ADDING SUPPORT FOR C++ CONTRACTS TO CLANG

…and some thoughts around their application

Javier López-Gómez

8th April 2019

Computer Science and Engineering Department,
University Carlos III of Madrid

Who am I?

- A CS PhD. student (Computer Architecture and Technology Area)
- Spent some time hacking the Linux kernel, embedded software, electronics... (low-level stuff!)
- Now: working on Clang for the last year

Agenda

Agenda

In C++17…

- Compile-time: `static_assert(…)`
- Run-time: C89 `assert(…)`
- …or other (non-standard) user-defined macro/function.

But `assert(…)` is a macro which expands to nothing for a production build.
This might be improved!

But in C++20 we might have...

Declaration (probably in a header file):

```
int f(int x)
  [[expects default: x > 0]]        // low-cost precondition
  [[expects audit: sanity_chk(x)]] // high computational cost
  [[ensures ret: ret > 0]];        // postcondition
```

Definition (in `.cpp` file):

```
int f(int x) {
  ...
}
```

Agenda

The C++ contract TS (P0542R5) (1/5)

P0542R5: a proposal to support contracts in C++.

Contract: the set of preconditions, postconditions and assertions associated to a function.

- Precondition: What are the *expectations* of the function? —Evaluated at function entry
  `[[expects: …]]`
- Postconditions: What must the function *ensure* upon termination? —Evaluated at function exit
  `[[ensures: …]]`
- Assertion:

The C++ contract TS (P0542R5) (1/5)

P0542R5: a proposal to support contracts in C++.

Contract: the set of preconditions, postconditions and assertions associated to a function.

- Precondition: What are the *expectations* of the function? —Evaluated at function entry
  `[[expects: …]]`
- Postconditions: What must the function *ensure* upon termination? —Evaluated at function exit
  `[[ensures: …]]`
- Assertion:

The C++ contract TS (P0542R5) (1/5)

P0542R5: a proposal to support contracts in C++.

Contract: the set of preconditions, postconditions and assertions associated to a function.

- Precondition: What are the *expectations* of the function? —Evaluated at function entry
  `[[expects: …]]`
- Postconditions: What must the function *ensure* upon termination? —Evaluated at function exit
  `[[ensures: …]]`
- Assertion:

The C++ contract TS (P0542R5) (1/5)

P0542R5: a proposal to support contracts in C++.

Contract: the set of preconditions, postconditions and assertions associated to a function.

- Precondition: What are the *expectations* of the function? —Evaluated at function entry
  `[[expects: …]]`
- Postconditions: What must the function *ensure* upon termination? —Evaluated at function exit
  `[[ensures: …]]`
- Assertion:

The C++ contract TS (P0542R5) (1/5)

P0542R5: a proposal to support contracts in C++.

Contract: the set of preconditions, postconditions and assertions associated to a function.

- Precondition: What are the *expectations* of the function? —Evaluated at function entry
  `[[expects: …]]`
- Postconditions: What must the function *ensure* upon termination? —Evaluated at function exit
  `[[ensures: …]]`
- Assertion: Do I need to define this?

The C++ contract TS (P0542R5) (1/5)

P0542R5: a proposal to support contracts in C++.

Contract: the set of preconditions, postconditions and assertions associated to a function.

- Precondition: What are the *expectations* of the function? —Evaluated at function entry
  `[[expects: …]]`
- Postconditions: What must the function *ensure* upon termination? —Evaluated at function exit
  `[[ensures: …]]`
- Assertion: ~~Do I need to define this?~~ A predicate that should hold at a specific location of the function body.
  `[[assert: …]]`

The C++ contract TS (P0542R5) (2/5)

You can include an assertion level `[[assert HERE: …]]`…

axiom. Not evaluated at run-time (useful for static analysis/optimizer).

default/audit. Indicate the relative computational cost of the checks.

A translation is carried out in a specific build level (off, default, audit).

The C++ contract TS (P0542R5) (3/5)

`ensures`-only: an identifier may be introduced

```
[[ensures default HERE: …]]
```

and can be used to refer to the return value of the function.

