Checking Coding Rules in OO Languages Using CRISP

Guillem Marpons[1]

Joint work with:
Julio Mariño Manuel Carro Ángel Herranz
Lars-Åke Fredlund Juan José Moreno-Navarro

[1]Universidad Politécnica de Madrid
gmarpons@fi.upm.es

COST Action IC0701 Meeting, Madrid December 2008

Motivation: C++ "Strange" Behavior

```
class A
{
public:
  A();
  virtual void func();
};

class B : public A
{
  B() : A() {}
  virtual void func();
};
```

```
A::A() {
  func();
}

B *d = new B();

// A::func or B::func?
```

Motivation: C++ "Strange" Behavior

```
class A
{
public:
  A();
  virtual void func();
};

class B : public A
{
  B() : A() {}
  virtual void func();
};
```

```
A::A() {
  func();
}

B *d = new B();

// A::func or B::func?
```

Coding Rule:

"Do not invoke virtual methods of the declared class in a constructor or destructor."

Coding Rules

Definition

Coding Rules constrain admissible constructs of a language to help produce more reliable and maintainable code.

Standard coding rule sets do exist, e.g.:

- High-Integrity C++ (HICPP): general C++ applications
- MISRA-C (C language): automotive industry / embedded systems

Many organisations need to write their own rule sets or adapt existing ones.

Other Tools

Proprietary tools:

- Compilers: IAR Systems (C)
- QA: Parasoft, Klocwork, Coverity, Semmle Code (Java)

Free software:

- Checkstyle (Java)
- Gendarme (ECMA CIL, Mono and .Net)

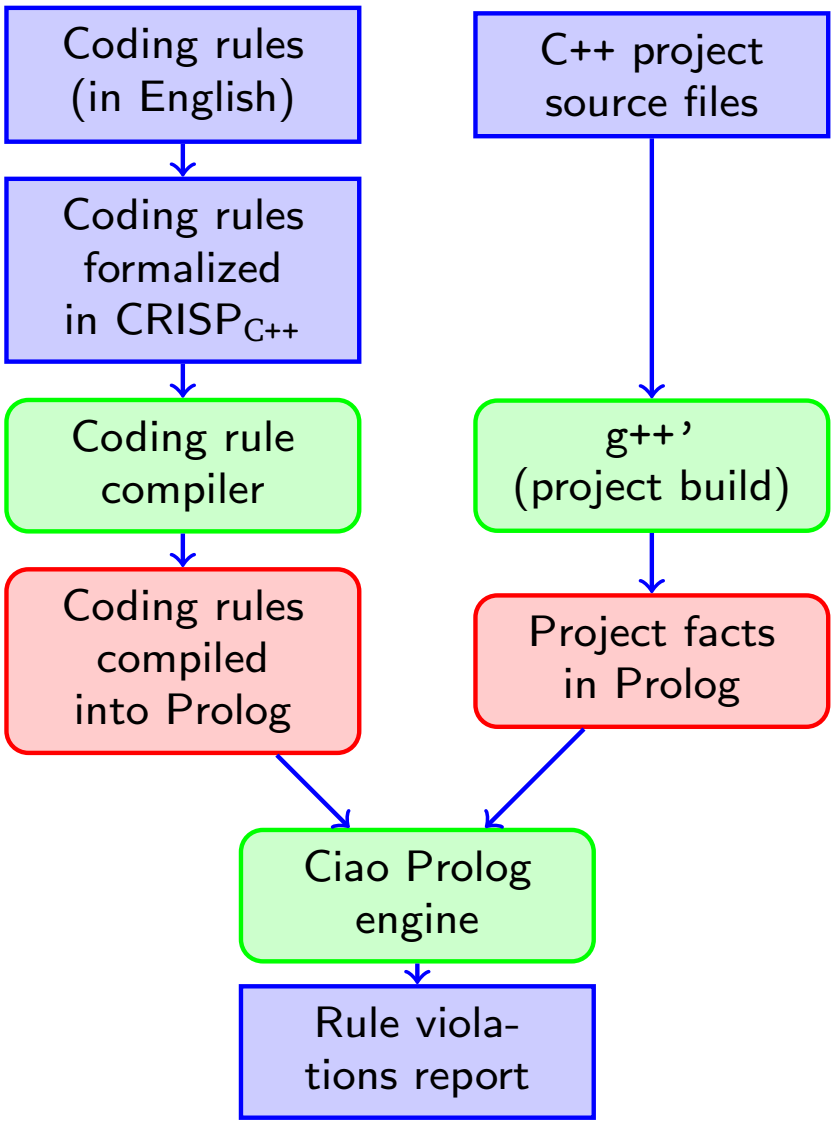
Drawbacks:

