Minutes of ARG Meeting 59

22-24 June 2018

Lisbon, Portugal

Attendees: Raphaël Amiard (except early Friday, late Saturday, and Sunday), Steve Baird, John Barnes (except the majority of Sunday), Randy Brukardt, Alan Burns (only Friday afternoon via WebEx), Peter Chapin (left early Sunday), Jeff Cousins, Brad Moore, Erhard Ploedereder (left early Sunday), Jean-Pierre Rosen, Tucker Taft, Tullio Vardanega (except early Sunday).

Observers: Pat Rogers (Friday only).

Meeting Summary

The meeting convened on Friday, 22 June 2018 at 12:15 hours WEST (Western European Summer Time) and adjourned Sunday, 24 June 2018 at 12:45 hours WEST. The meeting was held in a conference room in the VIP Executive ARTS Hotel. The meeting covered all of the regular AIs, most of the "simple" and "related to the instructions" Amendment AIs, and a few other Amendment AIs.

AI Summary

The following AI was approved:

AI12-0275-1/02 Make subtype_mark optional in object renames (7-0-0)

The following AIs were approved with editorial changes:

AI12-0119-1/10 Parallel operations (7-0-2)
AI12-0183-1/06 Presentation errors in Ada 2012 post Corrigendum 1(11-0-1)
AI12-0189-1/05 loop-body as anonymous procedure (5-0-2)
AI12-0226-1/02 Make objects more consistent (7-1-2)
AI12-0237-1/01 Getting the representation of an enumeration value (6-0-1)
AI12-0256-1/01 Aspect No_Controlled_Subcomponents (10-0-0)
AI12-0263-1/01 Update references to ISO/IEC 10646 (10-0-1)
AI12-0272-1/01 Contracts for generic formal parameters (10-0-1)
AI12-0277-1/01 "accessibility level of the body of F" (9-0-2)
AI12-0278-1/01 Implicit conversions of anonymous return types (11-0-0)
AI12-0283-1/01 Nonblocking and remote calls (8-0-3)
AI12-0285-1/01 Syntax for Stable_Properties aspects (9-0-0)

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

AI12-0020-1/04 'Image for all types (7-0-0)
AI12-0021-1/01 Additional internationalization of Ada (11-0-0)
AI12-0205-1/01 Defaults for generic formal parameters (11-0-0)
AI12-0212-1/06 Container aggregates; generalized array aggregates (11-0-0)
AI12-0230-1/02 Deadline Floor Protocol (10-0-2)
AI12-0242-1/03 Reduction expressions (8-1-2)
AI12-0251-1/01 Explicit chunk definition for parallel loops (10-0-1)
AI12-0262-1/01 Map/Reduce attribute (10-0-1)
AI12-0266-1/03 Parallel container iterators (11-0-0)
AI12-0267-1/04 Data race and non-blocking checks for parallel constructs (7-0-1)
AI12-0279-1/01 Nonpremptive dispatching needs more dispatching points (11-0-1)

The following AIs were discussed and assigned to an editor:

AI12-0190-1/04 Anonymous functions
AI12-0251-2/02 Parallel loop chunking libraries

The following AIs were discussed and voted No Action:

AI12-0009-1/06 Iterators for Directories and Environment_Variables (7-0-0)
AI12-0188-1/01 Add container iteration supporting iteration over keys (7-0-0)
AI12-0202-1/01 Tampering_Check (8-0-3)

The following AI was discussed and placed into hold status:

AI12-0197-3/02 Generator Functions (9-0-1)

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

AI12-0119-1/09 Parallel operations (11-0-0)
AI12-0267-1/03 Data race and non-blocking checks for parallel constructs (11-0-0)

Detailed Minutes

Welcome

Randy congratulates Steve Baird as having been confirmed by WG 9 as our new Rapporteur. He notes that Steve has delegated all duties to Jeff until the completion of the minutes review, expected to be in mid-July. In particular, Jeff will run this meeting as he has in the past.

Apologies

Ed Schonberg sent apologies. Alan Burns also sent apologies, but then attended most of Friday's session via WebEx.

Previous Meeting Minutes

No one has any changes to the minutes of Electronic Meeting #58C. Approve minutes by acclamation.

Date and Venue of the Next Meeting

We discuss the dates of the next meeting. HILT is November 5-6. Randy notes that if we have the meeting in October, we can have an extra electronic meeting in mid-December. The more meeting time that we can have before the end of March, the better. The group agrees that this is a worthy goal.

Tucker announces that he can't make any weekend in September or October. We note that the weekend meeting dates were chosen in part because travel was cheaper that way – but that hasn't been true (at least not in the same way) for many years. We could use weekdays if that makes more sense.

Tucker proposes several sets of dates. Erhard thinks that he can't make any of them. Tucker's attendance is critical (as is Steve's and Randy's), everyone else is optional.

We settle on the Sunday-Monday-Tuesday dates of October 21-23 for meeting #60. WG 9 would be Monday morning. We'll meet at the AdaCore offices in Lexington Massachusetts.

We'll schedule electronic meetings next time; the rough plan is a meeting in the middle of each month (December, January, February, and March).

The following in-person meeting (to finish up lose ends on Ada 2020) will be immediately following the Ada-Europe meeting on June 14-16, 2019 (with WG 9 morning of the 14th), in Warsaw, Poland.

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. For those who have volunteered, there's no time better than the present for writing some tests.

Simple AIs

Randy suggests setting aside two hours to work on Simple AIs. He suggests the time after the break on Saturday afternoon. This is generally approved.

