Minutes of the 56th ARG Meeting

8-10 October 2016

Pittsburgh, Pennsylvania, USA

Attendees: Raphael Amiard, Steve Baird, Randy Brukardt, Jeff Cousins, Gary Dismukes, Brad Moore, Erhard Ploedereder, Jean-Pierre Rosen (Saturday afternoon via Google Hangouts); Florian Schanda, Tucker Taft.

Observers: Pat Rogers (Saturday only).

Meeting Summary

The meeting convened on Saturday, 8 October 2016 at 13:08 hours and adjourned at 12:45 hours on Monday, 10 October 2016. The meeting was held in Salon 1 at the Pittsburgh Marriott City Center. The meeting covered all of the normal AIs and some of the amendment AIs on the agenda.

AI Summary

The following AIs were approved:

AI12-0004-1/03 Normalization and allowed characters for identifiers (5-0-4)
AI12-0125-3/08 Add @ as an abbreviation for the LHS of an assignment (8-0-1)

The following AIs were approved with editorial changes:

AI12-0058-1/04 The Fortran Annex needs updating to support Fortran 2008 (9-0-0)
AI12-0164-1/03 Max_Entry_Queue_Length aspect for entities (8-0-1)
AI12-0171-1/01 Ambiguity in Synchronous_Task_Control semantics (8-0-1)
AI12-0179-1/02 Failure of postconditions and language-defined units (10-0-0)
AI12-0186-1/03 Profile freezing for the Access attribute (7-0-2)
AI12-0193-1/02 Postcondition failure for a task entry (9-0-1)
AI12-0196-1/05 Concurrent access to Ada container libraries (9-0-0)
AI12-0199-1/01 Abstract subprogram calls in class-wide invariant expressions (7-0-2)
AI12-0200-1/01 Improve reentrancy requirements for language-defined subprograms (9-0-1)
AI12-0203-1/01 Overriding a nonoverriddable aspect (10-0-0)
AI12-0204-1/01 Renaming of a prefixed view (8-0-0)

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

AI12-0064-2/04 Nonblocking subprograms (9-0-0)
AI12-0111-1/04 Tampering considered too expensive (9-0-0)
AI12-0127-1/06 Partial aggregate notation (9-0-0)

The following AIs were discussed and assigned to an editor:

AI12-0119-1/02 Parallel operations
AI12-0139-1/03 Thread-safe Ada libraries
AI12-0197-3/03 Generator Functions

The intention of the following AIs were voted, but then were discussed again later in the meeting (the final results are above):

AI12-0186-1/02 Profile freezing for Access attribute (6-0-4)

Detailed Minutes

Welcome

The WG 9 meeting has given us two agenda additions: we need to recommend an ARG program of work.

Also, AI12-0125-3 was returned to us for further consideration.

Apologies

John Barnes, Alan Burns, Bob Duff, Jean-Pierre Rosen, Ed Schonberg, and Tullio Vardanega apologize for not being able to attend.

Previous Meeting Minutes

Approve minutes: 9-0-0.

Date and Venue of the Next Meeting

Our next meeting will be associated with the 2017 Ada-Europe conference, June 12-16, 2017, in Vienna, Austria. The WG 9 meeting will be the morning of June 16, and the ARG meeting will start the afternoon of the 16th and run until early afternoon on June 18th. Steve Baird notes that he won't be able to attend those dates (his daughter graduates from college that weekend). That was discussed during the WG 9 meeting, and the Ada-Europe organizers indicated that they strongly preferred a meeting following the conference. We'll have to do without Steve.

The following meeting would normally be associated with a HILT conference, but none is planned for 2017. We discuss briefly some possible locations for the meeting (and WG 9 as well). Several European members express a preference for an east-coast location. If we pick later dates (such as December or January, which better balances our schedule), there also is a preference for a warmer location. Someone laments that Greg Gicca no longer is in the Ada industry (the two meetings he hosted in St. Pete Beach certainly solved the warmer location issue). A suggestion is made that Pat host us in the Houston area again. That led to a discussion of the amazing cab fares from the Houston International Airport (which is a long ways from Houston). Jeff states that the cab fare was more than his air fare to Madrid or Pisa. Someone suggests that it was probably cheaper for Randy to drive to/from the meeting from Wisconsin than it was to take a cab to/from the airport to the meeting. [Editor's note: For the record, that wasn't true: it cost Randy about $300 in gas, oil, and tolls to drive to and from the meeting; the cab fare was reported to be about $125 each way. OTOH, with lower gas prices today, it might be true now.]

No decision about that meeting was made at this time; it's more than a year in the future.

Program of Work

WG 9 asks to recommend a program of work (which WG 9 will turn into instructions to the ARG).

It is suggested to start from the 2009 *Instructions to the ARG* document.

After some scrambling, Tucker finds the old document and pulls it up on his laptop (hooked to the very expensive [see Thanks] projector).

We want to change the order of bullets (move containers from first to third, and multicore goes from third to first).

Should anything be added? Someone suggests improving interfacing to C++ and Python. That doesn't get much support (most feel that it would be helpful, but not of the highest importance).

Randy notes that we've repeatedly been asked by some national bodies (at the WG 9 level) to improve Internationalization support. We should add a bullet about that, as the fourth bullet:

- Enhanced support for Unicode;

Drop the first paragraph of the introduction, we ended up ignoring that anyway.

We turn to discussing the schedule. Tucker notes that we won’t get the instructions until 2017. Should we do a smaller Amendment in 2018? Randy notes that customers don’t like version updates; it seems cruel to do one for not much language. Tucker notes that a new revision is a lot of work.

Tucker suggests just dropping the schedule for the reply to WG 9. A discussion on this topic would be useful, probably on the WG 9 list.

Prioritization of AIs

Randy notes that some AIs that less obviously fit into the intended program of work are sucking up a lot of discussion time, both at meetings and on the ARG list. He notes that he didn't finish readying all of the new AIs in part because of time and mental energy spent on the generators proposal.

It’s not clear how to apply prioritization ahead of time on AIs.

Jeff notes that we have a system of Low, Medium, and High priority for AIs. He tries to run the meeting so that we cover the higher priority items first. Randy says that we currently set that by a guess when the AI is created, and we tend to give almost everything Low priority.

