Minutes of Electronic ARG Meeting 58C

2 April 2018

Attendees: Steve Baird, Randy Brukardt, Jeff Cousins, Gary Dismukes, Bob Duff, Brad Moore, Jean-Pierre Rosen, Ed Schonberg, Tucker Taft, Tullio Vardanega (until 13:45 EDT).

Observers: Edward Fish (after 11:55 EDT), Justin Squirek.

Meeting Summary

The meeting convened on Monday, 2 April 2018 at 11:00 hours EDT and adjourned at 14:01 hours EDT. The meeting was held using Zoom. The meeting covered just a portion of the oversized agenda, but did cover the only regular AI.

AI Summary

The following AIs were approved with editorial changes:

AI12-0075-1/08 Static expression functions (10-0-0)
AI12-0220-1/03 Pre/Post for access-to-subprogram types (9-0-1)
AI12-0264-1/01 Overshifting and overrotating (8-0-2)
AI12-0265-1/01 Default_Initial_Condition for types (8-0-2)
AI12-0269-1/02 Aspect No_Return for functions reprise (7-0-2)

The following AIs were approved for intent, but they require a rewrite:

AI12-0112-1/03 Contracts for container operations (8-1-1)
AI12-0119-1/08 Parallel operations (9-0-1)
AI12-0266-1/01 Parallel container iterators (7-0-3)

Detailed Minutes

Previous Meeting Minutes

No one has any changes to the minutes. Approve minutes: 10-0-0.

Date and Venue of the Next Meeting

The next meeting is the in-person meeting in Lisbon, June 22-24, 2018.

ACATS Tests

Please volunteer at the Doodle poll for volunteers for ACATS tests at https://doodle.com/poll/f862izpz9nyp2y22. At this time, 5 people have volunteered.

Prioritization of Amendment AIs

Randy distributed the results of our previous poll on AI priority, just after midnight last Tuesday. He marked easy AIs in that listing.

Tucker had suggested that we split out the “thought to be easy” AIs and have a separate poll on those. He didn't vote for the low value/little work AIs, but these make various little clean-ups that would otherwise fall through the cracks. Randy says that there are 10-12 of these AIs.

We can use that to give those their own order (they help give us a sense of progress). This plan meets with general approval.

Randy has updated the color coded map of all of the Amendment AIs proposed for Ada 2020 with the changes from the last meeting. Find it at http://www.ada-auth.org/ai-files/grab_bag/2020-Amendments.html.

Scope of Revision:

We have to submit a scope (which usually is a list of AIs that we intend to include) to WG 9 for their June meeting. This is our last meeting before that list is due. Randy notes that we don't need to decide the exact list now, but we do have to decide on a plan so that Jeff and Randy can provide the list as required.

Randy notes that we can still drop things in the future, but we can't add anything not in the scope unless we can find a way to tie it to the things in the scope. So we can leave some AIs that we might not finish, but we should at least cut the list to a realistic length.

Tucker suggests dropping everything after the first 10 or 15 AIs. Randy notes that the 16th AI is AI12-0230-1, the Deadline Protocol revision. We probably need to give the real-time folks a bone. Jeff notes that there is a large gap in scores (18 to 12) between the 16th and 17th items.

Randy checks and notes that everything after the 16th AI was voted for by no more than 2 people (out of 13).

So drop everything with less than 15 points, that is not “easy” (marked as “simple” in the agenda, with a couple of additional issues). People can try to make a case to keep any of these if they wish.

Jean-Pierre speaks in favor in “coderivation”; he is surprised more people didn't vote for it. Randy notes that he and Steve tried to work out a solution to that one but it doesn't work for tagged types or dispatching. So he didn't vote for as he considered it insolvable and he had more than ten that he wanted to vote for. Perhaps others felt the same. We ask Jean-Pierre to make an attempt to come up with a solution.

Randy will send around a list of the AIs to be dropped, as a Letter Ballot change their status to “Hold”. For now, that will just defer them to Ada 2028 or whatever the next version of Ada ultimately will be. (We don't want to spend time now discussing them enough to actually kill them.)

Randy reminds the group that we are supposed to have a mostly finished draft to circulate next March -- only about 11.5 months from today.

Unfinished Action Items

We skipped this usual agenda item for this meeting. Members are reminded as always to do their homework.