WG 9 Extending Scope

Pat Rogers asked WG 9 to extend our scope to include his extended Ravenscar proposal. This was accepted. Pat should be submitting a proposal in the near future.

Thanks

Thanks to Jeff for running the meeting for all of these years. Jeff thanks us for putting up with him.

Thanks to Ada-Europe for the accommodations.

Thanks to Randy for all of the work he does. Jeff says that you have to be the Rapporteur to appreciate all of the work Randy does.

Unfinished Action Items

Steve Baird says that AI12-0016-1 needs to be done someday, but there is little customer demand for a correct implementation. So this on somewhat indefinite hold.

Steve also agrees that AI12-0210-1 needs to be done, but he is focused on other AIs.

His AI12-0243-1 is on the agenda for discussion; he has a model, but no wording yet.

Randy notes that he is working on AI12-0112-1; at this point, he doesn't need further input from the ARG. Tucker wants an action item to send Randy the GNAT formal container packages; he already ought to have an action item to help Randy run the containers on GNAT.

Randy should give Steve necessary information for disaster recovery of ARG materials if Randy is incapacitated.

Florian Schanda has left Altran and has a new job. No one knows if he will continue any Ada work. We need to reassign his AIs. AI12-0188-1 was an alternative to AI12-0189-1, it is too late now for a totally new idea. AI12-0197-3 is an alternative that we didn't have much will to pursue. Vote to put both of these AIs on hold: 9-0-1. [Editor's note: later in the meeting, we decided to vote AI12-0188-1 no action, so this hold vote was not recorded in the AI.]

Tucker Taft didn’t work on AI12-0079-1 and AI12-0111-1 as he didn’t see a lot of changes. He is reminded that it is difficult to make progress if updates aren't made. AI12-0191-1 is important, but it less important than the others. AI12-0235-1 is less important still (but it is easy!). In all cases, time was limited so doing everything wasn't possible.

Current Action Items

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

Raphaël Amiard:

- AI12-0205-1
- AI12-0212-1 (Propose “some syntax” to Tucker)

Steve Baird:

- AI12-0016-1
- AI12-0020-1
- AI12-0210-1 (get guidance from the SPARK group, then update the AI)
- AI12-0243-1

Randy Brukardt:

- AI12-0112-1
- Give Steve Baird recovery materials and information should he become incapacitated

Editorial changes only:

- AI12-0119-1
- AI12-0183-1
- AI12-0189-1
- AI12-0226-1
- AI12-0237-1
- AI12-0256-1
- AI12-0263-1
- AI12-0272-1
- AI12-0277-1
- AI12-0278-1
- AI12-0285-1

Alan Burns:

- AI12-0230-1 (assist Tucker Taft)

Peter Chapin:

- AI12-0021-1

Brad Moore:

- AI12-0242-1
- AI12-0262-1
- AI12-0266-1
- AI12-0279-1

Tucker Taft:

- AI12-0079-1
- AI12-0111-1
- AI12-0191-1
- AI12-0212-1
- AI12-0230-1 (help from Alan Burns)
- AI12-0235-1
- AI12-0251-1
- AI12-0267-1
- Send Randy examples of the “formal” containers for AI12-0112-1; help Randy with running of containers tests for the proposed specifications.

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 13 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

AI12-0009-1/06 Iterators for Directories and Environment_Variables

This is covered adequately by AI12-0189-1.

No Action: 7-0-0.

AI12-0020-1/04 'Image for all types

Tucker tells Steve that protected functions are not blocking. So that the wart involved does not actually exist. Remove that text.

A question is raised about the item

For scalars - unchanged

in the Proposal section. This is about Put_Image, so that needs to say that the image is the same as the 'Wide_Wide_Image in previous versions of Ada.

Erhard asks about Get_Image. We decided not to do that now, but it is open for future Ada. (He then leaves [this discussion happened on Sunday], probably not because of this answer.)

Are the streams necessary? Tucker has several personal buffering streams. Randy notes that Claw has one as well. It seems valuable to have these somewhere, but they're generally useful (not just for this AI). This fact makes the naming of them and the operations they contain more important.

Randy notes that the Claw version has a Clear operation to discard the contents. Steve says you can do this by reading all of the contents. That has a lot of memory overhead, in most implementations Clear would just write a single value. Tucker agrees we should add Clear to Ada.Streams.Buffered (or whatever we call them).

Tucker would like wording that says when you write something it is immediately available for reading.

These streams should be Nonblocking, and probably Pure except for the unbounded which is Preelaborated.

Wording is needed to specify that this is a FIFO.

Tucker would prefer to name these "Buffered_Streams". Then he changes his mind and advises "FIFO_Streams". Jean-Pierre suggests "Storage_Streams" since they are similar to Storage_IO.

Straw_Poll: Buffered: 1; FIFO: 5; Storage: 2.

Peter leaves.

The stream types should be Stream_Type. Lots of people hate _Type, but no better idea was suggested. And the type name has to differ from the parameter name.

Turning to the actual image rules:

Should the image of a null array be []? No, that doesn't show the bounds, and those matter for debugging.

The Wide_Wide_Image definition has a lengthy piece of code; that should be an AARM Implementation Note, it's not interesting to users.

In !proposal:

```
{procedure} S'Put_Image
  (Arg : T; Stream : access Counted_Stream'Class);
```

(That is, **procedure** is missing.)

Tucker wonders why Put_Image needs to be a Counted_Stream. It doesn't need the counted stream operations, so it can be any stream.

Tucker suggests that we just let the implementation figure out how to do this. So we don't need the stream definition to be here.