There is no reason that we can't change these priorities as we have more information. We ought to consider updating these priorities for any AI that seems to have the priority wrong.

Randy should prioritize his work by the AI priorities as much as possible, especially if he can't finish it all.

Thanks

Thanks to Randy Brukardt for taking the minutes.

Thanks to Jeff Cousins for whatever he does. [Editor's note: I can only record the resolutions, no matter how silly they are.] He's the Rapporteur and runs the meeting, of course.

Thanks to Tucker Taft for making the arrangements; the pretzels were amazing (and so was the amount he paid to rent the projector)!

Unfinished Action Items

AI12-0016-1 and AI12-0020-1 will get done when Steve gets around to them.

Steve says that AI12-0075-1 needs discussion for an alternative approach. We add it to the agenda (but never get to it).

Erhard notes that the only change requested for AI12-0139-1 was to change the name from Safe to <TBD>. He would like to discuss the other details first. We add this AI to the agenda.

Randy hasn't worked on AI12-0112-1; he plans to work on it soon. He's been waiting for Nonblocking, Global, and stable properties to get closer to being finished; all of those will be used in the updated annotations. (The plan is to use that AI to provide the values of the new annotations, so that won't have to be done in the individual AIs for the new features.)

Jean-Pierre notes that AI12-0197-2 (passive tasks) doesn't need to be tied to generators, so he didn't update it now.

Florian says that he ran out of time. AI12-0017-1 is big, and will like help. Randy volunteers to help. Priority is lower than others. Florian notes that he thinks it will be important for future versions of SPARK, and it makes sense for Ada as a whole.

He believes that container aggregates are valuable, and he will work on them in the next cycle.

Tucker says that he also ran out of time, especially as he forgot to take his laptop to Pittsburgh. So he lost the day before the deadline as it was shipped to him. Thus he didn't work on AI12-0079-1, AI12-0191-1, and parallel reads.

Current Action Items

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

Raphael Amiard:

- AI12-0139-1 (determine which language-defined units would benefit from having task-safe versions)
- AI12-0197-3 (with help from Steve Baird)

Steve Baird:

- AI12-0016-1
- AI12-0020-1
- AI12-0075-1 (hold: asked for an ARG discussion, which has not happened)
- AI12-0127-1 (with Florian Schanda)
- AI12-0197-3 (assist Raphael Amiard)
- New AI for Container aggregates (help Florian Schanda; see discussion of AI12-0189-1 at meeting #55)
- New AI to generalize nonoverridable to allow values (see discussion of AI12-0164-1) [This was assigned AI12-0206-1 after the meeting.]

Randy Brukardt:

- AI12-0017-1 (assist Florian Schanda)
- AI12-0064-2 (with help from Tucker Taft)
- AI12-0112-1
- AI12-0187-1 (write wording)

Editorial changes only:

- AI12-0058-1
- AI12-0164-1
- AI12-0171-1
- AI12-0179-1
- AI12-0186-1
- AI12-0193-1
- AI12-0196-1
- AI12-0199-1
- AI12-0200-1
- AI12-0203-1
- AI12-0204-1

Brad Moore:

- AI12-0119-1

Erhard Ploedereder:

- AI12-0139-1 (develop rules for the most important units identified by Raphael Amiard)

Jean-Pierre Rosen:

- AI12-0197-2

Florian Schanda:

- AI12-0017-1 (with help from Randy Brukardt)
- AI12-0127-1 (assist Steve Baird)
- Provide better examples for AI12-0197-3
- New AI for container aggregates (with help from Steve Baird; see discussion of AI12-0189-1 at meeting #55)
- Test container operations noted by Erhard using a lot of data. (See discussion under Unfinished Action Items at meeting #55)

Tucker Taft:

- AI12-0064-2 (assist Randy Brukardt)
- AI12-0079-1
- AI12-0111-1
- AI12-0191-1
- Parallel reads for existing containers (see discussion of AI12-0139-1 in Madrid, possibly related to AI12-0111-1)

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 9 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

AI12-0004-1/03 Normalization and allowed characters for identifiers

Randy explains the result of his latest trip down the Unicode rabbit-hole. The AI now recommends that all Ada source be in Normalization Form C, which eliminates redundant ways of encoding source. This matches the WC3 recommendation for Internet content.

We then add a Legality Rule that identifiers only include characters that could appear in text that is normalized to Normalization Form KC. There is a Unicode property associated with this, and we've added functions to Ada.Characters.Handling and the wide versions thereof to return the value of this property for any character.

Approve AI: 5-0-4.

AI12-0058-1/04 The Fortran Annex needs updating to support Fortran 2008

The declaration for Double_Complex is missing in the !wording.

Add after B.5(10):

```
package Double_Precision_Complex_Types is
   new Ada.Numerics.Generic_Complex_Types (Double_Precision);

type Double_Complex is new Double_Precision_Complex_Types.Complex;

subtype Double_Imaginary is Double_Precision_Complex_Types.Imaginary;
```

B.5(21):

> ... declarations {are permitted} for the character types corresponding to Fortran character kinds 'ascii' and 'iso_10646', which in turn correspond to ISO/IEC 646:1991[,] and to UCS-4 as specified in ISO/IEC 10646:2011 [are permitted].

In B5(31), “new Integer8” should be “range 1 .. 220”. Brad notes that the example is trying to show that one wants to define your own type. Perhaps just drop that example, as it doesn’t show enough different from regular Ada code.

Usually, the “with” is on a separate line and indented; make that change to Standard_Deviation.

Gary: in !discussion

> ... entities are described {in C}[to C functions], by defining structs and functions to create, manipulate, and inquire descriptors and the objects they describe[, s]{. S}imilarly interoperation with Fortran modules {(}which are similar to Ada packages{)} are not mentioned.

Gary: in !summary. “...implementation {advice}[device]...”

Tucker suggests a more radical change to the summary:

> {In the Fortran Annex, update}[Update] obsolete references and remove implementation {advice}[device] that is considered to be bad practice[from the Fortran annex], and add better support for double precision complex arithmetic.

Approve AI with changes: 9-0-0.

AI12-0064-2/04 Nonblocking subprograms

> ... This aspect is inherited by overridings of dispatching [subprograms]{operations, unless directly specified}.

Get rid of the Editor’s note following.

We do not need to say that “that S statically denotes a subtype”. Delete that and the following Editor’s Note.