Current Action Items

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

Raphaël Amiard:

- AI12-0214-1 (split into two AIs; help from Tucker Taft)
- AI12-0218-1 (get Thomas Quinot to write a detailed proposal, hold otherwise)
- AI12-0253-1 (assist Ed Schonberg)
- AI to make type marks optional in object renames (see discussion of AI12-0236-1 of meeting #58)

Steve Baird:

- AI12-0016-1
- AI12-0210-1 (get guidance from the SPARK group, then update the AI)
- AI12-0243-1

Randy Brukardt:

- AI12-0017-1 (assist Florian Schanda)

- AI12-0112-1
- Create straw poll for “easy” AIs.
- Create letter ballot to defer AIs with lousy scores.

Editorial changes only:

- AI12-0075-1
- AI12-0220-1
- AI12-0264-1
- AI12-0265-1
- AI12-0269-1

Alan Burns:

- AI12-0230-1 (with assistance from Tucker Taft)

Brad Moore:

- AI12-0119-1 (assist Tucker Taft)
- AI12-0234-1
- AI12-0266-1

Florian Schanda:

- AI12-0017-1 (with help from Randy Brukardt)
- AI12-0188-1 (see discussion of AI12-0189-1 in Vienna)
- AI12-0197-3 (lower priority than others)

Ed Schonberg:

- AI12-0253-1 (with help from Raphaël Amiard)

Tucker Taft:

- AI12-0079-1
- AI12-0111-1
- AI12-0119-1 (with assistance from Brad Moore)
- AI12-0191-1
- AI12-0214-1 (assist Raphaël Amiard)
- AI12-0230-1 (assist Alan Burns)
- AI12-0235-1
- AI12-0246-1
- AI12-0270-1
- AI for 'Value for composite types, similar to AI12-0020-1 (see discussion of AI12-0020-1 in Unfinished Action Items of meeting #58)

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 12 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

AI12-0075-1/08 Static expression functions

Tucker rewrote 6.8(5/4) to make better sense. Otherwise, the AI was unchanged.

Tucker would like to change the term to “type-invariant preserving”. After a bit of discussion we agree; “enforcing” is not the important principle (particularly for static analysis). There are multiple uses of the term that need to be changed.

Approve with AI with changes: 10-0-0.

AI12-0112-1/03 Contracts for container operations

Tucker thinks that the subheaders should include the preconditions/postconditions, as they are important to understanding the entire definition of the subprogram. Jumping back and forth between the spec and the description is not pleasant.

Tucker notes that nonblocking and global are of use for all tasking operations (not just parallel ones). He also notes that that the SPARK version of the containers had to define the additional reading routines. He agrees that it is important to have them here as well.

Bob thinks it is big waste of time. He admits that he probably would have to make the changes in GNAT, so it would have some effect on his work. Tucker asks whether he thinks it makes the Standard better. Bob agrees that it does.

Randy notes that one of the advantages of doing this for the containers is to show new Ada users how the various contracts would be used on real code. Ada programmers for decades have used the runtime packages as examples of Ada style, but we don't have anything in the Standard really using preconditions and postconditions.

Randy is asked how much time would be needed to do this. He thinks maybe 40 hours.

Bob will be willing to run a test version of this against the GNAT test suite. He worries, though that the old checks in the body would cover up the precondition checks. Tucker notes that setting the GNAT checking flag to False would allow testing without actually removing the existing checks. [Editor's note: probably that could be done using the existing GNAT suppression flag; that wouldn't affect the new preconditions yet.]

We look at some of Randy's questions by number.

(3) Tucker and Bob say they think it is OK to use Pre rather than Pre'Class. It's likely that any extended container would be used by itself; no one is using multiple kinds of containers and dispatching to them. (It would be different if we had defined an interface.)

(4) Randy wonders if it is OK to effectively prevent calling formal subprograms of containers packages for some operations, like the state queries. There seems to be no reason to call element equality from Length or Capacity, even though the English definition allows that generally. The group (at least Tucker) concurs that this is a good strategy.

(6) Is using prefix notation OK? Yes, do it when you need to, otherwise use regular notation.

(8) Tucker thinks we should avoid trying to do much tampering checking with contracts. Randy argues that the actual check needs to be in the preconditions (so it is suppressible); whether we need to try to describe the tampering state with contracts is a separate issue. Tucker is now confused; he would like to see possible alternatives to handling tampering checks.

