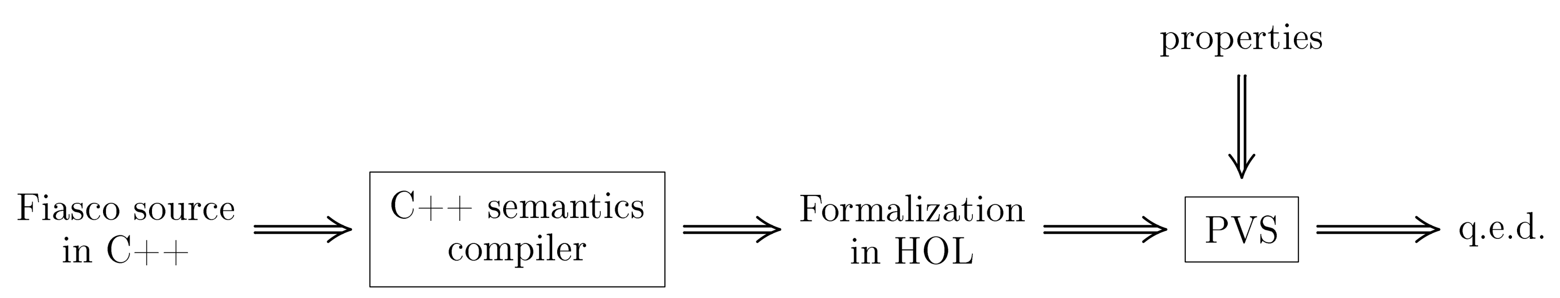
The Semantics of C++ Data Types: Towards Verifying low-level System Components

Hendrik Tews **Michael Hohmuth**

TU Dresden, Germany `vfiasco@os.inf.tu-dresden.de`

Context The VFiasco Project

- see `http://www.vfiasco.org`
- Verify essential properties of the Fiasco microkernel operating system



Aims for this work

- semantics for C++ data types (e.g., `int`, `union{ ... }`, `struct{ ... }`)
- suitable for reasoning about dirty operating system internals
- scales well to different C++ implementations
- supports popular unsafe C++ idioms (casts)
- **detects programming errors**

Errors we want to detect

- **first example:**

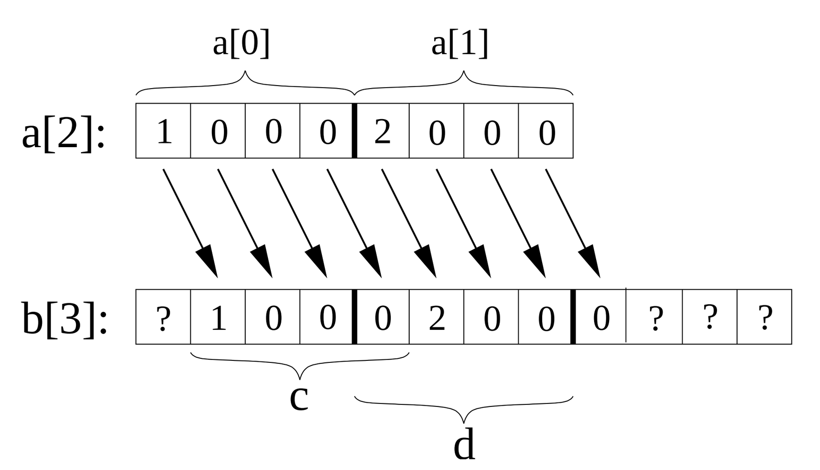
```
T a = * reinterpret_cast<T *>(50)
```

is correct *only if* an object of type T has been stored at address 50 before.

- **second example:**

```
unsigned a[2] = { 1, 2 };
unsigned b[3];
memcpy(static_cast<char *>(static_cast<void *>(b)) + 1, a, sizeof(a));
unsigned c = * static_cast<unsigned *> (static_cast<void *>
                    (static_cast<char *> (static_cast<void *> (b)) + 1));
unsigned d = b[1];
```

– memcpy is correct and works as follows:



– the Variable c is initialised with 1 because it reads a valid integer (provided there are no alignment problems)

– however the access of b[1] *is an error*, because no integer has been stored at b[1] (although this is ok with all C++ systems, the result is 512 on little-endian machines)

Approach: Take the C++ Standard literally

- for every C++ type model *values* and their *object representation* (in memory)
- every value has *at least one* object representation and every object representation corresponds to *at most one* value
  - there are values with different object representations (+0, −0)
  - some object representations correspond to no value
  - **The C++ standard admits lisp-like runtime typechecking!**
- **Observation 1** *A C++ program that does not crash on any C++ implementation must be type correct in the following sense:*
  - *For any type T which is not a character type, it accesses objects of type T only at memory locations that contain a correctly initialised object of type T.*
  - *It accesses members of any non-POD type T only at memory locations that have been initialised by a constructor or assignment operator for T.*
- For every type T any C++ implementation fixes its set of values and their object representations. This forms a *model* of T.
- As semantics of T we develop a specification for all possible models of T.
- The available proof power corresponds precisely to the guarantees of the C++ Standard.