- Lack of appropriate extensibility mechanisms
- Ambiguity in natural language
- Interoperability is difficult

Proposed Approach

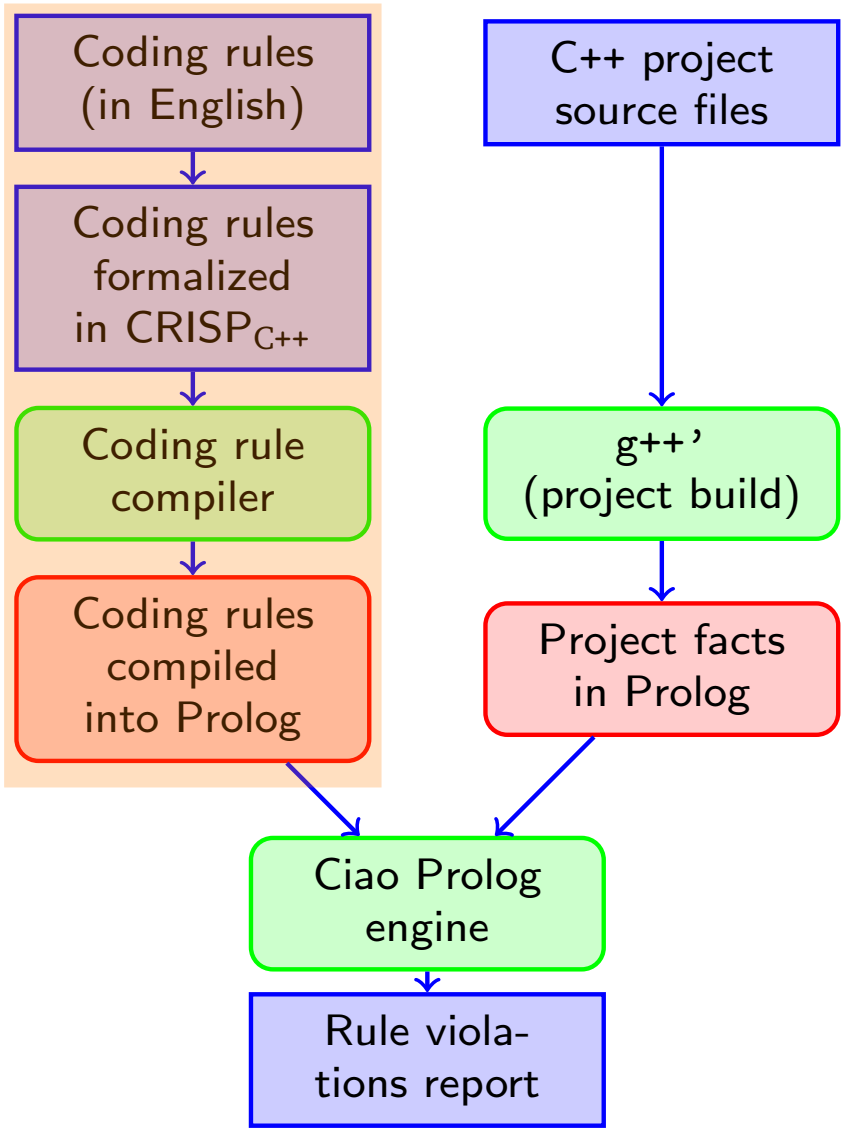
1. Formalize rules in a logic-based specification language that is executable: CRISP
2. Use GCC for gathering necessary program information