Randy is asked if he has been updating the wording. Yes, he has, removing everything that can reasonably be described as a precondition. In a few cases, he's also been able to remove wording and move it into a postcondition

(taking advantage of the 1.1.3 rule). For instance, Move used to say that the Source was empty after a Move, but now that's just part of the postcondition. Several people say they approve.

We look briefly at the check suppression. Randy notes that we have to use a pragma Suppress, as we don't want users to have to recompile the runtime, and the Assertion_Policy is determined when a package is compiled. We also don't want implementers to have to repeat precondition checks, so the erroneous execution for a failed check is needed. We could of course define a new pragma for this purpose, but Suppress has the right meaning.

Randy had considered having a single check for all language-defined preconditions, but precondition_check sounds like *any* precondition, not just the language-defined one.

Edward suggests naming the check (after a bit of prompting with the usual terminology) "Disable_Language_Defined_Precondition_Check". That's crazy long; Randy had tried and rejected "Language_Defined_Unit_Precondition_Check" for the same reason.

Tucker asks some questions to Bob about the GNAT Container_Check. The GNAT pragma Suppress (Container_Check) only applies to the instance of a container. Tucker notes that since Suppress is a permission to omit, this seems compatible with GNAT's definition. Nobody has to follow a permission.

Randy is asked precisely what he is planning to do. He's just planning to do the containers (children of Ada.Containers), he has limits. (And probably in the order of definition if time doesn't permit doing all of them.) He's like to do this to the entire runtime, but that's too much this time around. Tucker notes that SPARK has annotated many of the numerics libraries – Tucker shows us an example. It would be nice to do those someday. We probably don't want to do that now unless we get a volunteer.

Approve intent of AI: 8-1-1. Bob votes against (still thinking it is too much work for the value); Gary abstains.

AI12-0119-1/08 Parallel operations

Tucker had suggested a reorganization of this wording – in particular, put more of the text into Chapter 9. He also would like to get rid of the term "executor".

Tullio says that tasks are user-level entities; "executor" exists only indirectly to the user. We shouldn't talk about it unless we have to.

Bob notes that we already have the term "processor" in Chapter 9, we don't need "executor" to mean virtually the same thing.

Jean-Pierre says that he agrees with Tullio.

Randy notes that he thinks that the bulk of the loop definition should be in Chapter 5; when these things are standard usage in the future, it would odd for them to be separated from other similar statements.

Jean-Pierre notes that the part that is related to tasking should be in Chapter 9, and that is OK. He continues that most of the subprogram definitions are in chapter 6, but some is in chapter 9 (entries, protected subprograms).

Tucker and Brad should take this and work out how to organize this. Randy notes that Tucker will be the lead, since this is a word-smithing task rather than a design task. He also notes that putting two people equally in charge of a task sometimes leads to both saying that they were waiting on the other to start first, so by unofficial policy he no longer allows co-homework.

Jeff and Justin note that interweaving the rules with the usual loops seems fine because in 10 years that will be norm.

Tucker suggests moving some of the description to chapter 9, leaving the basic wording in chapter 5.

Steve notes "objects of the loop parameter" is weird. Steve will send comments to ARG so that Tucker and Brad can those into account.

If we don't need "tasklet" or "executors", then we'll get rid of them. (Randy notes that we currently have four similar concepts: task, tasklet, processor, executor – we ought to be able to do better than that.)

Approve intent of AI: 9-0-1. Bob abstains.

AI12-0220-1/03 Pre/Post for access-to-subprogram types

Steve notes that one could use a wrapper once for each type – it doesn't require a wrapper for each subprogram.

This is just extra checks; there is no requirement that they match. (SPARK may be different, of course.)

The point of 'Access is where any matching of the contracts would happen.

Gary suggests that we change the subject name to make it clear that this only applies to named types.

Tucker muses that we could allow dotted names for aspects to deal with anonymous types. Definitely not this AI.

Steve asks whether the naming change is safe. Randy notes that it isn't hard to construct a case where the meaning of the name would change, using Size or Alignment. Tucker notes that any such use would have to be static, and no parameter name is static, so such an incompatibility would always be caught at compile-time. We can't think of any aspect allowed on named access-to-subprogram types that could cause a runtime inconsistency.

We would be OK with a compile-time incompatibility from the naming, it's pretty pathological. Add some discussion about the incompatibility.

“Intent: ...” is an author's note. Either delete them or mark them as “author's note”.

The last sentence of 39/3 should be an AARM Implementation Note.

Remove the question from the vicinity, too.

