Minutes of Electronic ARG Meeting 60G

9 May 2019

Attendees: Steve Baird, Randy Brukardt, Jeff Cousins (joined 11:15 EDT), Gary Dismukes, Bob Duff, Brad Moore (joined 12:01 EDT), Jean-Pierre Rosen, Ed Schonberg, Justin Squirek, Tucker Taft, Tullio Vardanega (joined 12:09 EDT).

Observers: Quentin Dauprat, Richard Wai.

Meeting Summary

The meeting convened on Thursday, 9 May 2019 at 11:05 hours EST and adjourned at 13:56 hours EST. The meeting was held using Zoom. The meeting covered most of the AIs on the agenda that are associated with existing features.

AI Summary

The following AI was approved:

AI12-0328-1/03 Meaning of limited type and record type in 4.5.2(28.1/4) (9-0-0)

The following AIs were approved with editorial changes:

AI12-0191-1/08 Clarify “part” for type invariants (11-0-0)
AI12-0303-1/02 Some constants must be covered by Global aspects; extensibility (9-0-2)
AI12-0304-1/02 Image attributes of language-defined types (11-0-0)
AI12-0330-1/01 Add items to the Glossary (11-0-0)
AI12-0331-1/01 Order of finalization of a subpool (9-0-0)
AI12-0332-1/01 Implementation Permission for Default_Initial_Condition (9-0-0)

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

AI12-0240-6/02 Global aspect and access types used to Abstract Data Types (6-0-5)
AI12-0326-2/01 Bounded errors associated with procedural iterators (10-0-1)
AI12-0333-1/01 Predicate checks on out parameters (11-0-0)

The following AIs were discussed and assigned to an editor:

AI12-0302-1/01 Default Global aspect for language-defined units

Detailed Minutes

Welcome

Steve welcomes everyone.

Apologies

Tullio Vardanega said that he had a conflict and might join an hour late. Brad Moore had a last minute conflict and also might join an hour late. Erhard Ploedereder (whose apology got stuck in the Internet until after the meeting) had classes to teach during the meeting and could not attend.

Previous Meeting Minutes

No one has any changes to the minutes (all previously sent suggestions have already been applied). Approve minutes: 8-0-0.

Date and Venue of the Next Meeting

There isn't sufficient time before our next face-to-face meeting for another electronic meeting, so our next meeting will be June 14-16 in Warsaw Poland, immediately after the Ada-Europe conference.

Richard asks about joining the meeting electronically. We should be able to arrange such access. He wonders about the three day schedule. It is explained that we usually meet for parts of three full days (including a sometimes 10 hour day in the middle) when we meet face-to-face.

Possible Schedule Slip

AdaCore has requested more time to prototype and evaluate Ada 2020 features. There is a concern that we could be standardizing features that need significant changes to be usable in practice.

Randy asks how long a delay we want. AdaCore discussed internally one to two years.

We discuss how to communicate this to WG 9. (Since WG 9 controls the schedule, not the ARG.) Jean-Pierre suggests that AdaCore directly ask WG 9 about this request. We (the ARG) do not require a delay, so it is weird for us to ask for one. Others think that it would be more political for the ARG to ask for the delay. In any case, Pat Rogers is already aware of the request.

Tucker thinks that we should create a “committee draft”, essentially the document that we would submit now (when we've finished the open issues). We then can orient future work to be on top of that relatively stable platform.

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.

Current Action Items

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

Steve Baird:

- AI12-0016-1
- AI12-0326-2

Randy Brukardt:

Editorial changes only:

- AI12-0191-1
- AI12-0303-1
- AI12-0304-1
- AI12-0330-1
- AI12-0331-1
- AI12-0332-1

Tucker Taft:

- AI12-0240-6
- AI12-0302-1
- AI12-0333-1
- Create an AI for a Modifies aspect (see discussion of AI12-0302-1)

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 19 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

AI12-0191-1/08 Clarify “part” for type invariants

The summary still uses “screened”, but that's gone. The new term is “components of the nominal type”.

Gary wonders if “nominal type” is defined. It isn't, and the term is “components of the nominal type”, “nominal type” is not used by itself. Steve notes that we could end up with conflicts between that and this term. Tucker would hope that we'd be smart enough to avoid that. [Editor's note: There's a first time for everything. Grin.]

If we wanted to, add right after the nominal subtype: “The nominal type is the type of the nominal subtype.” Some people want us to do that. The editor needs to check if any existing uses of “nominal type” use some different definition.

Approve AI with changes: 11-0-0.

AI12-0240-6/02 Global aspect and access types used to Abstract Data Types

The idea is to think of these objects made up as a bunch of pieces as a single object. We then only need to reference that single object in Global aspects, not the individual parts. If we have to cover the parts individually, given that they're private and hidden from clients, we can only use the most general specifications (covering the body of the package if the pool is declared there or just **all**). That kills all parallelism.

Gary notes two misspellings: “Nothwithstanding” in the Legality Rules. The third word of the Erroneous Execution section is spelled “Erronous”.