So split the FIFO_Stream definition into a separate AI. (It seems generally useful, so we should save it.)

Steve wonders what happens if one wants to be in bounded environment. Tucker suggests a restriction: Max_Image_Length, which takes a value. If exceeded, you get an implementation-defined result. Probably want Implementation Advice that no dynamic allocation is used by 'Wide_Wide_Image when this is specified.

Max_Image_Elements goes away.

The examples probably should be in the Example part of the RM.

Steve asks if the format of record and array aggregates is specified too loosely. Tucker replies that he thinks it is over-specified. He thinks that we should just say that it is written in the form of an aggregate. He doesn't care how many spaces are displayed. Tucker would like to allow suppressing extra spaces. Probably use an implementation permission to do that.

Steve notes that a confirming Put_Image for a tagged ancestor, changes the Image of descendants. Seems OK.

Tucker would like to see **protected** shown in protected objects. They are very different from regular records. We agree that we should use a rule similar to that for tasks (without the id).

Tucker asks whether we should allow implementations to economize array aggregates by using ranges, or others, or changing order, or even positional if the lower bound would be correct. No decision was made.

Brad asks what you get if you do this on an Unchecked_Union. Steve says that it could be illegal (actually, raise a Program_Error). Randy notes that Read and Write raise Program_Error – B.3.3.

Square brackets for access is a problem now since we are using them for some aggregates; use @<hex value> instead.

Approve intent of AI (including split): 7-0-0.

AI12-0021-1/01 Additional internationalization of Ada

Tucker suggests adding packages like Wide_Text_IO to handle the names. Several people note that we already have a package with that name! There are two things that could be Wide: the file names, or the text written to the file. We need all of the possible combinations (file naming and file contents being unrelated).

Erhard suggests just overloading the names; he wants to use the same file type regardless of the way the file is named. Tucker comments that that would break code using string literals, which is very common.

Steve suggests child packages Wide_File_Names and Wide_Wide_File_Names for each I/O package. [Editor's note: Ada.Directories probably should just have a Wide_ and Wide_Wide_ version, since it doesn't do much other than file manipulation; specifically, there's no contents to worry about.] These would include just the operations taking file names.

Command_Line should do something similar. Environment_Variables also should have a Wide and Wide_Wide_ version.

Tucker suggests not changing exception message, because it would require changing the meaning of syntax incompatibly. Randy suggests that we recommend using UTF-8 encoding for exception messages that need non-ASCII characters so we don't cause incompatibilities. Some users encode non-textual information into exception messages (including binary data), and breaking that sort of code would not be acceptable.

Tucker says we should add Wide_Exception_Information and Wide_Wide_Exception_Information to Ada.Exceeptions. These include exception names and often other identifiers.

Peter is drafted to write this one (no one volunteered!)

Approve intent of AI: 11-0-0.

AI12-0119-1/09 Parallel operations

Brad notes that IRTAW had some questions. We discuss each one in turn:

- There was question about a mention of a tasklet; Tucker intended to eliminate that from all wording.
- Should Set_Priority be deferred after the parallelism? Tucker asks why? Multiple set priorities would be a race condition, and there is no problem with it. Alan concurs.
- There was a concern about the "first" exception being chosen. It's an arbitrary choice in the current wording.
- There was a concern about exception handlers on a parallel block being confusing. We've been over that before, and we (as a group) don't agree.
- There was a suggestion that there be a documentation requirement to explain how parallel operations are implemented. A documentation requirement on "how it is implemented" would not fly; it's way too general. A specific documentation requirement surely would be considered (but it needs to be proposed).
- There was a complaint that Yield is marked that it allows blocking. This is existing Ada semantics (see the now deleted line of D.2.1(7)); we've just made it more explicit. Note that Yield_to_Higher is **not** blocking.
- What happens for abort? Abort is a potentially blocking operation, so it isn't allowed in a parallel loop. [Editor's note: But what if some other task aborts this one? Presumably, all of the threads are aborted; does the wording cover that?]
- What happens if Set_CPU is used? Every thread of control in a task shares the properties of the task. So all threads execute on a CPU if one is specified. Randy worries that this would require preemption and moving of the threads while executing. That could require a lot of overhead for a feature that would very rarely be used. Tucker suggests that we give an implementation permission for Set_CPU and similar operations to be deferred until the end of the parallel construct.

In the proposal, "or task objects" needs to go (they would raise Program_Error as potentially blocking operations). There also are uses of "tasklets" in the proposal.

Spelling error in the wording : "parellel" in 5.5(9/4). Also in the second paragraph of !problem "mainain" and "perforamance".

Start of 5.6.1:

> A parallel_block_statement encloses two or more handled_sequence_of_statements to be executed concurrently.

Randy asks if we can simplify to "logical threads" in general. Someone asks what was wrong with "tasklet"s. Tucker explains that "tasklet" doesn't work well with tasks. We want a term that includes the task's thread of control along with all others.

Steve asks about the bounded error cases; these don't have a consequence. "**Raise** Program_Error;" or block the entire task, as well as working normally.

Erhard would like to replace the examples with searching an array in parallel (or maybe a tree). And Foo/bar probably doesn't belong in the Standard.

Steve wonders if the termination wording could be simplified somehow.

"Attempt to cancel..." doesn't seem to be defined. We don't want to make things abnormal because of "exit" or "raise". (11.6 makes certain exception uses abnormal, but that's not true in general). This needs careful consideration.

Tucker will try to give us a redraft before the end of the meeting.

Approve intent of AI: 11-0-0.

