Minutes of ARG Meeting 60

21-23 October 2018

Lexington, Massachusetts, USA

Attendees: Steve Baird, Randy Brukardt, Gary Dismukes, Bob Duff, Jeff Cousins (electronically, Monday other than late afternoon and Tuesday before last break), Brad Moore, Ed Schonberg, Justin Squirek (Monday and Tuesday), Tucker Taft, Tullio Vardanega (electronically, Monday other than late afternoon and Tuesday).

Observers: Justin Squirek (Sunday; he was made an official member during Monday's WG 9 meeting), Pat Rogers (Monday only, except late afternoon).

Meeting Summary

The meeting convened on Sunday, 21 October 2018 at 9:15 hours EDT and adjourned Tuesday, 23 October 2018 at 13:25 hours EDT. The meeting was held in the conference room at AdaCore's Lexington Massachusetts offices. The meeting covered all of the regular AIs, most of the "simple" and "technically complete" Amendment AIs, and a few other Amendment AIs.

AI Summary

The following AIs were approved:

AI12-0251-1/02 Explicit chunk definitions for parallel loop (5-0-2)
AI12-0289-1/01 Implicitly null excluding anonymous access types and conformance (5-1-1)

The following AIs were approved with editorial changes:

AI12-0020-1/06 'Image for all types (8-0-0)
AI12-0212-1/07 Container aggregates; generalized array aggregates (5-0-2)
AI12-0230-1/03 Deadline Floor Protocol (7-0-3)
AI12-0235-1/03 System.Storage_Pools should be pure (7-0-0)
AI12-0249-1/01 User-defined numeric literals (10-0-0)
AI12-0267-1/06 Data race and non-blocking checks for parallel constructs (5-0-2)
AI12-0276-1/01 Admission policy defined for acquiring a protected object resource (7-0-3)
AI12-0279-1/03 Nonpreemptive Dispatching needs more dispatching (9-0-1)
AI12-0286-1/01 Allows_Exit aspect should be used on language-defined subprograms (7-0-0)
AI12-0287-1/02 Legality Rule for null exclusions in renaming is too fierce (5-0-1)
AI12-0290-1/02 Restriction Pure_Barriers (10-0-0)
AI12-0291-1/01 Jorvik Profile (10-0-0)
AI12-0292-1/01 Various cleanups for Ada 2020 (7-0-0)
AI12-0293-1/01 Add predefined FIFO_Streams packages (8-0-0)
AI12-0295-1/01 User-defined string literals (6-0-4)

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

AI12-0079-1/08 Global-in and global-out annotations (8-0-1)
AI12-0208-1/04 Predefined Big numbers support (10-0-0)
AI12-0213-1/01 Unify record syntax (7-0-0)
AI12-0236-1/03 declare expressions (6-0-2)
AI12-0250-1/01 Iterator filters (7-1-1)
AI12-0262-1/03 Map/Reduce attribute (5-0-3)
AI12-0266-1/06 Parallel container iterators (6-0-1)

The following AIs were discussed and assigned to an editor:

AI12-0111-1/07 Stable containers to reduce tampering checks
AI12-0210-1/01 Type Invariants and Generics

The following AIs were discussed and voted No Action:

AI12-0060-1/00 Overriding indicators on protected subprograms (5-0-2)
AI12-0221-1/01 Defaults for in out parameters (7-0-0)
AI12-0248-1/03 Null array and empty container aggregates (7-0-0)
AI12-0251-2/01 Parallel loop chunking libraries (7-0-0)

The following AI was discussed and placed on hold:

AI12-0296-1/01 User-defined character and null literals (7-0-3)

Detailed Minutes

Welcome

Steve welcomes everyone to the meeting.

Apologies

Jean-Pierre Rosen sent apologies. Both Jeff Cousins and Tullio Vardanega sent apologies for missing Sunday's session (both attended electronically on Monday and Tuesday).

Previous Meeting Minutes

No one has any changes to the minutes of Meeting #60. Approve minutes by acclamation.

Date and Venue of the Next Meeting

We discuss possible dates for the electronic meeting in December. We settle on December 10 at 11 AM EST.

We'll decide on the date for the other meetings later (most likely during a preceding meeting).

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

Thanks

Thanks to AdaCore for the accommodations.

Thanks to Bob for getting the morning goodies and coffee. Thanks to Tucker for getting our lunch on Friday.

Thanks to the Editor (Randy) and Rapporteur (Steve) for all of their hard work.

Unfinished Action Items

We have to reassign AI12-0021-1, since Peter left after one meeting. Justin will take the AI. He claims to be interested.

Raphael is reducing his ARG participation, so we should reassign AI12-0205-1. Brad volunteers.

We get side-tracked into a discussion of this feature. Several people question the need, especially for the formal package default. Bob and Randy both give examples where a default for a formal type would be useful. Randy suggests that the Vector container should have a default index type of Integer, for those cases where one doesn't care about the exact type. (Ada essentially does this with array declarations that just give bounds.)

