Minutes of Electronic ARG Meeting 60D

26 February 2019

Attendees: Steve Baird, Randy Brukardt, Gary Dismukes, Bob Duff, Jeff Cousins, Brad Moore, Jean-Pierre Rosen (arrived at 11:55), Ed Schonberg, Justin Squirek, Tucker Taft, Tullio Vardanega.

Observers: Richard Wai.

Meeting Summary

The meeting convened on Tuesday, 26 February 2019 at 11:02 hours EST and adjourned at 14:00 hours EST. The meeting was held using Zoom. The meeting covered some of the AIs on the agenda.

AI Summary

The following AIs were approved with editorial changes:

AI12-0208-1/09 Predefined Big numbers supported (8-1-2)
AI12-0281-1/01 CPU affinity for Protected Objects (9-0-2)
AI12-0311-1/02 Suppressing client-side assertions for language-defined units (9-0-1)
AI12-0314-1/01 Title of 13.13.1 (10-0-0)
AI12-0315-1/01 Image attribute clause improvements (10-0-0)
AI12-0317-1/05 Simplifying the rules for newly constructed objects (10-0-0)
AI12-0318-1/01 No_IO should exclude Ada.Directories (10-0-0)

The intention of the following AIs were approved but they require a rewrite:

AI12-0234-1/03 Compare-and-swap for atomic objects (10-0-1)
AI12-0250-1/02 Iterator Filters (9-0-2)
AI12-0282-1/01 Atomic and Volatile generic formal types (10-0-1)

The following AI was discussed and assigned to an editor:

AI12-0319-1/01 Nonblocking for Unchecked_Deallocation is wrong

The following AIs were discussed and placed on hold:

AI12-0297-1/01 Defaults for generic formal packages and formal **in out** objects (9-0-1)
AI12-0305-1/01 Bounded Big Integers (9-0-1)
AI12-0316-1/01 Preconditions for checking Task_Ids (11-0-0)

Detailed Minutes

Welcome

Steve announces “Welcome to another remote ARG thingy”.

Apologies

Erhard Ploedereder said that he is on vacation and probably won't be able to attend. Jean-Pierre Rosen said that he had another overlapping meeting and would be arriving late.

Previous Meeting Minutes

Randy notes that Gary had sent some typos in the minutes, which he corrected and posted yesterday. No one has any further changes on the minutes. Approve minutes: 10-0-0.

Date and Venue of the Next Meeting

We've already decided that the next electronic meeting will be on March 11th at 11:00 EDT. It is noted that our west coast members will be in Paris that day, so we could start earlier. Randy grouses that is fine if we want the editor to be 2/3rds asleep. We note that Daylight Savings time starts on March 10th in the US, while Summer Time doesn't start in Europe until March 31st. So there is one less hour difference than usual on March 11th – the meeting will start at 1600 hours Paris time. We decide to use the original time of the meeting.

Our next face-to-face meeting is scheduled for June 14-16 in Warsaw Poland, immediately after the Ada-Europe conference.

RM Review Plan

RM review assignments have been distributed, and two reviews have been received. The deadline is that for the next meeting (March 7th). Early reviews are encouraged!

Unfinished Action Items

Steve Baird has the only unfinished action item, the "dynamic accessibility is a pain to implement" AI (AI12-0016-1). We did not spend any time talking about it this time. Brad sent a draft of AI12-0312-1, but it was late and without any review from his coauthors. It needs quite a few updates.

Current Action Items

The combined unfinished old action items and new action items from the meeting are shown below.

Everyone:

- Review their assigned portion of the draft Standard.

Steve Baird:

- AI12-0016-1

Randy Brukardt:

- AI12-0319-1

Editorial changes only:

- AI12-0208-1
- AI12-0281-1
- AI12-0311-1
- AI12-0314-1
- AI12-0315-1
- AI12-0317-1
- AI12-0318-1

Gary Dismukes:

- Determine if Atomic_Components need to be included in the generic formal derived type rule of C.6(12) (see discussion of AI12-0282-1).

Brad Moore:

- AI12-0234-1
- AI12-0312-1 (update with comments from Jeff & Randy in e-mail)

Tucker Taft:

- AI12-0250-1
- AI12-0282-1

Detailed Review

The minutes cover detailed review of Ada 2012 AIs (AI12s). The AI12s are presented in numeric order, which is not necessarily the order in which they were discussed. Votes are recorded as "for"-"against"-"abstentions". For instance, a vote of 6-1-2 would have had six votes for, one vote against, and two abstentions.

