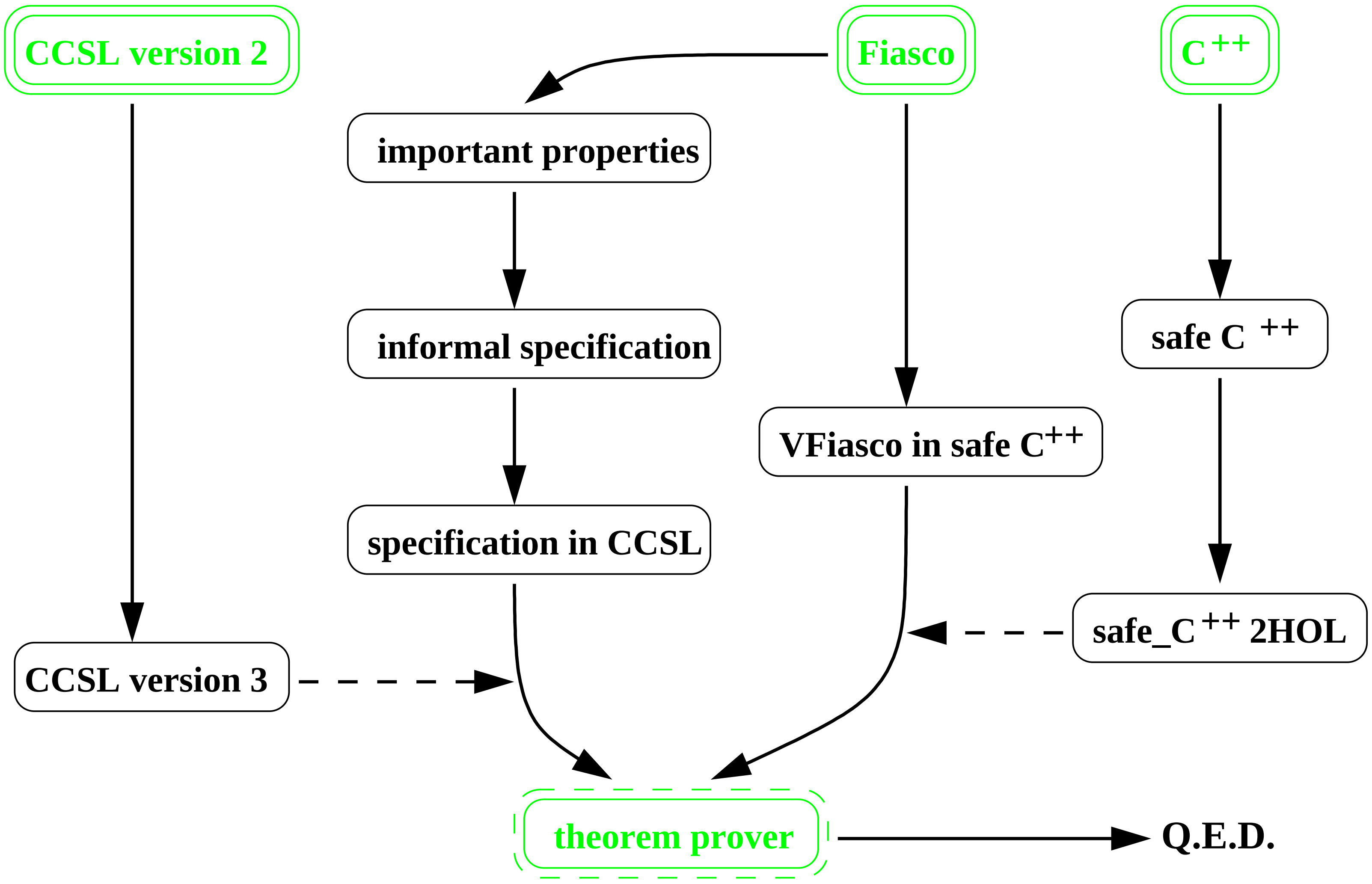
VFiasco — First Contact

Michael Hohmuth

Shane G. Stephens

Hendrik Tews

- general introduction into the project
- FAQ Specification & Verification
  - theorem provers demo
  - modelling imperative programs: state transformers
  - structured data
  - Modules


CCSL version 2
Fiasco
C++
important properties
informal specification
specification in CCSL
safe C ++
VFiasco in safe C++
safe_C ++ 2HOL
CCSL version 3
theorem prover
Q.E.D.

Problem

```
int n;
void incr_n(void) { n++; }
   ....
/* n == 2 */
incr_n();
/* n == 3 */
```

dependency on state

type state = . . .

incr_n : state ⇒ state

accomodate for failures

incr_n : state ⇒ $\mathrm{state} + \mathbf{1}$

Statements are functions

$\mathrm{state} \Rightarrow \mathrm{state} + \mathbf{1}$

Expressions

$\mathrm{state} \Rightarrow \mathrm{result} \times \mathrm{state} + \mathbf{1}$

Composition

$$
\begin{array}{lcl}
(\mathrm{stmt}_1 \; ; \; \mathrm{stmt}_2)(s) & = & \mathrm{match} \;\; \mathrm{stmt}_1(s) \;\; \mathrm{with} \\
 & & \quad \begin{array}{lcl} \mathrm{state} \;\; s' & : & \mathrm{stmt}_2(s') \\ \mathrm{Failure} & : & \mathrm{Failure} \end{array}
\end{array}
$$

state of the computer $\cong$ state of the main memory

$$\textbf{type}\ \text{memory} \quad = \quad \mathbb{N} \Rightarrow \text{byte}$$

How to model a variable?

- variables names are indices (instead of strings)

$$\textbf{type}\ \alpha\ \text{VREC} \quad = \quad \text{nat}$$

- conversion functions

$$\begin{aligned} \text{to\_value} &: \text{byte list} \Rightarrow \alpha \\ \text{from\_value} &: \alpha \Rightarrow \text{byte list} \\ \text{type\_size} &: \alpha \Rightarrow \text{nat} \end{aligned}$$

- properties

$$\begin{aligned} \textsf{persistent} &: \text{to\_value}(\text{from\_value}\ x) = x \\ \textsf{storable} &: \text{length}(\text{from\_value}\ x) = \text{type\_size}\ x \\ \textsf{constant} &: \forall x : \alpha, y : \alpha\ .\ \text{type\_size}\ x = \text{type\_size}\ y \end{aligned}$$

```
(* typed values layer *)

types 'a VREC = "nat"

consts
  to_value    :: "byte list => 'a"
  from_value  :: "'a => byte list"
  type_size   :: "'a => nat"
  example_val :: "'a VREC => 'a"

axclass type_val < "term"
  persistent: "to_value (from_value x) = x"
  storable:   "length (from_value x) = type_size x"
  constant:   "!(x::'a) (y::'a). type_size x = type_size y"
```

Modules