The only known example that makes any sense at all for **in out** formal objects is a default storage pool. But since Ada doesn't require the standard pool to have an associated object, that's not very compelling (defaulting to some random user-defined pool doesn't seem very likely to be useful). No one has an example for a default formal package.

Brad is directed to split the AI into a defaults for formal types AI and a separate AI for the other defaults, and focus on the wording formal type part.

On Tuesday, Gary takes AI12-0205-1 from Brad so he isn't overloaded.

Steve will take AI12-0191-1 off of Tucker's hands.

Randy only worked on AI12-0112-1 a small amount (to add Allows_Exit as needed, see AI12-0286-1 below); he would like to see a final version of the stable containers AI (AI12-0111-1) before doing much more.

AI12-0016-1 has the same answer as last time.

Current Action Items

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

Steve Baird:

- AI12-0016-1
- AI12-0191-1
- AI12-0208-1
- AI12-0210-1
- Create an AI to determine which language-defined types have a language-defined Put_Image with a specified value (see discussion of AI12-0208-1).

Randy Brukardt:

- AI12-0112-1
- AI12-0262-1 (review when Brad has updated)
- Check on needed rule to ban accept statements in the body of a procedural loop (see AI12-0286-1)

Editorial changes only:

- AI12-0020-1
- AI12-0212-1
- AI12-0230-1
- AI12-0235-1
- AI12-0249-1
- AI12-0267-1
- AI12-0276-1
- AI12-0279-1
- AI12-0286-1
- AI12-0287-1
- AI12-0290-1
- AI12-0291-1
- AI12-0292-1
- AI12-0293-1
- AI12-0295-1

Gary Dismukes:

- AI12-0205-1

Bob Duff:

- AI12-0236-1

Brad Moore:

- AI12-0242-1 (see AI12-0262-1 for most recent discussion)
- AI12-0262-1 (primary author)
- AI12-0266-1

Justin Squirek:

- AI12-0021-1
- AI12-0213-1

Tucker Taft:

- AI12-0250-1
- AI12-0262-1 (review when Brad has updated)

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 14 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

AI12-0020-1/06 'Image for all types

Steve tells us about the AI.

> For an array type T, the default implementation of T'Put_Image generates an image based on (named, not positional) array aggregate syntax using calls to the Put_Image procedures of the index type(s) and the element type to generate images for values of those type{s}.

Bob suggests that the form of the address for access object should be the proper address for the machine. For instance, the segment:offset form of the 16-bit 8086. So we should allow almost anything.

Tucker suggests using "(access <anything>)".

> Otherwise the image is a left parathesis followed by "ACCESS", a space, and a sequence of graphic characters other than space or right parenthesis, representing the location of the designated object, followed by a right parenthesis, as in (ACCESS 03BEEF04).

"For a [(specific)] type extension, ..."; there isn't any other kind of type extension.

"Corresponding_Specific_Type" is annoying.

The results should by default should allow only space, <cr>, <lf>, and graphic characters. We note that any identifier could happen here, as well as any character literal.

Tucker wants arrays to use square brackets as the delimiters. We add a requirement to the wording to that effect. Randy notes that using square brackets means that null arrays and singleton arrays can use the same optimizations as other aggregates.

There's a stray period after Max_Image_Length.

Tucker wants this restriction to be in Annex H, because the only users that don't want dynamic allocation are Annex H users.

We want to just have this raise Program_Error if the restriction is violated. If there are multiple of these restrictions, effectively the smallest one is used, but there is no formal rules for that. This should work in some sense like Max_Entry_Queue_Length.

Approve AI with changes: 8-0-0.

Later, Tucker notes that 4.10 should be titled "Image Attributes". Make that as an additional editorial change to this AI.

AI12-0060-1/00 Overriding indicators on protected subprograms

Ed notes that if GNAT is allowing this, there is a B-Test missing that checks this is not allowed.

There doesn't seem to be much interest in allowing indicators on subprograms that aren't primitive, nor are bodies of primitive subprograms where overriding isn't known for the specification.

No Action: 5-0-2 (Bob, Gary abstained).

AI12-0079-1/08 Global-in and global-out annotations

Tucker recently sent a new version of this AI. He removed all of the Reachable stuff after conversations with Randy determined that Reachable wasn't going to be useful in practice.

Since that wasn't the version on the agenda, we decide to defer further discussion of this AI until everyone has a chance to review the (now simpler) AI.

Approve intent: 8-0-1. (Bob abstained)

AI12-0111-1/07 Stable containers to reduce tampering checks

Correct the summary:

> Simpl{i}fy "tampering with elements" to be equivalent to "tampering with {cursors}[elements]" for containers of elements of a definite type. For containers of an indefinite type, the original definition remains.

There is concern about the size of the additional package. Could we describe it solely with English? That would be possible, but Randy argues that we've been working hard on all new and modified interfaces to reduce the amount of English descriptions used. And the actual definition of these routines is contained in three English paragraphs – it couldn't be shorter. So we're primarily talking about the size of the specification of the package.