Approve AI with changes: 9-0-1. Jeff abstains; Tullio had voted in favor before he left.

AI12-0264-1/01 Overshifting and overrotating

Typo: ... We use [the] iteration ...

Steve would prefer to just answer the question without describing the operations. Randy notes that this isn't as easy as just noting a result: for rotate, the Amount is effectively moded by *n*, for the usual shifts, the answer is zero, for the arithmetic shift, the answer is either 0 or -1. The algorithms given in the AI allow deriving the answers. After discussion, he decided to withdraw his suggestion, as he just has a vague idea and the AI has a concrete proposal.

Approve AI with changes: 8-0-2 Steve, Bob abstain.

AI12-0265-1/01 Default_Initial_Condition for types

Tucker claims that we don't need 'Class on these contracts, because you always know the type for default-initialization. There is no such thing as a dispatching call.

The inheritance rules then make sense; you can add new checks but can't get rid of ones belonging to ancestors; these are all **and**ed together.

Add the wording from the penultimate e-mail for 1.1.3(17.1/5).

Copy the example from the e-mail into the !example section.

Approve AI with changes: 8-0-2. Brad, Bob abstain.

AI12-0266-1/01 Parallel container iterators

This defines parallel iteration for containers.

[The Editor's internet fails at this point – comes back up 10 minutes later – some discussion is missed here.]

Tucker and Brad discuss the Advised_Split parameter to Split. They agree that this is an upper bound on the split and not a mandated amount of split.

Tucker does not like returning a classwide Chunk_Array. That seems expensive. It would be better if Chunk_Array was an array. Can you have an array of formal incomplete types? No.

If we made it a child generic, then Chunk_Array could be a formal parameter. But that's annoying to instantiate (and implicit instantiation did not make the priority cut).

Tucker suggests that the length function would tell you how many cursor “chunks” are requested. If there aren't enough elements for the chunks, then empty iterations (start = finish) are returned for the extras.

Ed wonders how filters work here. In general, you'd want to filter as part of the iteration; if the filter is expensive, you'd want it to be parallel. Moreover, you don't want to read the element twice, and most filters depend on the element. There are some special cases possible, but it doesn't seem that they'd be likely enough to have a special mechanism for that. (One could imagine a set of Booleans to select the elements to return, but that seems like more overhead than just iterating and skipping the iteration code when filtered out.)

Tucker suggests using a procedure Split, rather than a function Split. That would at least eliminate returning a class-wide object.

Tucker wonders why we have a parallel reversible iterator. Brad notes that we're adding “parallel” to a (sequential) reversible iterator. So these iterators support (sequential)“forward”, (sequential) “reverse”, and “parallel” (no real order).

Brad will keep this one.

The wording in some places says that there is a “set of loop parameter objects”. Use that description everywhere (it changes in the text).

Approve intent: 7-0-3. Jean-Pierre, Gary, and Bob abstain.

AI12-0269-1/02 Aspect No_Return for functions reprise

We discuss whether we want to add additional permissions to the basic rule that Tucker proposed. In particular, do we want to allow omitting the return statement? Bob argues that there doesn't seem to be any reason to treat these functions specially. If we were going to relax that rule, we need to do it more generally than just this case. But we haven't been able to agree on that in the past. No one argues in favor of making a change to the return rule specifically for this case.

Tucker asks about allowing other kinds of return so long as they are preceded by a raise statement. Randy notes that this won't be a commonly used feature; he would prefer that it is as simple as possible without impacting the rest of the RM. Tucker notes that the feature he suggested seems like flow analysis, which Ada has never asked Ada compilers to do for Legality Rules. There is no enthusiasm for this change.

The discussion which says “we could go further” should be “we considered”.

Steve worries that the wording is too vague; it seems to apply to any return statement nearby (nested, for instance). The wording should say that the rule only is enforced on return statements that “apply to” the nonreturning function. Ed asks about extended return statements. Those could be made to work, but again let's stay simple.

> Any return statement that applies to a nonreturning function or generic function shall be a simple_return_statement with an expression that is solely a raise_expression or a call on a nonreturning function.

We discuss whether we need the word “solely” in this wording altogether too long. Several people claim we don't need it (why was not recorded). The ultimate conclusion is not crystal clear (it seems to depend on who was recording it) , but more people thought that we were dropping it than keeping it.

Approve with changes: 7-0-2 (Steve, Jeff abstain)