AI12-0119-1/10 Parallel operations

Tucker rewrote the !proposal, moved transfer of control into 5.1; added the definition of parallel constructs; moved the bounded error here. He also added an example that searched a binary tree.

Jean-Pierre notes a typo. "when multiple[r] processors are available."

"...it causes it to complete..." too many its.

Erhard is concerned that the wording only allows deferring cancellation after an abort deferred region, as opposed to always.

We suggest the following rewording of the cancelation text:

> When a logical thread of control is canceled, it causes it to complete as though it had performed a transfer of control to the point where it would have finished its execution. Such a cancellation is deferred while it is executing within an abort-deferred operation (see 9.8), and may be deferred further, but not past a point where the logical thread initiates a new nested parallel construct, or reaches an exception handler.

Steve notes that it is important to note that exception handlers are relevant only when not in an abort-deferred region.

Jeff notes that the deferring of migration when the CPU is changed that we agreed to on Friday is missing.

Steve thinks that initiating a parallel construct should be an abort completion point. That makes sense.

Approve AI with changes: 7-0-2 (John and Peter abstain)

Tucker will provide wording changes to the editor.

AI12-0183-1/06 Presentation errors in Ada 2012 post Corrigendum 1

Erhard notes that D.5 doesn't have any policies. Question (2) and Wording (2) ought to refer to D.4 (as does the ! corrigendum section). Also, the paragraph reference should be (7/5).

Approve AI with change: 11-0-1 (Alan abstains).

AI12-0188-1/01 Add container iteration supporting iteration over keys

This is adequately covered by AI12-0189-1.

No Action: 7-0-0.

AI12-0189-1/05 loop-body as anonymous procedure

The second Legality Rule appears twice.

The aspect specifies that the routine is designed to allow exiting (cleanup will be handled only via finalization).

We had a brief discussion of the syntax. Using **do** instead of **loop** didn't gain much support – the examples are loops. We also considered optionally allowing **parallel**, but the implementation of the subprogram determines how looping is executed, so the keyword would only add some restrictions and a possibly false promise.

Approve AI with change: 5-0-2 (Steve, Jean-Pierre abstain).

AI12-0190-1/04 Anonymous functions

Tucker claims that applying functions is universal; returning functions is much less used. So he claims that Erhard's objection (no closure support) is not very compelling (he is not here at this point).

Someone complains that the !proposal doesn't match the !summary.

Jean-Pierre argues that there is no blocking problem here, you can always declare this explicitly.

Randy wants to hear Erhard's objections before voting this. We don't want to kill it without more input.

Keep alive: 6-0-1 (Jeff abstains).

AI12-0202-1/01 Tampering_Check

Randy notes that Ada 2020 has three other ways to minimize these checks: using stable views (in AI12-0111-1); suppressing container checks (in AI12-0112-1); and there are fewer tampering checks for some containers (also in AI12-0111-1).

Having a fourth minimization sounds like we are throwing things at the wall.

Randy notes that Suppress(Tampering_Check) in GNAT is effective at the point of the instantiation. That's important to the implementation (since most of the cost is centered in setting/resetting the tampering state in Reference/Constant_Reference), but that's not how Suppress is defined, so we would have to change a bunch of rules to support this as defined in GNAT.

No Action: 8-0-3 Steve, Tuck, John abstain.

AI12-0205-1/01 Defaults for generic formal parameters

Several people comment that they prefer the **or use** syntax over previous suggestions.

We get side-tracked by the formal package proposal. Tucker says that if the parameters are <>, then it could be useful. Otherwise, it doesn't seem very useful, since it seems unlikely that some default would match. AI12-0268-1 proposes another mechanism to determine a default in such cases (a form of implicit instances).

Jean-Pierre asks whether we could have <> for these (similar to formal subprograms). Raphael suggests that they aren't very likely to have the same name; as none of these entities are overloadable (objects, types, packages).

Erhard would like to drop the **in out** case. Subprograms don't have defaults for in out parameters, why should generics. Randy notes that we have an AI for defaults for **in out** parameters for subprograms, so the idea isn't that strange. Tucker would prefer to **or use** for **in out** parameters; it can be a name, not an expression. For **in**, it is an assignment; **in out** is not any sort of assignment (it's a renames, effectively).

Raphael volunteers to take this one.

Approve intent of AI: 11-0-0.

AI12-0212-1/06 Container aggregates; generalized array aggregates

4.2.3 (two places) ought to be 4.3.3.

Tucker explains various parts of the proposal in detail.

Peter suggests that Empty has a parameter of subtype Natural. Steve suggests that this is name resolution, so the subtype isn't appropriate. Since these are compiler-generated, these aren't going to ever generate negative numbers, and forcing Natural is just extra pain. So that isn't worth changing.

John notes that in the examples, there is no declaration of S.

"Assign S to {be} an empty set".

The use of "when" in these examples isn't helpful, as that hasn't been seen before and it belongs to a different AI.

Randy suggests that the map key is something other than a discrete, 'cause that's the "cool new feature" here, we need to show it off. Use something non-discrete, like float or constrained strings.

Peter asks if the comma is too subtle for separating the separating the table and the key. Jean-Pierre agrees and suggests using colon.

```
M := (for P of Table, P.Key => P.Value);
M := (for P of Table: P.Key => P.Value);
```

John suggests double colon:

```
M := (for P of Table:: P.Key => P.Value);
```

Randy suggests vertical bar:

```
M := (for P of Table | P.Key => P.Value);
```

More even worse suggestions are thrown out. We agree to let Tucker marinate on this topic.

But Raphael doesn't like the right arrow. That's standard aggregate syntax!

