Minutes of the 53rd ARG Meeting

26-28 June 2015

Madrid, Spain

Attendees: Steve Baird, John Barnes (except last hour of Saturday, all of Sunday), Randy Brukardt, Alan Burns (except last hour of Friday, Saturday after midafternoon, all of Sunday), Jeff Cousins, Gary Dismukes, Brad Moore, Erhard Ploedereder (except Friday afternoon, late Saturday morning), Jean-Pierre Rosen, Florian Schanda (missed part of Friday),Tucker Taft, Tullio Vardanega (Saturday morning only).

Observers: None.

Meeting Summary

The meeting convened on Friday, 26 June 2015 at 16:30 hours and adjourned at 13:00 hours on Sunday, 28 June 2015. The meeting was held in a conference room at ETSI-T. The meeting covered all of the normal AIs and many of the amendment AIs on the agenda.

AI Summary

The following AI was approved:

AI12-0160-1/01 Adding an indexing aspect to an indexable container type (8-0-1)

The following AIs were approved with editorial changes:

AI12-0061-1/03 Index parameters in array aggregates (10-0-0)
AI12-0086-1/03 Aggregates and variant parts (7-1-1)
AI12-0155-1/06 Freezing of operation of incomplete types with completions deferred to a body (6-0-3)
AI12-0162-1/01 Memberships and Unchecked_Unions (8-0-2)
AI12-0165-1/01 Operations of class-wide types and formal abstract subprograms (7-0-5)
AI12-0166-1/01 External calls to protected functions that appear to be internal calls (8-0-1)
AI12-0167-1/01 Type_Invariants and tagged-type and formal abstract subprograms (11-0-1)
AI12-0168-1/01 Freezing of generic instantiations of generics with bodies (9-0-0)
AI12-0169-1/01 Aspect specifications for entry bodies (7-0-2)

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

AI12-0059-1/03 Object_Size attribute (7-0-2)
AI12-0064-1/04 Nonblocking subprograms (10-0-0)
AI12-0156-1/01 Use subtype_indication in generalized iterators (7-0-1)

The following AIs were discussed and assigned to an editor:

AI12-0009-1/01 Iterators for Directories and Environment_Variables
AI12-0015-1/00 Ada unit information
AI12-0017-1/01 Compile-time checked exception specifications
AI12-0075-1/01 Static expression functions
AI12-0079-1/02 Global-in and global-out annotations
AI12-0087-1/01 Reading the default value of a subtype
AI12-0125-1/03 Add Object'Inc and 'Dec
AI12-0127-1/01 Partial aggregate notation
AI12-0128-1/03 Exact size access to parts of composite atomic objects
AI12-0139-1/00 Containers and multitasking
AI12-0140-1/02 Access to unconstrained partial view when full view is constrained
AI12-0163-1/01 Deterministic queue servicing for FIFO_Queueing
AI12-0164-1/00 Max_Queue_Length aspect for protected entries
AI12-0170-1/01 Abstract subprogram calls in class-wide precondition expressions

The following AIs were discussed and voted No Action:

AI12-0014-1/01 Postconditions on subprogram bodies (8-0-1)
AI12-0019-1/01 Generic formal record types (9-0-0)

AI12-0026-1/02 Task_Safe aspect (7-0-1)
AI12-0057-1/01 Unchecked_Access for discriminant-dependent subcomponents (5-0-4)
AI12-0115-1/01 Add Size_Is_Multiple_Of aspect (10-0-0)
AI12-0122-1/01 Add 'Base for all types (7-0-1)
AI12-0126-1/01 Add Interfaces.Shifting (5-0-3)
AI12-0161-1/01 Unicode equivalents for Ada operator symbols (8-0-0)

Detailed Minutes

Apologies

Bob Duff apologizes for not being able to attend.

Previous Meeting Minutes

We have four sets of minutes to approve (the Portland meeting and three sets of minutes for phone meetings). As there are no objections, we vote for all four sets as a group.

Approve minutes: 6-0-3. (Several people abstain saying that they can't approve minutes for a meeting they did not attend.)

Date and Venue of the Next Meeting

We set the next meeting dates by e-mail to October 16-18, 2015. Robert is very ill, so we can't assume to be able to meet at his house. [Editor's note: Robert died June 30th, just a couple of days after this meeting.] Vermont Technical College, Randolph Center, Vermont, might be possible. Lexington MA will have a new office for AdaCore, so that would be plan B, AdaCore New York would be plan C (but that's rather expensive).

The summer meeting next year will be in Pisa, Italy. There is a heavily-attended festival the following weekend, which might make lodging difficult. After discussion, we decide to meet before the Ada-Europe meeting, in order to miss the tourist weekend. We'll probably need a room for ARG away from the conference site for the weekend days. The dates would be June 11-13, 2016. WG 9 would get a slot on Monday afternoon.

WG 9 Instructions

WG 9 has instructed the ARG to create a new document (either a Corrigendum or Amendment) on a three year schedule. We also have the possibility of creating Technical Specifications for things that we intend to standardize, but aren't ready for that step yet. We believe that the parallel and related facilities probably should be in a Technical Specification, as they really ought to have an implementation before we lock the wording into stone.

