Document number: N3576
Date: 2013-03-12
Author: Herb Sutter (hsutter@microsoft.com)

SG8 Concepts Teleconference Minutes – 2013-03-12

1. Opening and introductions

Austern called the meeting to order at 9:05 am U.S. Pacific time.

1.1 Roll call of participants

Matt Austern (chair)

Walter Brown

Magne Haveraaen

Dietmar Kühl

Alisdair Meredith

Jason Merrill

Nevin Liber

Bjarne Stroustrup

Herb Sutter

Andrew Sutton

Faisal Vali

Tony Van Eerd

Daveed Vandevoorde

Larisse Voufo

Ville Voutilainen

Jeremiah Willcock

1.2 Adopt agenda

Adopted by acclamation.

1.3 Approve minutes from previous meeting

Not applicable.

1.4 Review action items from previous meeting

Not applicable.

2. Technical topics

2.1 Do we continue work on static if, or focus attention on concepts lite?

Austern summarized that there were two static if papers considered over the past year at Kona and Portland.

Vandevoorde: Would not like to close the door on static if, which is not currently analysis-friendly but could be progressed into a compiler analysis-friendly version.

Willcock: Concepts lite replaces part of the uses of static if but other parts could still be useful, concepts lite does not subsume static if completely.

Stroustrup: Would like to see a better behaved substitute for preprocessor techniques but the current static if proposals don't seem to be a good solution. Concepts lite covers one of the three things static if would cover and is not intended to be a general mechanism for replacing all template metaprogramming, just specifying better interfaces including overloading.

Austern: We cannot close the door on static if or any proposal, but should focus on concepts lite if that's a short term deliverable.

Sutter and Brown said that they are not planning to bring an updated static if paper to Bristol.

```
Straw poll: Should we defer discussion of static if until at least after
Chicago?

    ++    +     =     -     --
    8     3     1     2     1
```

- Meanings for the above columns are usual in WG21 straw polls: ++ strongly in favor, + weakly in favor, = neutral, - weakly against, -- strongly against.

2.2 Brief review of concepts lite from Andrew and Bjarne.

Stroustrup and Sutton: This paper was the result of more than a decade's work, the Palo Alto concepts summit and resulting technical report, and experience with the gcc implementation as low-hanging fruit with low effort and immediate benefits.

Vandevoorde: Are constraint predicates overloadable?

Sutton: Only with respect to the number of template parameters, no constraints-based overloading.

Vandevoorde: What if I did it anyway?

Sutton: Goal is to find a single definition of constraints for all types. Overloading would lead to situations where the definition could change based on requirements. If you overload constraint predicates anyway, the predicate is no longer 'expanded inline' (i.e., it becomes effectively a predicate 'atom'); it's

not an error per se. If the call target isn't a constraint predicate, then it's just a regular `constexpr` call (an atom).

2.3 Relationship between concepts lite and concepts: is this the right stepping stone?

Ideas about what full concepts might look like -- what's "lite" about concepts lite

Austern: We want small incremental steps. This discussion is about whether this the right incremental step. What has been left out so that this is "lite"?

Stroustrup: "Lite" defines the checking of calls to a template. It does not describe the checking of the template implementation itself against its constraints. The growth path from concepts lite is toward something complete like what is described in the Palo Alto TR. We expect full compatibility between concepts lite and full concepts, in particular that every use of concepts lite will work once you check the definitions provided the definitions only use the facilities required from the users. It is elegant that there is no distinction between constrained and unconstrained templates other than if you say `typename` the only constraint is that it's a type; no need for late check as such as everything in the definition is late checked. As John Spicer had said on the reflector, some syntax may be needed once we do checking.

Compatibility between concepts lite and concepts

Meredith: Likes current elegant lite solution. Concerned about the feasibility of extending that to not end up with two template worlds of constrained and unconstrained template.

Stroustrup: Palo Alto TR describes such a unified world but definitely not ready to propose now, feedback and help with further work will be appreciated.

Does there need to be a syntactic distinction?

Austern: Concerned about potential compatibility issue whereby the same syntax may not be able to used as in the Palo Alto TR.