Ed wonders if the description of what compound objects allow dereferences is too vague. The formal rules are given in the Legality Rules. We don't try to check that there is a compound object that can actually reach the value being dereferenced, as that would sometimes run afoul of privacy. (Dynamic checks have no such problem.) Tucker notes that in theory at least, it would be possible to test reachability and thus detect the cases of erroneous execution. But such checks would be expensive, as in general each dereference would require walking the graph of references for every visible compound object. [Editor's note: that's not practical without an additional aspect and some additional rules requiring the absence of cycles in the graph; the aspect is needed to allow “back pointers” such as previous in a doubly-linked list to exist without creating cycles].

Tucker notes that we don't allow non-limited non-controlled types, as uncontrolled assignment would make it impossible to control the use of the pointers (particular to enforce one-per-object rules). However, we don't try to check for problems as that proved to be quite complex to do properly.

Jeff wonder if there is “suppress” somewhere; having that explicit would make the source of the erroneous execution clearer. He thinks that might ease some of Richard's concerns. But it seems strange to “suppress” checks that aren't really checkable. Richard says that he was simply concerned that these are unchecked, which doesn't seem like the way Ada does things.

Randy notes that he would like to propose a checked version to build on top of these, we should have time to do that during the delay. But he notes that we need a framework to build on top of, and some (reasonable) designs might not be allowed by a restrictive scheme.

Approve intent: 6-0-5 Abstain: Brad, Jeff, Tullio, Justin, Ed

AI12-0302-1/01 Default Global aspect for language-defined units

There's no wording here. There is an open issue as how to deal with **in** parameters that are handles where the underlying data structure gets modified (like random numbers and files).

Randy notes that Tucker used the list of Ada 2012 packages, he hasn't added the Ada 2020 packages to the list in the introduction of Annex A yet, and those need to be covered here.

Randy suggests that we decide whether we should define a Modifies aspect, or whether to use Global for this purpose. Global is a broader club, there is clearly more information with Modifies. Two files would conflict if we used existing Global features to handle them, whereas a version with Modifies could allow them to be used in a parallel construct.

Tucker will create a separate AI with “Modifies”. He'll then update this AI to use Modifies and list the missing packages.

Jean-Pierre notes a typo, 6 paragraphs before the Pure group: “the retured reference”.

Tucker suggests not to worry about typos since much of this text is exploring the parameter problem and we're going to handle that separately and get rid of the bulk of this text.

Keep alive: 11-0-0.

AI12-0303-1/02 Some constants must be covered by Global aspects; extensibility

Tucker notes that the wording we settled on for AI12-0332-1 would be a better choice here, too:

> Implementations may extend the syntax and semantics of the Global aspect in an implementation-defined manner.

Gary wonders why “different” is in square brackets in the !proposal. That's to say SPARK's rules are different from the ones given here.

Gary notes a misspelling: “accomodate” in the third paragraph of !proposal.

Jeff notes a typo in the third paragraph of the !problem:

Such constants have to be included {in} the Global {aspect} for subprograms that read them, ...

In the example in the !problem:

```
type Object_Type is new Ada.Finalization.Limited_Controlled {with}[is] record

-- Global => null {is}[has to be] OK here [(and I believe it is)], we're
-- reading a by-copy component of the parameter without any
-- dereference. [There's]{There are} no possible side effects here.

-- left {as}[for] an exercise for the reader.
```

Approve AI with changes: 9-0-2 Bob, Jean-Pierre abstain.

AI12-0304-1/02 Image attributes of language-defined types

We had seen this before, why is it back? Randy had some questions that we forgot to address last time, and he still would like to discuss them. [Editor's note: I suspect that the reality was that I forgot to change the status of the AI, my message said that I didn't intend to bring this back. And I had totally forgotten about this one already, so I had to go fishing to see why it still was here. But I look better with the initial explanation!]

We have a long discussion about the meaning of “Implementation Requirements”. Eventually, we agree that the Implementation Advice text is better as a requirement. So let's change all of the “should”s to “shall”s and make this a requirement.

Randy also suggests that the AARM note at the end here be rewritten as Implementation Advice. An implementation can still ignore it if they think some type is too obscure.

We briefly discuss what this rule means vis-a-vis the cursors for the containers. We don't really have a conclusion outside of the Vectors, since the image of an access type is not generally useful to the reader. One could show the designated element, but that isn't really right, either. Randy suggests that there is nothing that would really be meaningful.

> *Implementation Advice*
> For each language-defined private type T, [implementations are encouraged to ensure that] T'Image {should} generate[s] {an} images that would be meaningful based only on the relevant public interfaces, as opposed to requiring knowledge of the implementation of the private type.

Gary: AARM Note: For any language-defined scalar type T, T'Put_Image should not be specified; the Image attribute needs to return the language-defined image for such types. This is important for compatibility: the Image attribute {has}[have] been available for scalar types for many Ada revisions, and programs can (and do!) depend on its behavior.

The first AARM Note should start with a capital 'N'.

Randy asks about “imaginary” in Generic_Complex_Types. It is a private type whose full type is specified to be a real type. We discuss it for a while, but we decide it is not important enough, and the Advice covers it well enough.

