Reserve more freedom for `atomic_ref<>` implementers

Author: Olivier Giroux (NVIDIA) This paper: P1298R0 Date: 2018-10-08.
Audience: SG1

1. What is `atomic_ref<>`?

The facility for atomic access to non-atomic types introduced by http://wg21.link/p0019 (R8 as of this writing). As an example, an object of type `atomic_ref<int>` adapts the interface of `atomic<int>` to an underlying object of type `int`.

2. Why should implementers ask for more freedom?

The specification for the `atomic_ref<>` types allows less implementation diversity than the corresponding `atomic<>` types for non-*lock-free* types, in ways that affect other discussions ongoing in WG21 (like *freestanding*). In particular, the current proposal reflects a dislike in SG1 for implementations that use embedded locks (locks embedded inside `atomic<>` objects) instead of lock tables (locks in the library, associated by an address hash).

Implementers should ask for the freedom to only implement `atomic_ref<>` that are either *lock-free* (by un-defining or implementation-defining the effects of the constructor for non-*lock-free* `atomic_ref<>` types), or can make use of embedded locks (by un-defining concurrent invocations of the constructor, perhaps conditionally to non-*lock-free* types, or by introducing new types a user can use to manage the lifetime and associativity of the lock).

There is potentially a lot of design space here, only the simples step will be proposed here.

3. What is proposed?

Make things less defined. Choose one of the following:

Proposal A: un-define construction of non-*lock-free* `atomic_ref<>`.

This is the simplest and most powerful lever. For now, users can check for this using `is_always_lock_free`, and we can relax this later.

-

> ***atomic_ref(T& obj);***
>
> *Requires:* ***`is_always_lock_free`*** *is* ***`true`*** *and the referenced object shall be aligned to* `required_alignment`*.*

Proposal B: un-define concurrent construction of `atomic_ref<>`.

Equally simply:

> *While an~~y~~* `atomic_ref` *instance~~s~~ exist~~s~~ which reference~~s~~ the* `*ptr` *object, all accesses to that object shall exclusively occur through this ~~those~~* `atomic_ref` *instance~~s~~.*

This is not likely to be as popular as proposal A, but could be modified to the less stringent *implementation-defined*. This still would effectively remove the concurrent construction feature from portable code, but implementations are likely to agree that at least instances of *lock-free* `atomic_ref<>` can be constructed concurrently.