Our Rule Checking Procedure



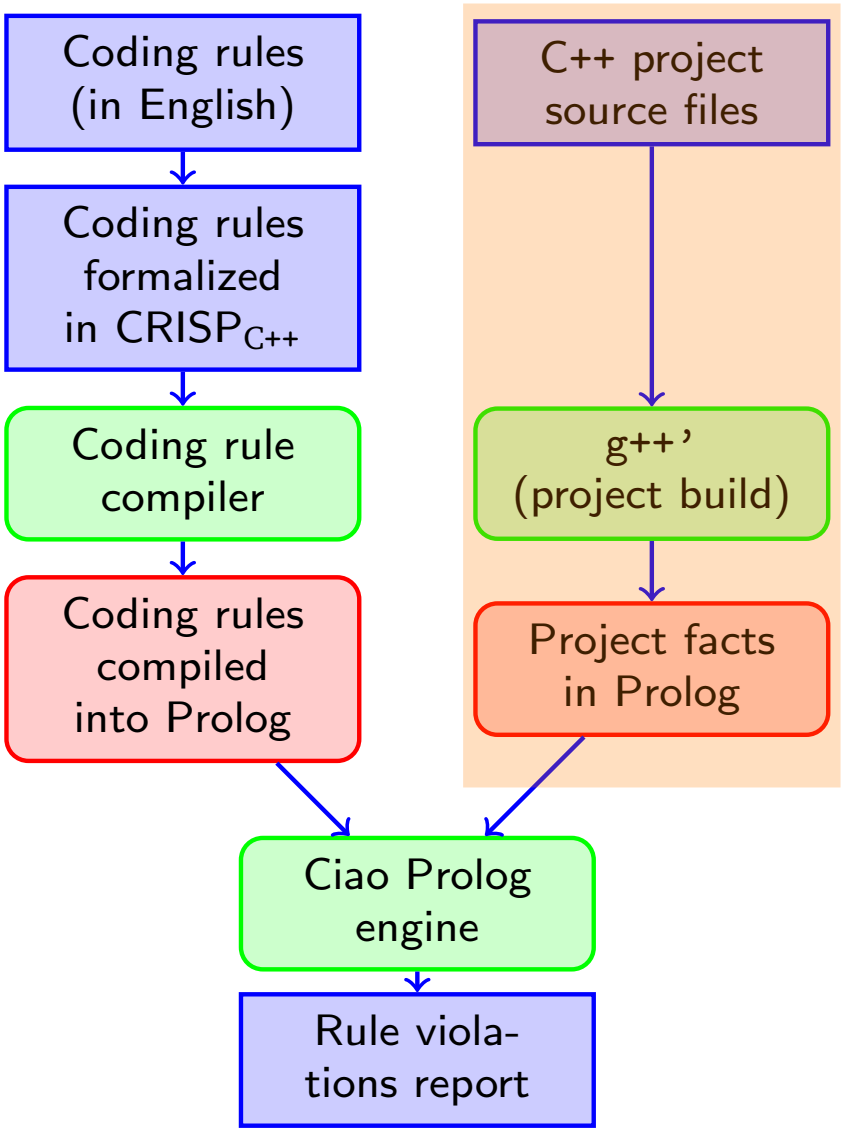
1. Coding rule(s) written **once** in the logic-based formalism
2. Extract program information (+ analysis information if available) using GCC, and store it
3. Search (using a Prolog engine) for a counterexample