PVS Formalisation

- Assume a memory with the following interface:

```
memory_read : [State, Address -> Byte]
memory_write : [State, Address, Byte -> State]
```

with the following types:

State the states of the memory (in the Fiasco verification State is a very complex type)
Address available addresses (typically a subset of the natural numbers)
Byte the contents of bytes

- The semantics of a C++ type T is given as a record

```
Data_type_structure : Type = [#
  size : nat,
  to_byte : [Data, Address -> list[Byte]],
  from_byte : [list[Byte], Address -> lift[Data]]
#]
```

Data Type parameter, the type of values of T
size the length of the object representation
to_byte converts a value into its object representation
from_byte converts an object representation into a value (failure possible)

- the following properties are required:

```
pod_data_type? : PRED[Data_type_structure] = Lambda(ct : Data_type_structure) :
  (Forall(d : Data, a : Address) : length(to_byte(ct)(d,a)) = size(ct)) And
  (Forall(d : Data, a : Address) : up?(from_byte(ct)(to_byte(ct)(d,a), a))) And
  (Forall(d : Data, a : Address) : down(from_byte(ct)(to_byte(ct)(d,a), a)) = d ) And
  (Forall(l : list[Byte], a : Address) : up?(from_byte(ct)(l, a)) Implies length(l) = size(ct))
  (Forall(d : Data, a1,a2 : Address) : up?(from_byte(ct)(to_byte(ct)(d,a1), a2)))
```

– the object representation of all values is size(ct) bytes long.
– from_byte is a left inverse of to_byte (i.e., from_byte is defined on all results of to_byte and yields the original value).
– from_byte fails on objects of the wrong size
– for POD types the from_byte function is successful even if the object has been moved

(There is an important distiction between POD and non-POD types, please consult the paper.)

Example: Semantics of Booleans

```
Cxx_Bool: THEORY
BEGIN
  Semantics_bool: TYPE = bool
  IMPORTING Abstract_Data[Semantics_bool]
  dt_bool_exists : AXIOM EXISTS (x: (pod_data_type?[Semantics_bool])): TRUE
  dt_bool : (pod_data_type?[Semantics_bool])
END Cxx_Bool
```

To ensure consistency we also develop a model for every data type (consult paper).

Example: Structures consider `struct Z { int x; char y; };`

```
Cxx_struct_Z: Theory
Begin
  Semantics_Z : Type = [# x: Semantics_int, y: Semantics_char #]
  Offsets_Z : Type =   [# x_offs: nat, y_offs: nat #]
  Importing Abstract_Data[Semantics_Z]

  dt_Z_exists : Axiom Exists (x: (pod_data_type?[Semantics_Z])): true
  dt_Z : (pod_data_type?[Semantics_Z])
  offsets_Z : Offsets_Z

  offs_order_Z: Axiom 0 = x_offs(offsets_Z) And
    x_offs(offsets_Z) + size(dt_int) <= y_offs(offsets_Z) And
    y_offs(offsets_Z) + size(dt_char) <= size(dt_Z)

  from_byte_whole_Z: Axiom
    Forall(bl: list[Byte], a : Address):    up?(from_byte(dt_Z)(bl,a)) Implies
      Let x = down(from_byte(dt_Z)(bl,a)),
          lift_a = from_byte(dt_int)(sublist(x_offs(offsets_Z), size(dt_int))(bl), a + x_offs(offsets_Z)),
          lift_b = from_byte(dt_char)(sublist(y_offs(offsets_Z), size(dt_char))(bl), a + y_offs(offsets_Z))
      IN
          up?(lift_a) And down(lift_a) = x(x) And up?(lift_b) And down(lift_b) = y(x)

  from_byte_parts_Z: Axiom
    Forall(bl: list[Byte], a : Address):    length(bl) = size(dt_Z) Implies
      Let lift_x = from_byte(dt_Z)(bl, a),
          lift_a = from_byte(dt_int)(sublist(x_offs(offsets_Z), size(dt_int))(bl), a + x_offs(offsets_Z)),
          lift_b = from_byte(dt_char)(sublist(y_offs(offsets_Z), size(dt_char))(bl), a + y_offs(offsets_Z))
      IN
        up?(lift_a) And up?(lift_b) Implies
          up?(lift_x) And down(lift_a) = down(lift_x)'x And down(lift_b) = down(lift_x)'y
End Cxx_struct_Z
```