Erhard asks if an entry that has a queue size of 0 is nonblocking. Steve notes that such an entry could requeue to a routine that does have a queue.

AARM Reason that starts “This in turn...” should be deleted.

We discuss the problem with tagged equality. We look at an example:

```
type T tagged null record with Nonblocking => True;
function "=" (Left,Right : T) return Boolean with Nonblocking => True;

type T2 is new T with record
    X : ZZ; -- Nonblocking specified as False for ZZ.
end record;
-- function "=" (Left, Right : T2) return Boolean
--    needs with Nonblocking => False;

A, B : T'Class;

if A = B then -- This promises to be nonblocking
```

However, if A and B are both objects of type T2, the actual equality called (dispatched to) will be blocking. That's not going to work.

Florian suggests that we allow Nonblocking on types, and that specifies the value for implicit declarations and default for primitives of a type.

We need to try to make equality nonblocking as much as possible.

Tucker starts musing about how we could make a stream routine Nonblocking.

```
procedure Blah_Write (S : access Root_Stream; ...)
    with Nonblocking => S'Nonblocking;
```

Randy notes that S'Nonblocking has to be static, as this controls Legality Rules. We don't know the nonblocking value for an actual parameter until the routine is called.

> AARM Ramification: specifying Nonblocking {as}[is] False imposes no requirements. Specifying Nonblocking {as}[is] True imposes additional compile-time checks to prevent blocking, but does not prevent deadlock. {A} pragma Detect_Blocking can be used to ensure that Program_Error is raised in a deadlock situation.

Approve intent: 9-0-0.

AI12-0111-1/04 Tampering considered too expensive

Tucker explains the major changes to the AI. Stable was changed to a nested package. Various deferred constants were eliminated as they don't work in that form.

Change the second paragraph of the summary to:

> Replace any prohibition of "tampering with elements" checks with a prohibition of "tampering with cursors" checks for containers of elements of a definite type. For containers of an indefinite type, any prohibition of tampering with elements remains.

We need the "tampering of elements" checks for indefinite elements, because if we change the complete element it can change shape. The typical implementation will deallocate the existing element and allocate a new one. We have to allow such an implementation.

Erhard complains that removing that check makes the parallel version less safe because two tasks can then write the same element (which is not allowed but otherwise undetected). That's not the purpose of the tampering check; it is only accidental when it detects such errors. [Editor's note: Specifically, calling Replace_Element on the same element from two different tasks is never detected by the tampering check; while Replace_Element is a tampering with elements routine, it does not itself prohibit tampering.]

Tucker says that for the indefinite containers the stable view would not have operations that tamper with elements.

The stable view prohibits tampering with cursors for the associated "unstable" view; the idea is the overhead for doing that is incurred only once. Tampering with cursors is prohibited until the stable view is destroyed.

Raphael implemented this for Vectors, this was mostly easy, a few changes were needed to the package. He had some issues with coextensions, but using those isn't necessary. If the stable "owns" the underlying vectors, then it has to deallocate them, but that's easy since the type is controlled anyway. We just have avoid deallocating a

coextension; Randy notes one way to make sure that doesn't happen is to use pragma Restrictions (No_Coextensions);

Typo in !wording from Gary: "the Stable version{s} of these..."

Steve notes that !proposal "Two essential properties {are}[is] ..."

What needs to be done on this AI? We need the wording sections for the other kinds of container. We need real wording to handle the changes to tampering with elements; the indefinite versions depend on the wording in the definite versions (they're just differences from the definite versions), and just deleting the wording from A.18.2 as proposed in this draft leaves it defined nowhere. Randy thinks that moving the tampering with elements wording to A.18.11 is going to be very different than anything that exists since A.18.11 just defines the differences from A.18.2 (there are very few today).

Approve intent of AI: 9-0-0.

AI12-0119-1/02 Parallel operations

Tucker wants to talk about parallel blocks. He wondered if there should be a declare part to the block. Usually, either it is local to one branch, or needs to live after the end. It would only be useful for local communication between parallel operations, which probably isn't very common. Maybe not enough to bother with.

Tucker discusses completion. He says that it is common to want to return from one of these tasklets; think of multiple searches being done, and as soon as one finds an answer then it returns.

The model is that one of the tasklets completes, and then other tasklets stop immediately. Which means that an exception might be lost; it would be as if that tasklet didn't execute to the point of the exception.

If a tasklet is changing global variables, they would need to be synchronized.

Steve wonders if ATC could be expressed using a parallel block. Seems like it.

Erhard notes that this is a fork or join operation. In normal execution, you would have to wait for all three to get finished. It's odd that exceptions and other control flow doesn't follow this model.

Obviously, you can't get rid of exceptions. And you would hardly want to complete the other tasklets if there is an exception.

Raphael notes that he agrees with Erhard, it seems unusual to allow an early return. But that is a common use case.

Steve notes that this has all of the same problems as ATC; you have to write all of your code to be abort-safe. Assignment to a global would disqualify it (that's because it could be aborted and then further use might be erroneous).

One could have exit to mean to go to the end of block. (That doesn't work for regular blocks because it's not backwards compatible, but that would be OK here.)

Randy notes that it doesn't matter what the other tasklets do if one has failed. Florian notes that even in sequential code, you can't tell where it dies (Tucker points out 11.6).

Steve wonder if we should give the user control, as to whether the branches have to stop at the same time (like "with abort"). Randy notes that in such a case we could require no unsynchronized writes if "with abort" is used, and allow them otherwise.

Steve asks Jeff if ATC is allowed in their applications. No, they don't allow ATC.

Florian wonders if the annotations would be an issue. He notes that SPARK can infer annotations. Tucker notes that tools can infer annotations, but they have to be able to output them. Perhaps have a Synchronization Policy to suppress the legality checks.

Moving on to parallel loops. Brad notes that this is a more difficult problem to solve. Most of the trouble comes from loops that need to do a reduction (which is a large fraction of them).

Various options for the syntax (where parallel goes); we'll ignore that for now 'cause it isn't important.

Brad explains the Reducer aspect.