There seems to be no agreement on this point, and no one seems to be changing opinions. We decide to move on without an ultimate decision on this one.

Keep Alive: 6-0-1 (Ed abstains)

AI12-0208-1/04 Predefined Big numbers support

Why is "**" by Integer missing from the interface?? The subtype of the second parameter is different for the integer and rational versions.

Do we even want these to be tagged? Bob argues that there is an implementation that would get twice as large. He explains that the implementation would be a value for numbers that are sufficiently small (most them, if the value is 64-bit) and a pointer at a bit-string otherwise. Adding a tag would double the size of the item in that case. (This implementation requires it to be clear when a value is a pointer, which is hard to do on modern OSes that randomize heap locations, but there are ways around that.)

The original reason was to make constructors easier (by allowing prefixed calls), but that's not an issue if we have literals. So drop the interface and the tagged stuff. The package structure stays in the same.

Copy the A.18.2(253/2) rule for unbounded; we don't want any storage leaks.

We discuss the Type_Invariant. Tucker claims that it would be possible to write that as a postcondition, for ONE of those functions.

Bob would rather make it a written rule. Steve is very against that. Randy notes that we're trying to reduce the amount of written rules. We eventually agree to keep the Type_Invariant.

Tucker suggests To_Unbounded should be From_Bounded, From_Unbounded to To_Bounded.

All of the overridings have to be removed as these are no longer derived from an interface.

Someone wonders why To_String/From_String are not Image/Value. That would be confusing, since 'Image is also available.

Side issue: For which Language-defined types do we specify what Put_Image returns? We need to create an AI to answer that question. Action item to Steve to do this.

Here we do want specify Put_Image. We start discussing the infamous leading blank. To_String does not have the leading blank, Put_Image would put the annoying leading blank. We will have visible Put_Image routines.

We should consider using extra parameters for To_String. Support something like Text_IO.

What is the Image of the Rational case? Probably again use something like Text_IO. Fixed or Float? They are close. Try to figure something reasonable where it doesn't make sense.

Do we want Text_IO.Big_Num_IO? We did that for other language-defined types (Unbounded_String, .etc). If we have those, we don't have to hair up To_String. No consensus is arrived at.

Bob asks if the default initialization should be 0. He notes that is evil to initialize to a good value, and he would like the possibility to detect uninitialized values.

Steve suggests that it gets defined to an uninitialized value, and then a subtype that excludes the unitialized value. Then those are used on the operations. The check could be suppressed (using a mechanism similar to the one Randy defined for containers).

Tucker suggests that the names would be Big_Integer and Valid_Big_Integer. This seems consistent with existing languages. Bob says he can live with that (and he will complain about it).

Need a function Is_Valid for a Big_Integer. Do we provide a function to create an invalid Big_Integer? It doesn't seem necessary.

Approve intent: 10-0-0.

AI12-0210-1/01 Type Invariants and Generics

Tucker wonders if it would be better to use visibility. The rest of the group doesn't understand how that could work. The place checks are needed is when the specification is visible to the world, and the body can see the full type.

Tucker suggests that the we need checks when declaration is in the scope (not immediate scope) of the partial view (which is essentially the world for the library-level packages), and the body is in the scope of full type. Steve says that the body of the instance is not in the scope of the full type. That's the problem.

So Tucker suggests defining "extended scope" as Steve has it (to cover the instance body).

Bob thinks “extended immediate scope” is a contradiction. We try other terms. “Inherited immediate scope” is suggested. Uggh. “Expanded immediate scope” - that seems betters, as a generic instance is “expanded”, at least informally. We agree to use that as the term.

The child case that Steve wrote in e-mail is already plugged, because type D has no invariant (so obviously there is no check), and any conversion to T will make a check. So there isn’t a problem with that case.

Tucker suggests that the proposed change will in fact fix the formal derived type case as well. We ought add a explanation of that to the !discussion. Steve will send the editor a discussion to add to the AI.

Tucker notes that the existing wording (from our last fix) has a parenthetical remark that should be removed as it won’t be necessary. (from 7.3.2(8.4/5)).

Gary, in the !question, change a couple of whichs to thats. (But not all of them!). Change the first two “whichs” in the 3rd paragraph.

Approve AI with changes: 3-0-4 (Bob, Gary, Brad, Ed abstain)

This does not pass; we need a majority of the voters to vote to approve.

There is a concern about the implementation cost, including in shared generics.

We decide that we won’t try to fix this, and thus we need a Ramification. The AARM note on holes needs to be updated in that case. This solution should be captured in the discussion of that AI, so we can use it again if this was come up as a problem in practice.

So this AI goes back to Steve for extensive revisions.

AI12-0212-1/07 Container aggregates; generalized array aggregates

Tucker suggests solving the problem of using [in the grammar by saying that they represent themselves if at the start and end of the production. But the are such productions – for mode **in**, and for parameter_profile.

We now decide to put '[' for literal uses, and do the same thing for '|' (removing that hack).

So something like “for symbols that appear in the meta-notation, the symbols when surrounded by ‘ represent themselves.”

