Integrating Concepts: “Open” items for consideration

John H. Spicer, Hubert S.K. Tong, Daveed Vandevoorde

Project: ISO/IEC JTC 1/SC 22/WG 21/C++
Document #: P0691R0
Date: 2017-06-17
Revises: None
Working Group: Evolution
Reply to: Hubert Tong
hubert.reinterpretcast@gmail.com

Table of Contents

Background/Context for the current discussion

General issues with the TS status quo

Specific issues for consideration

- Unadorned concept name treatment: placeholder type versus satisfaction value versus metatype versus introducer
- One concept definition syntax, first-class concept definitions, and removal of the current Concept Name Resolution
- Abbreviated function syntax support for both “same” and “separate”
- Concept overloading
- “greppable” syntax to identify templates
- Atomic constraint identity
- Disambiguate requires-clause syntax with parentheses
- Require redeclarations to use consistent syntax

Acknowledgements

Background/Context for the current discussion

The C++ committee (WG 21) decided to produce Technical Specifications (TSes) as a way to develop extensions to the C++ language without modifying the base International Standard (IS). What was known as "Concepts Lite" was developed as one such TS, now published as ISO/IEC TS 19217. That feature is the first C++ core language feature to have become a TS project[1], and the functionality it provides is desired for the IS (although some believe that the functionality is not complete as a solution to Concepts).

Efforts to propose applying the changes from the TS working draft, maintained by Andrew Sutton, into the working draft for C++ have thus far failed to gain consensus within WG 21. The point of contention hinges over whether the TS has proven to be sufficiently mature to represent the "final shape" of Concepts aside from minor bug fixes or not. This further breaks into what constitutes a minor or not-so-minor change: even if changes are agreed upon, perhaps we do not have to wait for them before merging Concepts into the IS. This paper summarizes a number of issues, none of them new, which may elicit strong opinions if left without resolution.

General issues with the TS status quo

In terms of the readiness of the TS for integration into the IS, one of the more obvious issues is that the TS was developed to be an isolated TS.

From that origin, we are left with:

- some design properties which mostly originated from convenience (the property that **`concept`** is a *decl-specifier*),
- some experimental design elements which were fine for an experimental TS, but require evaluation for inclusion into the IS; and
- wording that was reviewed without the benefit of wider implementation or usage experience.

There has been a lack of substantive changes to the TS since its publication. We now have later (for certain groups, probably as expected) development of abstract models for consistency from "from-spec" implementation efforts. One early example is the last-minute change before publication of the TS to define signatures using expressions as opposed to constraints.[2] We also have feedback from usage experience that asks for changes or indicates that some functionality present in the TS is not essential.

---

[1] The New Work Item Proposal (NWIP) for the Concepts TS, ISO/IEC JTC 1/SC 22 N 4800, is dated 2013-05-17. The Transactional Memory TS, which was published before the Concepts TS, had an NWIP dated 2014-03-12.

[2] Andrew Sutton, "Editor's report for the Concepts TS", WG21 N4554, 2015. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4554.pdf

At the same time, there are outstanding concerns raised since before the TS was approved. The committee has had the benefit of time to see if these concerns would lessen; however, at least some of them have not faded. Especially in this space, there are differing evaluations of the TS experience in terms of its effectiveness in gathering sufficient usage or design feedback (i.e., whose feedback is important and significant). We have heard back that people are using Concepts with the GCC implementation; however, whether as a toy or a tool, these users are currently a group that chose to use the feature in its current form. The committee also answers to those who have not chosen to use Concepts; unfortunately, active feedback from the latter group is unlikely to materialize before the presence of Concepts in a ratified revision of the IS is a *fait accompli*.

Perhaps what has not been really stressed before is what appears to be a general lack of from-spec *testing*. A sound mid-to-low-level design requires considering interaction with the existing language. Preferably, precedents or lessons learned from the existing language are used in determining a direction. From parsing to partial specializations to "just" calling a function concept, the record on Concepts shows that more work is to be done.[3] Conveniently, some changes to the "high level" design could entirely sidestep problems from the implementation standpoint.

Changes are planned for Concepts, and some of them will partially resolve the set of issues listed in this paper. If agreement is reached on a direction to resolve the issues that remain, then we are in a good shape for shipping Concepts in the version of C++ slated for 2020.

Specific issues for consideration

Unadorned concept name treatment: placeholder type versus satisfaction value versus metatype versus introducer

Preference:
Consistently use the value/metatype interpretation (require use as a placeholder type to have some syntax nit); the use as an introducer in the *template-introduction* syntax is a mostly orthogonal discussion.

Previous related polls:

- Do we want to visually distinguish template declarations as in some additional syntax in the shortest form? 9 strongly for | 6 | 14 neutral | 13 | 9 strongly against

[3] Parsing issues stem from Concepts Issue 21. Issues with the reference implementation in GCC have been reported for partial specializations (https://gcc.gnu.org/bugzilla/show_bug.cgi?id=81043), and for calling a function concept (https://gcc.gnu.org/bugzilla/show_bug.cgi?id=79711).

Status:
Proposals providing the "syntax nit" are available; however, the committee has not closed on a direction.

Positions:
Expect to see strong opposition to having abbreviated function syntax (also known as "terse syntax") without the "syntax nit".

