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

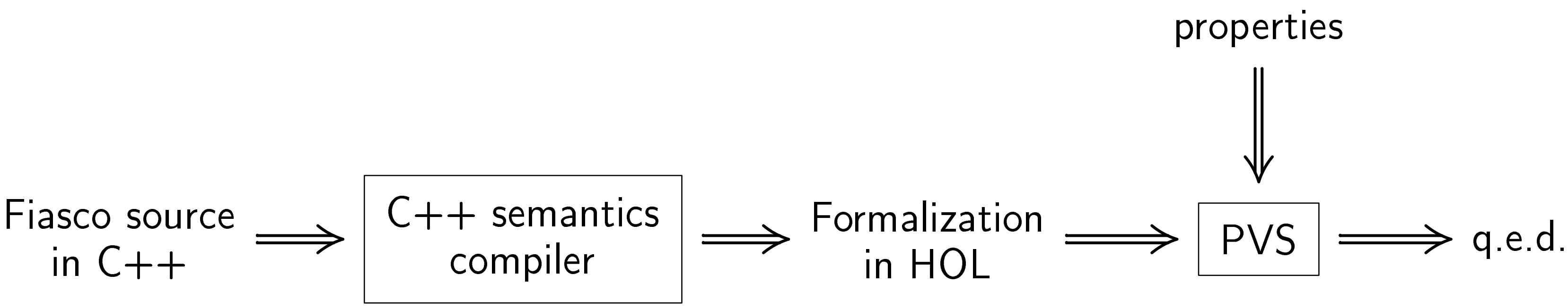
Hendrik Tews   Michael Hohmuth

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



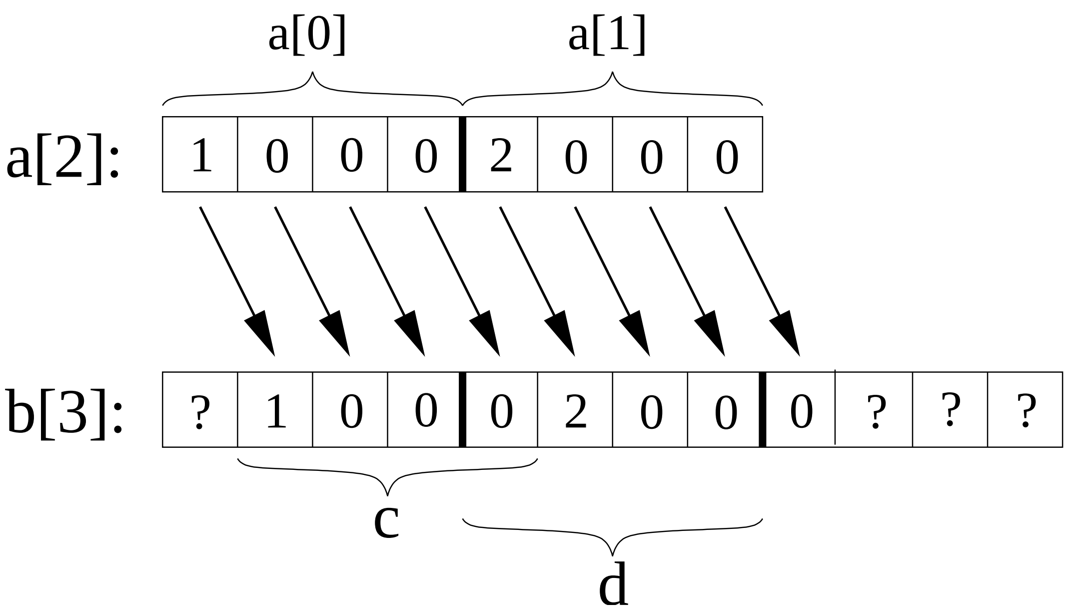
Aims for this work

- 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**

```
unsigned a[2] = { 1, 2 };
unsigned b[3];

memcpy((char *)(b) + 1, a, sizeof(a));
unsigned c = * (unsigned *) ((char *)(b) + 1);          //    √

unsigned d = b[1];                                      //    ↯
```



Verify an operating system!

- low level programs *need to bypass* the typesystem
- "dirty" tricks are important for efficiency
- operating systems are traditionally written in C or C++

. . . it is also a theoretical challenge!

C++ Standard

- talks about objects (memory representations of values)
- objects have an address, **a type**, and occupy some memory
- distiguishes between values, value representation, object representation
- reading in memory is only guaranteed to work if the memory contains a *valid* object representation

Observation

- the standard admits lisp-like runtime typechecking
- to prove type safety it is sufficient to prove normal termination on any C++ implementation

Approach for the semantics of data types

- capture only what standard guarantees
- admit different models

For more details visit the poster!