Alan suggests that EDF enhancements and super-Ravenscar could be a Technical Specification, rather than directly going to the Standard.

If we do an Amendment, it should be aimed at smallish improvements, not major features. The next major version of the language would be no sooner than the following update (with a 3 year cycle, that would be 2021). We agree that we want to issue some of the smaller improvements (like the **for** in aggregate syntax) as soon as reasonably possible; it's obnoxious to deny those to users for many years after the details have been decided.

Procedural note: Now that we are creating an Amendment, we now can give final approval (or No Action) to Amendment class AIs, and in particular, can revisit Hold and Promising class AIs.

Thanks

Thanks to Ada-Europe for the accommodations (excepting the outdoor temperatures – the Sunday high was 41C).

Thanks to the editor (Randy Brukardt) for taking the minutes.

Thanks to the Rapporteur (Jeff Cousins) for running the meeting.

Old Action Items

AI12-0002-1 is reassigned to Gary.

Tucker will try to get in contact with Van Snyder, in order to work on AI12-0058-1.

Erhard notes that he found the example of container usage for AI12-0111-1. He'll send it out once he's been able to sanitize it. Tucker notes that Bob and others rewrote part of the AdaCore implementation to improve the performance numbers. They should get involved with the AI.

Current Action Items

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

Steve Baird:

- AI12-0016-1
- AI12-0020-1
- AI12-0075-1
- AI12-0127-1 (with Florian Schanda)
- AI12-0128-1
- AI12-0156-1

Randy Brukardt:

- AI12-0059-1 (with help from Tucker Taft)
- AI12-0112-1 (with Ed Schonberg)
- AI12-0125-1

Editorial changes only:

- AI12-0061-1
- AI12-0086-1
- AI12-0155-1
- AI12-0162-1
- AI12-0165-1
- AI12-0166-1
- AI12-0167-1
- AI12-0168-1
- AI12-0169-1

Alan Burns:

- AI12-0164-1

Jeff Cousins:

- AI12-0015-1
- AI12-0087-1

Gary Dismukes:

- AI12-0002-1

Brad Moore:

- AI12-0009-1
- AI12-0119-1
- AI12-0163-1
- Concurrent access to separate elements of a container (see discussion of AI12-0139-1)

Erhard Ploedereder:

- AI12-0139-1 (protected containers)
- If possible, acquire some examples where Ada containers are too slow compared to other containers (for AI12-0111-1).

Florian Schanda:

- AI12-0017-1
- AI12-0127-1 (with Steve Baird)

Ed Schonberg:

- AI12-0058-1 (with Van Snyder and Tucker Taft)

- AI12-0112-1 (with Randy Brukardt)

Van Snyder:

- AI12-0058-1 (with Ed Schonberg and Tucker Taft)

Tucker Taft:

- AI12-0058-1 (produce wording, with Ed Schonberg and Van Snyder)
- AI12-0059-1 (help Randy Brukardt as needed)
- AI12-0064-1
- AI12-0079-1
- AI12-0111-1
- Alternative to AI12-0125-1
- AI12-0140-1
- AI12-0170-1
- Parallel reads for existing containers (see discussion of AI12-0139-1, probably related to AI12-0111-1)

Detailed Review

The minutes cover detailed review of Ada 2012 AIs (AI12s). The AI12s are presented in numeric order, which is not necessarily the order in which they were discussed. Votes are recorded as "for"-"against"-"abstentions". For instance, a vote of 6-1-2 would have had six votes for, one vote against, and two abstentions.

If a paragraph number is identified as coming from the working Ada 202x AARM, the number refers to the text in draft 7 of the Ada 202x AARM. Paragraph numbers in other drafts may vary. Other paragraph numbers come from the final Ada 2012 AARM; again the paragraph numbers in the many drafts may vary.

Detailed Review of Ada 2012 AIs

AI12-0009-1/01 Iterators for Directories and Environment_Variables

What would this usage look like for Ada.Directories?

```
for Dir_Entry of Ada.Directories.Entries("/", "*.a") loop
```

If Entries returns an array of Directory_Entry_Type, then the above would work.

That seems like a solution.

What about Environment_Variables?

Perhaps an array of Pairs, of a limited private type with two selector functions (Name and Value); then one can iterate over that. ("Pair" is the name of the type; that's already mentioned in the text of the package.) The type would "need finalization" so that the memory could be cleaned up. (The components would have to be pointers at strings or the like.)

The function would be called "All_Variables". That would give something like:

```
for Pair of Ada.Environment_Variables.All_Variables loop
```

Brad will take this AI back.

Someone notes that the name "Pair" might conflict via use clauses, so use "Name_Value_Pair" instead.

Keep alive: 9-0-0.

AI12-0014-1/01 Postconditions on subprogram bodies

This would not replace the Pre or Post. These are additional assertions.

Assert_on_Entry is relatively useless. It's essentially the same as Assert. Steve would like to drop Assert_on_Entry since he thinks it's just unnecessary regularity.

Florian wants to call it Post, but then you'd have two posts. Steve jokingly suggests Pre_Post.

This doesn't seem sufficiently valuable to pursue.

No action: 8-0-1.

