Response to Core Safety Profiles (P3081)

Document #: P3543R0
Date: 2024-12-17
Project: Programming Language C++
Audience: EWG, SG21
Reply-to: Mungo Gill
<mungo.gill@me.com>
Corentin Jabot
<corentinjabot@gmail.com>
John Lakos
<jlakos@bloomberg.net>
Joshua Berne
<jberne4@bloomberg.net>
Timur Doumler
<papers@timur.audio>

Contents

1 Abstract

The authors of this paper believe that several of the features proposed by [P3081] are not yet sufficiently mature for adoption. Paper [P3081], “Core safety profiles” was introduced to EWG in Wrocław. Some of the Committee members involved with writing [P2900] have raised several concerns; this brief paper enumerates those concerns and makes constructive recommendations for how to quickly proceed towards safe adoption of the most urgently needed and uncontentious aspects of [P3081] into C++26.

2 Revision History

R0: December 2024 (midterm mailing)

Initial revision

3 Categorization of Features of [P3081]'s Safety Profiles

[P3081] proposes some changes to the current behavior (including to currently defined, currently erroneous, and currently undefined behavior) of ISO C++. These proposed changes are addressed individually in the sections below.

3.1 Rejection of unsafe program constructs (Language Profiles)

This proposal involves restricting the set of C++ programs (or translation units) that will compile (or compile without warning) to those using a known safe subset of the language that does not include some of the features that developers are likely to misunderstand and misuse. The resulting errors will enable those developers to then address each construct in turn by either replacing it with a safer construct or annotating it as being exempt from checking.

For ease of discussion, this paper will use the term *Language Profiles* to describe such restrictions, which are necessarily applicable only at compile time. Language Profiles necessarily do not affect code generation. For example, removing a profile preserves all behaviors, and recompiling without a profile produces the same generated code (modulo metadata, such as timestamps) as it did before. An important corollary for profile interoperability is that if a program (or translation unit) is compiled under two different profiles, the behavior is the same as that when compiled under either profile separately or when compiled with no profiles.

Language Profiles, i.e., [P3081]'s Safety Profiles as restricted to just subsetting the set of compilable valid programs, is perhaps the least controversial aspect of the [P3081] proposal. It is the feature least likely to conflict with other, more established efforts, such as Contracts, and the one that adds the most value. Although the authors of this paper would, in principle, be in favor of such a change, some concerns do remain.

— Choosing applicable entities and scopes must be well defined and clearly specified with research, examples, and implementation experience. If very granular scopes are pursued, clearly defining whether that granularity is at the lexical or logical level is needed, and the ramifications of this decision on real usage of profiles must be examined.

— Giving developers a means to selectively suppress Language-Profile checking on sections of code is absolutely needed and should again have associated research regarding to which scopes it should apply (and exactly what profile will apply to uses of a type and where a suppression needs to be placed must also be well defined and clearly specified). In particularly, note that semantic restrictions might relate to the use of an entity or to how that entity is declared, defined, or instantiated, and which of the relevant profiles is applicable for each individual use must be clear.

— A mechanism is needed to control the restrictions at a more fine-grained level than is currently proposed by [P3081] since some of [P3081]'s suggested profiles currently contain or have the future potential to contain numerous individual rules. In addition, the chosen syntax needs to allow for any future extension to apply more nuanced control over how various features are used without necessarily disabling them entirely. As an example, developers might wish to allow `static_cast` for integral-to-integral conversions but not for downcasting. Such a mechanism could entail much more fine-grained profiles or explicit exploration of how profiles (both Standard-provided and vendor-provided) will interact when they have overlapping rule sets. In particular, for many rules, a necessity will be the ability to suppress the specific rule in a given context, even if that rule might be a part of a wide number of profiles.

3.2 Implicit insertion of runtime checks

3.2.1 General approach

This aspect of the paper is arguably one of the least well developed and most controversial parts of [P3081] and would actively change the semantics of currently working, correct programs, even those without UB. (For an example, see [P3081]'s proposed behavioral changes to `union`.) In particular, SG21 has been working to provide a mechanism — Contracts — for defining, conditionally, what is currently undefined behavior, and any approach to doing so in a separate proposal must coordinate with that work.