Our Rule Checking Procedure



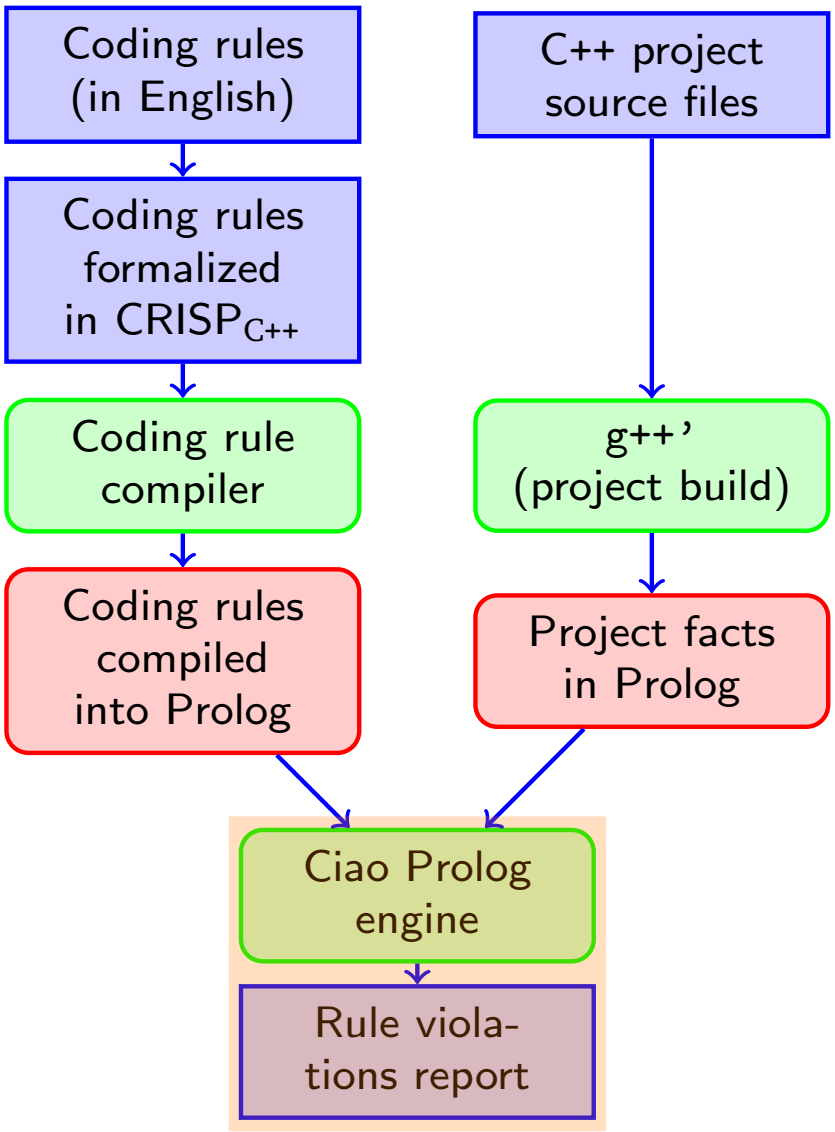
1. Coding rule(s) written **once** in the logic-based formalism
2. Extract program information (+ analysis information if available) using GCC, and store it
3. Search (using a Prolog engine) for a counterexample

Our Rule Checking Procedure



1. Coding rule(s) written **once** in the logic-based formalism
2. Extract program information (+ analysis information if available) using GCC, and store it
3. Search (using a Prolog engine) for a counterexample

Our Rule Checking Procedure



1. Coding rule(s) written **once** in the logic-based formalism
2. Extract program information (+ analysis information if available) using GCC, and store it
3. Search (using a Prolog engine) for a counterexample

CRISP Building Blocks 1: Sorts

Variable, DataMember, LocalVariable
Function, MemberFunction, Constructor
Type, PointerType, Record
Scope, Namespace, Record, CompoundStatement
Operator
ArgumentTypeInFunctionType
ClassMember
Thing