Randy notes that the comma doesn't read well as this is used in a comma-delimited list already. It parses, but would be confusing.

Tucker (re)suggests using vertical bar. Jean-Pierre and John agree that sounds better. He later determines that it wouldn't work well, either, since it could appear in/near other lists separated by vertical bars. That's the same problem as with comma.

Raphael argues the key mapping belongs to the value, not to the iterator.

```
M := (for P of Table => P.Key | P.Value);
```

Tucker argues that we've had the key on the left of the arrow since the beginning of time.

Raphael writes:

```
M := (for A in I.Iterator do Key(A) => Element(A));
M := (for A in I.Iterator do Element(A));
```

The second one doesn't fit with the existing syntax. The first would be a possible replacement for the comma, though. We remind Raphael that this is an extension of features that have existed in Ada since Ada 83. It would be bizarre to use a different syntax for an iterator over a range than for the range alone (which could be considered a shorthand for an iterator where we don't need a name for the iterator parameter. Specifically:

```
M := (for I in 1..10  => Element(A)); -- Previously approved Ada 2020 syntax
M := (1..10 => Element(A)); -- Ada 83 syntax
```

Raphael finally sees the light and withdraws most of his previous discussion.

We turn to the syntax for empty aggregates.

Some people think that <> means unspecified thing, as opposed to nothing. Various examples are given: unconstrained array declarations, aggregate components, generic formal packages.

Straw poll: 4 vote against using <>; several people abstain wanting to see an alternative (they don't like <>, either, but they need a better idea before killing it).

Straw poll: Prefer empty parentheses to (<>): 6-2-3.

Jean-Pierre suggests that () is similar "" (that is, null string). But he thinks both are ugly.

Steve says that he uses () as a placeholder for code that isn’t finished yet. He'd hate to lose that possibility.

Randy asks about singletons. In particular, containers that only support positional operations (like lists) don't allow singleton aggregates. It would be possible to write a singleton aggregate using a phony iterator, but if writing a null range is too annoying for some, a phony iterator would be ten times more annoying.

Tuck makes a radical suggestion: use [] for “containers” (including optionally for array aggregates, but NOT record aggregates).

This solves multiple problems (empty aggregates, singletons), and also does so for array aggregates as well.

Straw poll: who likes this – 10-0-1.

Steve notes that the container type can be limited; that doesn’t work with empty being a constant. We need to either make the container type non-limited only or disallow constants for limited types.

Randy notes that the element type has to be nonlimited. We need a rule to require that.

Tucker asks if only allowing this on private types is appropriate. That seems OK.

Raphael thinks that the difference between Add_Named and Add_Indexed is complicated. Hard for writers of containers to understand. He would prefer to use only one. Tucker tried that, but it was a mess. Add_Indexed does completeness/non-overlap checks, while Add_Named does not.

Steve notes that the new aspect needs to be nonoverriddable.

Steve wonders if the wording works of the element is an anonymous access type. Randy wonders why we would care.

Steve wonders what happens if the element type is an array – an appropriate applicable index subtype needs to be defined. He writes an Ada 95 example:

```
    type T is array (1..2) of String (1..3);
    X : T := (1..2 => (others => ‘x’));
```

The same should be true for a container aggregate. Randy notes that the wording covers this (the definition of “applicable index constraint” does not care what kind of aggregate the element is in). But Tucker notes that we need to have an element subtype in order for that and probably other rules to work. (We try to have a nominal subtype is almost all contexts, and this seems like a context where that will be needed.) Most likely, we need that the element subtypes shall be statically matching; and the element subtype is determined from this. We need to make sure that the applicable index constraint is handled.

Action item to Raphael to propose some syntax with Tucker. Tucker will revise the AI.

Approve intent of AI: 11-0-0.

AI12-0226-1/02 Make objects more consistent

A value conversion of an object makes a new object. It’s important that this doesn’t force lots of copying for **in** parameters.

Tucker would prefer to talk about objects. Are there any composite objects that are not objects? No. If we make that change, all of the text dealing with values can be eliminated.

...value conversion [of a composite type]{of an object} either creates a new anonymous object...

Erhard would prefer the problem statement wouldn’t promise so much, something more like “make qualified expression and value conversion be more similar/regular/consistent”.

Jeff notes that there is a plural vs. singular is confused in the first couple of sentences. Tucker suggests that it uses singular.

Approve AI with changes: 7-1-2.

Jean-Pierre disagrees with the problem, and makes the claim that no real user ever complained about this. Jeff and Peter abstain.

AI12-0230-1/02 Deadline Floor Protocol

Alan had "minimum relative deadline", he has since gotten rid of it. He says that two styles are used, either using relative deadlines or absolute deadlines.

Erhard wonders what happens when both relative deadlines and absolute deadlines are used. Alan says that the intent is that the absolute deadline is used for scheduling (relative deadlines are converted to absolute deadlines).

Erhard wonders if you could have an either-or but not both. Mixing isn't intended. Alan says that the intent is that the rules are well-defined, as he couldn't see a way to statically avoid mixing (these are all subprograms).

This uses deadline floor protocol for accessing protected objects.

Steve wonders what happens if you forget to put the pragma in. All tasks would run as if it is FIFO.

Deadline Floor is replacing an attempt to get deadline dispatching working with existing ceiling locking.

Tucker again volunteers to review the wording in detail.

Erhard wonders about wording that says "the deadline". Alan says that "the deadline" means "absolute deadline"; if "relative deadline" was meant, this is explicitly said. It should be explicitly mentioned in the text that this is the case.

Approve intent: 10-0-2 (Steve and Randy abstain).

