Minutes of the 41st ARG Meeting

18-20 June 2010

Valencia, Spain

Attendees: Steve Baird, John Barnes, Randy Brukardt, Alan Burns (except Sunday), Brad Moore, Erhard Ploedereder (left early on Sunday), Jean-Pierre Rosen, Ed Schonberg, Tucker Taft, Tullio Vardanega.

Observers: Greg Gicca, Steve Michell (Friday only).

Meeting Summary

The meeting convened on 18 June 2010 at 11:40 hours and adjourned at 14:10 hours on 20 June 2010. The meeting was held in the conference room at Adeit on Friday, and in a conference room at the Reina Victoria Hotel on Saturday and Sunday. As expected, the meeting covered about three quarters of the entire agenda.

AI Summary

The following AIs were approved :

AI05-0122-1/01 Private with and children of generics (9-0-0)
AI05-0142-4/07 Explicitly aliased parameters (7-0-3)
AI05-0151-1/07 Allow incomplete types as parameter and result types (10-0-0)
AI05-0209-1/01 Universal operators of fixed point types (again) (7-0-3)

The following AIs were approved with editorial changes:

AI05-0114-1/01 Conflicting definition of Letter (7-0-1)
AI05-0127-2/02 Adding Locale capabilities (9-0-0)
AI05-0137-2/01 String encoding packages (6-0-3)
AI05-0139-2/06 Syntactic sugar for accessors, containers, and iterators (8-0-2)
AI05-0144-2/07 Detecting dangerous order dependences (9-0-0)
AI05-0146-1/04 Type invariants (8-0-1)
AI05-0147-1/10 Conditional expressions (9-0-0)
AI05-0155-1/03 'Size clause on type with nonstatic bounds (9-0-1)
AI05-0159-1/08 Queue containers (10-0-0)
AI05-0163-1/01 Pragmas in place of null (8-0-1)
AI05-0165-1/01 Inheriting non-conformant homographs (9-0-0)
AI05-0170-1/03 Monitoring the time spent in interrupt handlers (10-0-0)
AI05-0171-1/04 Pragma CPU and Ravenscar profile (8-0-2)
AI05-0173-1/01 Testing if tags represent abstract types (7-0-1)
AI05-0174-1/03 Implement Task barriers in Ada (10-0-0)
AI05-0176-1/07 Quantified expressions (7-0-3)
AI05-0184-1/03 Compatibility of streaming of containers (8-0-0)
AI05-0185-1/01 Wide_Character and Wide_Wide_Character classification and folding (8-0-0)
AI05-0210-1/01 Correct Timing_Events metric (10-0-0)
AI05-0211-1/01 No_Relative_Delay should not allow relative timing events (10-0-0)
AI05-0216-1/01 No_Task_Hierarchy is still wrong (8-0-1)
AI05-0217-1/01 Record extensions and "immutably limited" (10-0-0)
AI05-0219-1/01 Pure permissions and limited parameters (8-0-1)

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

AI05-0153-1/06 Subtype predicates (7-0-2)
AI05-0167-1/03 Managing affinities for programs executing on multiprocessors (10-0-0)
AI05-0182-1/01 Preciseness of S'Value (7-0-1)
AI05-0183-1/04 Aspect Specifications (9-0-1)
AI05-0188-1/05 Case expressions (9-0-0)
AI05-0189-1/02 Restriction No_Allocators_After_Elaboration (8-0-0)
AI05-0206-1/01 Remote_Types packages should be able to depend on Preelaborated packages (8-0-0)
AI05-0214-1/01 Default discriminants for limited tagged types (10-0-0)

The following AIs were discussed and assigned to an editor:

AI05-0051-1/08 Accessibility checks for class-wide types and return from nested-extension primitives
AI05-0110-1/00 Characteristics of generic formal derived type are not inherited
AI05-0111-2/01 Specifying a pool on an allocator (without reachability checks)
AI05-0115-1/04 Aggregate with components that are not visible
AI05-0117-1/01 Memory barriers and volatile objects
AI05-0125-1/01 Nonoverridable operations of an ancestor
AI05-0190-1/03 Global storage pool controls
AI05-0191-1/01 Aliasing predicates
AI05-0200-1/00 Mismatches in formal package declarations
AI05-0202-1/01 Task_Termination and Exceptions raised during finalization
AI05-0215-1/00 Errors in AI05-0030-2
AI05-0218-1/00 Generics and volatility
AI05-0220-1/00 Definition of "needed component"
Future AI to extend the base on literals to 36.