CRISP Building Blocks 2: (Binary) Relations

Function	*calls*	Function
Record	*hasImmediateBase*	Record
Variable	*hasType*	NonFunctionType
Function	*hasType*	FunctionType
Thing	*isDefinedIn*	Scope
Scope	*isNestedIn*	Scope
Record	*hasMember*	MemberFunction
Record	*hasMember*	DataMember
Record	*hasBase*	Record
Record	*isPrivateBaseOf*	Record
Record	*isVirtualBaseOf*	Record
PointerType	*hasPointedType*	Type
FunctionType	*hasReturnType*	Type
Record	*hasFriend*	Record
Record	*hasFriend*	MemberFunction
ClassMember	*hasVisibility*	Visibility

Example of Rule Formalization

Rule HICPP 3.3.13:

"Do not invoke virtual methods of the declared class in a constructor or destructor."

Example of Rule Formalization

Rule HICPP 3.3.13:

"Do not invoke virtual methods of the declared class in a constructor or destructor."

```
rule          HICPP 3.3.13'
violated by   Caller : MemberFunction; Callee : VirtualFunction
when          exists R : Record such that
              (
                    R hasMember Caller
                and R hasMember Callee
                and
                (
                    Caller is Constructor
                    or Caller is Destructor
                )
                and Caller calls+ Callee
              )
.
```

Auxiliary Sorts and Relations

```
relation    F : Function oveloads F' : Function
when        exists S : Scope ; N : String such that
            (
                    F  isDefinedIn S
                and F' isDefinedIn S
                and F  hasUnqualifiedName N
                and F' hasUnqualifiedName N
                and F \= F'
            )
.
```

```
sort        M : ClassMember is PrivateClassMember
when        exists V : Visibility such that
            (
                M hasVisibility V and V is 'private'
            )
.
```

Integration into GCC

- GlobalGCC Project (Eureka/ITEA). Enhance the GNU compiler collection with:
  - Project-wide static analysis
  - Global optimization
  - Coding rule checking
- Huge potential user base
- All facts of a project in a single file
- User interface. Two steps:
  1. `g++ -fipa-codingrules -fipa-codingrules-file=FILENAME`

     Pass appropriate `CXX` to `./configure`
  2. `checkrules -s RULE_SET [-r RULES] FILENAME`

Code available at `http://www.ggcc.info`

Experimental Results

Project	KLoc	Load Time	# Violations (Checking Time)			
			3.3.1	3.3.2	3.3.11	3.3.15
Bacula	20	0.24	0 (0.0)	3 (0.0)	0 (0.0)	0 (0.0)
CLAM	46	1.62	1 (0.0)	15 (0.5)	115 (0.1)	0 (0.2)
Firebird	439	2.61	16 (0.0)	60 (1.0)	115 (0.2)	0 (0.3)
IT++	39	0.42	0 (0.0)	6 (0.0)	12 (0.0)	0 (0.0)
OGRE	209	3.05	0 (0.0)	15 (0.9)	79 (0.2)	0 (0.3)
Orca	89	1.17	1 (0.0)	12 (0.4)	0 (0.1)	0 (0.2)
Qt	595	10.42	15 (0.0)	75 (10.5)	1155 (1.3)	4 (1.2)

All times expressed in seconds.

Work in Progress

1. Implement / Enrich the CRISP Language
2. Implement more rules with information given by other tools
3. Open our abstract representation of programs to external tools

Implement / enrich the CRISP language

- Quantification and true negation needed
  - Both performed over certain domains (sorts)
  - Infinite domains may appear with templates / generics
  - We have an implementation of constructive intensional negation
- Goals automatically reordered
- Extend CRISP to other languages: Java, Ada, C, Fortran, ...

Examples of Rules that Need Specific Analysis

Rule HICPP 3.3.13:

"Do not invoke virtual methods of the declared class in a constructor or destructor."

Rule HICPP 3.2.5:

"Ensure destructors release all objects owned by the object"

Rule HICPP 3.4.2:

"Do not return non-const handles to class data from const member functions"

Examples of Rules that Need Specific Analysis

Rule HICPP 3.3.13:

"Do not invoke virtual methods of the declared class in a constructor or destructor."