The authors of [P3081] make assumptions about what will or will not be integrable with the current and future direction of Contracts, while also declaring that the checks introduced by profiles shall have specific properties (such as runtime-settable per-profile violation handlers or guarantees of specific evaluation semantics). In particular, unlike [P2900], proposal [P3081] dictates some specific behaviors.

— The invocation of a new handler, with a new signature, for checks evaluated by different profiles, is directly in conflict with the global violation handler model of [P2900].

— The choice of evaluation semantic for a contract assertion — whether it is ignored, observed, or enforced (or quick-enforced if no violation handler should be invoked) is unclear but implied heavily (based on responses given by the community about what they expect profiles to do) to be dictated by the use of a profile. In contrast, [P2900] leaves this choice as implementation-defined behavior to make sure implementations have the freedom to provide users the configurability they will need while leaving more fine-grained syntactic control for future extensions.

— [P3081] does not attempt to ensure the inserted predicate is free from side effects; for example, an inserted `size()` call on an arbitrary object may be computationally intensive or require a database query.

Each of these points has been discussed heavily within SG21 and EWG over the course of many years, and the results of these discussions should be leveraged and not discarded.

[P3081] asserts that

> *"This can in the future be easily integrated with contracts when we get them. We do need the contract group/category extension so that we can have customized global handler specifically for Bounds checks"*

Such integration, however, is not possible given [P3081] as currently written and is unlikely to be true in future versions unless there is greater collaboration between the authors of [P3081] and SG21.

The introduction of new runtime checks when a Safety Profile is present absolutely *must* be done in a way that either exclusively uses facilities provided by Contracts or has strong consensus *in SG21* as being forward compatible with future extensions to provide such facilities. These checks are unquestionably runtime checks of correctness, and all such runtime checking incontrovertibly falls squarely within the purview of SG21.

Importantly, any runtime checks introduced by [P3081] for behavior that is undefined behavior in the Standard specification today are already preconditions introduced as implicit preconditions into the language itself by [P3100].

In particular, [P3100] proposes that preconditions of basic language constructs should instead introduce contract assertions whose runtime behavior can be controlled in the same way as that of any other contract assertion. This transformation includes any undefined behavior for which a condition can be readily identified, including array bounds violations, null-pointer dereference, integer overflow, and many more. In addition, by providing a well-defined behavior when those contract assertions are *ignored* (instead of *assumed*, as they currently are) [P3100] aims to minimize the negative risks and bolster security even when the checks are not enabled. Finally, as mechanisms for in-source control of contract assertions (i.e., labels) are introduced, SG21 will work to be able to apply the same groupings and vocabulary to core language preconditions as well.

The only proposal [P3081] offers beyond those of [P3100] for these implicit preconditions is to muddy what behaviors are conforming for such checks.

The authors of this paper suggest taking one of three courses of action with respect to runtime checking in profiles.

1. Remove all parts of the proposal that introduce new runtime checks and pursue those as supplementary papers that make their way through SG21 and SG23 to ensure that they integrate correctly with the rest of the C++ ecosystem, including Contracts.

2. Leverage [P2900] Contracts directly, and do not impose any new restrictions that are not otherwise imposed. In particular, describe each runtime check in terms of the equivalent contract assertions, having no indication of any particular evaluation semantic or any novel use of the violation handler. Select the desired safety checks from the set of core-language preconditions delineated in [P3100].

   — Each such contract assertion may have any of the available evaluation semantics, depending on how the program is being built (i.e., it is implementation defined).

   — When a contract assertion is invoked, the violation handler shall be invoked with a `kind` value of `implicit` and a `detection_mode` that captures the specific type of implicit contract assertion being violated.

   — A recommended (but not required) practice that certain semantics be preferred when the relevant profile is in effect would likely be acceptable.

3. Safety by default is being widely requested by government and industry alike. Instead of having the profile turn on any runtime dynamic safety apparatus (which will likely go largely unnoticed and unused), consider the approach of enabling all runtime safety checks by default — the same way library contract checks are treated — and then, only by explicit request, remove some or all. This approach has several benefits.

   — C++ takes a bold move to make itself safe by default instead of vice versa (great for public relations).

   — Novices who would never think to turn on profiles will get the benefit by default.

   — Established professional developers who need to claw back the performance can do so explicitly on the command line.

   Note that by adopting this approach, the problem would essentially reduce to approach 1 above.

