Frama-Clang, a C++ front-end for Frama-C

Frama-C Day

Virgile Prevosto

joint work with Franck Védrine

June 20th, 2016

Context

Stance Project

- `http://stance-project.eu/`
- Security analyzes with (among others) Frama-C
- C++ case studies and need for a C++ front-end

Others

- Developments with Trust-in-Soft inside the common lab
- Some industrial case studies (Bureau Veritas)
- Quite widespread interest for C++ analyzes with Frama-C

Summary

Introduction

Frama-Clang Basics

Evolved Features

C++ Standard Library

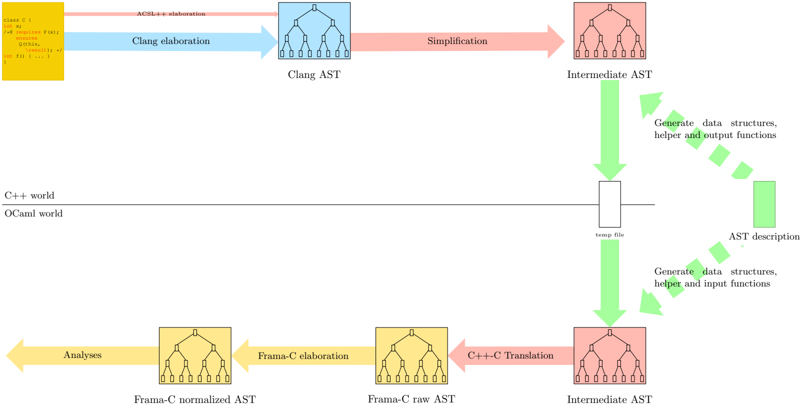
Conclusion

A Clang-based parser

Clang

- C/C++/Objective-C front-end to LLVM
- `http://clang.llvm.org/`
- Very good C++11 coverage
- Very good API (they even have comments)
- Quite easy to extend (especially for introducing ACSL++)

Ideal Toolchain Overview



```
class C {
int x;
/*@ requires P(x);
    ensures
     Q(this,
       \result); */
int f() { ... }
}
```

Expressions and Statements

Done by Clang

- Overloading resolution
- As well as `auto` pointers

Translation

- Translation mostly identity
- References are pointers (valid by constructions)
- Insertion of temporary variables with their default constructor...
- ... and destructor (still TODO)
- Explicit notion of `constexpr`

Base classes

Done by Clang

- checks for visibility (`private`, `deleted`)

Translation

- `struct` with a set of functions
- Generation of special methods (default constructor, copy, destructor)
- Accounting for `this` pointer in non-static methods

Inheritance

Single inheritance

- Base class is simply another field of the enclosing `struct`
- field access must take that into account
- cast/call of method of base class must use the appropriate field
- generate base constructor calls as needed.

Multiple Inheritance

- multiple sub-structures
- choose appropriate field depending on the context

Templates

Instantiation

- instantiations and specializations are done by Clang
- Frama-Clang translates only instantiations
- Similar to normal C++ code

Templates

Instantiation

- instantiations and specializations are done by Clang
- Frama-Clang translates only instantiations
- Similar to normal C++ code

Reordering

- Instantiated nodes are visited at template declaration
- Potentially before actual arguments
- Need to reorder intermediate AST nodes to obtain well-formed AST

ACSL++ Specifications

Main constructions

- Function contracts
- Assertions and loop annotations
- Definition of predicate and logic functions
- Same namespace rules as for C++ definitions

Terms

- Same as in ACSL
- symbols are qualified in the same way as C++ symbols
- no `private`/`public` distinction

Expanding Macros in annotations

- Access to Clang preprocessor's internal structures
- Expansion done during lexing phase of ACSL++ annotations
- Same restrictions as with expansion of macros in ACSL:
  - Only last value of a macro is used
  - "Clever" sequences of **#define** and **#undef** might not work as intended

Name mangling

- A unambiguous and valid C name for any global C++ symbol
- Use Itanium ABI
- Example: assignment operator of class `A`: `_ZN1AEaSO1A`