Stroustrup: Can be done in stages. First check callers without checking implementations; can write header containing only declarations and that is used to check the constraints without modifying existing code. Later check implementations; some things won't work because implementers did things they shouldn't. If we had an integrated proposal that did it all right away we'd get all these migration problems at once. By staging it we can incrementally migrate toward constrained templates.

Austern: So you are proposing a syntactic distinction in the future, so that even when we have full checking in the future some may not be checked?

Stroustrup: Possibly, but alternatives still being considered and worked on.

Sutton: Working on a gcc branch that will perform separate checking as a warning. There is a path from interface checking to separate checking. More information should be available later in the year.

Additional concerns, if any

```
Straw poll: Based on the papers, implementation, and discussion, are we
reasonably comfortable that we will probably find a good migration path
from concepts lite to a full separately checked concepts?

    yes    no    abstain
```

8	2	5

During the straw poll, several people spoke about favoring concepts lite, and that they were only responding to the future broader migration path question.

Stroustrup: The migration was intended to be not to something like the ornate C++0x design but to something much smaller and faster.

Voufo: In a future full concepts system, we will probably need a separate syntax for concepts definition.

Stroustrup: Agreed, as John Spicer also said.

Austern: For those who voted No or Abstain, what additional information would help increase your confidence?

Meredith: After seeing C++0x, no longer believes it's possible to have checked and unchecked template bodies coexisting in the same language, but this is a useful feature by itself even if we progressed no further.

Stroustrup: The main problems in checking definitions in C++0x concepts were the concept maps and the complexity those caused, and losing them when going into unchecked territory, and these problems have disappeared by not using concept maps.

Voufo: With or without concept maps, there is complexity with specifying separate checking for templates.

Van Eerd: Does concepts lite shut the door on other directions people wanted to go?

Stroustrup: Some of the problems with the C++0x proposal was imperfect merging of different ideas. Hopes this one has fewer problems because it's a simpler proposal and the major people who worked on concepts for C++0x were at the Palo Alto meeting, including Alex Stepanov and Sean Parent and Andrew Lumsdaine from the Indiana group, so hopes we can avoid the complexities.

Sutton: Concepts lite doesn't impose any particular model on how we do separate checking, not closing doors there.

Willcock: Has the sense that starting with this proposal it will foreclose on the ability to add concept maps later. Lookup rules will be more like the current ones.

Stroustrup: I don't believe it closes the door, as we still have the ability to check callers vs. definitions, as Sutton said. I would actually like to close that door because it leads into a swamp, but concepts lite itself does not close that door yet.

Willcock: Lookup for types that come from concepts will be different from normal lookup.

Stroustrup: That only affects the definition which we are free to specify separately later, and concepts lite is only checking the calling interface.

Sutter: In C++0x, concept maps were terrible because they were the default that imposed indirection overhead by default and we relied on being able to optimize them away. One of the nicest things about

this proposal is that it makes a clear distinction and separation between checking callers and checking definitions, and although the latter can still use concept maps in the future, by not mentioning concept maps in the lite proposal for checking callers means that concept maps and their indirection overhead will not be the default.

Meredith: Would eventually like to see some ability like concept maps for adaptation, but would give up on that idea to get something real.

Stroustrup: Also liked concept maps initially as providing a useful feature, but it was the source of compile time overheads and optimizer problems which made maps problematic when we don't know how widely the benefit will be used.

2.4 Feedback for Andrew and Bjarne on specific technical details of the proposal

Rules for atomicity and for deducing whether one concept subsumes another

Austern: Lots of questions and answers about this on the reflector. Were everyone's questions answered or more discussion needed?

There were no responses.

Stroustrup: Discussion seemed to be successfully worked out on the reflector.

Logical operations: conjunction, disjunction, negation

Austern: The proposal includes only conjunction and disjunction. Should negation be added?

Sutton: The issue is looking down the road to separate checking. Couldn't find a meaning for that operator in separate checking that had its usual Boolean meaning. End up in a netherworld of meanings, haven't been able to find the right meaning for that symbol in the constraints language.

Austern: Maybe add a line in the paper somewhere to explain why it's left out.

Sutton: Will add.

Any additional feedback based on the text of the paper

Brown: Concern about what is not in the paper – can I constrain other things with a requires clause, such as a free function?

