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

1. Introduction
2. Background
3. Supporting the P0542R5 TS in Clang
4. CSV: an extension to TSan that uses contracts
5. Conclusion
1. Introduction
2. Background
3. Supporting the P0542R5 TS in Clang
4. CSV: an extension to TSan that uses contracts
5. Conclusion
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):

```cpp
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):

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

1. Introduction
2. Background
3. Supporting the P0542R5 TS in Clang
4. CSV: an extension to TSan that uses contracts
5. Conclusion
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
  
  ```cpp
  [[expects: ...]]
  ```

- **Postconditions**: What must the function ensure upon termination? —Evaluated at function exit
  
  ```cpp
  [[ensures: ...]]
  ```

- **Assertion**: 
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
  
  \[\text{[[expects: ...]]}\]

- **Postconditions**: What must the function ensure upon termination? —Evaluated at function exit
  
  \[\text{[[ensures: ...]]}\]

- **Assertion**:
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:**
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**: 
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?
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: ...]]
  ```
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).
ensures-only: an identifier may be introduced

[[ensures default HERE: ...]]

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

```cpp
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;
};
```
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.

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

class contractViolation {
public:
    int lineNumber() const noexcept;
    string_view fileName() const noexcept;
    string_view functionName() const noexcept;
    string_view comment() const noexcept;
    string_view assertionLevel() const noexcept;
};
```
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/3rd-party libraries.
<table>
<thead>
<tr>
<th></th>
<th>Agenda</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Background</td>
</tr>
<tr>
<td>3</td>
<td>Supporting the P0542R5 TS in Clang</td>
</tr>
<tr>
<td>4</td>
<td>CSV: an extension to TSan that uses contracts</td>
</tr>
<tr>
<td>5</td>
<td>Conclusion</td>
</tr>
</tbody>
</table>
Required changes to the Clang FE (1/2)

**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.
- (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
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
}
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.
What? GNU libstdc++ `std::basic_string`

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

¹To 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

1. Introduction
2. Background
3. Supporting the P0542R5 TS in Clang
4. CSV: an extension to TSan that uses contracts
5. Conclusion
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).
Listing 1: Updated “boost/lockfree/spsc_queue.hpp” to use CSV

```cpp
#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)]];

    ...
};
```
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
CSV is maintained as a branch (CSV-src) at the clang-contracts repository: https://github.com/arcosuc3m/clang-contracts/
Agenda

1. Introduction
2. Background
3. Supporting the P0542R5 TS in Clang
4. CSV: an extension to TSan that uses contracts
5. Conclusion
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.
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.
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!

Thank you for listening!

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