AI12-0015-1/00 Ada unit information

Jeff says that his group does use GNAT.Source_Information. He is asked what parts of it are actually used? Jeff will need to research that to get an answer.

Jeff will take this and make a proposal.

Keep alive: 9-0-0.

AI12-0017-1/01 Compile-time checked exception specifications

Tucker cautions that Java and C++ both have given up on this. He suggests that the problem is that all of the units in between the raise and the handling have to be aware of this.

Erhard suggests that it causes problems with OOP, because a new extension can raise a new exception. But that is illegal. Additionally, the same sort of problem happens with Global.

Erhard notes that all coding standards require documentation of exceptions, it's unusual that the compiler can't know about those and help enforce them.

Tucker says that one thing that might be useful is a boolean as to whether there is any propagated exception with an explicit raise.

Florian will look at what SPARK will do, since it probably will do this someday.

Keep alive: 7-0-2.

AI12-0019-1/01 Generic formal record types

Randy says this was a fever dream of how we could do this usefully if we wanted to. But we don't want to.

No action: 9-0-0.

AI12-0026-1/02 Task_Safe aspect

Tucker thinks this is `Global => (In_Out => synchronized)`, so we don't need it separately.

No Action: 7-0-1.

AI12-0057-1/01 Unchecked_Access for discriminant-dependent subcomponents

Jean-Pierre comments that this would be OK, because it is named Unchecked something.

But Unchecked_Access is mostly checked in Ada. It's more like “Unchecked_Accessibility_for_Access”. We don't want to introduce additional holes.

No Action: 5-0-4.

AI12-0059-1/03 Object_Size attribute

Typo: problem after example: derviations

Penultimate paragraph, stray right parenthesis at end.

AARM Discussion: That's especially true [for] when following the Implementation Advice below.

“Pittsburg{h}” in next paragraph.

In wording: We don't need a special rule for the “inherently limited” cases, so get rid of the wording talking about those.

> If S is definite [and not inherently limited], {is zero or} denotes the size (in bits) ... The Object_Size of a subtype is {zero or} at least as large as the Size of the subtype. ...

Gary: in 13.3(9/1), the first comma should be a colon.

Tucker doesn't understand the Implementation Requirement. It says "would match", which is weird.

Randy tries to explain that static matching is changed to include the specified value of Object_Size. We want the default value of Object_Size to match, so that this changed rule doesn't cause an incompatibility.

Tucker proposes changing static matching to say that object size must match only if either subtype has a non-confirming specification for object size.

Then the Implementation Requirement can just say that if they statically match, the values must be the same.

Tucker would like to drop "aliased" here. The question arises as to what AdaCore does. Randy notes that the documentation isn't very helpful. Tucker looks in the on-line version of the GNAT documentation, and doesn't find much useful either.

Tucker wonders about 13.3(50). Randy says that he changed it so that Object_Size gets a priority over Size.

Absent representation clauses, the subtype doesn't come into play for GNAT. Tucker wonders what an 8-bit subtype of a 32-bit type does. One of the other GNAT implementers says that for GNAT, it would be 32 bits. So GNAT ignores 13.3(50). Randy notes that Janus/Ada ignores that advice as well, because most discrete subtypes have weird sizes that aren't helpful to users.

Tucker suggests that what's important is components; stand-alone isn't too important.

So the suggestion is that we use the same list of items for Object_Size as in 13.3(50-52). (all components and aliased stand-alone objects).

Randy worries that doing so might cause problems with other compilers. Steve notes that the problem with the old Rational compiler (now Apex from PTC) is with stand-alone objects. Thus, if we restrict the effect of object size to components, we don't need the escape hatch of the zero value. So Tucker wants to get rid of that.

In order to preserve the old implementation advice, we need Implementation Advice stating that if Size is specified for a subtype which allows for efficient independent addressability on the target architecture, then the Object_Size has the same value.

Typo in Reason: "That's need".

Randy will take this back, and Tucker will provide assistance and review.

Approve intent: 7-0-2.

AI12-0061-1/03 Index parameters in array aggregates

Tucker says that this is a start toward aggregates for containers. That seems a lot more complex than this proposal.

Typo: 2nd line of 2nd paragraph "would have use".

Jean-Pierre notes that <> is not allowed here. But you don't need it, because if you aren't going to use the identifier introduced by **for**, you can just use the discrete_choice_list directly.

Erhard would like **others** here. That's already allowed. Any named array choice that is currently allowed would allow adding the **for**.

Jean-Pierre suggests as less radical syntax change which would make it clearer that the **for** is just a prefix:

```
array_component_association ::=
   [for defining_identifier in] discrete_choice_list => expression
 | discrete_choice_list => <>
```

The problem with this is that we use "iterated_component_association" a lot. That's not worth changing (it would be hard to change, as "iterated_component_association" is the scope of the identifier; we have to have some syntactic construct to use for the scope).

4.3.3(31) is in the wrong place, it should be after 4.3.3(23.1/4). The group agrees.

Jean-Pierre notes a typo in the first paragraph:

> We don't need to change 4.3.3(17), since the wording [for] applies to all forms of array_component_associations, including these new ones.