Sutton: Answering from the implementation perspective then generalize… the implementation does allow you to constrain non-template declarations.

Brown: Please add something about that in the paper. Ability to overload on thing in the environment, such as endianness, would be very useful.

Sutton: Will add text speculating what else might be done with constraints.

Austern: This would be useful but would be a separate and future proposal please.

Van Eerd: The ability to talk about these things is a good sign of the future extensibility of this proposal.

Stroustrup: We are looking over the horizon to see there are no major problems lurking there, but don't want to be too specific until it's time.

Vandevoorde: Do you have measurements on how much memory is consumed with complex cascade of predicates that involve logical or? Does it multiply?

Sutton: It does, and if you design your constraints using large disjunctions you've probably done something wrong. We haven't seen that limit yet in our experience, not a great concern.

Vandevoorde: Suggested that "constraint predicates" be explicitly marked as such, that is, not just any `constexpr` function template with `bool ()` type would be a constraint predicate. This would allow for diagnosing unintended code.

Stroustrup: Please send examples.

Haveraaen: Paper around page 16 doesn't seem to fully specify the meaning of disjunction.

Sutton: Formal definition of disjunction is theoretical. Didn't feel it was useful to include formal definition in paper.

Stroustrup: Would you be happy with a reference?

Sutton: I have to write a research version of this paper and will include this. Can circulate pre-publication drafts.

Stroustrup: Help please for crafting working paper text. We would like it to be readable in the WP text and not suffer mission creep while it's being written down.

Haveraaen: Section 4.3 sets forth the current steps in detail, but it also would be helpful if the new rules were integrated into a step-by-step procedure in order to see when candidates are deemed viable and when they are injected into the set of overload resolution candidates.

Sutton: Will elaborate.

Haveraaen: In what sense can I extend the meaning of constraints? In the example of the complex number constrained to arithmetic, which is decomposed into being floating point or integral, if I have a matrix data type can I use that as an element type for complex numbers?

Sutton: In this case no because definition of arithmetic is fixed by the standard. A good constraint that allows complex numbers and real numbers wouldn't be specified this way. This definition is specifically taken out of the standard. Constraints should be written in a more general way. Will add to a discussion section at the end of the proposal.

Austern: Similarly the proposal should say more about how to define traits.

Stroustrup: We have the Palo Alto TR as an example.

2.5 Additional work that might need to be done in conjunction with concepts lite

Do we need the intrinsics discussed in the implementation section, or some other facilities to make defining concepts easier?

Vandevoorde: Section 5 discusses facilities about valid types, `__is_valid_type` and `__is_valid_expr`, that seem pretty compiler dependent and SFINAE capability. C++ doesn't define whether something can be typed.

Austern: And should these be standardized?

Stroustrup: We don't have an answer but would like suggestions. Wouldn't like to see concepts lite rejected because of these intrinsics, but the intrinsics would be useful. Perhaps separate them into a separate proposal?

Austern: Would like to see them completely part of the proposal, or completely gone and not even mentioned. For the latter, that would include examples of how to define useful concepts for the standard library without them.

Vandevoorde: You could write a library without them, but I suspect that no first-party library will do that. Worth standardizing so that we can all agree how it's done. There are type traits like `is_convertible` that have intrinsic support in all compilers, and I've spent weeks trying to emulate ones in different compilers that behave differently.

Stroustrup: Sutton encountered frustration without `__is_valid_*`, and after using them things became much easier.

Austern: Two questions – do we want these intrinsics, and do we want them as part of this proposal?

Vandevoorde: Yes and yes.

Stroustrup: I came in without an answer, and like Vandevoorde's answer.

```
Straw poll: Do we want these intrinsics, and have them be part of this
proposal?

    Unanimous consent.
```

Austern: Paper authors have more work to do, to define these intrinsics.

Several people volunteered to help the authors draft the wording.

Do we also need a library proposal?

Stroustrup: We are not ready for a proposal to put constraints on the library. We are experimenting with how to do it, we are quite happy with our experiments, but if a library proposal were part of this proposal we don't think it would pass. We want a library proposal as a proof of concept and we have a proof of concept implementation, but we don't have something ready for an International Standard. Adopting concepts lite now is the best path to get a constrained standard library for C++17.