Rule HICPP 3.2.5:

"Ensure destructors release all objects owned by the object"

Rule HICPP 3.4.2:

"Do not return non-const handles to class data from const member functions"

Examples of Rules that Need Specific Analysis

Rule HICPP 3.3.13:

"Do not invoke virtual methods of the declared class in a constructor or destructor."

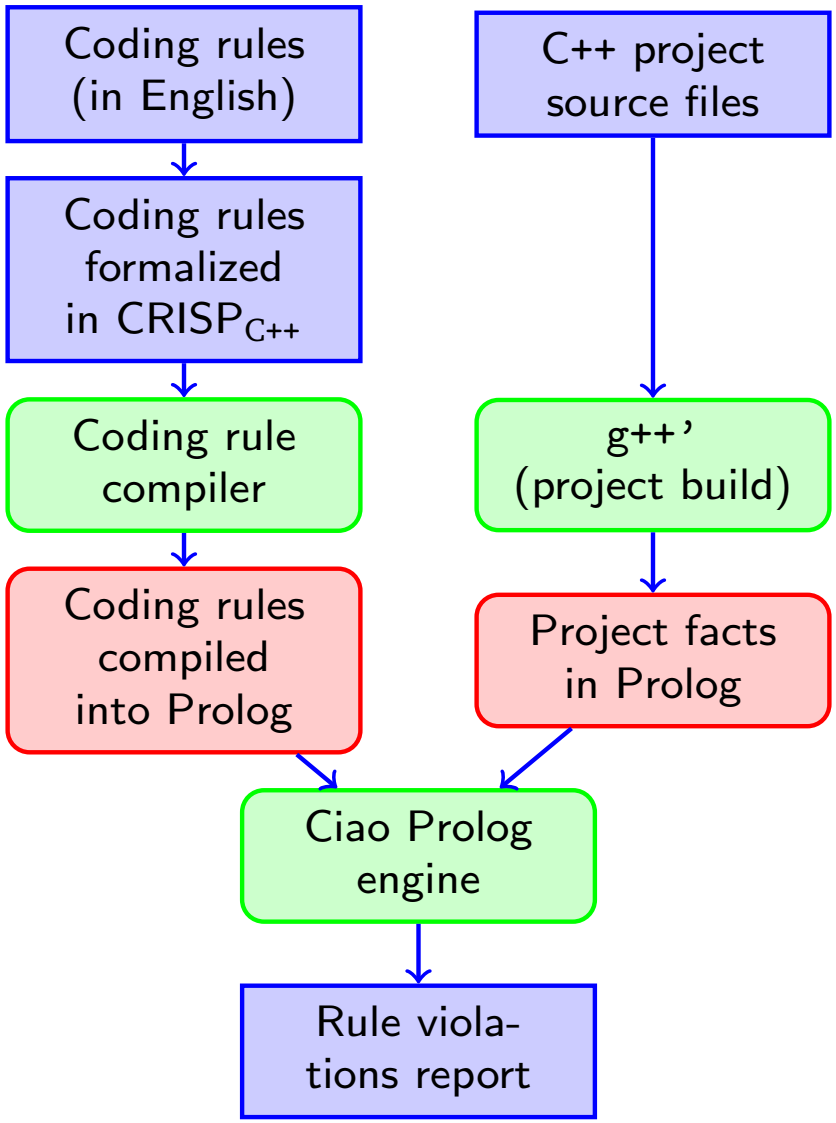
Rule HICPP 3.2.5:

"Ensure destructors release all objects owned by the object"

