto_storage_ptr
ADDRESS_FORWARD auto_storage_ptr 24
ADDRESS_GET     auto_storage_ptr a
ADDRESS_FORWARD auto_storage_ptr 8
ASSIGN          auto_storage_ptr 1 8
ADDRESS_FORWARD auto_storage_ptr 8
ASSIGN          auto_storage_ptr 2 8
ASSIGN          auto_storage_ptr 41 8
ADDRESS_BACKWARD auto_storage_ptr 32
ADDRESS_ASSIGN  scope_start1 auto_storage_ptr
GO 4
```

Reduced C++ to IML in More Details

```
class Base {
    int b;
    virtual void f() { }
    Base(int pb) : b(pb) { }
};

class Derived : Base {
    int d;
    void f() { } // overrides Base::f
    Derived(int pb, int pd) : Base(pb), d(pd) { }
};

int main() {
    Base* b = new Derived(3,4);
    b->f();  // Derived::f() is called
}
```

Reduced C++: Compilation 3

```
int main() {
    Base* b = new Derived(3,4);
    b->f();  // Derived::f() is called
}
```

```
// Перемещение указателя начала свободного
// стекового пространства на заданный размер
ADDRESS_FORWARD  auto_storage_ptr 8
```

```
COMMENT   function_main
ADDRESS_ASSIGN   b auto_storage_ptr       // Выделение памяти для объекта типа Base*
ADDRESS_FORWARD  auto_storage_ptr 8
HEAP_ALLOC       b 24                     // Так как объект b был создан через операцию new,
                                          // производится выделение памяти в динамической памяти
ASSIGN           auto_storage_ptr b 8     // Сохранение значения heap-указателя в стеке
ADDRESS_FORWARD  auto_storage_ptr 8
ASSIGN           auto_storage_ptr 3 8     // Сохранение аргумента (3) вызова конструктора в стеке
ADDRESS_FORWARD  auto_storage_ptr 8
ASSIGN           auto_storage_ptr 4 8     // Сохранение аргумента (4) вызова конструктора в стеке
ADDRESS_FORWARD  auto_storage_ptr 8
ASSIGN           auto_storage_ptr 82 8    // Сохранение номера "возвратной инструкции" в память стека
ADDRESS_BACKWARD auto_storage_ptr 24      // Возвращение указателя на начало свободной части стека
                                          // для возможности занять эту память следующим стекфреймом
                                          // и получить право читать эту информацию в новом scope
```

Reduced C++: Compilation 4

```
int main() {
    Base* b = new Derived(3,4);
    b->f();  // Derived::f() is called
}
```

```
ADDRESS_ASSIGN   scope_start2 auto_storage_ptr     // Создание нового стекфрейма
GO               35                                // Переход на начало вызываемой функции
ADDRESS_ASSIGN   auto_storage_ptr scope_start2     // Удаление верхнего стекфрейма
ADDRESS_PUT      virtual_function_go b             // Копирование адреса обьекта (this)
                                                   // для последующей модификации
ADDRESS_GET      auto_storage_ptr virtual_function_go    // Сохрание this в стеке
ADDRESS_FORWARD  auto_storage_ptr 8
ADDRESS_PUT      virtual_function_go virtual_function_go // Получение адреса виртуальной таблицы
ADDRESS_FORWARD  virtual_function_go 8       // Смещение внутри виртуальной таблицы
*                                            // для получения информации о вызываемом методе
ASSIGN           auto_storage_ptr 96 8       // Сохранение номера возвратной инструкции
                                             // для возвращения из вызова
ADDRESS_BACKWARD auto_storage_ptr 8          // Возвращение указателя на начало свободной части стека
ADDRESS_ASSIGN   scope_start3 auto_storage_ptr  // Начало нового стекфрейма
GO               virtual_function_go            // Переход на начало вызываемой функции
ADDRESS_ASSIGN   auto_storage_ptr scope_start3  // Закрытие стекфрейма (см. *)
ADDRESS_ASSIGN   auto_storage_ptr scope_start0  // закрытие изначального стек фрейма (main)
                                                // -> программа закончилась
```

Demo:
Interactive execution

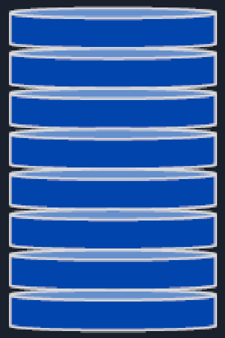
Evaluation

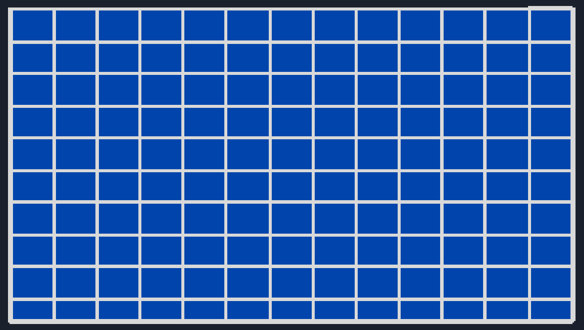
- Each bytecode instruction is assigned a predefined efficiency score (EF).
- Executed programs receive final performance score.
- Program scores allow to objectively evaluate and compare performance.

$$E = \sum C(B_i)$$

B_i – ith byte code instruction
$C(B_i)$ – EF for ith instruction
$i = 1, 2, \ldots, k$

Execution: VM memory model overview





Auto Storage (Stack)

1. Lightweight storage
2. Has predefined lifetime for stored values
3. Constant costs of operations

Heap

1. Heavy memory storage
2. Has no rules for value lifetime
3. Runtime evaluation of operation costs

Evaluation: Instructions' Costs

ADDRESS_ASSIGN	2	Copying address
ADDRESS_GET	4	Instruction requires copying address (1EF) and store it inside the receiver memory (2EF)
ADDRESS_PUT	4	Reverse instruction of the previous
ADDRESS_FORWARD	4	Takes address and parameter (2EF), applies arithmetic operation, stores new address
ADDRESS_BACKWARD	4	Same as previous
ASSIGN	(2 | 3) * N	2 if provider literal, 3 if provider is alias. N = sizeof / size_of_address
HEAP_ALLOC	2 + M	M = costs of heap management, 2 EF for allocation and assigning new address
HEAP_FREE	2 + M	Same as above
GO	2 | 3	2 If the value provider is a literal, 3 in case provider is an address
COMMENT	0	has no runtime impact