Approve AI with changes: 11-0-0.

AI12-0326-2/01 Bounded errors associated with procedural iterators

Allows_Exit => Parallel means that the subprogram can and must be used in a **parallel** loop.

Steve wonders how the unhandleable exception model works if a task contains the call on the loop body procedure. That usually would cause the task to silently disappear. Tucker thinks that the implementation would have to propagate the unhandleable exception to the activator. Steve doesn't think that works – where is the exception raised if the task is allocated, for instance? We couldn't decide how to make that work for exceptions in general, it seems unlikely that we (or any implementer) could do it here.

Someone suggests just banning the use of tasks in such a case. But Randy had suggested that someone might want to implement parallelism with the pool of tasks (especially if it needs to be compatible with Ada 2012 or earlier) – Brad's libraries do this. That seems reasonable, so simply banning tasks isn't going to fly.

Tucker suggests we should make that a Bounded Error. We agree that is the best solution. Tucker suggests that Steve propose some wording.

Approve intent: 10-0-1. Gary abstains.

Randy says to skip the other alternative on the agenda, since we probably aren't ready to vote it No Action.

AI12-0328-1/03 Meaning of limited type and record type in 4.5.2(28.1/4)

Tucker describes the changes.

Randy notes that there is an ACATS test like this. Steve notes that this is incompatible. Randy notes that GNAT gets this completely wrong using a bit compare even for limited types that in theory have no such compare, so it is already going to be incompatible in a worse way. However, one of the ACATS tests will have to be withdrawn as Tucker's Legality Rule makes it illegal.

Approve AI: 9-0-0.

AI12-0330-1/01 Add items to the Glossary

Randy asks about the definition of logical thread of control. Tucker suggests defining it by context, and italicizing the first use in 9(1/5).

Default Initial Condition
A property that [should hold]{holds} for ...

“turn off” doesn't sound like RM wording (even informally).

Suppress
{Permits disabling}[To turn off] a run-time check ...

Bob suggests:

Suppress
{A request to the implementation to disable}[To turn off] a run-time check ...

Tucker suggests:

Container Aggregate
A construct used to define {a}[the] value {of}[for] a type that represents a collection of elements, by explicitly specifying the elements in the collection.

We discuss Stable Properties whether “many” is the right word. This is informal, and the intended use is when “many” or “most” operations are involved. So we make no change.

Jean-Pierre notes that this AI should say there is no ASIS effect. And there's no ACATS tests needed, either.

Gary: remove hyphen from “pre-existing” in the summary.

Approve AI with changes: 11-0-0.

AI12-0331-1/01 Order of finalization of a subpool

Change the lead-in:

> ...has the following effects {in the given order}:

Gary notes that in !discussion: “ceasee” (second usage).

Tucker complains that the last item is rather passive. Randy notes that the technical term is “belong to a pool”. Steve notes that there isn't an “unbelong” verb. Someone suggests using “cease” here:

> 4. The subpool ceases to belong to any pool.

Approve AI with changes: 9-0-0.

AI12-0332-1/01 Implementation Permission for Default_Initial_Condition

Tucker thinks this wording is weird. We look at the wording proposed for Global:

> Implementations may extend the syntax for global_aspect_definition, as well as any of the syntactic categories that comprise it, in an implementation-defined manner.

A version of that is preferable:

> Implementations may extend the syntax or semantics of the Default_Initial_Condition aspect in an implementation-defined manner.

In the ACATS test section, “Argubly”.

Approve AI with changes: 9-0-0.

AI12-0333-1/01 Predicate checks on out parameters

Steve describes the problem: the rules as written require predicate checks on all inbound **out** parameters of composite types (since these are defined to use subtype conversions, and subtype conversions enforce all checks, exclusions, and predicates). That's a problem if the object is partially or fully uninitialized, as the check may be reading junk.

He goes on to note that there have been two suggestions for handling this case. We could leverage the existing initialization rule and make the checks only in the cases that default initialization is checked. Or, an option he finds appealing, we could simply say that no **out** parameters get predicate checks on the way in.

Tucker changes his opinion, the never check **out** parameters is simple. Randy notes that it matches what we do for scalar types. No one is supposed to be reading **out** parameters anyway, why should they care about the predicates? The reason we do subtype conversions for composites is to ensure that the bounds and discriminants are compatible with any constraint, as that could change the representation of the parameter (that is, the code generated). No such issue exists for predicates, they have no effect on representation. Steve suggests that checking a predicate on an inbound **out** parameter is similar to applying a predicate to the left-hand side of an assignment statement; that seems like clear nonsense.

There would be no change on applying other checks in this case (some of those probably could be omitted as well, but figuring out which ones seems like a can of worms best left unopened). And there would be no change for **in out** parameters or on the checks on out parameters upon return from the subprogram.

Ed notes that this matches what GNAT actually does in this case, so it probably is more compatible in practice. (Although we haven't checked if any other implementations implement predicates, specifically the newish ObjectAda, which implements a portion of Ada 2012.)

We agree to never check **out** parameters. Tucker will draft wording for this.

Approve intent: 11-0-0.