4.3.3(17) does need a change, though, because an iterated_component_association is not a discrete_choice_list. So we need to add:

> The discrete_choice_list of an array_component_association {including an iterated_component_association} is allowed...

Add after 4.3.3(43) (a new example):

```
G : constant Matrix :=
          (for I in 1 .. 4 =>
             (for J in 1 .. 4 =>
                (if I=J then 1.0 else 0.0))); -- Identity matrix
```

Typo in the Example:

```
type Task_Array {is array} (Worker_Indexes) of Worker;
```

Approve AI with changes: 10-0-0.

AI12-0064-1/04 Nonblocking subprograms

Tucker explains the name and wording.

Jean-Pierre says that the value is either True or Maybe. (Some laughter.) His point is that Nonblocking = False doesn't mean that the routine definitely is blocking, just that it isn't guaranteed to be nonblocking.

Steve asks if it can be specified for an entry; Tucker says that it does allow that. Steve wonders what the value would be for an entry.

Probably don't want to specify it on entries, as that would be confusing (they're always potentially blocking), even if the contents of the entry body are nonblocking.

Randy had noted in e-mail that we need mechanisms for formal subprograms and access-to-subprograms. We agree that these are needed.

Maybe it would be better to not allow aspect Nonblocking on any protected operations, only allowing it on the protected unit (that makes it a static check for the unit; it's always required dynamically).

Predefined operations should be covered. Randy notes that there is a problem with record equality, because it can incorporate user-defined operations. Tucker suggests that the value comes from the operations that make up predefined equality for a record. Randy notes that dispatching calls are possible for equality, so you can't know the extension components (and thus the extension component's equality operators). Ideally, we'd require all equality to be nonblocking, but that would be incompatible.

Florian notes that extensions require keeping the same blocking, so we have to be careful with standard libraries as someone can extend stuff. Storage_Pools, for instance, could not have Nonblocking = True.

Randy worries that containers will be tough, because we surely want them to be nonblocking. But that would require user-written hash functions to be nonblocking, which might be incompatible. (Again, if we had this for Ada 2005, they'd be required to be nonblocking.)

Tucker suggests allowing Nonblocking = False on an instantiation. Steve notes that formal instances would also need this possibility.

Randy notes that an alternative would be to somehow have the nonblocking state of the actual for a formal subprogram used as part of the definition of some of the exported operations. We'll have to have a mechanism like this for Global, so it might make sense to define it here, too. This would look something like:

```
with Nonblocking => Hash'Nonblocking and "="'Nonblocking;
```

Brad notes that a procedure call on something might actually be an entry. It could have been a renames of an entry or an interface primitive operation implemented by an entry. It appears that 9.5.1 doesn't handle that (already) – such a call has to be potentially blocking, but it's not in the 9.5.1 list. We need to fix 9.5.1 and here as well to support that.

Randy thinks that a rename of an entry would not have Nonblocking = True, as it (like all other properties) is inherited from the renamed entity. But we need to prevent someone from declaring Nonblocking = True for an entry (either in a renames or a primitive operation for an interface).

Approve intent: 10-0-0.

AI12-0075-1/01 Static expression functions

Steve had identified some issues that would need to be addressed. How should they be handled?

The aspect would require that all of the subtypes are static, and all operands would need to be static. (In particular, "statically unevaluated" would not apply outside of static subexpressions.) Only **in** mode parameters would be allowed.

For a use to be static, all actual parameters would need to be static (even if they're not used in the actual expression).

[What about completions? Would we allow the aspect on the specification of a (normal) function, which would require that the completion be an expression function that meets the static requirements? Allowing this would allow static functions to be used on private types. - Editor.]

Steve will take a crack at handling this.

Erhard suggests that no access to global variables be allowed in a static function.

Keep alive: 8-0-1.

AI12-0079-1/02 Global-in and global-out annotations

Tucker gives an overview of the proposal. A few interesting points: Input is assumed to be the default mode. The specification **synchronized** indicates no data race is possible; this specification is intended to support the parallelism work.

The SPARK implementation supports some of this notation (but not the access type stuff, obviously).

The notation is similar to aggregate syntax, but it's not an aggregate.

Steve worries that a call back can modify things that aren't local nor global to the current subprogram. Tucker suggests that he meant dynamically global.

The same things we talked about for Nonblocking (see discussion AI12-0064-1) are needed here: predefined operations, access-to-subprogram, and formal subprograms need handling. We also need to do something for language-defined packages (looks like trouble to the Editor).

Florian worries that SPARK allows no Global specifications and in that case it (SPARK) would generate them as needed. Tucker suggests that essentially you are adding (with a tool) the appropriate Global aspects. Certainly, that is OK. Portable Ada would require Global annotations.

For a package, does that mean only the hidden state, or all of the package state? Randy notes that we previously just covered the hidden state, as the other state can be explicitly mentioned.

Steve notes that SPARK as a case with limited views includes the whole state. Perhaps we should have limited vs. normal views expose this difference of all vs. hidden.

Randy wonders what happens with the language-defined packages, where they tend to be built on top of other packages that we (the language) can't talk about. This could constrain implementations in bad ways.