Tucker thinks that this looks like an automatic race condition. He prefers:

```
declare
   Min_Value : array (<>) of Integer
```

```
      with Reducer => Integer'Min := Integer'Last;
   Max_Value : array (<>) of Integer
      with Reducer => Integer'Max := Integer'First;
  Sum       : array (<>) of Integer
      with Reducer => "+"         := 0;
begin
   for parallel (Min_Value, Max_Value, Sum) I in 1 .. 1_000_000 loop
      Min_Value(<>) := Integer'Min(@, Arr(I));
      Max_Value(<>) := Integer'Max(@, Arr(I));
      Sum(<>)       := @ + Arr(I);
   end loop;
end;
```

[We quickly diverge to a brief discussion how the "unnatural" @ seems to have appeared in this example. It seems that even people who don't like the syntax seem to use it as soon as they can. Tucker admits that he seems to have hoisted himself on his own petard.]

Florian suggests:

```
parallel
   Min_Value : array (<>) of Integer
      with Reducer => Integer'Min := Integer'Last;
   Max_Value : array (<>) of Integer
      with Reducer => Integer'Max := Integer'First;
   Sum       : array (<>) of Integer
      with Reducer => "+"         := 0;
for I in 1 .. 1_000_000 loop
   Min_Value(<>) := Integer'Min(@, Arr(I));
   Max_Value(<>) := Integer'Max(@, Arr(I));
   Sum(<>)       := @ + Arr(I);
end loop;
```

Randy notes that this eliminates the need to repeat the reduction objects for the arrays in the loop header; that's good.

Each chunk has its own accumulator; there is a lot of "magic" going on behind the scenes.

Brad notes that the reduction function used after the loop is not necessarily the same as the combiner used within the loop. For instance, for concatenation the combiner in the loop could be Append and reduction function could be &.

Florian writes an example of summing an array:

```
My_Sum := 0;

parallel
   Sum : Integer (<>) := 0;
for I in 1 .. 1_000_000 loop
   Sum (<>) := @ + A(I);
reduce
   My_Sum := Sum(<>)'reduce("+");
end loop;
```

Some people suggest that block should be just "**begin**". Someone suggests "**then**".

```
My_Sum := 0;

parallel [(Num_CPUs*2)]
   Sum : array (<>) of Integer := 0;
for I in 1 .. 1_000_000 loop
   Sum (<>) := @ + A(I);
then
   My_Sum := Sum(<>)'reduce("+");
end parallel loop;
```

Steve asks whether you are allowed to query the bounds of the array. (Answered later.)

Someone suggests that we don't really need 'reduce:

```
My_Sum := 0;

parallel [(Num_CPUs*2)]
   Sum : array (<>) of Integer := 0;
for I in 1 .. 1_000_000 loop
   Sum (<>) := @ + A(I);
then
   for S of Sum loop
      My_Sum := @ + S;
   end loop;
end parallel loop;
```

The middle loop expands into two loops:

```
for Chunk_Index in Sum'Range loop
   for I in Split(1..1000000, Chunk_Index, Num_CPUs*2) loop
      Sum(Chunk_Index) := @ + A(I);
   end loop;
end loop;
```

Brad notes for an array with no reduction, the reduce block and declarations can be omitted.

```
parallel
for I in 1 .. 1_000_000 loop
   A(I) := @ * 2;
end parallel loop;
```

This seems like a way forward.

Steve wonders how tasklets interact with the Task_Id package. Tasklets belong to a task, and have the same id.

Florian wonders if the reducer block executes after an exit. He thinks that it would be valuable, because you don't want to assign to a global object in the loop for a search.

```
parallel
for I in 1 .. 1_000_000 loop
   if A(I) = Target then
      exit; – Tucker suggests exit with Index := I;
   end if;
end parallel loop;
```

Raphael thinks it is really an implicit generic.

Erhard suggests:

```
My_Sum from Sum with "+";
```

but we still don't have the needed keywords.

In the **then** block, the array is a normal unconstrained array (with 'First, 'Last, etc.)

Keep Alive: 9-0-0.

AI12-0125-3/08 Add @ as an abbreviation for the LHS of an assignment

Tucker noted that some people at AdaCore thought that this was not "Ada". Thus it was sent back to us from WG 9. "It looks like a combination of Ada and Python."

Florian says he likes the @. Raphael says that he doesn't particularly like @, but it was the best idea we have now.

Someone suggests that "The left-hand side is where the value is at, and the sign for at is @."

Randy heard a similar but slightly different explanation: "The Assignment Target can be abbreviated AT, and @ is the AT sign."

We turn to the endless list of alternatives to @.

We start with the idea of "<Target>". That might be confused with the greater than and less than operators. Florian thinks a lot to write. Raphael notes that is would force people to not use the shorthand when it is better to avoid it.

"`:+`" is less powerful than @.

"`()`" looks like zero. It's also often used as a placeholder in examples and partially written code.

"<>" is ambiguous in an aggregate. It also could cause confusion with the greater than and less than operators.

"`[]`" works, but it seems like a waste to use a set of bracketing characters for this. It seems like we'll have a better use someday.

"`[target]`". Gary thinks that including an id in it makes it harder to read.

"<<>>" doesn't-require a new lexeme. It's also extremely visible, and it looks better in text. We digress into how important the font of presentation is. We decide that that's not in our field of concern.

We look at what the best ideas look like when used as a name:

```
<<>>(I)
<<>>.Disc
[](I)
[].Disc
@(I)
@.Disc
```

Steve wonders if we should consider making the renaming aspect of this feature more obvious. A number of ideas on that line are suggested:

```
LHS : This_is_a_very_long_name := LHS + 1;
   -- Looks like a type in an object declaration
This_is_a_very_long_name is LHS := LHS + 1;
This_is_a_very_long_name is LHS in := LHS + 1;
(declare LHS is This_is_a_very_long_name := LHS + 1)
This_is_a_very_long_name [VLN] := VLN + 1;
```

Gary does not like having to choose another name for the shortcut. [Editor's note: Bob sent an e-mail concurring with this opinion shortly after this discussion.]

We take a "can live with" straw poll:

@	6-1-2
<<>>	7-1-1
[]	5-2-2
name [id]	5-4-0

Looks like the last two should be eliminated.

Straw poll: Prefers @ -- 3; Prefers <<>> -- 3; abstain 3.

That's so conclusive that we decide to head to lunch and try again afterward.

After lunch, the discussion continues.

Florian asks Tucker precisely what "not Ada" means? Tucker suggests that it is readability.