The following AIs were discussed and voted No Action:

AI05-0111-1/09 Specifying a pool on an allocator (7-0-0)
AI05-0127-1/02 Adding Locale capabilities (9-0-0)
AI05-0186-1/05 Global-in and global-out annotations (9-0-1)
AI05-0195-1/01 Uninitialized scalars (6-0-2)
AI05-0213-1/01 Formal incomplete types (8-0-2)

Detailed Minutes

Meeting Minutes

John had a few fixes, which he will send to the editor after the meeting.

Approve minutes by acclamation.

Date and Venue of the Next Meeting

Next meeting: October 29-31, 2010. Fairfax VA co-located with the SIGAda conference.

Should there be a February meeting? We ought to be finalizing the Ada 2012 Amendment around then. So we should plan on a meeting. Where? New York would work. Tentatively, February 18-20, 2011.

[Note: The five of us who went to the very slow lunch after the meeting on Sunday (Greg, John, Randy, Steve, and Brad) discussed the topic of the location of the February meeting and were unanimous that we'd rather go somewhere warmer than New York in February, especially given how expensive New York is to stay at. We talked about whether Greg should host something again, or whether there is another site that would work. West coast (i.e. drafting Gary or Joyce or Steve) seems a bit far to travel for the Europeans. No one minded going back to St. Pete, though. We'll need to discuss this again at the next meeting.]

RM Review

Randy reports that he didn't finish the AARM draft until the week before this meeting, and thus he decided to delay the start of the review until after the draft update for this meeting. He expects to start the review in mid-July, but will leave a long time to do it with an ending date of October 1.

ASIS

WG 9 decided to delay the ASIS standard for a year in order to include Ada 2012 support. It was also suggested that we treat ASIS changes on an ongoing basis, in particular we ought to include any ASIS changes needed to each new AI. We started doing that with the wording for the Standard, and that eliminated the need to revisit all of the AIs years later (and also caused many problems to show up early).

Randy says it makes most sense to start that next time, that is with the AIs that will follow the completion of Ada 2012. With 220 AIs so far, it would be a lot of work to go back and consider all of them. Jean-Pierre says that not many AIs would require ASIS changes and those would be easy. After some argument with Jean-Pierre's position, Tucker announces that we have our volunteer for this task! Jean-Pierre is given an action item to look at the Ada 2012 AIs and propose ASIS changes.

Scope

WG 9 approved the scope without comment. Note that this limits new Amendment items to those that we can reasonably relate to the ones in the scope document; anything else has to wait until next time (no matter how small).

Correct classification of AIs

Randy sent around a note commenting that there were a number of Binding Interpretation AIs that appear to be pure extensions. He recommended that we consider changing some or all of these to Amendment classifications, specifically AI05-0015-1, AI05-0030-2, and AI05-0032-1.

Each of these three AIs defines a new feature that extends existing features; there is no semantic problem with the existing features, they are just more limited than is prudent.

Randy is asked if this would cause a problem with the scope document. He replied that there would be no problem as he included them in the scope document in anticipation of a possible classification change.

Change AI05-0015-1, AI05-0030-2, and AI05-0032-1 to Amendment classification.

Approve change in classification: 10-0-0.

Thanks

Thanks to the host (Ada_Europe) for the accommodations on Friday, and to Tullio Vardanega for making the weekend arrangements.

Thanks to Randy Brukardt for his valuable efforts as editor, webmaster, and more.

Old Action Items

Pascal didn't do his task. Ed and Randy had tasks assigned by the last Amendment phone call that they were not able to complete. Otherwise, all of the action items were finished.

New Action Items

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

Steve Baird:

- AI05-0110-1
- AI05-0218-1

- AI05-0220-1
- Participate in the Amendment subcommittee.

John Barnes:

- Draft AI to extend the base on literals to 36.

Randy Brukardt:

- AI05-0111-2 (simple version)
- AI05-0119-1 (from Amendment subcommittee)
- AI05-0177-1 (from Amendment subcommittee)
- AI05-0182-1
- AI05-0214-1
- Participate in the Amendment subcommittee.