Steve notes (referring to the new wording to be added before 4.3.3(21)) that we need some wording to deal with the case where the first iteration has one length, and the second iteration gets a different length.

The second bullet should say that Program_Error is raised if a different number of values is produced by the second iteration.

Bob notes that are two checks, in that you have to avoid writing over memory you don’t own, and you also would want to detect ending too soon.

Ed thinks that this entire rule is an implementation detail. Randy notes that if this stuff is all user-defined, you can tell what it does.

Tucker suggests an Implementation Permission.

> The first step of evaluating a iterated_component_association can be omitted if the implementation can determine the number of elements by some other means.

And we should have Constraint_Error be raised if it doesn’t fit.

For example, if the result is constrained, one can determine the needed size from that.

This has to go after 4.3.3(31). This is a long ways away, so we need more intro:

> When evaluating iterated_component_associations for an array_aggregate that contains only iterated_component_associations with iterator_specifications, the first step of evaluating a iterated_component_association can be omitted if the implementation can determine the number of values by some other means.

Turning to a new topic. The key syntax uses **use**. Tucker tried lots of ideas.

The line 781 example should be:

```
-- The above could have been written using an explicit key_expression:
M := [for Key of Keys use Key => Integer'Image (Key)];
```

Approve AI with changes: 5-0-2 (Bob, Gary abstain)

AI12-0213-1/01 Unify record syntax

Tucker suggests "**end record** <id>"; as an option, there is no interest in leaving out the **record**. **End** by itself is disgusting.

Gary suggests that we would allow the name on end of the record representation clause. Tucker argues against that, it doesn't declare anything. Others argue that it could improve readability; it also would be consistent, otherwise you would have to worry about the context of **end record**. We eventually agree that it makes sense to allow it anywhere **end record** occurs.

The optional identifier has to match appropriately. Randy complains that a record_definition (which is the construct with the **end record**) does not iteself have an identifier. Rather, it is part of other constructs (record_type_definition, record_extension) which themselves in turn are part of a type_declaration, which is where the defining identifier comes from. So it is not simple to describe the identifier that has to match. It should be possible to describe this (perhaps by saying the "enclosing type_declaration"), we'll leave it to the author to figure that out.

Justin will take the AI.

Approve intent of AI: 7-0-0.

AI12-0221-1/01 Defaults for in out parameters

This problem can be handled with overloading. This feature hides explicit state modification from the call (a call that modifies state may only have literal parameters and a Global specification of no globals touched yet it still could modify state).

No Action: 7-0-0.

AI12-0230-1/03 Deadline Floor Protocol

Tucker tries to explain what is going on here.

Brad notes a typo in the !problem, paragraph 3, "For such {a} task..."

There is a missing comma in the !proposal: "...while a task accesses a PO{,} ..."

Gary also wants a comma in the same paragraph, "again{,} ..."

Bob sees a typo further down in the !proposal: "There {are} two styles..."

Gary, helping his reputation as Mr. Punctuation, notes another missing comma: "program mode changes{,}

Gary notes that the name of the configuration pragma is known, so the "?" should be deleted.

Gary has yet another comma to add: “anonymous type of a single_protected_declaration){,} ...”

And another to remove: “The Relative_Deadline aspect is an expression[,] of type ...”

There shouldn't be a break in the middle of the parameter definition for the subprograms in the EDF package. Format it more like the Round_Robin package or the Calendar package.

Brad notes missing commas in two of the bullets: “When a task executes a protected action{,} …” Similarly in the next bullet.

> The initial absolute ... is the result of adding the value returned by a call of Real_Time.Clock added to the value of the expression specified for the aspect Relative_Deadline aspect, where this entire computation, including the call Real_Time.Clock, is performed between task creation and the start of its activation.

Steve wonders if this is well-enough defined. We decide it is, because we can’t really be more specific in a way that would really constrain it. (It should be close to, but before, activation.)

Gary suggests to avoid “i.e.”, thus: “i.e. added to a ready queue” should be “that is, was added to a ready queue”

Approve AI with changes: 7-0-3 (Bob, Steve, Gary abstain)

AI12-0235-1/03 System.Storage_Pools should be pure

Gary “in [the]{a} distributed program”. “).” should be “.)”.

Approve AI with changes: 7-0-0.

AI12-0236-1/03 declare expressions

Randy tries to explain why a declare expression shouldn’t be a master. He says that finalization in the middle of an expression would be a new concept, and it would substantially complicate both the model and the implementation.

Tucker suggests that renaming of function calls are not a problem. He goes on to say that it could be treated like an aggregate – it should be no more complex than an aggregate. Bob notes that there is no practical difference between that and declaring an object (and that aggregates are pretty complex).

Tucker suggests that if we adopt Randy's suggestion that a declare expression is not a master, then we should allow finalization to be done early. Randy notes that he had suggested that years ago but got no traction. One has to be careful with the rules for that so that one doesn't introduce tasking (and thus data races) where none existed before. [The rejected solution (2) in the discussion of AI05-0066-1/09 suggests a possible rule, and some of the issues with allowing early finalization. It is recommended reading for the author of this AI – Editor.]