Erhard notes that <<>> doesn't work with > or < very well.

```
This_is_a_very_long_name := (if <<>> > 5 then Foo else Bar);
```

That's especially true if there are no spaces. We digress into a discussion about whether we care about the readability of poorly formatted code. No conclusion is drawn.

Tucker suggests using "`(<>)`".

```
This_is_a_very_long_name := (<>)(I).Field + 1;
```

Is this ambiguous with an aggregate? No, because a choice is required in an aggregate.

Florian says that he prefers a solution with a single character; any syntax is hard to understand for unfamiliar users.

Tucker says that he does not agree.

A preference straw poll: 4 prefer @, 1 prefers <<>>, 4 prefer (<>).

Eliminate the loser, and retake the poll: 5 prefer @, 4 prefer (<>).

This result was predictable.

Randy thinks that (<>) looks like an aggregate (even though it is not an aggregate). That harms understandability.

Steve suggests that we need a greater consensus before changing the AI.

It is suggested is to pass the AI by comp.lang.ada.

Tucker wonders if it is would better to unify this with **let** somehow. Randy doesn't like this idea.

Randy should present this proposal to comp.lang.ada and ask for comments and alternative suggestions (preferably not already rejected), wear fireproof suit.

Keep alive: 8-0-1.

On Monday morning, Jeff asks to reconsider this AI. This AI was discussed at dinner Sunday night by the subset of the ARG that was present. The dinner group worried that opening this to the public would just rehash all of the ideas and arguments that we already have had. After all, Randy has counted more than 30 options that we've considered at and since the Vermont meeting, and we considered some additional ideas at this meeting. And there were additional ideas before that.

In addition, two members that are not here (Ed and Bob) supported the current AI on the ARG list. In addition, Ed says that most people get used to it rapidly and quickly depend on it. (Case in point: Brad's parallel loop and block examples.)

Keeping this open is just making work for ourselves.

We have looked extensively and have no better ideas. Most people get used to it rapidly (as Tucker showed by using it in the parallel loop examples he wrote for us yesterday – and he was the person that wanted to reconsider it!).

Randy wonders if we ought to make the term "assignment target" to better motivate the @ (AT sign). Apparently, he is the only one that thinks that is a good idea.

Approve AI as is: 8-0-1.

We discuss what should happen at WG 9. We suggest that we recommend that it be brought to a separate WG 9 vote (not lumped in with a dozen other AIs), and Tucker agrees to allow it to be brought to such a vote. (Consensus does not mean unanimity!)

[Editor's note: The subject of the AI was "a abbreviation", which has been corrected here and in the AI, so the AI was changed – by one letter.]

AI12-0127-1/06 Partial aggregate notation

Change the title of the AI to "Delta aggregates".

Florian explains the changes to the AI: the multidimensional array case was removed; record and array were syntactically split.

Randy notes he changed much of the text to use the syntax forms.

Change, and reorganize the resolution wording:

> For a record_delta_aggregate, the expected type and {any} applicable index constraint of the expression in each record_component_association {are}[is] defined as for a record_component_association occurring within a record aggregate.
>
> For an array_delta_aggregate[s], the expected type and {any} applicable index constraint of the expression in {an} array_component_association {are}[is] defined as for an array_component_association occurring within an array_aggregate of the type of the delta aggregate. The expected type for each discrete_choice in an array_component_association is the index type of {the} type of the delta aggregate.

Randy says that 4.3.3 (10-15.1) defines applicable index constraints; that does not belong here. 4.3.3(14) clearly applies to the expressions of delta aggregates. So strike all of that text. Giving:

> For a record_delta_aggregate, the expected type of the expression in each record_component_association is defined as for a record_component_association occurring within a record_aggregate.

> For an array_delta_aggregate, the expected type of the expression in an array_component_association is defined as for an array_component_association occurring within an array_aggregate of the type of the delta aggregate. The expected type for each discrete_choice in an array_component_association is the index type of the type of the delta aggregate.

"...shall be {one}[1]." This is stilted.

> For an array_delta_aggregate, the type of the delta aggregate shall be a nonlimited one-dimensional array.

Drop the paragraph about the array base expression.

The ramification to needs to say type rather than *base*_expression.

> For a record_delta_aggregate, the order of the subsequent assignments is unspecified. For an array_delta_aggregate, the subsequent assignments (including all associated expression evaluations) are performed in the order in which the array_component_association_list is given.
>
> AARM Ramification: Within a single array_component_association, the order of assignments is unspecified.

4.3.3(17/5) needs some updating to support the possibility that the association occurs in a array_delta_aggregate.

The notion that a single association don't have overlapping choices isn't handled, because that was handled globally by 4.3.3(18/5).

Randy asks that we check all of the rules for record and array associations, for similar problems. He especially wonders about the each component only appears once ("needed") rules for records.

This sort of detail seems like a job for Steve Baird, so he is the co-author given the lead on this (hopefully last) revision of the AI.

Approve intent of AI: 9-0-0.

AI12-0139-1/03 Thread-safe Ada libraries

Change the name from Safe to Task_Safe.

Steve asks if one global lock is sufficient. Tucker notes that the question is what do callbacks allow?

Erhard suggests that every call is atomic; they're like protected procedures. They'd be implemented as protected actions, so callbacks have to be nonblocking. Randy notes that would be a usage difference from the "normal" containers; changing wouldn't be quite as easy as changing the prefix.

Do separate containers have separate locks? Yes. The wording must make that clear.

> {The i}[I]mplementation should provide for mutual exclusion at a {fine}[high] degree of granularity{, at a minimum of at least one lock per object of each type with task-safe operations. For containers, the Stable nested package will acquire exclusive access to the underlying upon creation, with no further synchronization associated with individual operations.}

Randy notes that a parallel loop that modified the same element more than once would still have a race condition with these rules. Someone notes that such a loop is already suspicious; it would be a problem for the "usual" semantics anyway (as which version of the element would be the final one would be nondeterministic).

The idea is that there task_safe versions, that the user would use when they need that. So using some "regular" packages and some Task_Safe packages would be the usual use case. Thus, it would be best if as many of the types, exceptions, and constants are shared as possible.

> We define the package Ada.Task_Safe as a new root for a duplicate of the entire Ada library and postulate thread-safeness for entire content of the package, including child units. Some of these we might decide must be simply renames. Others we might allow them to be renames, depending on the implementation.