Name mangling

- A unambiguous and valid C name for any global C++ symbol
- Use Itanium ABI
- Example: assignment operator of class `A`: `_ZN1AEaSO1A`
- `frama-c -print`: **operator**=

Name mangling

- A unambiguous and valid C name for any global C++ symbol
- Use Itanium ABI
- Example: assignment operator of class `A`: `_ZN1AEaSO1A`
- `frama-c -print`: **operator**=
- `frama-c -cxx-demangling-full`: `A::`**operator**=

Name mangling

- A unambiguous and valid C name for any global C++ symbol
- Use Itanium ABI
- Example: assignment operator of class `A`: `_ZN1AEaSO1A`
- `frama-c -print`: **`operator`**`=`
- `frama-c -cxx-demangling-full`: `A::`**`operator`**`=`
- `frama-c -cxx-keep-mangling`: `_ZN1AEaSO1A`

Name mangling

- A unambiguous and valid C name for any global C++ symbol
- Use Itanium ABI
- Example: assignment operator of class `A`: `_ZN1AEaSO1A`
- `frama-c -print`: **`operator`**`=`
- `frama-c -cxx-demangling-full`: `A::`**`operator`**`=`
- `frama-c -cxx-keep-mangling`: `_ZN1AEaSO1A`
- Demangling also occurs in messages

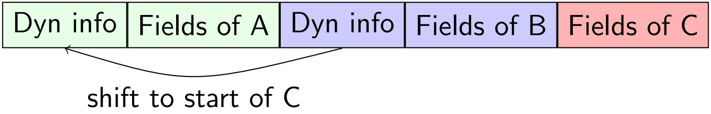
Name mangling

- A unambiguous and valid C name for any global C++ symbol
- Use Itanium ABI
- Example: assignment operator of class `A`: `_ZN1AEaSO1A`
- `frama-c -print`: **`operator`**`=`
- `frama-c -cxx-demangling-full`: `A::`**`operator`**`=`
- `frama-c -cxx-keep-mangling`: `_ZN1AEaSO1A`
- Demangling also occurs in messages
- Also works for giving function names as options:

```
frama-c -slevel-function A::f:10 -main A::g
```

Virtual functions

- Represented as function pointer
- Virtual method table for each class
- Corresponding field in the **struct**
- Possibly need to shift the **this** pointer

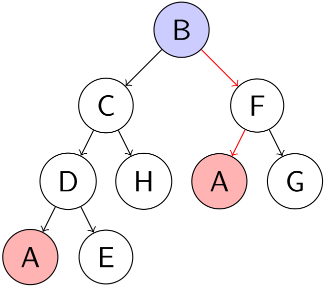


Dyn info | Fields of A | Dyn info | Fields of B | Fields of C

shift to start of C

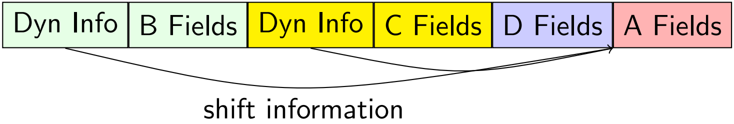
Dynamic Casts

- `dynamic_cast<A*>(b)`
- Find a path in the inheritance graph between dynamic type of `b` and `A`
- Notion of distance in case multiple choices are possible
- Keeping runtime type information
- Graph traversal algorithm



Virtual Base Classes

- Only one copy of each **`virtual`** class in a given object
- Put after the non-virtual fields
- Again, some shifts are required



Dyn Info | B Fields | Dyn Info | C Fields | D Fields | A Fields

shift information

Exceptions at C Level

- Introduction of new nodes `Throw` and `TryCatch` in Cil AST
- Code transformation in Frama-C kernel to generate pristine C AST
- Translation from C++ to C straightforward
- Except for a whole-program pass to deal with inheritance