If a paragraph number is identified as coming from the working Ada 202x AARM, the number refers to the text in draft 18 of the Ada 202x AARM. Paragraph numbers in other drafts may vary. Other paragraph numbers come from the final consolidated Ada 2012 AARM; again the paragraph numbers in the many drafts may vary.

Detailed Review of Ada 2012 AIs

AI12-0208-1/09 Predefined Big numbers supported

The main changes were to change the name of the type and package to Big_Real, since the intent is that it is an implementation of a type like *universal_real.* Most users just want more precision for their math; the implementation technique of rationals is not interesting. It's not mathematically a real number (as it can't represent irrational numbers), but that ship sailed decades ago for Ada.

There also were some minor improvements (a unary "+" for Big_Real, removal of the use clause for Big_Integer in Big_Real (it was only used twice, and the usage of operators is easy to confuse with the ones being defined).

We discuss whether Big_Integer should have a unary "+" for Big_Integer. Do we need the "+" (Integer => Big_Integer)? It seems that "+" isn't needed, and then we can and should add the unary "+".

Tucker also suggests getting rid of "+" (Big_Integer => Big_Real) as well. We agree that there should be no unary "+" operators where the argument and result types differ.

We discuss the names of functions "Is_Present" and "No_Big_Integer". Tucker and others had not liked these. Steve included a list of 6 possibilities determined after (private) discussion with Tucker:

1) Is_Null and Null_Big_Integer
2) Is_Null, Not_Null, and Null_Big_Integer.
3) Is_Valid and Invalid_Big_Integer.
4) Is_Defined and Undefined_Big_Integer.
5) Is_Present and Null_Big_Integer.
6) Is_Present and No_Big_Integer (as in the current proposal).

We quickly eliminate choice 2; someone imagines writing "**not** Not_Null". In general, "null" feels like the wrong name, null usually means absence of something and is at least partially usable, not something that exists but isn't usable. So we also eliminate choices 1 and 5.

We take a straw pool (vote for one) on the remaining choices:

3) Tuck, Randy, Jeff, Brad, Gary -- 5
4) Ed, Richard, Justin, J-P -- 4
6) Steve, Bob -- 2
Abstain: Tullio -- 1

Choice 6 has substantially less support. We take a straw poll between choices 3 and 4. Steve and Bob change their choice to choice 3. No one else tries to change their vote.

That's 7 for choice 3, and 4 for choice 4. That seems clear enough. So we choose "Is_Valid", "Invalid_Big_Integer".

Approve AI with changes: 8-1-2.

Bob is opposed. He is concerned that there are mistakes in this package that will cause problems during implementation, and he doesn't like "Big_Real" that can't represent PI and other irrational numbers. Gary, Jeff abstain.

AI12-0234-1/03 Compare-and-swap for atomic objects

Brad proposed this library based on the C library for GCC. Ed (??) notes that the AdaCore implementation would be built on top of that library, so we surely don't want incompatible semantics.

The type Test_And_Set seems over-specified. Brad says that the C library has three such types, 1, 2, and 4 bytes. [Brad notes after the meeting that he was misunderstood. He was talking about the bulk of the operations in the C library come in those sizes (and 8 and 16 bytes as well). A generic is needed to support those. However, there is only a single byte Test-and-Set type.] It would be better to leave the size to the implementers.

```
type Test_And_Set_Type is mod <implementation-defined>
   with Atomic;
```

Steve suggests that names ending with "_Type" are ugly. It is noted that members with naming complaints have to offer an alternative. After a few seconds of "ummm"s, he slinks away grumbling.

Should the fence routines be there? No, too target-dependent.

Program_Error should be raised if these aren't implemented on some architecture; no one is likely to be handling this exception so a dedicated exception is not needed.

It is suggested that test-and-set should be declared in one of the generics, not using a built-in-type. C doesn't have generics so it had to provide fixed size operations. [As noted above, Brad comments after the meeting that this applies to the other operations, not to test-and-set.]

Steve thinks this is too much. The critical need is for Compare-and-Swap, everything else can be built with it.

Randy thinks that atomic addition is a quite common need, and building it out of Compare-and-Swap is painful. Bob notes that if someone has a target with an atomic add, they would not want to have to jump through hoops to get access to it semi-portably. Someone notes that atomic increment is a more common atomic operation, perhaps the specification should support that directly. [Editor's note: When I said "atomic addition", I was thinking specifically about addition of a static value, not arbitrary addition of two objects. I believe the Intel chips support that, but not arbitrary additions of objects. Not sure how that would be represented in a generic; maybe a generic function with a formal **in** object providing a constant value.]