The C++ contract TS (P0542R5) (4/5)

- By default, a violated contract invokes `std::terminate()`.
- Alternatively, the user can specify a handler (per-translation). `std::terminate()` may optionally be called after return.

```
void (const std::contract_violation &); // the type of a handler

class contract_violation {
public:
  int line_number() const noexcept;
  string_view file_name() const noexcept;
  string_view function_name() const noexcept;
  string_view comment() const noexcept;
  string_view assertion_level() const noexcept;
};
```

The C++ contract TS (P0542R5) (4/5)

- By default, a violated contract invokes `std::terminate()`.
- Alternatively, the user can specify a handler (per-translation). `std::terminate()` may optionally be called after return.

```
void (const std::contract_violation &); // the type of a handler

class contract_violation {
public:
  int line_number() const noexcept;
  string_view file_name() const noexcept;
  string_view function_name() const noexcept;
  string_view comment() const noexcept;
  string_view assertion_level() const noexcept;
};
```

The C++ contract TS (P0542R5) (5/5)

A contract…

- …has no observable effect on a correct program (except performance): UB if side-effects.
- …might be a convenient to provide additional information to the optimizer/3*rd*-party libraries.

Agenda

Required changes to the Clang FE (1/2)

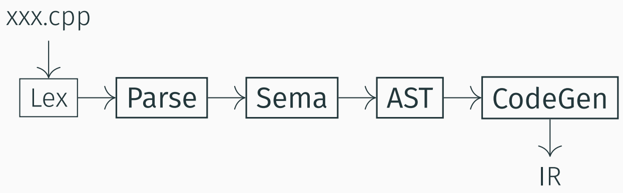

Figure 1: Patched Clang components

Parse. Updated due to the proposed grammar changes for contract attributes.
Sema. Most of the code is here (Decl injection, merging attributes, instantiation, etc.)
AST. Small changes to the *ASTContext* and *FunctionDecl* classes.
CodeGen. Run-time checks code generation.

Required changes to the Clang FE (2/2)

- *(1)*: copy the `f` FunctionDecl; the copy (`g`) owns the original body of `f` will be forced inline.
- *(2)*: body of `f` replaced (synthesized): evaluates pre-conditions + calls `g` + evaluates post-conditions.

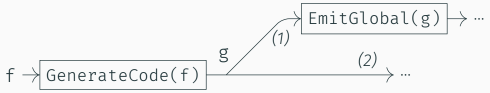

Figure 2: CodeGen for functions that have pre/post-conditions

Required changes to the Clang FE (3/3)

```
int f(int x)
  [[expects: x==2]];
...

int f(int x) {
  return x;
}
```

```
define i32 @_Z1fi ( i32 returned %x)
local_unnamed_addr #0 {
entry:
  % cmp = icmp eq i32 %x, 2
  br i1 %cmp, label %if.end,
        label %if.then
if.then:
  tail call void
        @_ZSt9terminatev() #2
  unreachable
if.end:
  ret i32 2
}
```

Applying the “p1290r0” fix

ISSUE: Assuming contracts that were not checked was a source of UB.

FIX: Do not assume unchecked contracts (except `axiom` (depending on the “axiom mode”)
Added the `-axiom-mode=` command line option.

Evaluation (1/2)

What? GNU libstdc++ `std::basic_string`

How? Replaced the `__glibcxx_assert` macro by `[[assert: …]]` or `[[expects: …]]` and compared the run-time overhead (10000 iterations).

Evaluation (2/2)

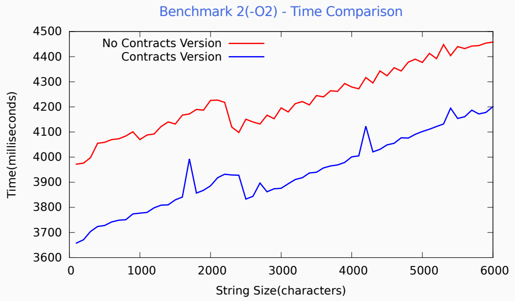

Figure 4: Swap characters (-O2)

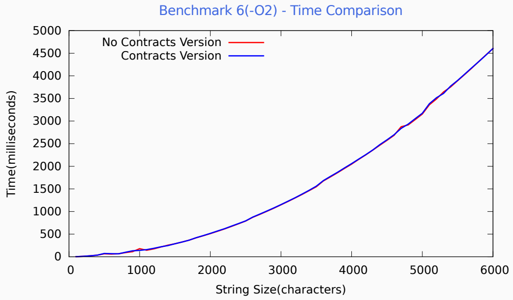

Figure 5: Find and replace 3-char substring in a random string (-O2)

DEMO: a P0542R5-enabled Clang

Try it: `http://fragata.arcos.inf.uc3m.es/`

Open-sourced (GitHub)[1]:

`https://github.com/arcosuc3m/clang-contracts/`

[1] To be rebased on top of the current development branch and submitted for code review.