Steve wonders what the Global for Calendar.Clock would be, because Global => null suggests that subsequent calls will return the same value. Tucker suggests that some volatile object would be read. Which brings up that Volatile is also an interesting problem yet to be addressed.

Also need **access** subtype_mark here (anonymous access).

Florian notes that we need some mechanism for referencing the set for a formal subprogram (the actual thereof) or access-to-subprogram. That needs to be added for subprograms that call these sorts of things (think Hash in containers).

Steve wonders if we need a mechanism for naming a global list so it doesn't have to be repeated. Florian says that the subprogram naming mechanism (formal subprogram) would serve that purpose.

Steve notes there are issues with anonymous access-to-subprogram; also via a dispatching call on a class-wide parameter.

Tucker seems to think that **synchronized** will be used often as "don't worry, be happy". But of course that only works for the parallel uses, not the purity/analysis uses.

Constants that aren't really constants (many composite constants) need to be handled somehow.

Keep alive: 10-0-0.

Much later, Steve notes that if you do an allocator, the storage pool should show up on the global list somehow. Randy notes that this really is about implicit calls getting reflected in these annotations – not only storage pool calls (Allocate, etc.) but also Initialize/Adjust/Finalize and calls that come from default initializations.

AI12-0086-1/03 Aggregates and variant parts

Should we change this from Promising to something else??

The wording is still weird, it's the existing part that is causing the trouble.

Tucker will try reword 4.3.1(17/3) again. After some other discussion, he comes up with:

> If a variant_part *P* is nested within a variant *V* that is not selected by the discriminant value governing the variant_part enclosing *V*, then there is no restriction on the discriminant governing *P*. Otherwise, the value of the discriminant that governs *P* shall be given by a static expression, or by a nonstatic expression having a constrained static nominal subtype. In this latter case of a nonstatic expression, there shall be exactly one discrete_choice_list of *P* that covers each value that belongs to the nominal subtype and satisfies the predicates of the subtype, and there shall be at least one such value.

Approve AI with changes: 7-1-1.

Jeff is opposed because he does not think this is worth bothering with.

AI12-0087-1/01 Reading the default value of a subtype

What happens if the Default_Value is unset? Probably illegal plus Program_Error in a generic body.

Someone suggests an 'Has_Default_Value attribute that could be tested before asking for 'Default_Value. Jeff says that would be analogous to calling Exists before attempting to read an environment variable. [Editor's note: Actually, calling Exists is a potential race condition; it should be avoided in favor of handling the exception or using the two parameter Value. 'Has_Default_Value at least would be safe in that the value would be determined at compile-time. But it would only work in a generic body, as the matching use of 'Default_Value would be statically illegal in other contexts.]

Florian notes that you can declare an uninitialized variable to find out the default value.

What's the use case for this attribute? One doesn't want to duplicate the value across the program. The example:

```
pragma Assert ((for all I in 1..10 => A(I) /= S'Default_Value));
```

It would prevent mix-ups. It's much like First/Last.

Punt on the Normalize_Scalars part, because that typically sets the default to an invalid value.

Do we want to do the same thing for Default_Component_Value? Seems like it should be available.

Jeff will attempt words.

Keep alive: 7-1-1.

Gary does not think we need to add this, it's just another little feature of minimal value.

AI12-0115-1/01 Add Size_Is_Multiple_Of aspect

This seems to have been a very specialized need, insufficient for including in the Standard.

No Action: 10-0-0.

AI12-0122-1/01 Add 'Base for all types

Steve asks about an extension of a constrained type; that has to keep the constraint.

“italized” in the !proposal.

Florian says that he finds Predicate_Free would be useful in various cases.

Steve asks about private predicates; that is discussed in the !discussion.

Now one has to do the two-step declaration:

```
type T_Base is ...
subtype T is T_Base ...
   with Dynamic_Predicate => ...;
```

The work-around is not **that** painful. Visibility of T'Base would matter, and in a parameter it would have different properties in the specification vs. the body when there is a partial view. Would it even conform??.

There seem to be a lot of worms crawling out of both of these ideas (T'Base and T'Predicate_Free). Neither idea is worth trying to get all of the worms back in the box.

No Action: 7-0-1.

AI12-0125-1/03 Add Object'Inc and 'Dec

Tucker wonders if these can be renamed as subprograms.

What did we do for X'Image? It's a value (so the answer is no).

Jean-Pierre suggests that we define S'Inc which is a procedure, and then X'Inc is defined in terms of it in the same way as X'Image.

Erhard would suggest 'Incr and 'Decr.

Florian asks why did we not do X'Succ. That doesn't solve the problem of duplicating the X.

Someone asks about the problem with +=? That doesn't resolve well, there are cases where X := X + 1; and X :=+ would resolve differently.

Tucker suggests that this is an operator which is automatically defined if "+" is defined, following the model of "/=".

Jean-Pierre complains that this is new syntax. X'Inc is not new syntax, it's much less intrusive.

Tucker thinks that it might make sense that "+=" is not allowed to be redefined. Tucker would like to write this up as an alternative, which would be AI12-0125-2.

Straw poll (should there be a mechanism): 8-0-1.

C syntax: 6-2-1.