One has to look at type-equivalence between task_safe and normal on a package-by-package basis. It's probably obvious for the pure packages. Steve adds that in some cases these are generic.

Steve asks to kill the AI. He thinks it is a morass. He does not get any immediate support.

Randy suggests that we concentrate on the containers, as we've had requests for that. Tucker thinks the others work naturally (no changes would be needed to the language or implementations). Randy says that's silly, and gives the example of Text_IO. Some compilers intersperse letters when Put_Line is called from multiple tasks.

Tucker suggests that is something that we should simply say that it is not allowed, for both the Task_Safe and classic versions of the library. Randy says that's fine, but the important point is that we need to make these requirements explicit. We cannot generalize this approach. The blanket rule approach is wishful thinking.

So we need to prioritize what we work on: probably Text_IO, containers.

Steve asks if packages that have global state are covered by A(3). Randy notes that AI12-0052-1 answered that very question, see AARM A(3.b.1/4).

Erhard wonders if the last paragraph of A.18.X described Stable. It's broader than that.

Send back to Erhard; for e-mail discussion.

Raphael will do a review of all of the existing packages to see which ones would benefit from having task-safe versions.

Erhard will concentrate his efforts on the most important packages to make task safe. (Informed by Raphael's work.)

Keep alive: 7-0-2.

AI12-0164-1/03 Max_Entry_Queue_Length aspect for entities

On Saturday, Randy notes that Pat is here and can give us some guidance on AI12-0164-1. Is this aspect important to extended Ravenscar? Pat answers that it isn't used directly by extended Ravenscar (the associated restrictions are lifted), but it is important for usages of extended Ravenscar, in order aid scheduleability analysis. It is especially important to have a finer granularity of specifying the queue length of entries.

On Sunday, we consider the AI itself.

Typo in !proposal "for a{n} entry declaration".

Drop the Editor's Note.

Merge the two definitions of the aspect:

> For an entry_declaration, a task type (including the anonymous type of a single_task_declaration), or protected type (including the anonymous type of a single_protected_declaration), the following language-defined representation aspect may be specified:

Why -1? The existing restriction allows Max_Entry_Queue_Length = 0, which means nothing can be queued. Clearly, the default is unbounded, and that is different than nothing.

But we need wording so the value can't be -2.

> If aspect Max_Entry_Queue_Length for a type has a nonnegative value, aspect Max_Entry_Queue_Length for an individual entry of that type shall be a nonnegative value less then or equal to the value of the aspect for the type.

What does -1 mean for this aspect? It means that no additional restriction is applied.

> If directly specified, the aspect_definition shall be a static expression no less than -1. If not specified, the aspect has value -1 (representing no additional restriction on queue length).

Steve asks about the following case:

P1 is a protected type with Max_Entry_Queue_Length 5, P1.E has Max_Entry_Queue_Length 3.

type P2 is new P1 with MEQL => 2.

Is this illegal? We think it is. Steve thinks that this is just a ramification. Tucker thinks this needs wording.

Change the Legality Rule to:

> If the Max_Entry_Queue_Length aspect for a type has a nonnegative value, the Max_Entry_Queue_Length aspect for every individual entry of that type shall not be greater than the value of the aspect for that type.
>
> AARM Ramification: This can apply to a derived type if Max_Entry_Queue_Length is specified on the derived type.

Erhard worries that this would require a different implementation for a derived task body. After some discussion, we agree that there is a problem.

We consider using nonoverridable here, but that doesn't work without work. 13.1.1(18.2/4) says that those are all names. Other rules depend upon that.

Randy and Tucker argue about whether the restriction applies to types or objects. Tucker's position is that types don't belong to partitions, so the restriction only applies to objects of the type. Randy feels that allows types that could never be used. Florian notes that for static analysis one would want to check the types, not the objects.

Erhard suggests that the types appear in every partition where they are used; one can't imagine an object in a partition without a type.

The Post-compilation rule also needs "shall not be greater than".

> If a restriction Max_Entry_Queue_Length applies to a partition, any value specified for the Max_Entry_Queue_Length aspect specified for the declaration of a type or entry in the partition shall not be greater than the value of the restriction.

Steve will take an AI to generalize nonoverridable to allow values (as in here). [This was assigned AI12-0206-1 after the meeting.] We add the following to the Legality Rule:

> The Max_Entry_Queue_Length aspect of a type is nonoverridable.

Tucker asks to add an editor's note:

> [Editor's note – nonoverridable needs to be generalized to allow its usage with a numeric-valued aspect.]

We discuss the lead-in text. Tucker wants to change it, but Randy objects to changing the wording that is standard in Annex D. Steve concurs with Randy, we want to be consistent. Tucker finally relents.

We consider making the header as follows:

> For a task type (including the anonymous type of a single_task_declaration), protected type (including the anonymous type of a single_protected_declaration), or any of their entries, the following language-defined representation aspect may be specified:

But eventually we decide to leave this alone.

Tucker will send his edited version of the AI so Randy gets all changes.

Approve AI with changes: 8-0-1.

AI12-0171-1/01 Ambiguity in Synchronous_Task_Control semantics

Steve complains that wording suggests Program_Error happens, or the other things happen. But the other task might do something else on this list.

> It is a bounded error for two or more tasks to call Suspend_Until_True on the same Suspension_Object concurrently. For each task, Program_Error might be raised, the task might proceed without suspending, or the task might suspend, potentially indefinitely. The state of the suspension object might end up either True or False.

Typo: "fundemental" is misspelled in the discussion.

Approve AI with changes: 8-0-1.

AI12-0179-1/02 Failure of postconditions and language-defined units

The checks associated with any such postcondition or type invariant …

"runtime" should be "run time".

Tucker wants to eliminate "failure".

Replace the second paragraph with:

> The evaluation of any such postcondition or type invariant expression shall either yield True or propagate an exception from a raise_expression that appears within the assertion expression.

Erhard thinks this is too low-level to be in 1.1.3.

So replace all of this in 1.1.3 with:

> The implementation of a language-defined unit shall abide by all postconditions and type invariants specified for the unit by this International Standard (see 11.4.2).

And all of the previous wording and notes goes into 11.4.2 (in an Implementation Requirement)

Approve AI with changes: 10-0-0.