We discuss additional issues. We don't want task waiting in a declare expression; it would be hard to have a rule to do that early. We also want the accessibility to be very local so that no one can safely store accesses to these objects.

This leads to a set of restrictions: No limited types, no explicitly aliased objects, no anonymous access types, very local accessibility.

We take some straw polls:

A declare expression should not be a master, and we should have a permission for early finalization. This passes 8-0-0.

Declarations should be non-limited only for declarations. This also passes 8-0-0.

Bob suggests that within a declaration expression, a function call (whether explicit or implicit) returning a limited type is illegal.

Randy is concerned that a transformation from:

```
       C : Lim := (if B then F(<<long>>, 1) else F(<<long>>, 2));
```

to:

```
       C : Lim := (declare
                      A renames <<long>>;
                   begin
                      (if B then F(A, 1) else F(A, 2)));
```

would be illegal. This seems to be an example of the motivating case for this feature; disallowing it seems uncomfortable.

Tucker agrees that there shouldn't be restrictions on the expression returned.

Bob argues that both of these are hard to implement – that seems true but the effort seems mainly for the first (which is already legal, and which presumably is already implemented), and the second would be essentially the same.

An accessibility rule is complicated. Just say no 'access in a declare expression. And also 'unchecked_access.

Someone asked for use clauses in declare expressions. Randy is very against, as that would cause the visibility of parts of an expression to differ. It's hard enough to understand meaning when use clauses are involved without having the same name in different parts of the same expression having different meanings.

Another straw poll: Disallow use clauses in a declare expression. This also passes 8-0-0.

So to summarize: In a declare expression, there are:

- No limited top level objects.
- No use clauses.
- Constants and object renames only.
- No 'Access.
- No anonymous access.

Initially have a non-limited result restriction? It seems for many issues, the rules should be consistent with those for conditional expressions. And those allow limited results.

Justin asks if the **constant** can be dropped. No, that would hurt readability.

Approve intent: 6-0-2 (Bob, Justin abstain).

AI12-0248-1/03 Null array and empty container aggregates

AI12-0212-1 provides the missing capability by allowing the use of square brackets. The other changes did not have broad support anyway.

No Action: 7-0-0.

AI12-0249-1/01 User-defined literals

We need numeric_literals that are non-static, so 4.9(3) needs an update.

The Implementation Permission should only allow early evalution (especially at compile-time) only if Global => null.

Steve notes that the preference rule for the use of "normal" literal has a conformance issue. The literal can mean different things for the specification and for the body (since the body has visibility on the full type). That gives indigestion.

We decide to disallow them on things that have the same kind of literal. The rule should be that the full view is illegal if the partial view has the wrong kind of literal aspect.

Ed says that he thinks this is not important enough, with the exception of Bignum. Steve says that the numeric ones are enough. Tucker argues that using Unbounded_String is pain because of the lack of this feature. Randy notes that fixing that would be wildly incompatible, as many routines are overloaded with String.

Steve suggests that if the literal subprogram is a static expression function, then the literal can be static as well. That would be fine if we allowed composite types to be static, but we don't. We'd need a generalization of static to make that work – that was an idea we decided not to pursue for Ada 2020.

We discuss whether it is clear that the profile here is a Name Resolution Rule. We certainly want it to be like the other similar aspects, like Default_Iterator and Variable_Indexing. None of these are clear, but the ACATS tests (and the fact that Iterator is overloaded) certainly treat this as an resolution rule.

Since we're adding Legality Rules to prevent literals on types that already have that kind of literal, the discussion should be fixed on this point.

We’re not going to try to change any existing packages (as Unbounded_Strings would be incompatible).

Approve AI with changes: 7-1-2 (Tullio, Jeff abstain, Ed objects, he is concerned about the complexity and effort, the cost-benefit ratio is insufficient.)

Gary decided he wants to change his vote to no. Tullio wonders if he should vote against. This would fail to pass. So the discussion continues. Bignum really needs this, and Gary says he would be more in favor if we only did numeric literals.

Justin, Tuck, and Randy all argue that string literals are important (even if we can't use them in the existing Ada.Strings packages). No one wants to argue for the character literal or null cases.

So split this into three parts: numeric, string, and char/null.

Approve numeric literal AI with changes: 10-0-0. (This will remain as AI12-0249-1)

Approve string literal AI with changes: 6-0-4 (Jeff, Tullio, Gary, Ed abstain). (This will be AI12-0295-1)

Hold character and null literal AI: 7-0-3 (Bob, Ed, Justin abstain) (This will be AI12-0296-1)

AI12-0250-1/01 Iterator filters

Tucker explains the (empty) proposal. Ed gets confused by Tucker’s example.

We end up looking at an example in the proposal of AI12-0212-1.

Randy asks when this is useful. Ed says for aggregate constructors and for quantified expressions.