Reasons:

- A "syntax nit" makes abbreviated function syntax clear from the immediate context.
- Concept names are not always sufficiently distinctive; we should not make it necessary for users to use Hungarian notation or advanced development tools in order to skim code.
- Consistently applying syntax keeps options open for language extension points. Constrained `template <auto>` is an example where using additional syntax for the placeholder-type use of a concept name would naturally fit.
- Having syntax provides a way to encourage/constrain name lookup to find a concept name.

Note: first-class concept definitions (discussed below) would remove the confusion that a name used for a concept is also the name of a function or variable.

One concept definition syntax, first-class concept definitions, and removal of the current Concept Name Resolution

Preference: Adopt

Previous related polls:

- Explore removing the distinction between function-like and variable-like concept definitions? 24 strongly for | 23 | 1 neutral | 0 | 0
- Explore removing the bool? 20ish strongly for | 10ish | 5 neutral | 0 | 0

Status:
The direction from the polls is compatible with papers proposing "first-class concepts definitions". Papers appear to be in-flight, with the possibility that proposals will be discussed as part of the 2017 Toronto meeting.

Positions:
Expect to see strong opposition to keeping concept definitions as also functions or variables.

Reasons:

- Various grammar ambiguity and code readability issues are caused by concept names also being function or variable names.
- The function aspect introduces interaction with function overload resolution, which appears to be an unnecessary complication.
- A more integrated approach to handling concept names as part of name lookup would result from this work.

Abbreviated function syntax support for both "same" and "separate"

Preference: Adopt

Status:
Proposals covering this are available. The committee discussion is (perhaps necessarily) entangled with the general treatment of unadorned concept names.

Positions:

- Expect to see strong opposition to swapping the default without changes to syntax.
- Expect to see at least moderate opposition if this functionality is not added.
- Expect to see moderate opposition if the syntax closes doors on extensibility such as constrained return type deduction.

Reasons:

- Neither "same" nor "separate" is clearly more correct than the other except that "same" is more convenient for use in non-deduced contexts.
- Input from committee members with recognized usage experience:

> Eric Niebler: I consider it very important to change the way deduction works for the "simple" form such that for `` `foo(Concept x, Concept y)` ``, the types of `` `x` `` and `` `y` `` are allowed to be deduced as distinct.

Concept overloading

Preference: Weakly in favour of removal

Status: This item is entwined with the more general discussion on concept definitions.

Positions: Expect strong opposition to overloading concepts using functions.

Reasons:
Feedback from committee members with usage experience of Concepts through experience with the Ranges TS that the ability to overload concepts on arity has limited use, and using different names would work just as well.

> Eric: As the author of the ranges TS I would be perfectly happy with this change to remove overloading.

"greppable" syntax to identify templates

Preference: Explore alternatives to the *template-introduction* syntax.

Status: Papers, such as P0587R0, are available. No clear guidance was provided by the committee.

Positions: Expect moderate opposition to the novelty of the introduction syntax.

Reasons:

- As given in P0587R0, the *template-introduction* syntax has limited use and overloads the syntax.
- Alternatives can have both increased expressivity and increased similarity to the existing language. For example:

```
template <T, U, V, N>
{ C: T, U; typename V; int N; }
void foo(T t, U u, V (*v)[N]);
```

Atomic constraint identity

Preference:
Adopt a formulation which ties the identity of an atomic constraint to a specialization of a concept definition.

Status:
Committee guidance is to adopt. Papers appear to be in-flight, with the possibility that there will be a proposal discussed in the 2017 Toronto meeting.

Previous related polls:

- Identity of atomic constraints doesn't depend on token sequence?
  tons strongly for | dozen | 1 neutral | 0 | 0

Positions:
Expect tension if wording review for integrating Concepts begins before incorporating changes in this area.

Reasons:

- Change avoids fragile accidental matching or failure to match.
- Change avoids requiring implementations to differentiate between equivalent versus functionally equivalent expressions.

Disambiguate *requires-clause* syntax with parentheses

Preference: Adopt

Status: This is a possible resolution for Concepts Issue 21.

Positions: Expect some opposition to adding yet another parsing vexation.

Reasons:

- Unexpected parsing is unfriendly.
    - Have already seen evidence of code using function-style cast to **`bool`**, which does work to mostly avoid unexpected parsing.
- The grammar ambiguity increases the cost of extending the language; for example, new forms of expressions may eat tokens that currently belong after the *requires-clause*. In essence, leaving the ambiguity (and merely applying an ambiguity resolution rule) is laying a trap.
- Users cannot resolve the issue simply by consistently adding parentheses around the intended constraint.

```
template <typename T>
requires (bool(T()))
int (*fp)();
```

Require redeclarations to use consistent syntax

Preference: Adopt

Status: This needs wording.

Previous related polls:

- Restrict redeclarations to identical forms?
  15 strongly for | 14 | 13 neutral | 2 | 3 strongly against

Positions:
Expect tension if wording review for integrating Concepts begins before incorporating changes in this area.

Reasons:

- Lack of precedent in the existing language.
- Extremely fragile; adding parentheses could break a program.
- Rewrite rules can produce ASTs that cannot be expressed as "flat" expressions in C++ source.

Acknowledgements

The authors would like to thank Jens Maurer and certain other individuals consulted during the development and planning for this paper. Whatever mistakes are present are the sole responsibility of the authors.