But wait, that's not all!

C++ contracts may also be used as annotations for static analyzers (`axiom`) or to interface third party libraries.

To prove this point, we built something on top of this…

Agenda

ISSUE: TSan and lock-free data structures

ISSUE: *ThreadSanitizer* reports false positives using a Boost lock-free SPSC queue (only 1 producer + 1 consumer).

FIX: extend *ThreadSanitizer* to honour user-defined data structure semantics (that use C++ contracts).

CSV: a TSan extension (1/2)

Listing 1: Updated “boost/lockfree/spsc_queue.hpp” to use CSV

```
#include "csv.h"

template <typename T, typename... Options>
class [[csv::checked]] spsc_queue {
private:
  [[csv::event_sets(init_events, prod_events, cons_events, nts_events)]];
public:
  …
  bool push(T const &)
      [[expects audit: !init_events.empty()
          && init_events.happens_before(csv::current_event())]]
      [[expects audit: !prod_events.concurrent(csv::current_event())]]
      [[csv::add_current(prod_events)]];
  …
};
```

CSV: a TSan extension (2/2)

```
====================
WARNING: CSV: rule violation at …/spsc_queue.hpp:854
  `!prod_events.concurrent(csv::current_event())`

  Stack trace:
  #0 __csv_violation_handler /home/…/tsan/rtl/tsan_csv.cc:45
(+0x4915f0)
  #1 boost::lockfree::spsc_queue<T>::push(T) <null>
(+0x4b8f99)
  …
```

Figure 6: If a rule is violated the user gets a descriptive trace

Open Source: TSan—CSV patches

CSV is maintained as a branch (`CSV-src`) at the clang-contracts repository:

`https://github.com/arcosuc3m/clang-contracts/`

Agenda

Conclusions

Support for contract-checking in C++...

- Helps to detect more programming errors (improves correctness).
- Run-time checking can be enabled/disabled (Safety—Run-time overhead).
- Q: Can I throw an exception/log violations?
  A: Use a violation handler!
- Portable and standard way of providing information to the optimizer/third party libraries.

Few issues to be fixed: P0542R5 *Sec. 2.3*, late-parsing, and contract inheritance.

Conclusions

Support for contract-checking in C++...

- Helps to detect more programming errors (improves correctness).
- Run-time checking can be enabled/disabled (Safety—Run-time overhead).
- Q: Can I throw an exception/log violations?
  A: Use a violation handler!
- Portable and standard way of providing information to the optimizer/third party libraries.

Few issues to be fixed: P0542R5 *Sec. 2.3*, late-parsing, and contract inheritance.

Conclusions

Support for contract-checking in C++...

- Helps to detect more programming errors (improves correctness).
- Run-time checking can be enabled/disabled (Safety—Run-time overhead).
- Q: Can I throw an exception/log violations?
  A: Use a violation handler!
- Portable and standard way of providing information to the optimizer/third party libraries.

Few issues to be fixed: P0542R5 *Sec. 2.3*, late-parsing, and contract inheritance.

Conclusions

Support for contract-checking in C++...

- Helps to detect more programming errors (improves correctness).
- Run-time checking can be enabled/disabled (Safety—Run-time overhead).
- Q: Can I throw an exception/log violations?
  A: Use a violation handler!
- Portable and standard way of providing information to the optimizer/third party libraries.

Few issues to be fixed: P0542R5 *Sec. 2.3*, late-parsing, and contract inheritance.

Conclusions

Support for contract-checking in C++...

- Helps to detect more programming errors (improves correctness).
- Run-time checking can be enabled/disabled (Safety—Run-time overhead).
- Q: Can I throw an exception/log violations?
  A: Use a violation handler!
- Portable and standard way of providing information to the optimizer/third party libraries.

Few issues to be fixed: P0542R5 *Sec. 2.3*, late-parsing, and contract inheritance.

Thanks!

2019 / BRUSSELS
LLVM EUROPEAN DEVELOPER'S MEETING

¿?

Thank you for listening!

`https://github.com/arcosuc3m/clang-contracts/`