AI12-0186-1/02 Profile freezing for Access attribute

This has to be changed to a Binding Interpretation.

Improve the summary:

> Using the access attribute does not freeze the profile of the associated access type (nor of the subprogram denoted by the prefix); that only happens upon a call of a value of the access type, use as a generic actual parameter of an instance, the occurrence of a body, or at the end of the declarative part.

Erhard notes that any expressions involved in a subtype get elaborated at the point of the type declaration, so they have to be frozen. But the name in the subtype_indication does not freeze, by AARM 15.b. This rule won't do that. Gack.

```
package P is
   X : constant Integer;
   type A is access Rec(3*X); -- Elaborates X, must be frozen so it is illegal.
private
   X : constant Integer := 77;
   ...
end P;
```

Nothing is frozen for an access-to-subprogram; but access-to-object has to freeze the constraint but not the access type.

Tucker will try again. Difficulty should be Hard.

Approve intent of AI: 6-0-4.

AI12-0186-1/03 Profile freezing for the Access attribute

On Sunday, we look at the new wording Tucker sent overnight.

> At the place where a subtype is frozen, its type is frozen. At the place where a type is frozen, any expressions or names within the full type definition cause freezing{, other than the name that is the subtype_mark used in specifying the designated subtype in an access-to-object type definition, or expressions or names that appear within the designated profile of an access-to-subprogram type definition}; the first subtype, and any component subtypes, index subtypes, and parent subtype of the type are frozen as well. For a specific tagged type, the corresponding class-wide type is frozen as well. For a class-wide type, the corresponding specific type is frozen as well.

Randy notes that the elaboration of a subtype_indication is not necessarily at the place of freezing of a subtype. So this doesn't make any sense.

13.14(8) says that expressions are freezing where they occur (other than a few exceptions that don't apply here). So these constraints freeze immediately. We don't need a special rule to handle those.

Erhard wonders why we need that first sentence in the first place. For other kinds of types (arrays, records, etc.)

So revert this wording to the text from version /02 of the AI:

> At the place where a subtype is frozen, its type is frozen. At the place where a type is frozen, any expressions or names within the full type definition cause freezing{, other than those that occur within an access_type_definition or an access_definition}; the first subtype, and any component subtypes, index subtypes, and parent subtype of the type are frozen as well. For a specific tagged type, the corresponding class-wide type is frozen as well. For a class-wide type, the corresponding specific type is frozen as well.

We discuss commas in this text for a while. No change is made.

We ask the editor to include a discussion about the constraint case that confused us at the end of the discussion. (That is, the expressions of a constraint already caused freezing, so we don't care about them for this rule.)

Approve AI with changes: 7-0-2.

AI12-0193-1/02 Postcondition failure for a task entry

The date and version were not changed on this AI, should 2016-10-06, /02.

It doesn't make sense to talk about 'Old here. But we need to say something about the cleanup of such objects. Change the middle of the replacement wording to:

> Execution of the rendezvous consists of the execution of the handled_sequence_of_statements, performance of any postcondition or type invariant checks associated with the entry, and any finalization associated with these checks, as described in 6.1.1 and 7.3.2.

Change difficulty to Medium.

Approve AI with changes: 9-0-1.

AI12-0196-1/05 Concurrent access to Ada container libraries

Steve's AI12-0200-1 (which we discussed Saturday) redoes A 3(4), so that shouldn't be in here.

In the wording "container object" should be replaced by "element" (Brad left the "object").

Steve says that he finds that it is important that "just use a cursor" implies that the operation does not tamper with cursors. Others don't think that helps, so make no change for that.

Fix the A.18(2.a) Ramification: "A cursor is not considered to overlap with {other elements of} the associated container, ...

5.o {and concurrent calls to operations of the container have overlapping parameters.}

In A.18.2(125), reentrant is spelled wrong, and "is {not} considered to [not] overlap with any [other] object Redundant[(including itself)]".

Tucker would like to talk just about overlap – he thinks that "reentrant" is talking about the wrong time. After all, the text in A says that all language-defined routines are reentrant. So instead use:

> For the purposes of determining whether the parameters overlap in a call to To_Cursor, the container parameter is not considered to overlap with any object Redundant[(including itself)].
>
> AARM Ramification: Without the preceding rule, concurrent calls to To_Cursor on the same container would interfere by the reentrancy rules in Annex A, since the container object of the concurrent calls would overlap with itself. We want these to not interfere, for example to allow the Vector elements to be split into separate "chunks" for parallel processing.

[Editor's note: I changed the above to a Reason rather than a Ramification in the AI, since it is mainly talking about why we have the rule.]

Make a similar change to A.18.2(133/3):

> For the purposes of determining whether the parameters overlap in a call to Replace_Element, the Container parameter is not considered to overlap with any object Redundant[(including itself)], and the Index parameter is considered to overlap with the element at position Index.

Similar changes need to be made to the remaining wording (they're all the same).

Does this change to Replace_Element work for indefinite containers? Yes, so long as the (implicit) allocator is task-safe. We digress into a discussion of whether the implementation-defined storage pool has to be task-safe. (It overlaps in concurrent calls, after all.) We decide to not decide this question.

Make this a binding interpretation.

Approve AI with changes: 9-0-0.

AI12-0197-3/03 Generator Functions

This draft is based on Steve Baird's syntax proposal. This makes a generator a type. There also is an abstract generator type.

The "abstract" generator should have **abstract** in the syntax. Steve notes that this isn't like an abstract type, in that it is indefinite, meaning that you can declare objects of it (necessarily initialized by a type conversion).

It's not a tagged type, but the operations dispatch. Why isn't it tagged? Then one could have a generator interface which is much like the synchronized ones (similar rules).

Steve wants these generated explicitly, as the accessibility is defined there. If 'Class is used, the accessibility comes from the specific type.

Since dispatching operations are involved it makes more sense that these are tagged. If they're tagged, we can have a new kind of interface. Then most of the rules follow from existing interface rules. Concrete generators would "implicitly" derive from such an interface, so conversions would be allowed. Seems like it would work.

We turn to the unanswered question of whether **yield** should be allowed in nested units. Several people say that this seems important. But then one needs rules to handle if the wrong guy yields. (Assuming only units that can yield that are statically enclosed by a generator type.)

Steve suggests that 'Access could be used to pass a subprogram out to some other task. Tucker doesn't think that can happen.