Library Usage

```
#include <iostream>
```

- C++ Standard Library is large (700 pages in C++11 standard)
- Widely used by programs, including embedded ones
- Heavily templated
- Contains many non-trivial constructions
  - `#include <memory>`
  - `#include <functional>`

Using System library

Pros and Cons

✔ Readily available and complete

✘ Often contains compiler-specific features

✘ Using another `-machdep` amounts to cross-compilation

✘ No ACSL++ annotation in the library

Current situation

- primary target: GNU libc++
- Supports `<iostream>` (including dependencies)
- Supports `<vector>` and `<map>` (including dependencies)
- Other components to be considered depending on case studies

Frama-Clang's specific headers

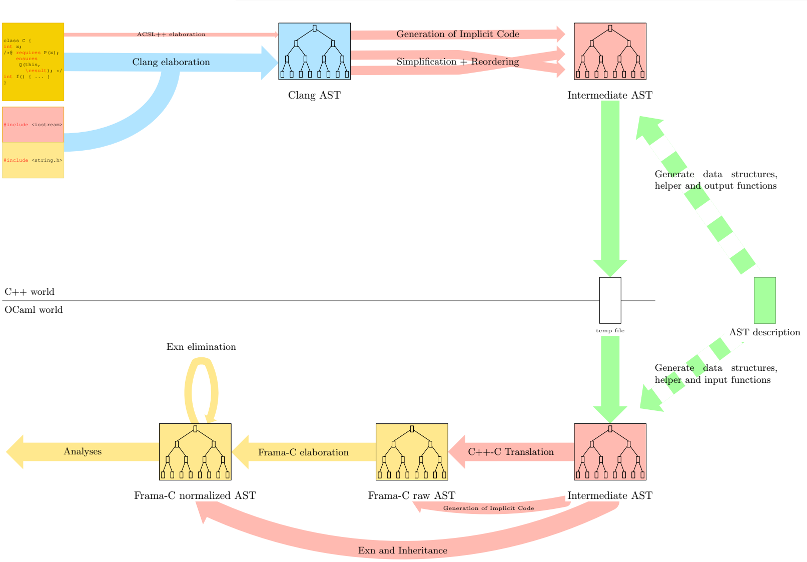
Why reinventing the (squared) wheel?

- No need to handle compiler built-ins or other non-standard features
- Make Frama-C's own built-ins accessible
- Better interaction with Frama-C C library (`machdep.h`)
- Provide ACSL++ annotations

Current status

- Adaptations required at C std lib level
- 10 wrappers over C headers (and their ACSL annotations)
- 15 pure C++ header files (sometimes incomplete)
- Support for `iostream` and dependencies

Real Toolchain



```
class C {
int x;
/*@ requires P(x);
    ensures
     Q(this,
       \result); */
int f() { ... }
}
```

```
#include <iostream>
```

```
#include <string.h>
```

Current State

Development size

Intermediate AST generator	1000
Generated C code	*18000*
Generated OCaml code	*7500*
C++ code	41000
ACSL++ handler	*20000*
OCaml code	7000
STL Headers	2000
Total	51000

Current State

Main results

- Covers major C++ features
- Support for basic ACSL++ constructions
- Partial support for Standard Library
- Still requires a lot of polishing (with the help of TiS and FOKUS)
- Able to analyze real-world programs of moderate size (~1kLoC)

Release

- In the coming days
- LGPL licence
- Compatible with Clang 3.8.0 (and Frama-C Aluminium)
- For adventurous users only!

Next steps for translation

- Complete handling of virtual inheritance
- Polish existing features
- Extend STL support (and add annotations)
- Take into account more C++11 features (e.g. Function objects)

Next steps for analyzers

Under design

- Abstract away (with ACSL predicates and contracts) operations dealing with inheritance through clever pointer arithmetic (selection of virtual member function, dynamic cast, ...)
- Enhance WP's handling of indirect calls

Longer Term

- Propose specific lattices in EVAlhalla for tracing inheritance
- Modular analysis over templates instead of instances