Towards Checking Coding Rule Conformance Using Logic Programming

G. Marpons[1] **M. Carro**[1] **J. Mariño**[1] **A. Herranz**[1] **L. Fredlund**[1] **J. Moreno-Navarro**[1,2]
(1) Universidad Politécnica de Madrid (2) IMDEA Software

Coding Rules

Constrain admissible constructs (e.g. forbidding error-prone features or coding styles) to help producing safer code.

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

MISRA-C (C language): automotive industry standard

High-Integrity C++ (HICPP): sponsored by a private company

Javacard: addressing specific restrictions of Java Smart Cards

Enormous diversity in:

- Program features involved
- Analysis techniques required
- Static enforceability

We focus on **structural rules**, which deal with **relationships** between **static entities** in the code (classes, member functions, etc.), e.g.:

Rule HICPP 3.3.15:

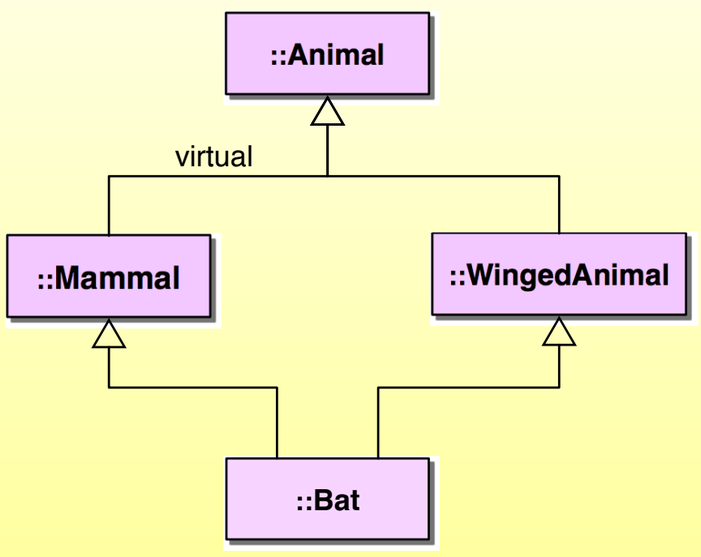
"Ensure base classes common to more than one derived class are virtual."

Natural language is **inherently ambiguous**: Which inheritance links must be tagged as "virtual"?

A framework to formalise coding rules is necessary to statically check that programs conform to a given set. We are developing such a framework in the environment of the GGCC project.

Knowledge Base About a Program

A set of classes violating rule HICPP 3.3.15:

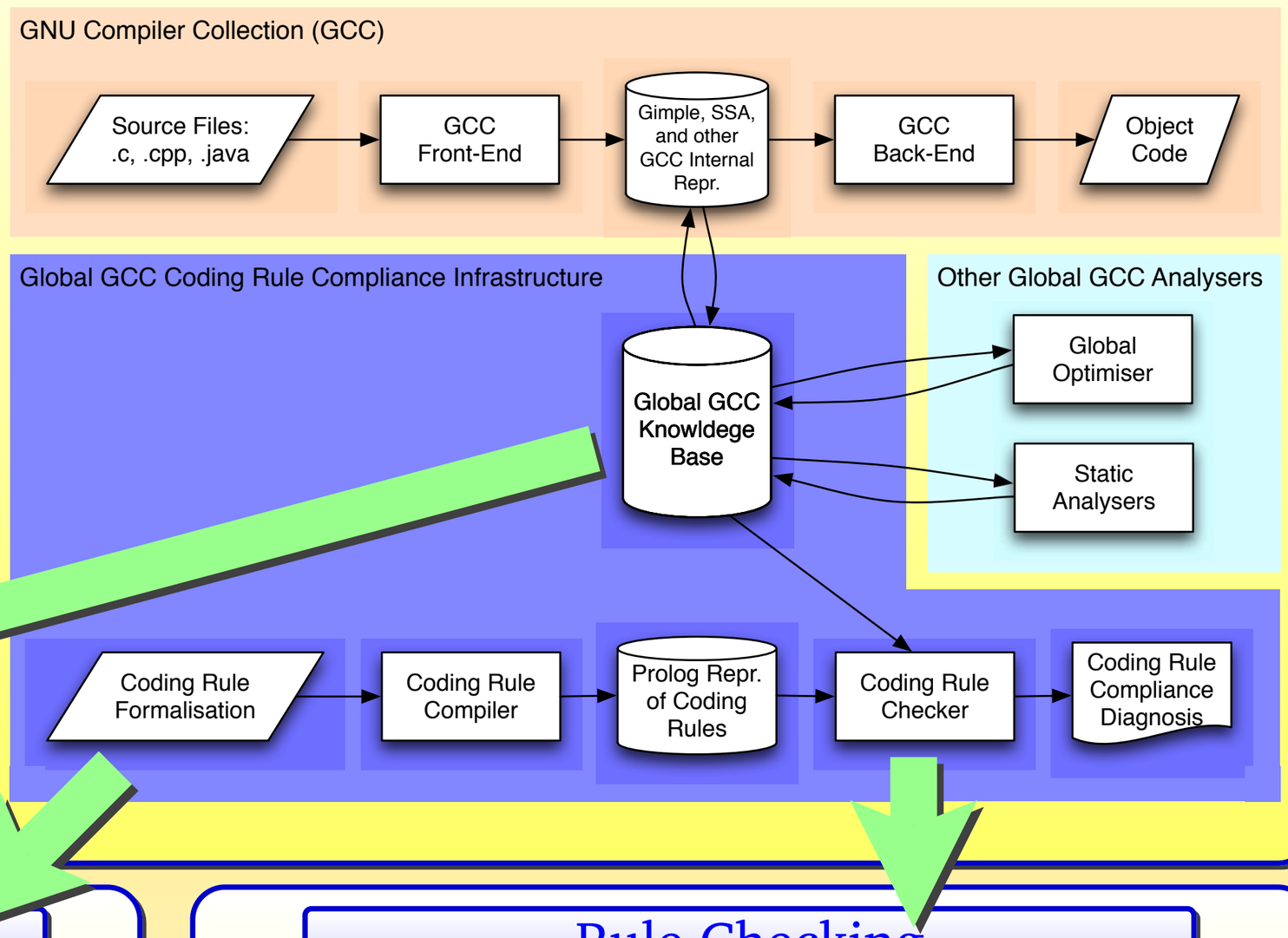


Program properties gathered during compilation for the above example and relevant to rule HICPP 3.3.15:

```
class('::Animal').
class('::Mammal').
class('::WingedAnimal').
class('::Bat').
direct_base_of('::Animal','::Mammal').
direct_base_of('::Animal','::WingedAnimal').
direct_base_of('::Mammal','::Bat').
direct_base_of('::WingedAnimal','::Bat').
virtual_base_of('::Animal','::Mammal').
```

The Global GCC Project

- ITEA funded (2006-08) consortium of industrial / research partners
- Goal: improving static analysis capabilities of the GNU Compiler Collection (GCC)
- **Global GCC knowledge base:** share information among different GGCC analysers



Rule Formalisation

Based on **first-order logic** and written in a domain-specific language which is translated into Prolog and that:

- Formalises standard coding rule sets in a declarative style
- Makes it easier for the final user to define additional coding rules
- Provides a collection of predefined predicates about program facts (such as `class/1`, `base_of/2`, or `in_call_graph_of/2`)
- Quantification over certain domains
- Constructive negation

Rule Checking

Rule HICPP 3.3.15 translated into Prolog

```
violate_hicpp_3_3_15(A, B, C, D) :-
      class(A), class(B), class(C), class(D), B \= C,
      direct_base_of(A, B), direct_base_of(A, C),
      base_of(B, D), base_of(C, D),
      \+ virtual_base_of(A, C).
```

We do not code the rule itself, but **its negation**. Any program that satisfies the negated rule thus violates the coding rule.

Predicates coding rule violations are queried against facts describing a program. A successful resolution flags a rule violation, providing a witness.

Work partially supported by PROFIT grants FIT-340005-2007-7 and FIT-350400-2006-44 from the Spanish Ministry of Industry, Comunidad Autónoma de Madrid grant S-0505/TIC/0407 (PROMESAS), Ministry of Education and Science grant TIN2005-09207-C03-01 (MERIT/COMVERS) and EU IST FET grant IST-15905 (MOBIUS).