AI12-0237-1/01 Getting the representation of an enumeration value

The names of the attributes aren't liked. Tucker suggests 'Torep and 'Derep.

Tucker later suggests 'Val_of_Rep and 'Rep_of_Val.

Jean-Pierre suggests 'To_Rep and 'From_Rep.

Steve suggests 'Enum_Rep and 'Enum_Val; he notes that these don't do anything useful for integers; but the only time they should be potentially be used on an integer is within in a generic on a formal discrete type. Most of the group agrees with this argument.

Jean-Pierre now suggests 'To_Enum_Rep and 'From_Enum_Rep. We decide to use these names. [Your editor has no recollection or notes of such a decision; in any case, it was reconsidered after the meeting].

Tucker wonders if we should deal with bias here. After some discussion, Tucker decides that we don't say anything.

[Editor's note: The following note, which talks about using Unchecked_Conversion immediately after we define an attribute to avoid the need for using Unchecked_Conversion, also needs revision.]

Approve AI with change: 6-0-1 (Jeff abstains).

AI12-0242-1/03 Reduction expressions

"of an array {type}..."

Randy notes that container type isn't defined, we need to resolve to any composite type and use a Legality Rule to restrict to types with appropriate iterators.

Erhard notes that there are some synchronized checks left in this definition. These belong to AI12-0267-1 and don't belong here.

We should say the version with **parallel** is a parallel construct here, otherwise, both versions are the same semantically.

Jean-Pierre notes a typo under reduction attributes, middle paragraph: "whether injecting parallelism [with]{will} improve instead of detract from performance."

Also "sequential processing [with] by simply replacing the parallel version of the attribute with the sequential one."

John thinks "summarize the prefix object" is weird.

More typos: "ka Combiner_Subprogram"; A "combiner_Subprobram".

Tucker finds it strange to put this in the array section, should be elsewhere. Tucker suggests using 4.5.9 (an new subclause). The group agrees.

Operational aspect is defined[.]{;}

Steve suggests that the aspect Associative is defined such that the compiler can assume that it is true. Tucker suggests that it is a Bounded Error if this aspect is making a promise that isn't True. Tucker says we don't want this to be an assumption; we want to allow reordering.

Erhard suggests that we call it something else. We don't want to specify something that is mathematically False. Maybe Reordering_Allowed.

Some of this wording needs to be updated to use the new wording from AI12-0119-1. "executor" appears, for instance.

Brad comments that an alternative would be not to have the Associative check. Erhard argues that this is a lot of complexity for just a little bit of safety. Tucker says that there are not many operations that don't qualify. But "-" and "/" don't qualify, so the ones that don't qualify aren't that obscure. Even so, we decide to drop the Associative aspect and the associated checks.

Brad asks if we want to support combiners with different types for the parameters. He suggests that we'd need an identity value to apply to the initial value of each chunk, if we allow the combiner to have different types.

Jean-Pierre again suggests raising Constraint_Error for any empty array. Tucker argues that is weird to have to a special case that requires explicit handling in almost every usage.

Peter notes that Scala has separate "Reduce" and "Fold". Reduce has no order, whereas Folding has an order, an initial value, and possibly different parameters.

Raphael suggests that we don't put this into the language. He suggests that we could define it as a generic. He doesn't use it that often.

Tucker says that he finds this as one of the most important parallel features.

We should look at supporting parameters of different types; that perhaps is a different compatibility. We may want to write that up separately, since it seems a lot more complex.

Steve wonders if we would ever want a 'Reduce to be static. We'll leave that for the AI author.

Approve intent of AI: 8-1-2. Raphael votes against. He says the feature can be implemented sufficiently with existing language features and has insufficient value.

Since we didn't change the owner, this AI remains assigned to Brad.

AI12-0251-1/01 Explicit chunk definition for parallel loops

Peter says that OpenMP does something similar. It seems necessary for some such mechanism to exist to handle real-world code.

We take a brief digression to look at Brad's alternative (AI12-0251-2), then take a straw poll that shows a strong preference for this alternative (see discussion of AI12-0251-2 for details on the poll).

Tucker will take this AI.

Approve intent of AI: 10-0-1

AI12-0251-2/02 Parallel loop chunking libraries

Brad says this is just a library. Tucker says that this is just a long-winded way to provide just a chunk index.

Peter notes that the example here is a lot harder to understand.

Straw poll: Prefer first alternative: 9; 2 abstain; prefer second alternative: 0.

We'll keep the second alternative around while we see how messy the first actually is.

Keep alive: 11-0-0.

AI12-0256-1/01 Aspect No_Controlled_Subcomponents

Tucker asks if this should be No_Controlled_Parts. Yes, it seems it should apply to the object itself. Subcomponents should be changed to parts generally in the AI.

Steve wonders if it can be specified on an interface. We agree that there is no obvious problem with doing so.

The AARM Note is False; the explicit specification is not the same as the default. So delete it entirely; no one sees any problems with that.

Approve AI with changes: 10-0-0.

AI12-0262-1/01 Map/Reduce attribute

Tucker suggests writing the object case (that is, AI12-0242-1) in terms of an implicit iterator of this form. He also suggests putting the **parallel** in front of the **for** here (since it would be allowed there), and just have a single Reduce attribute.

Tucker suggests that Reduce has to be symmetric if it is parallel, otherwise can be asymmetric. That way, we don't need a separate initial value.

There is a typo in the problem: "convinient". The first sentence of Name Resolution contains a bad use of "it's", but that has to be replaced by real wording anyway.

Straw poll on Tucker's suggestions: 8-0-3. So we should focus on this one.