Tucker suggests splitting this into two AIs, one which is the bare minimum, and one is which is the rest.

Brad will take the AIs back and attempt to provide an update.

Approve intent: 10-0-1 (Gary abstains)

AI12-0250-1/02 Iterator Filters

Richard asks if the use of **all** in "`for all C in Integer when <blah> => xxx`" is confusing. We don't expect this to be used a lot with quantified expressions, since it could have been included in the expression.

Randy notes that the same is true for loop statements, which could easily use an inner if statement. Tucker notes that eliminating a level of indentation can make some code more readable.

The primary purpose of filters is for constructors (aggregates) and reductions. There is no alternative if the goal is to reduce a set of elements in an aggregate or reduction.

The primary objection is just a lot of complexity – Randy notes that Ada 2020 has around 39 different iterators, and arguably adding filters doubles that number. Iterators tend to be the hottest code in most programs, so implementers have to pay a lot of attention to the efficiency of the generated code (taking the easy way out is not a good idea).

Randy notes that this proposal does not allow array aggregates to use a filter on an index form. That's because the index iterator in an array aggregate is just giving a name to the index; it still follows all of the normal rules for a named array aggregate choice (including completeness checks). None of that makes sense for a filter (which would change the number of operations available). It would have to work much more like the container iterator choices.

He goes on to note that this will cause a problem for reduction expressions, since we won't syntactically allow filters in a value_sequence. He wonders if we have to allow filters in array aggregates, even though the rules (both static and dynamic) would have to be very different than the no filter case. [Editor's note: they could work exactly like the container iterators, and probably be restricted to a single range, which is already true for value_sequences for reductions.]

Tucker suggests an intent vote, since this reduction issue needs study. Moreover, the proposed wording here was written before the massive rewrites of 5.5 and 5.5.2 to insert parallel iterator semantics. This needs to be done again.

Approve intent: 9-0-2. (Randy, Jeff abstain).

AI12-0281-1/01 CPU affinity for Protected Objects

Supposedly this is already implemented in GNAT. There is skepticism about this. Much later, Gary notes that the GNAT documentation says that CPU can be used on a protected type. So it does appear to be implemented.

Tucker suggests that wording stay as singular rather than changing to plural. For instance,

D.16(10/3):

> The CPU aspect shall not be specified on a task {or a protected }interface type.

Bob: The second paragraph of the proposal seems to imply that if a task happens to be on the right CPU, it is OK. The actual rules require it to be assigned to that CPU. Tucker suggests using “assigned” rather than “executing”.

Brad notes that this prevents deadlock cases that can happen for a busy-wait lock. (But not all deadlocks, of course, an object still can deadlock by calling itself somehow.)

Tucker wonders if “starting a protected action” is well-defined. After much discussion, Tucker finds a definition of that term in chapter 9. So he withdraws the objection.

Jeff complains about D.16 (8.a/3). “Calling task for a protected type” doesn't make much sense.

Tucker suggests saying “...for a protected operation,}”. It is noted that the aspect is defined for a type, but since this is a short description, it is fine.

Ed wonders why you can't specify CPU for objects. He's thinking of an aspect CPU on an object declaration. You would use a discriminant of a type for that. That is the Ada 95 solution for Priority, which was copied here when CPU is defined. Tucker points out that a discriminant can be used for a heap-allocated object, while an aspect could not.

Someone notes that aspect CPU is not necessarily static. And in particular, it can be specified non-statically to have Not_A_Specific_CPU. So the compiler cannot, in general, eliminate the lock. It can only do it when CPU is specified statically, but that will be a common case. Indeed, it is required for Ravenscar/Jorvik programs.

Gary has a pile of typos: !discussion “initialization” and “finalization”.

!problem 2nd sentence “can[ ]not”. “multi[-]tasking”. “lock{-}[ ]free” (adjective). “deadlock{-}[ ]free”.

!discussion “straight[ ]forward”. Same line “lock{-}[ ]free”.

!proposal “stand{-}[ ]alone”

Approve AI with changes: 9-0-2 (Bob, Gary abstain)

AI12-0282-1/01 Atomic and Volatile generic formal types

In C.6(12), “atomic components” is ambiguous; it could be referring to any old components that are atomic, rather than just the Atomic_Components aspect. Thus, the wording should directly refer to “Atomic_Components”. Probably:

> If the Atomic_Components aspect is True for a generic formal type, then the aspect shall be True for the actual type.