The intent is that these are allowed on all iterators, including aggregates, quantified expressions, and for loops.

Tucker suggests dropping the static formulation.

Randy complains that this is orthogonal with parallel and the complexity of iterators in the first place. So this adds quite a bit of extra work in every case (of which there are many).

Approve intent: 7-1-1 (Gary abstains, Randy opposes, too much complexity for the value).

AI12-0251-1/02 Explicit chunk definitions for parallel loop

Bob wonders if modular types are allowed for the simple_expression; that seems to be the case. That clearly would complicate the implementation (not a big deal, though), as different representations would have to handled (this is dynamic, most “any integer” things are static).

Steve says that task storage size in a pragma allows any integer type. It's a dynamic value as well, so there already is a precedent for this case.

Chunks here can be effectively empty.

Approve AI: 5-0-2 (Gary and Bob abstain)

AI12-0251-2/01 Parallel loop chunking libraries

We selected the other alternative.

No Action: 7-0-0.

AI12-0262-1/03 Map/Reduce attribute

Tucker would prefer putting the square brackets into value_generator:

value_generator ::= '[' [**parallel**] iterated_component_association ']'

This is based on the array iterator.

Tucker suggests calling this a value_sequence.

Approve intent: 5-0-3 (Steve, Gary, Bob abstain).

Brad will take this, and Randy and Tucker will review afterward.

AI12-0266-1/06 Parallel container iterators

Tucker worries that the use of chunk here might be incompatible with AI12-0251-1. We decide to look at that first, then later we returned to this AI.

We want the number of chunks from AI12-0251-1 to be passed as Maximum_Split. And the index from AI12-0251-1, should be mapped into the chunk index. New wording is necessary for that.

The Static Semantics of 5.5.2 does not need any change at all. The AARM note changes are needed, though.

The parallel discussion should be in 5.5.2(10, 12, 13) – that is, the Dynamic Semantics.

Several people complain about wording that mentions creating a single loop object. We need to make it clear that there is one such object per chunk. [Note that AI12-0294-1 rewords the basic for loop to reduce this problem – it might be usable as a pattern – Editor.]

Tucker suggests calling "Iterations" either "Num_Iterations" or "Iteration_Count".

The discussion then turns to whether this value is even worthwhile. It doesn't provide by itself enough information to determine the best chunk sizes to use. Moreover, it's not necessarily easy to calculate, for instance, for a container iteration between two given cursors. Iterating an extra time to figure that out seems like madness.

So we would drop that requirement, and allow chunks to be empty. That requires that Chunk_Finished be called at the beginning of the loop, to be sure that the chunk isn't empty.

Can Split return zero chunks? Randy thought not, since something has to be iterated, but others think that is also silly. So we need to adjust the subtypes. [Editor's note: This seems inconsistent with the semantics for the chunk_specification, which raises Program_Error if there are zero chunks.]

It's odd that the two similar subtypes have different names. Maybe we only need one subtype anyway, since zero needs to be in both.

Tucker wonders if these interfaces should be in different package. We look at how the existing iterators work, and there a lot of details that would have to be duplicated if we made the parallel iterator separate. So that isn't very practical.

Steve wonders what happens when a parallel iterator is Split twice. Tucker wonders if splitting could be part of creation. That doesn't work very well with the existing interface design.

So calling Split twice should raise Program_Error. If Split isn't called at all, Chunk_Count and Start_of_Chunk raise Program_Error.

Tucker would like to make the names more similar to other iterators: "First_in_Chunk", "Next_in_Chunk". (These return Null_Cursor if the chunk empty or this is the last element of the chunk, respectively.)

We could add function Is_Split, and then that could be precondition (Pre'Class) on the various routines.

We don't need to force Nonblocking on First and Next, as those would only be used for sequential loops. That's also true for Split_into_Chunks, as that occurs first.

But allowing blocking would be an issue for nested loops. We would want to ensure that an iterator object and all of its operations are nonblocking if the instance is nonblocking.

Approve intent: 6-0-1 (Bob abstains).

AI12-0267-1/06 Data race and non-blocking checks for parallel constructs

Parallel_Conflict_Checks should include Known_Conflict_Checks (at least for tasks). Use wording like that for the All_Conflict_Checks. It makes the most sense for this to have "levels" of checking, to be used depending on your tolerance for rejection.

The wording says "The implementation shall impose restrictions related to possible ..."

One needs a consequence for this, especially as to when this is enforced.

Add a sentence at the end of Implementation Requirements:

"If a compilation unit violates one of these restrictions, the unit is illegal."

Tucker suggests alternatively dropping the Static Semantics and Implementation Requirement headings (making this all a giant Legality Rule). Then, replace the lead-in to the policies with:

> Certain potentially conflicting actions are disallowed according to which conflict check policies apply at the place where the action or actions occur, as follows:

In the Known_Conflict_Checks bullet: "... and any {named} task type ..." as a task with a single instance cannot conflict.

Known_Conflict_Check needs to be enforced only in a single unit, and the wording needs to make that clear.