Approve intent of AI: 10-0-1. Raphael abstains.

Since we didn't change the owner, this AI remains assigned to Brad.

AI12-0263-1/01 Update references to ISO/IEC 10646

!wording

2.1(3.1/3) the changes of 2011 are missing 2017.

1.1.4(14.2/3), “readble” in the AARM.

2.3(5/3):

Two @fa<identifier>s are

Get rid of the @fa and angle brackets.

Jeff wonders if the remaining AARM notes should also be changed. Randy says that he checked all of the notes and did not change ones that were specific to a particular version of the language. It wouldn't make sense for an Ada 95-specific note to reference a 2017 Standard.

Approve AI with changes: 10-0-1. Steve abstains.

AI12-0266-1/03 Parallel container iterators

The proposal uses “tasklet”, that should be fixed.

Side discussion: should we allow **parallel** in front of **for** in aggregates. Tucker thinks that it is would be easy to parallelize implicitly. But you don’t always want that (Raphael is particularly concerned about this point), and there is value to **parallel**: it allows checks, and tells the compiler that parallelization is requested. This is much like inline, which is a hint to the compiler; it still can inline it itself. But that doesn’t make the hint useless.

Advised_Split is the maximum number, so it should be Maximum_Split.

Raphael suggests that this work on the stable view of the the container, because otherwise you would have parallel tampering checks. (Editor: Not sure that really is a problem – one would set the tampering state before starting the iteration, only once, regardless of the view. The actual checks are just reads of the state – those should always work in parallel.)

Tucker would prefer to have Get_Start_Cursor and Get_End_Cursor.

How does the chunk iteration end? There is an equality check. But that doesn’t work, because it’s not part of the interface. One can't assume any operations are available for an incomplete type – we can't require the type to be nonlimited.

So we need a function Split_Finished (Cursor, Split_Index) return Boolean; which returns True when the cursor is such that the chunk is finished.

Peter asks why this uses the Ada.Containers.Count_Type. Brad says it was the best type; but we didn’t want a dependence on Ada.Containers since other people might make containers.

Tucker would rather add a minimum_chunk_size parameter, so that the number of iterations doesn’t have to be calculated. That could be very expensive if the bounds are dynamically determined or for custom containers. One probably has to do it once anyway, doing it twice would be insane.

Steve wonders why the chunks are explicit here; they're implicit in the parallel constructs. The container implementer has to provide this information, so we have to make it more explicit.

Tucker suggests that we use the “chunk” terminology explicitly; there's nothing to be gained by avoiding it. (Brad groans; during a previous discussion he was asked to remove it.) Erhard suggests that “partition” also works. Peter notes that has another meaning in Ada; Erhard withdraws his suggestion.

Randy notes that this interface can be implemented with a statically allocated internal array, since the size requested is a maximum, and one can always return less. This matters for the bounded containers, which are not supposed to use dynamic allocation.

Approve intent of AI: 11-0-0.

AI12-0267-1/03 Data race and non-blocking checks for parallel constructs

The new sentence in 5.1 seems to indicate that something has to be both a parallel block and loop. Obviously only one is needed to be a parallel construct. Rewording is suggested.

“conflicting actions would be allowed during the execution of the calls”: “allowed” bothers Steve, because we are going to turn around and disallow the action. Tucker says these are different things.

Erhard is concerned that these checks could be too conservative. Essentially lock-free stuff and pointer stuff would not be allowed. Randy and Tucker note that lock-free implementations using atomic objects are sequential, so these doesn’t trigger these checks.

“Known to refer” implies a conservative check. Raphael worries that a conservative check would miss many problems, giving an illusion of correctness.

Why are some things “assign”s and others “update”s? Use one or the other.

Peter notes that if we make it very safe, future versions can allow more as the language gets smarter. If we allow stuff that is clearly unsafe, we are stuck in the future because of compatibility concerns.

Raphael wonders if we could have a pragma to activate such checks for tasks.

Straw poll would we like the default for a parallel construct to be illegal unless it is clearly safe? 8-1-2. Erhard worries that programmers will be tempted to throw the safety equipment into the water when they get an error.

Tucker suggests a policy for setting the amount of checks. Default: Very safe for parallel, no checking for Ada tasking. Also: all very safe; Only if obviously wrong; and Don’t bother me.

Erhard is still concerned about writing various lock-free algorithms. That seems like low-level code that probably fits best under the “Don't bother me” policy. One hopes that Ada compilers would use lock-free implementations for protected objects when possible; such code is much safer and portable than any direct use of a lock-free algorithm.

So four levels of policy, no runtime checks. The name of the policy should be Conflict_Check.

Approve intent of AI: 11-0-0.

AI12-0267-1/04 Data race and non-blocking checks for parallel constructs

On Sunday, Tucker presents a substantially revised AI.

Tucker forgot to update the !proposal completely. Must remove discussion of runtime checks.

The nonblocking stuff has not changed.

In a protected action, we don’t create any threads of control. Erhard suggests that this rule belongs in AI12-0119-1, since it doesn’t have anything to do with checks. The group agrees.

For the new protected action wording, Steve prefers:

> Instead, all parts of the parallel construct execute {in arbitrary order} using the same logical thread of control as that of the protected action.

Tucker worries that could cause issues. Steve wants to ensure that a reader doesn’t think that this turns the loop into a normal forward loop. Maybe an AARM note is enough for that purpose.

...be [either]{one of} Unchecked,

Jean-Pierre asks whether this policy can apply to part of a declarative part. It should be able to. Steve wonders about the rules for these policies in generics. We suggest to copy the rules for Assertion_Policy or Suppress for handling scope and generics for this policy.