C.6(6.1/3) is odd, in that is shuffles items around to get the effect rather than just deleting the extra “or”. So it appears to have more change than it really does. Use instead:

> For an object_declaration, a component_declaration, [or] a full_type_declaration{, or a formal_complete_type_declaration}, the following representation aspects may be specified:

Tucker suggests allowing this for Independent as well.

Randy notes that we would need a separate sentence about generic matching for “Independent” in that case; C.6(12) doesn't apply to Independent. (Should it? Dunno.)

Gary wonders if the formal derived type rule in C.6(12/3) needs to cover atomic components. Perhaps, it would be necessary to figure out what problem the derived type rule is trying to prevent. Maybe it cannot happen for an array component. He is given an action item to look into it.

Tucker suggests moving the generic matching rules into a separate paragraph. Steve suggests that we assign that to Tucker to wordsmith.

Approve intent: 10-0-1 (Bob abstains).

AI12-0297-1/01 Defaults for generic formal packages and formal in out objects

The parent, more important AI (AI12-0205-1) was put on hold, so this one surely has to be held.

Hold AI: 9-0-1. (Jeff abstains).

AI12-0305-1/01 Bounded Big Integers

We have had trouble deciding between a generic and a discriminant and how the capacity ought to be specified. It seems best to defer this one.

Hold AI: 9-0-1. (Brad abstains).

AI12-0311-1/02 Suppressing client-side assertions for language-defined units

Formatting will be like 11.5 (two lines each check).

“singlar” --> “singular” (multiple places).

Kill the last sentence of (1) in !discussion. Randy argues that the word “transform” is missing from that sentence, which sways no one.

Approve AI with changes: 9-0-1 (Bob abstains).

AI12-0314-1 Title of 13.13.1

There are now multiple packages in the subclause, so “The Package Streams” is not an appropriate title.

Typo in !corrigendum, the @ is missing on @dby.

Approve AI with changes: 10-0-0.

AI12-0315-1/01 Image attribute clause improvements

Fix the summary:

> Add a{n} introduction to the Image [A]{a}tt{r}ibute clause.

Two other places with "universal[ ]{_}integer"

> Allow Image attributes for entities with type universal[ ]{_}integer, and the [values can be the] prefix of an Image attribute {can denote a value}.

In the !problem, make a similar change "to allow the prefix of an Image attribute to denote a value ... and similarly for the rest of the sentence.

Approve AI with changes: 10-0-0.

AI12-0316-1/01 Preconditions for checking Task_Ids

Steve worries about defining a Boolean function that returns only one of True or False.

Bob suggests have it return Truth:

```
subtype Truth is Boolean range True .. True;
```

Someone suggests naming the function "Validate_Task".

There is a lot of discomfort with the idea of a Valid_Task function that does not return False. Users could call it, and the result would not be what they might expect from the name. The counter-argument is that this is an expression function, it's obvious what it does.

Tucker suggests putting the AI on Hold, since it isn't critical in any way (there's probably not a lot of SPARK using Annex D packages), and we're having trouble with the definition.

Gary "Preconditions [is]{are} used to check..."

Jean-Pierre: The function is "Is_Terminated" He also notes that the semantics of Is_Terminated raises Program_Error for a null task id, so the separate test for that case is not needed.

Hold AI: 11-0-0.

AI12-0317-1/05 Simplifying the rules for newly constructed objects

6.2(10): "[If For" should just be "[For"

Steve asks whether the implicit True of an if_expression is a constituent. It is not defined with an equivalence rule; the wording just says that the result is True in that case.

Tucker suggests a To-Be-Honest note to cover this. (If an if_expression does not have an **else** clause, "True" is a constituent of the expression and it can be the evaluated constituent.)

Jeff: !discussion "inent" (intent).

Ed: suggests dropping "out of thin air" from !proposal.

Approve AI with changes: 10-0-0.

AI12-0318-1/01 No_IO should exclude Ada.Directories

Subject “No_IO should apply to Ada.Directories”. As either include or exclude is going to be confusing (Directories should be included on the list of packages to be excluded).

Start the !discussion with:

> The modification to H.4(24/3)...

Add a summary!

Approve AI with changes: 10-0-0.

AI12-0319-1/01 Nonblocking for Unchecked_Deallocation is wrong

Randy notes that Steve brought up another issue (privately) yesterday – the blocking of the storage pool's Deallocate operation needs to be taken into effect. Another related issue is what requirements are placed on the Standard pool. These are related enough that they all should be addressed together. Randy will take this back, and we'll discuss it next time.