Jean-Pierre suggests that problems would happen from nested generators:

```
generator Outer
   procedure Foo is -- includes yield
   generator Inner
      Foo; -- which generator is yielded?
   end Inner;
end Outer;
```

Florian suggests that not allowing yield in nested subprograms is acceptable. Steve isn't sure, since traversing recursive data structures (such as trees) need such things.

Florian says that Python only allows directly given yields, and then uses recursive generator calls. There is concern about the overhead of that.

Tucker tries to figure out how a recursive nested procedure would work. Steve says that it either **yield**s or completes (which completes the generator). Tucker thinks it needs to be declared specially with some special syntax to support allowing **yield**.

Florian suggests banning most of the problematic cases (like nested tasks).

Florian notes that recursive generators handle the recursive tree walk case:

```
generator body Walk_Tree (N : Tree_Node) yield Tree_Node is
begin
   yield N;
   for C in N.Children loop
      for Element in Walk_Tree(C) loop
         yield Element;
      end loop;
   end loop;
end Walk_Tree;
```

But that is going to create a new stack (context) for each recursive call, which would kill performance.

Jean-Pierre asks what happens if Next is called by itself. Steve suggests that it would raise Program_Error. Could we ban naming of itself within the body? Perhaps, but that would be very different than the normal Ada practice (where there almost always is an expanded name for every object).

We definitely need more examples (Florian already has an action item for that). We need those to really determine the value of any of the alternatives here.

Keep Alive: 8-1-1.

Jeff voted against because he does not believe it is worth the substantial effort (both here and for implementers) that it will take to include it.

AI12-0199-1/01 Abstract subprogram calls in class-wide invariant expressions

Tucker added a definition of corresponding expression; it applies only to nonabstract descendants, in order to avoid the problems.

Erhard wonders why we have this concept at all. An example is shown:

```
type T(D : Discrim) is tagged private
     with Type_Invariant'Class => Valid(T);

function Foo(X: T) return Discrim is abstract;

function Valid(X : T) return Boolean is
begin
    return X.A > X.B;
end Valid;

type T1 is new T with private;

-- function Valid(X : T1) return Boolean;
```

The inherited Valid is called for invariant checks on T1. If it had been overridden, the new routine would have been called. Randy notes this is necessary for static binding; see AI12-0113-1 for the reason for the whole corresponding expression semantics. This AI is just applying that idea to Type_Invariants (which is clearly necessary for the same reasons).

Tucker notes that you could have:

```
type T1(D3 : Fun; D4 : Discrim) is new T(D4) with private;
```

Thus we have the two bullets:

- References to T are replaced by with references to T1;
- References to discriminants of T are replaced with references to the corresponding discriminant of T1, if any, or to the specified value for the discriminant when the discriminant is specified by the derived_type_definition for some type that is an ancestor of T1 and a descendant of T (see 3.7).

We don't need to handle non-discriminant components, because this has to be a private type (not the full type). Randy thinks about this and disagrees. If the parent has visible components, those could be references to those components in the type invariant expression.

```
type To is tagged record
    A, B : Integer;
end record;

type T is new T0 with private
     with Type_Invariant'Class => T.A > T.B;
```

So replace the first bullet with:

- References to nondiscriminant components of T (or to T itself) are replaced with references to the corresponding components of T1 (or to T1 as a whole);

AARM Note: The only nondiscriminant components visible at the point of such an aspect specification are necessarily inherited from some nonprivate ancestor.

Gary: No hyphens in non-abstract.

The entire corrected wording:

> If the Type_Invariant'Class aspect is specified for a tagged type *T*, then [the invariant expression applies to all descendants of *T*] {then a *corresponding expression* also applies to each nonabstract descendant *T1* of *T*

Redundant[(including *T* itself if it is nonabstract)]}. {The corresponding expression is constructed from the associated expression as follows:

- References to nondiscriminant components of *T* (or to *T* itself) are replaced with references to the corresponding components of *T1* (or to *T1* as a whole);
- References to discriminants of *T* are replaced with references to the corresponding discriminant of *T1*, if any, or to the specified value for the discriminant when the discriminant is specified by the derived_type_definition for some type that is an ancestor of *T1* and a descendant of *T* (see 3.7).

Gary: drop 3 hyphens in the !question: “re-interpreted”, “non-dispatching”, “non-abstract”.

Replace the !summary with:

Class-wide type invariants do not apply to abstract types, to avoid various problems. Define the notion of a "corresponding expression" for a class-wide type invariant, replacing references to components as appropriate, taking into account rules for corresponding and specified discriminants when applying them to a nonabstract descendant.

!discussion also has “non-abstract”. Gary is in fine form this morning.

Approve AI with changes: 7-0-2

AI12-0200-1/01 Improve reentrancy requirements for language-defined subprograms

The implementation shall ensure that each language-defined subprogram is reentrant in the sense that concurrent calls on any two (possibly the same) language-defined subprograms perform as specified, so long as all pairs of objects (one from each call) that {either} are denoted by parameters that could be passed by reference{,} or designated by parameters of an access type are nonoverlapping.

Correct the !standards to just have A(3/4).

Approve AI with changes: 9-0-1.

AI12-0203-1/01 Overriding a nonoverriddable aspect

Gary notes that the subject misspells nonoverridable.

Jean-Pierre points out a typo in the recommendation, “have” rather than “has”. “that has to be covered”.

Tucker suggests quoting the “if not overridden” in the question.

Jean-Pierre suggests marking the new text Redundant.

Approve AI with changes: 10-0-0.

AI12-0204-1/01 Renaming of a prefixed view

Steve notes that !summary says generalized_reference but should say generalized_indexing.

In the Language Design Principle, “If” should be “When”.

Steve would like the principle to say (this is a complete replacement):

When the Standard says that some construct C1 has equivalent dynamic semantics to some other construct C2, there should be a language rule that says that C1 is illegal if C2 is illegal.

4.1.3, from Tucker:

{be legal as the prefix of an Access attribute reference}.

Reorganize that further:

For a prefixed view of a subprogram whose first parameter is an access parameter, the prefix shall be legal as the prefix of an Access attribute reference.

Gary suggests changing the summary:

...must be {renameable}[able to be renamed] as an object.

Approve AI with changes: 8-0-0.