Keep alive: 9-0-0.

AI12-0126-1/01 Add Interfaces.Shifting

"Shoft" (twice) in !problem

Tucker suggests adding an assertion that the modulus is a power of 2.

Why does this declare a new type? That rather defeats the purpose, and it's Address_to_Access_Conversions all over again. Tucker says that he thought we wanted these primitive. That seems unnecessary, and someone can use renames if they really want that. So drop the type "shiftable".

Jean-Pierre says that this is the value of this is rather low, and the cost is also rather low.

Florian says that cryptographic algorithms look ugly because the use of "shift_left", etc. That's not really this AI, he will make a separate item for that.

No Action: 5-0-3.

AI12-0127-1/01 Partial aggregate notation

Fix spelling of "trapazoid".

Fix the problem and example to make more sense:

```
procedure Foo (Fighters : in out Rec)
   with Post => (Fighters = (X => Fighters'Old.X,
                             Y => Fighters'Old.Y, ...
```

```
                                    Colour => Black,
                                    Shape  => Trapezoid));
```

and

```
procedure Foo (Fighters : in out Rec)
   with Post => (Fighters = (Fighters'Old with delta Colour => Black,
                                                     Shape  => Trapezoid));
```

The array case should be handled. But it doesn't work well for multi-dimensional arrays, because you want to be able change just a single element [say, (3,4)].

Assign to Florian and Steve to provide an array definition as well as an early cut at wording.

Examples might help.

Keep alive: 9-0-0.

AI12-0128-1/03 Exact size access to parts of composite atomic objects

Randy suggests that we look at Robert's suggestion toward the end of the Appendix.

C.6(20) is wrong for writing volatile components that aren't the size of a machine scalar, those have to have a pre-read and it doesn't allow that. Argubly that volates the Dewar rule, but it should be said explicitly.

C.6(22) has to be modified in the case of the new aspect (whatever it is), as the components are also volatile but we wouldn't want this rule to apply (it's the problem, in fact).

Tucker doesn't think that this should say anything about “volatile”; this is somewhat orthogonal.

This should be over and above “Atomic” and “Volatile”. Tucker thinks it might be even the default; Erhard objects that we don't want to force word reads vs. byte reads. And we really don't want to change behavior (thus potentially breaking programs that currently work.)

Tucker thinks that we need to support this on types, so it can be passed to subprograms.

It shouldn't be allowed to pass one of these subcomponent by-reference (the called routine would not know to use the larger size).

Erhard argues that this should be no action, because the users are “just plain lazy”. They should be writing full record operations, not operations on individual components. Randy and Tucker disagree. Randy notes that even if the user wanted to do the right thing (as Erhard means it), they could easily accidentally write a component. And the results of that could be really bad.

Jean-Pierre suggests that there is an aspect that prevents accessing subcomponents if an address clause is given. That seems to confuse multiple issues.

Randy had proposed that the object that represents the hardware register be given an aspect (name TBD, Exact_Size_Only in the AI) that prevents access to the components of that object. It can only be copied whole; it doesn't apply to the type as that's used to declare the temporary (and one definitely has to access the components of the temporary). That forced people to write the correct code, and makes C.6(20) true for such objects. Randy agrees with Erhard on one point: it's not necessary for the language to do this for the user, it needs just to do enough to prevent accidents.

Tucker would like some sort of specification that we get independent addressibility on the whole item, and on nothing less. [Editor's note: not sure what independent addressibility has to do with this.] He would like it to work on components of a larger type. For instance, the top level represents a set of registers (one example would be low address, high address, and access rights such as read/write for an MMU), the intermediate level a register, and the bottom level bit fields within a register (the access rights register in the example), and the new aspect would be applicable to the intermediate level, not top or bottom.

Jean-Pierre/Erhard suggest calling the aspect Dependent_Subcomponents.

Steve will take a stab at writing up Tucker's ideas on this.

Keep alive: 9-0-0.

Fix the difficulty on this one to Hard.

AI12-0139-1/00 Containers and multitasking

What issues do we have?

Parallel reads should be possible in existing containers. Assign this to Tucker (it's probably related to AI12-0111-1) as new AI.

Concurrent access to containers? Specifically, referencing separate elements of a container in parallel. (Consider a vector add, each element is added to the corresponding element and assigned into a result vector's elements.) Brad will take this as a new AI. This probably would be additional requirements.

General purpose concurrent reads/writes of any container. (Any series of operations.) We could imagine adding protected sets, protected maps, etc. Iterators and the like are a problem. It might make sense for this to be a Technical Specification as some prototyping might be required. Erhard will take this AI (139).

Keep alive: 10-0-0.

AI12-0140-1/02 Access to unconstrained partial view when full view is constrained

Randy tries to explain the problem: look at UNC2 in the !discussion of the AI.

Steve notes that characteristics inherit for “composite types”, there is no corresponding wording for access types.

Tucker argues that there is only one view of a type at any particular point of of code. No matter where the use originated, in his view.

Randy doesn't see any mechanism for this change of views, in particular there aren't any rules that would make that be the case. In particular, 3.10.1(2.1-2.6) depend on that not being the case; otherwise, we would have a ripple effect. Because of 3.10.1(2.1-2.6), one can't apply the Dewar rule in this case.