> If this policy applies to two concurrent actions occur within the same compilation unit, they are disallowed if they are known to denote the same object (see 6.4.1) with uses that potentially conflict. For the purposes of this check, any parallel loop may be presumed to involve multiple concurrent iterations, and any named task type may be presumed to have multiple instances.

Steve wonders if expanded instance bodies get enforced. It's not required because Legality Rules aren't enforced in expanded bodies, but the permission allows that. The permission also allows it to be enforced in subunits or anything else that the compiler can figure out.

Approve AI with changes: 5-0-2 (Gary, Ed abstain)

AI12-0276-1/01 Admission policy defined for acquiring a protected object resource

!wording is missing after !proposal.

... Unless {there is an}[the FIFO_Spinning] admission policy (see D.4.1) [is] in effect

The bracket in the first paragraph. D.4.1 is a redundant bracket.

There are too many t's in ticketing. Say that fast three times.

In the Dynamic Semantics, drop the brackets and add the missing period to the first paragraph.

As follows{:}[.] But there's only one bullet here and one rule, so get rid of this header and just define it!

In the discussion: "A sensible approach might{,} for instance{,} be to ..."

Approve AI with changes: 7-0-3 (Gary, Justin, Bob abstain)

AI12-0279-1/03 Nonpreemptive Dispatching needs more dispatching

This adds a way for a yield to appear in longer runs.

It is required that the Yield not be called if there was a block in the body. The wording should say "if and only if" in that case.

Tucker explains that there is counter in the implementation, and Yield is called only if that counter hasn't changed during the call. That can be implemented easily by making copy of the counter when the subprogram is called, and calling Yield only when the counter is unchanged. If the counter is modular, this would incorrectly call Yield only when exactly 2**N (where N is the number of bits in the counter) task dispatching calls occurred during the call – no one could tell the extra overhead in that case, even if N was 16. So this is a viable implementation (of course, the implementation could determine that a task dispatching point exists on some or all paths and eliminate the check in those cases).

Randy suggests that this should be described in the !discussion.

The inheritance model is confused.

> If a Yield aspect is specified True for a primitive subprogram S of a tagged type T, then the aspect is inherited by corresponding primitive subprograms of each descendant of T. If the Yield aspect is specified for a dispatching subprogram that inherits the value of the aspect, the specified value shall be confirming.

The summary is from a previous version of AI. It should be redone as follows.

> A yield aspect is provided to ensure that a associated subprogram encounters at least one task dispatching point during its each invocation.

Brad wonders about logical threads of control. Ed notes that parallel operations don't block. Randy notes that Yield is potentially blocking, so Yield is incompatible with Nonblocking => True.

Tucker adds this to !discussion (but it later gets moved to be a rule).

!recommendation "Non_Blocking" should be "Nonblocking".

The last sentence of the !recommendation is old and should be deleted.

It's important to ensure that the normative wording makes it clear that a call to Yield is statically included in the body.

Tucker starts rewording, and after 10 minutes or so we decide to let him finish that over lunch.

After lunch, we look at Tucker's wording and decide it is good.

We then discuss whether Nonblocking True is incompatible with Yield True. Randy says that the Legality Rule 9.5(57/5) will fail in that case (the implicit Yield is illegal in a Nonblocking => True routine). We then get hung up on whether that is going to cause a problem in an expanded generic body. Legality Rules aren't rechecked in bodies. We decide to add an explicit rule to ensure that no subprogram has Nonblocking => True and Yield => True. [This isn't enough, see later ARG discussions – Editor.]

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

AI12-0286-1/01 Allows_Exit aspect should be used on language-defined subprograms

Ed asks what this does. It makes exit and other things legal in a procedural iterator body. It essentially is a declaration that the subprogram either doesn't need clean-up, or that it uses finalization to clean-up.

Steve asks if a terminate alternative is allowed. Accept statements of all kinds shouldn't be allowed, as they aren't allowed in a procedure. We made need a rule to that effect – the editor is directed to check on that and add that to a cleanup AI.

Add to the discussion that the containers uses of Allows_Exit are handled in AI12-0112-1.

Approve AI with changes: 7-0-0.

AI12-0287-1/02 Legality Rule for null exclusions in renaming is too fierce

Randy says that he had to duplicate a lot of the wording text as he couldn't find a way to share it. Tucker suggests using sub-bullets:

- if the object_renaming_declaration occurs within the body of *G* or within the body of a generic unit declared within the declarative region of *G* and
  - the object_name statically denotes a generic formal object of mode **in out** of a generic unit *G*, then the declaration of the formal object of *G* shall have a null_exclusion;
  - the object_name statically denotes a call of a generic formal function of a generic unit *G*, then the declaration of the result of the formal function of *G* shall have a null_exclusion;

But now G is used before it is declared. So rewrite as:

- if the object_renaming_declaration occurs within the body of a generic unit *G* or within the body of a generic unit declared within the declarative region of generic unit *G*, then
  - if the object_name statically denotes a generic formal object of mode **in out** of *G*, then the declaration of the formal object of *G* shall have a null_exclusion;
  - if the object_name statically denotes a call of a generic formal function of *G*, then the declaration of the result of the formal function of *G* shall have a null_exclusion;