Meredith: Speaking personally, not as LWG chair, I was horrified in C++0x that we designed standard library concepts based on what implementations did in order to avoid breaking code rather than designing the constraints properly. We can't have a constrained library in C++14. If we do a Concepts TS that is decoupled from C++14 then it could contain a library when it's ready, or a rapid-turnaround Concepts Language TS and a longer-term Concepts Library TS so that hopefully both are in line when we get to C++17.

Vandevoorde: It's not unprecedented to have a core language feature that has obvious library application but is not used in the library, such as UDLs and some attributes.

Does anything need to be done about the interaction between concepts and polymorphic lambdas?

Stroustrup: Would hate to see generic lambdas deliver a practice that is very different from templates. Constraints should be universal. If we use the `auto` syntax for lambdas as discussed in Portland, then you use `auto` as the least constrained and throw in `Sortable` etc. in that syntactic spot, and in that place you need the `concept` keyword to make them syntactically distinguished.

Sutter: Should be careful to unify not only generic lambdas and concepts lite, but also make sure it's consistent and regular with functions, such as if we allow `[]( auto s ) { }` and `[]( Sortable s ) { }` then we should allow also something like `auto sort( Sortable& s ) -> void` as a synonym for `template<Sortable Cont> auto sort( Cont& s ) -> void`.

Vandevoorde: There are problems (order-of-processing issues) with old-style function declarations when the return type is already parsed, would be better to do this only for new-style function declarations with the return type at the end.

Stroustrup: Does this group give guidance to pursue constrained generic lambdas, and making lambdas and functions as similar as possible, as part of the concepts paper this week, with the caveat that some kind of syntactic disambiguation would be useful?

```
Straw poll: Would we like the pre-Bristol concepts paper include
constrained generic lambdas and making lambdas and functions as similar
as possible, with the caveat that some kind of syntactic disambiguation
would be useful?

    ++    +     =     -     --
    5     6     3     2     0
```

2.6 Scheduling

Should this be targeted for C++14, C++17, or a TS?

Stroustrup: Want something in the C++14 timeframe, people outside the committee probably don't distinguish IS and TS, but C++17 would be too long. Probably neither C++14 nor a TS approved at essentially the same meeting as C++14 will be possible without splitting the proposal into language and library parts. Having a separated library proposal also enables what many including Stepanov want which is that we think we can do a library design better than in the past.

Austern: Two options have been suggested. 1. Shoot for getting concepts lite into the C++14 IS, core language not library. 2. Shoot for concepts lite TS in the same timeframe as C++14, core language only not library.

Sutter: Summarized IS and TS requirements and schedules. If we target C++14 we probably don't want to slide this in between CD (Bristol) and DIS (Chicago).

Stroustrup, Austern: We probably don't need to decide this now and can do it in Bristol.

Whatever the answer to the first question is: What milestone has to be reached by the end of Bristol, and the end of Chicago, to get us there?

Sutter, Austern, Stroustrup, Vandevoorde: Need fairly solid working paper text in the pre-Bristol paper if we want to get it into C++14.

Meredith: Not speaking as LWG chair. Prefers TS. Doesn't want to see anything in C++14 that changes the programming model.

Stroustrup: Doesn't change the programming model, changes the checking model.

Austern: And the documentation model.

There was tentative general agreement on the call, subject of course to more concrete and broader discussion in Bristol, to:

- Separate language and library: Ship concepts lite language separately from applying constraints to the standard library.
- Probably ship concepts lite language support in C++14 or a TS in the same timeframe as C++14.
- Then ship a constrained standard library as a TS between the above and C++17.

3. Any other business

There was none.

4. Review

4.1 Review and approve resolutions and issues

There were none.

4.2 Review action items

Stroustrup and Sutton will update their paper for the pre-Bristol mailing.

5. Closing process

5.1 Establish next agenda

No teleconference is currently planned, but if/when planned will use the same agenda structure with appropriate points in section 2.

5.2 Future meetings

We will meet face-to-face in Bristol as SG8 or fold into EWG.

5.3 Adjourn

Adjourned by acclamation at 12:00 pm U.S. Pacific time.