Tucker will make an attempt to figure out a model that fixes this.

Keep alive: 7-0-0.

AI12-0155-1/06 Freezing of operation of incomplete types with completions deferred to a body

Randy explains that Tucker requested that he redo the rules so that incomplete types never freeze, adding Legality Rules to eliminate any problems. He added the narrowest possible Legality Rule and a notwithstanding rule to freezing, eliminating the special cases from 13.14(3/4).

Steve would like a redundant note: “In contrast, freezing a partial view does have an effect.” Tucker suggest that this is an AARM note.

Redo the summary to: “freezing of an incomplete view has no effect”. Drop the parenthetical remark.

That doesn't satisfy everyone. Change the summary to:

> Freezing of an incomplete view has no effect. In particular, it does not freeze the completion of the view.

It's still considered sloppy. One more time:

> Freezing of an incomplete view of a type has no effect. In particular, it does not freeze the completion of the view.

Change the subject similarly.

Approve AI with changes: 6-0-3.

AI12-0156-1/01 Use subtype_indication in generalized iterators

Steve wonders if the accessiblity of the anonymous access type would be a problem; we would want the accessibility to be the same as that of the array type.

Tucker wonders if the loop parameter has the correct accessibility, there doesn't seem any rule for that. That probably matters even without this change (one can iterate over an array or container without giving the element subtype).

Give to Steve to figure out whether there is a problem and if so how to fix it.

Approve intent: 7-0-1.

AI12-0160-1/01 Adding an indexing aspect to an indexable container type

Randy explains the problem. The fact that this could cause Pre'Class to mean something different than the same expression evaluated explicitly shows that this needs to be fixed.

Approve AI as is: 8-0-1.

AI12-0161-1/01 Unicode equivalents for Ada operator symbols

Jean-Pierre says that this would create more problems then it would solve. Even accented letters causes problems for programs.

Several people bring up ideas for additional operator symbols, but no one speaks in favor of using Unicode characters for those. Other ideas are definitely a separate proposal.

No Action: 8-0-0.

AI12-0162-1/01 Memberships and Unchecked_Unions

Adjust the wording to make it clear what subtype mark we're talking about:

> Evaluation of a membership test if the subtype_mark {of a membership_choice} denotes a constrained unchecked union subtype and the {*tested*_simple_expression}[expression] lacks inferable discriminants.

Jean-Pierre has suggested instead:

> Evaluation of [a] {an individual} membership test if the subtype_mark {(if any)} denotes a constrained unchecked union subtype and the {*tested*_simple_expression}[expression] lacks inferable discriminants.

We prefer this second version.

Approve AI with change: 8-0-2.

AI12-0163-1/01 Deterministic queue servicing for FIFO_Queueing

Randy notes that there was a form of mass hysteria over this proposal. Everyone was talking about a new configuration pragma – but we have such a configuration pragma, Queuing_Policy. The group agrees that this is indeed a new queuing policy, essentially combining FIFO_Queuing with D.4(12).

Randy had suggested the name Deterministic_FIFO_Queueing. Tucker suggests Ordered_FIFO_Queueing. (He says that we don't want to suggest that the others aren't deterministic.)

Jean-Pierre notes that textual order is programable:

```
select
   accept E1;
else
   select
      accept E2;
   else
      select
         accept E3;
         ...
```

Brad will take the AI and provide wording.

The ASIS section should say “No effect.”, as this is just a parameter to an existing pragma.

Erhard says that the best idea is FIFO between the queues. That is, the entries are processed in the order of arrival. (That requires a sequence number in the queue.) That would be fair and deterministic. (As noted in e-mail, textual order is not fair, in that under heavy load the latter entries can starve.)

Keep alive: 8-0-1.

AI12-0164-1/00 Max_Queue_Length aspect for protected entries

Should this be per type or per object? Tucker suggests that it should be both. He says that it would be weird to have global (via the existing restriction) and specific entry, but not type level.

Should there be an attribute to query it? We'd have to define the value of the attribute when the aspect is unspecified – but that could be just the maximum number of tasks, it wouldn't have to be illegal like Default_Value (see AI12-0087-1).It is not clear what useful could be done with an attribute that wouldn't cause a race condition, so having an attribute doesn't seem necessary.

Tucker suggests calling the aspect "Max_Entry_Queue_Length" so that it clearly states what it is setting, and that way it has the same name as the related restriction.

Steve suggests that only smaller value (than the restriction) are supported. We don't want a situation where you set the restriction Max_Entry_Queue_Length, but still have to check all over the program to see if anyone has overridden it with aspect Max_Entry_Queue_Length.

It is suggested that one usually wants a particular Max_Entry_Queue_Length, but there are one or two exceptions.

Alan says that he thinks the aspect would be used normally in a non-Ravenscar program, where you know a particular type does not have more than 2 callers. And so on.

So we will only allow narrowing from the restriction.

These aspects would be static.

The subject should be changed to match the contents.

Alan will take the AI.

Keep alive: 10-0-0.