Tucker notes that there are no “checks” associated with these rules now; they're all compile-time. So the introduction is wrong. Change it to:

> This subclause determines what {restrictions are enforced}[checks are performed] relating to possible concurrent conflicting actions (see 9.10).

Erhard asks what happens if two different policies apply? The Parallel and All restrictions are on single actions, so it would be impossible for more than one policy to apply. That implies that the Known restriction only applies if both actions are Known (otherwise, the restriction on the action that is not Known would make that action illegal, or there is no restrictions on that action).

Jean-Pierre asks if “loop_parameter_index” covers renames and other known aliasing; we think that it should be “a name that statically denotes a loop_parameter_index”.

It is noted that other parallel looping things (parallel aggregate iterators, reductions) need to have this exception as well.

There is more detail needed in the definition of these restrictions.

Erhard wonders about statically checking concurrent actions, that is a runtime idea. Tucker says that we need to define “known to be concurrent”. We’re not going to try to do that here in this meeting.

Steve would like to see some examples. Erhard sent some examples, these would be useful.

Jean-Pierre says that the idea is no false positives for Known_Conflict_Checks; no false negatives for All_Conflict_Checks.

Triple-l in Parallel_Conflict_Checks.

Brad wonders if the items should be “Parallel_Data_Conflict_Checks”.

Erhard wonders if Known_Conflict_Checks would actually detect anything. Tucker thinks it would detect obviously wrong loops (where multiple iterations write the same global object), and that seems reasonably likely to happen – especially if someone slaps “parallel” onto a loop without much consideration.

Steve notes that there are “known to denote”, and “known to refer”, and Steve think Tucker is using the wrong one. Tucker agrees.

Approve intent with AI: 7-0-1 (Erhard abstains).

AI12-0272-1/01 Contracts for generic formal parameters

Steve complains that different views of the same subprogram have different contracts. Randy noted in the AI discussion that one already can have the same effect with an explicit subprogram declaration. Moreover, it already happens directly when using access-to-subprogram types. It make little sense to worry about it happening for formal subprograms while not caring that it happens for access-to-subprogram types.

Steve wonders about enabling of these aspects. That’s covered by 6.1.1(19/5), that says that the generic declaration is what matters. That seems like it works. We mostly care that it's well-defined and not so much what that definition is.

Various typos are noted in the !wording:

- 6.1.1(40/5): For a call on {a} generic subprogram....
- 7.3.3(2/5): The text says: “{,}[o]r” we need to move that square bracket so it says “[or]”.
- F.1(1): “{The r}[R]epresentation...”

Approve AI with changes: 10-0-1 Jeff abstains.

AI12-0275-1/02 Make subtype_mark optional in object renames

Randy notes that the subtype_mark is not trustworthy, so having it here does not provide any subtype information.

Steve argues this makes the code slightly harder to read. Tucker thinks that the extra type is noise, especially if you are renaming a view conversion.

Approve AI: 7-0-0.

AI12-0277-1/01 "accessibility level of the body of F"

The wording uses "modify", but the text doesn't show the changes using [] and {}. We need to add this.

Approve AI with changes: 9-0-2.

AI12-0278-1/01 Implicit conversions of anonymous return types

Change the summary:

> An implicit conversion to a named access type determine{s}[d] the "master of the function call".

Approve AI with one letter changed: 11-0-0.

AI12-0279-1/01 Nonpremptive dispatching needs more dispatching points

Randy noted that potentially blocking includes I/O. If someone is doing I/O character at a time, switching at each call to I/O would be outrageously expensive. (Essentially, a "yield" would needed at the start of each routine.)

Tullio notes that "optional" dispatching points are not useful, because they make analysis hard.

So the issue is that this is getting buried as part of calls. Tucker notes that this means that user-defined routines act differently than language-defined routines. Tucker suggests an aspect "with yield" that guarantees that there is a yield somewhere in the body of a routine.

The concern is routines that might not block (but usually block).

Brad gets this AI.

Approve intent: 11-0-1. Steve abstains.

AI12-0283-1/01 Nonblocking and remote calls

E.2.3(13/3):

- it shall not contain {nor}[or be] itself {be} a remote subprogram that is nonblocking (see 9.5);

That's different than the text nearby.

- it shall not be, nor shall its visible part contain, the declaration of a subprogram that is nonblocking (see 9.5);

This works since such a subprogram is a remote subprogram.

E.4(8):

This is backwards, the rule needs to apply to the containing operation. Something like the following is needed:

> A dispatching call with a controlling operand designated by a value of a remote access-to-class-wide type shall be in a construct that allows blocking (see 9.5).

It would be better to use wording like that in 9.5(57/5):

> A nonblocking program unit shall not contain, other than within nested units with Nonblocking specified as statically False, a dispatching call with a controlling operand designated by a value of a remote access-to-class-wide type.

Approve AI with changes: 8-0-3 Steve, Erhard, Peter abstain.

AI12-0285-1/01 Syntax for Stable_Properties aspects

Randy notes this depends on the 13.1.1 change in AI12-0212-1.

In paragraphs 7.3.4(7/5) and 7.3.4(10/5), the square bracketed text at the end of these is "redundant", not to be deleted.

In paragraph Add after 7.3.4(4/5): "A type property aspect definition is {a} list of names... [two places]"

Also, Brad points out "{an} optional **not**"

Typo in the paragraph before !discussion a "syntac".

Jeff notes **when** should be **with** in the example in !problem and also in !example.

Approve AI with changes: 9-0-0.