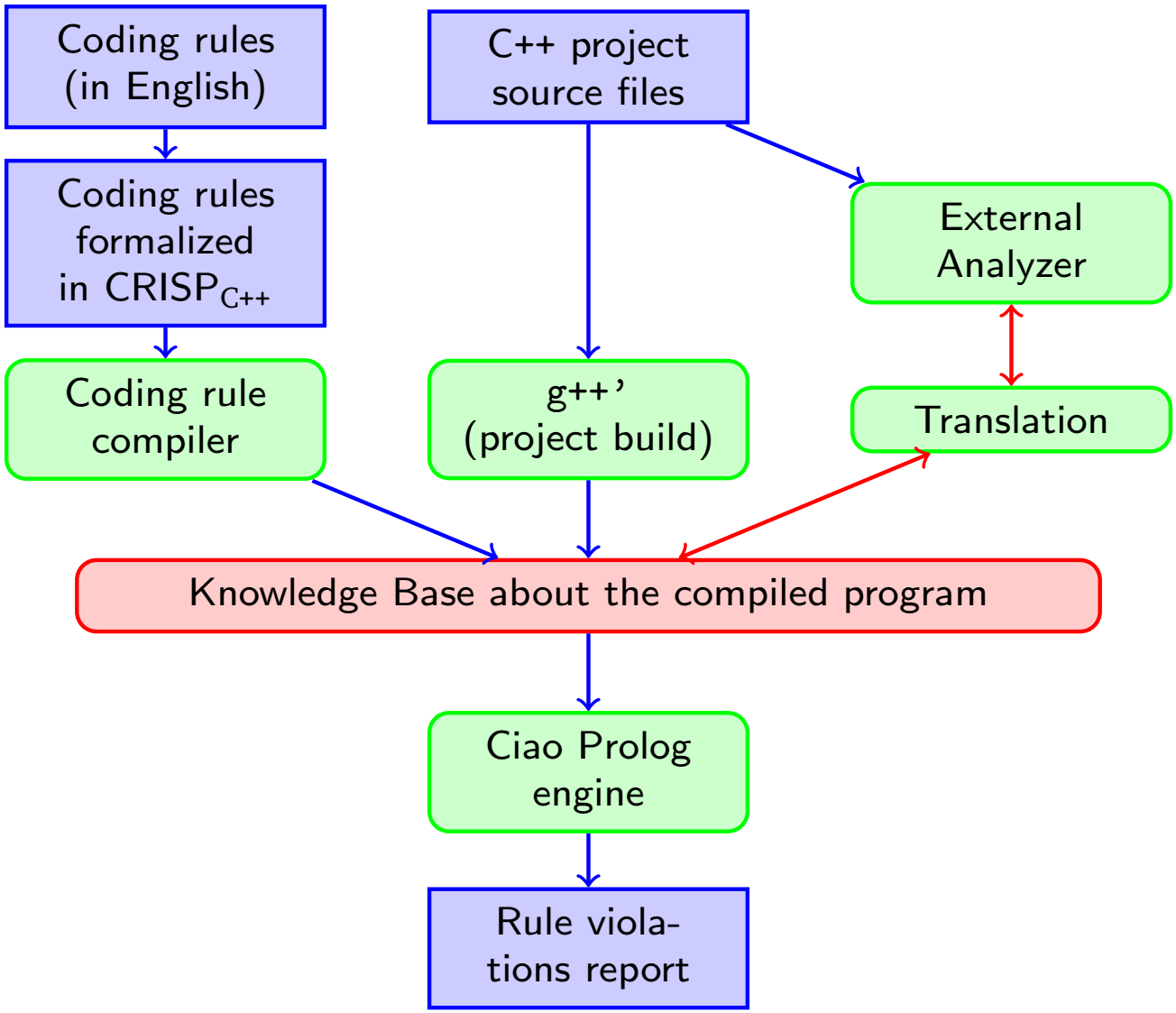
Rule HICPP 3.4.2:

"Do not return non-const handles to class data from const member functions"