AI12-0165-1/01 Operations of class-wide types and formal abstract subprograms

Typo in discussion: "Generic_Dispatching_Consturctor"

Tucker wonders why 12.6(9.1/3) says "may be available" instead "are available". Seems like that would be better, but it isn't necessary for the purposes of this AI.

12.6(8.e/3): This means that the actual is [either] a primitive operation …

Approve AI with changes: 7-0-5.

AI12-0166-1/01 External calls to protected functions that appear to be internal calls

Randy explains the problem. The default parameter case falls over in GNAT and Janus/Ada (that's an Ada 95 example).

The last sentence of 6.1.1(34/3) is unnecessary, as the action has to be started for that to happen. So just delete it as well.

9.5(7.1/3): "an protected" should be "a protected". A protected operation includes a protected entry, so it should just be "a protected operation".

> A call on a protected function that occurs within a precondition expression (see 6.1.1) or a default_expression of a parameter_specification for a protected operation shall not be an internal call.

Steve complains that the wording is ambiguous as to whether "a protected operation" applies to both preconditions and default expressions.

Also, Steve worries that a "reasonable" nested protected type (uh-huh) would be rejected by this rule. So we need to make it more specific.

Jean-Pierre notes that a nested protected type is not a useful thing, as within a protected action, nothing else can happen (you've already got exclusion, you don't need more). So don't try to fix the wording for that, it's not worth our time.

> A call on a protected function that occurs within a precondition expression (see 6.1.1) of a protected operation or a default_expression of a parameter_specification of a protected operation shall not be an internal call.

Refactor the wording:

> An internal call on a protected function shall not occur within a precondition expression (see 6.1.1) of a protected operation nor within a default_expression of a parameter_specification of a protected operation.

The following AARM reason is missing the closing parenthesis.

The first editor's note after the reason can be deleted. [Editor's note: I moved it into the discussion instead; it seems relevant to the design of the rule – otherwise we could just have banned internal calls in (any) default parameters and (any) preconditions.]

Approve AI with changes: 8-0-1.

AI12-0167-1/01 Type_Invariants and tagged-type and formal abstract subprograms

We have to explain to some ARG members who "Will Robinson" was. Apparently, not everyone remembers Lost in Space.

The summary is confused. The more we discuss it, it gets more and more wrong.

Replace it with:

> If the modification of an object of type T that is a component of a class-wide object happens within the scope of the full view of type T does not cause an invariant check for T.

Still not liked. Try instead:

> If an object of type T that is a component of a class-wide object is modified within the scope of the full view of type T, then there is no invariant check for T at that point.

Add an AARM note after 7.3.2(20/3):

> In all of the above, we are only referring to the parts that are known statically to be of type T.

Erhard objects to "known statically". Randy suggests that we could make this negative (talk about the cases that we don't require a check).

> To Be Honest: In all of the above, for a class-wide object, we are only referring to the parts of the specific root type of the class.

After 7.3.2(20.a/4):

> Despite this model, if an object of type T that is a component of a class-wide object is modified within the scope of the full view of type T, then there is no invariant check for T at that point.

Typo: "principal" should be "principle" toward the end of the question.

Add at the end of 7.3.2(23.a/4): "Similar holes exist for class-wide objects as discussed above."

Approve AI with changes: 11-0-1.

AI12-0168-1/01 Freezing of generic instantiations of generics with bodies

Randy explains that the wording change of AI12-0103-1 had an unintended consequence of changing the freezing rules for a generic instance.

First question: "uninitended".

"...will [have]{contain} a [construct] proper_body..."

"there is no distinct construct generic_body."

Approve AI with changes: 9-0-0.

AI12-0169-1/01 Aspect specifications for entry bodies

Where should the aspect specification go?

Straw poll: Can live with entry barriers before: 7, can't 2.

Tucker argues that the entry barrier should be part of the body, and the existing syntax is wrong.

Straw poll: Before entry barrier: 3; after entry barrier: 3; undecided: 3

This was the most conclusive straw poll ever. We'll leave the AI as written. Remove the editor's note from the discussion.

Approve AI with changes: 7-0-2.

AI12-0170-1/01 Abstract subprogram calls in class-wide precondition expressions

The change is missing for 7.3.2(5/4) "{nonabstract}" goes after "(notional)".

The operations of NT are also nonabstract, so the rule against calls of abstract subprogram does not trigger for a class-wide precondition or postcondition. There should be an AARM note to this effect after 6.1.1(7/4).

This takes care of the problem for derived types.

Randy worries that we have a problem for the initial declaration. He gives an example:

```
package Pkg3 is
   type AT is abstract null record;

   function F1 (X : At) return Boolean is abstract;

   function F2 (Y : At) return Boolean
      with Pre'Class => F1 (Y);
end Pkg3;

O : At'Class := ...;

if F2(At(O)) then ... -- OK, but the precondition calls F1 for AT.
```

This seems legal (F2 is not abstract, and there is no rule against type conversions to an abstract type), but the precondition would call an abstract subprogram.

Tucker will take this back and try to figure out a solution to prevent this problem. [Editor's note: Perhaps 6.1.1(18.2/4) or something like it needs to apply to the initial declaration.]

We're not going to try to solve the default parameter problem.

Keep alive: 12-0-0.