Ed suggests:

- if the object_renaming_declaration occurs within the body of a generic unit *G* or within the body of a generic unit declared within the declarative region of generic unit *G*, then
  - if the object_name statically denotes a generic formal object of mode **in out** of *G*, then the declaration of that object shall have a null_exclusion;
  - if the object_name statically denotes a call of a generic formal function of *G*, then the declaration of the result of that function shall have a null_exclusion;

The second paragraph of the discussion is confusing to Gary. Replace it with:

> In the case of formal objects, when the actual is a formal of another generic, both the object being matched and the actual object need to be generic formal objects of mode in out. (An actual of mode in is a constant.)

Gary: First paragraph of the discussion, second sentence, "where" should be "were".

Approve AI with changes: 5-0-1 (Steve abstains, Tucker is away getting lunch)

AI12-0289-1/01 Implicitly null excluding anonymous access types and conformance

Tucker argues that this is consistent with our previous decisions for conformance. Bob argues that there is an incompatibility with this change; Tucker says that it is easy to eliminate (add “not null” to both the spec and body; that's more readable anyway).

Approve AI: 5-1-1 (Bob is against, because of compatibility concerns; Gary abstains)

AI12-0290-1/02 Restriction Pure_Barriers

Pat Rogers explains the purpose: to have a more expressive barrier with no exceptions and no side-effects.

Tucker wonders if this isn’t essentially the same as predicate-static. He would like to ensure that the wording is as similar as possible.

- a membership test whose *tested*_simple_expression is a pure barrier eligible expression, and whose membership_choice_list meets the requirements for a static membership test (see 4.9);
- a short-circuit control form both of whose {operands}[relations] are pure barrier eligible expressions;
- a call to a predefined equality, ordering, boolean logical, or boolean logical negation operator, where each operand is a barrier eligible expression;

Randy e-mails these to the group.

Delete the details “specifical” for the !proposal about pure-barriers, cause it doesn’t match and it’s unnecessary.

Steve notes that an uninitialized protected component could cause an exception to be raised. That’s not usefully preventable.

On receiving the e-mail sent earlier, we reconsider the operator bullet.

- a call to a predefined equality, ordering, boolean logical (and, or, xor, not) operator, where each operand is a barrier eligible expression;

Steve suggests using relational rather than equality and ordering. Tucker wants to duplicate operator again.

Gary wants a hyphen in “barrier-eligible”.

- a call to a predefined relational operator or boolean logical operator (and, or, xor, not), where each operand is a pure-barrier-eligible expression;

Pat doesn’t seem to be getting ARG e-mails regularly, check that out – action item.

In the Proposal, drop the * after Pure_Barriers, and the note about GNAT. Jeff notes that the first line of the proposal has the wrong AI number (should be 291).

Improve the lead-in to:

> A scalar expression within a protected unit is said to be *pure-barrier-eligble* if it is one of the following:

Reword part of the Count bullet:

- ... denotes an entry declaration of the immediately enclosing unit;

Approve AI with changes: 10-0-0.

AI12-0291-1/01 Jorvik Profile

Steve suggests that there is an AARM note that shows the exact differences between the two profiles. Take that from the !proposal, put it right after Jorvik.

Steve suggests that one could define Ravenscar in terms in Jorvik, that is Jorvik with additional restrictions. While that's possible, it would give an appearance of change to Ravenscar users when there is no change. Since these by definition are the most conservative users, we don't want to give them any unnecessary FUD which might delay the adoption of Ada 2020. So we decide to leave the presentation as it is.

Gary suggests removing the hyphens in “electromechanical”, “burnout”.

Discussion: (See Proposal.)

Approve AI with changes: 10-0-0.

AI12-0292-1/01 Various cleanups for Ada 2020

Summary (4) should have “to”, not “too”.

Approve AI with changes: 7-0-0.

AI12-0293-1/01 Add predefined FIFO_Streams packages

Steve explains the purpose.

Steve put most of the definition of Read and Write into their postconditions. Tucker doesn’t like the organization of these. One part is an assertion about Item, and one part is an assertion about Last.

Tucker rewrites the Postcondition as:

```
overriding
procedure Read (
   Stream : in out Stream_Type;
   Item   : out Stream_Element_Array;
   Last   : out Stream_Element_Offset)
   with Post =>
      (declare
         Num_Read : constant Stream_Element_Count :=
            Stream_Element_Count'Min
               (Element_Count(Stream)'Old, Item'Length);
        begin
          -- Element_Count decreases by the number of elements read.
          -- Number of elements read is minimum of Item'Length and number of available items.
             Last = Num_Read + Item'First - 1
          and
          -- Element_Count decreases by number of elements read.
             Element_Count (Stream) =
                Element_Count (Stream)'Old - Num_Read);
```

Approve AI with changes: 8-0-0.