If the Committee instead continues down the path of developing a parallel system for runtime checking and simply hopes the systems can all be reconciled later, then the Committee risks shipping a Standard with no cohesive approach to reliably handling incorrect software. The authors of this paper are strongly against such a course of action.

3.2.2 Container duck-typing

The paper's suggested approach includes adding bounds checking to containers solely based on a duck-typed deduction using the availability of operators and member functions without knowing for sure what purpose the container serves, what those members do, or even whether the class is, in fact, a container at all. [P3081]'s proposal would cause a wide variety of existing, perfectly valid code to fail unpredictably at run time:

— Containers that are not zero-indexed

— Sparse containers

— Map-like types the compiler cannot detect as being map-like

— A two-color image where `size()` is in bytes but for which indexing is by pixel (i.e., by bit)

— Ring-like containers

— Containers that automatically grow.

[P3081] also fails to consider that some library vendors already (and will continue in the future to) choose to put contract preconditions on their `operator[]`, which would result in double and possibly inconsistent checking.

3.3 Silent changes to runtime behavior

[P3081] proposes that, in some cases, a Safety Profile setting should cause a change that will impact essential runtime behavior, e.g., on page 5:

> *"We should normatively require that the compiler actually perform a safe `dynamic_cast`, without changing the code which still says `static_cast`."*

The idea of deliberately modifying the defined runtime behavior of a program depending on which Safety Profile is active is dangerous. If a piece of code is genuinely unsafe, then the only viable solution is for that code itself to be modified so as to render it safe regardless of which Safety Profile is active.

Consider that profiles will evolve with and across groups and projects over time, but programs must always retain the same meaning across time and space or else must simply and safely fail to compile. The prime directive is necessarily that safety does not affect semantics other than to limit compilation (or compilation without warnings).

3.4 Offering "modernization" suggestions

Under this proposal, compilers would be mandated to produce "patch files" or similar containing code changes, with the expectation that the developer will commit these changes directly to their source control system. The authors of this paper do have some concerns about this proposed change.

— [P3081]'s approach is straying into the realm of what is considered QoI, and implementations already offer such modernizations.

— Any Safety Profiles requiring control flow analysis could have a noticeable impact, now or in the future, on compile times. This kind of gating where the problem is unbounded must not be legislated in the Standard because it would put an unreasonable burden on compiler vendors to accomplish an arbitrarily hard task in a reasonable amount of compilation time. Thus, strictly limiting what can be required to that which has a bounded time-complexity solution for all cases will be necessary.

— As more elaborate Safety Profiles are added, complex code structures might not have a single potential modernization, which becomes a problem if different toolchains result in different changes.

— Some of [P3081]'s suggested changes are not safety issues but merely stylistic preferences; historically, the Standard's role has not been to dictate to developers or implementers style or coding standards that are not purely objective and verifiable.

— At least one implementer has raised serious concerns about this aspect of the [P3081] proposal in discussion on the reflector. One must question whether valuable Committee time should be expended to add this feature if implementers have no intention of ever delivering it.

4 Conclusion

The authors of this paper are firmly convinced that, to increase immediate consensus in time for the C++26 deadline, all but the language-subsetting aspects (i.e., Language Profiles) be removed from [P3081], notably

— all runtime checks until a more mature proposal (designed in collaboration with, and approved by, SG21) can be brought forward or runtime checks that leverage [P2900] Contracts in the same manner as [P3100], with no new forward-incompatible restrictions on the expected behavior

— all "fix it" changes where the compiler is silently reinterpreting the developer's own choice of cast

— all mandated modernization suggestions

The authors of this paper also encourage forethought about how to incorporate more nuanced syntax for a user to express general design rules and coding standards beyond a binary yes/no to a given C++ feature, construct, or keyword.

5 Acknowledgements

This document is written in Markdown and depends on the extensions in `pandoc` and `mpark/wg21`, and we would like to thank the authors of those extensions and associated libraries.

Thanks to Lori Hughes for reviewing this paper and providing valuable editorial feedback.

6 References

[P2900] Joshua Berne, Timur Doumler, Andrzej Krzemieński and others. Contracts for C++.
https://wg21.link/p2900

[P3081] Herb Sutter. Core safety Profiles: Specification, adoptability, and impact.
https://wg21.link/p3081

[P3100] Timur Doumler, Gašper Ažman, Joshua Berne. Undefined and erroneous behaviour is a contract violation.
https://wg21.link/p3100