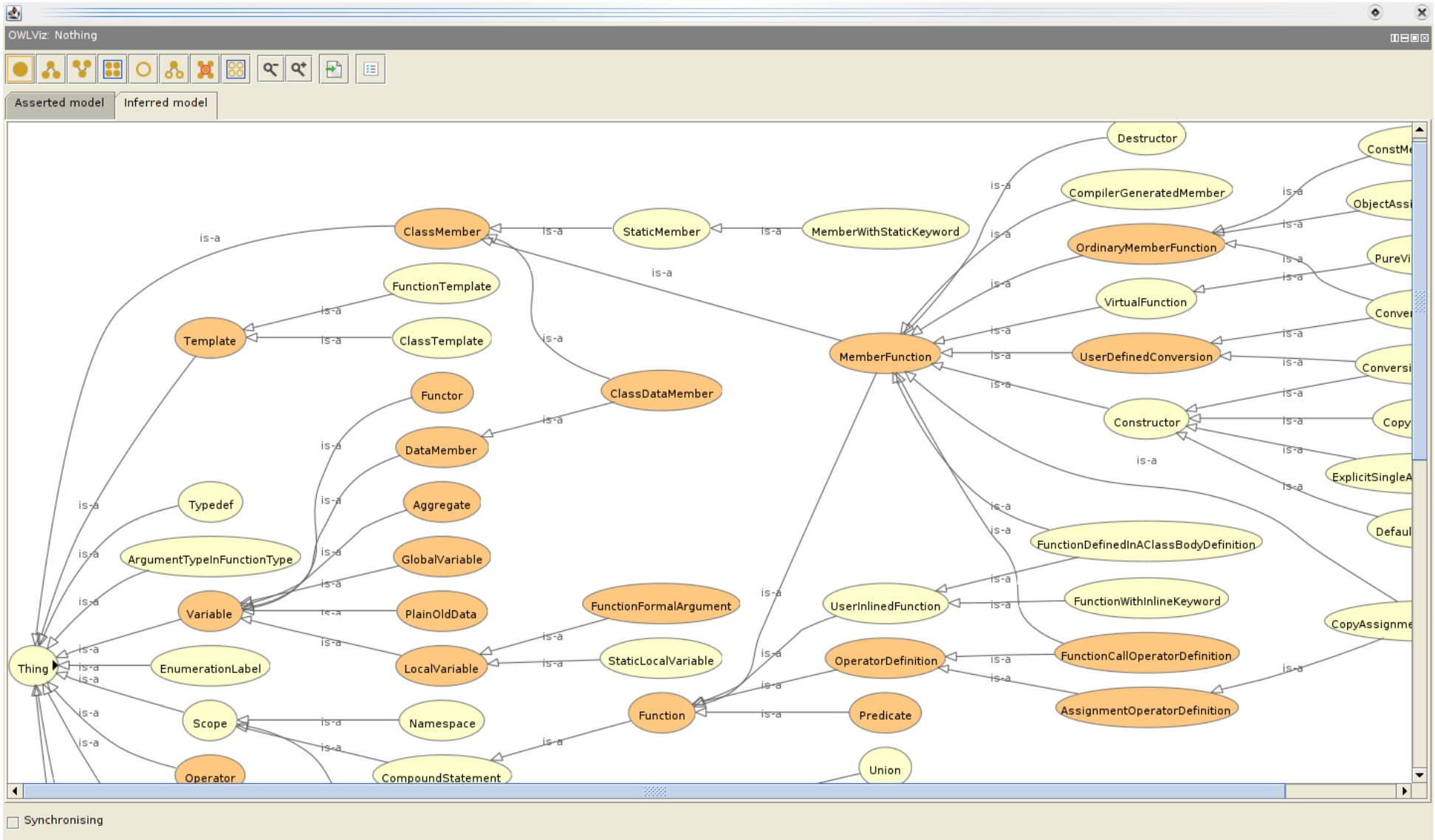
Integration of Information from External Analyzers



Integration of Information from External Analyzers



Open the Knowledge Base to External Tools



Thanks for your Attention!

- Questions?
- Comments?