Editorial changes only:

- AI05-0114-1
- AI05-0127-2
- AI05-0137-2
- AI05-0144-2
- AI05-0146-1
- AI05-0147-1
- AI05-0155-1
- AI05-0159-1
- AI05-0163-1
- AI05-0165-1
- AI05-0170-1
- AI05-0171-1
- AI05-0173-1
- AI05-0174-1
- AI05-0176-1
- AI05-0184-1
- AI05-0185-1
- AI05-0210-1
- AI05-0211-1
- AI05-0216-1
- AI05-0217-1
- AI05-0219-1

Alan Burns:

- AI05-0117-1
- AI05-0167-1

Gary Dismukes:

- AI05-0115-1

Bob Duff

- AI05-0190-1
- AI to add wording to explain which non-attribute aspects can be specified via aspect clauses, and the exact semantics for such specification. (See discussion of AI05-0183-1.)
- Participate in the amendment subcommittee

Pascal Leroy:

- Create AI to combine wording in AI05-0006-1 with other parts of the language (from his editorial review).

Brad Moore:

- AI05-0206-1

Erhard Ploedereder:

- AI05-0191-1

Jean-Pierre Rosen:

- Study all of the Ada 2012 AIs and propose ASIS changes to support the AI changes.

Ed Schonberg:

- AI05-0158-1 wording (with help from Steve Baird)
- AI05-0200-1
- Participate in the Amendment subcommittee.

Tucker Taft:

- AI05-0051-1
- AI05-0111-2
- AI05-0125-1
- AI05-0139-2
- AI05-0153-1
- AI05-0183-1
- AI05-0189-1
- AI05-0202-1
- AI05-0215-1
- Participate in the Amendment subcommittee.

Detailed Review

The minutes for the detailed review of AIs and SIs are divided into ASIS Issues (SIs) and Ada 2005 AIs (no SIs were considered at this meeting). The AIs and SIs 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 Amendment (with a /2 version), the number refers to the text in the Final Consolidated AARM. Paragraph numbers in earlier drafts may vary.

Detailed Review of Ada 2005 AIs

AI05-0051-1/08 Accessibility checks for class-wide types and return from nested-extension primitives

Steve Baird wonders if we need to even mention the constrained discriminant case; doesn't that fall out without additional wording? Tucker says he was trying to preserve the existing meaning.

Steve says in 4.8(10.1/2), the wording doesn't seem to work for access discriminants that come from defaults. Tucker says that doesn't matter, it is only the type that matters.

Steve says that doesn't work, you don't know statically whether or not you have access discriminants. Consider **new** T'Class'(P.**all**) where P is **access** T'Class; the specific object might have discriminants. But you cannot know that statically.

Steve prefers much less verbiage:

"A check is made that the accessibility level of the anonymous access type of each access discriminant (if any) is not deeper than that of the type of the allocator."

Tucker doesn't think this is right.

After more discussion, it is agreed that Steve and Tucker will take this off-line.

On Sunday, Tucker provided the following new wording, but we never got back to this AI to review and possibly approve it:

Replace 4.8(5.2/2):

> If the designated subtype of the type of the allocator has one or more unconstrained access discriminants, then the accessibility level of the anonymous access type of each access discriminant, as determined by the subtype_indication or qualified_expression of the allocator, shall not be statically deeper than that of the type of the allocator (see 3.10.2).

with:

> If the subtype determined by the subtype_indication or qualified_expression of the allocator has one or more access discriminants, then the accessibility level of the anonymous access type of each access discriminant shall not be statically deeper than that of the type of the allocator (see 3.10.2).

Modify 4.8(10.1/2) as follows:

> For any allocator, if the designated type of the type of the allocator is class-wide, then a check is made that the accessibility level of the type determined by the subtype_indication, or by the tag of the value of the qualified_expression, is not deeper than that of the type of the allocator. If the [designated subtype] {subtype determined by the subtype_indication or qualified_expression} of the allocator has one or more [unconstrained] access discriminants, then a check is made that the accessibility level of the anonymous access type of each access discriminant is not deeper than that of the type of the allocator. Program_Error is raised if either such check fails.

AI05-0110-1/00 Characteristics of generic formal derived type are not inherited

The problem is if the category changes, the category of the formal derived type needs to change as well. So 12.5.1(21) needs to change.

Steve Baird will take this one.

AI05-0111-1/09 Specifying a pool on an allocator

Too complex.

No Action: 7-0-0.

AI05-0111-2/01 Specifying a pool on an allocator (without reachability checks)

Jean-Pierre asks how Allow_Reference works. Tucker explains that the To count goes up one. A handle needs a list of handles that it has on other subpools. When that handle goes away, it has to decrement the reference counts all of designated subpools.

A handle is a pointer. When the reference count of a handle goes to zero, it is reclaimed. That does not have any effect on the items in the subpool.

The implementation has to worry about cycles between handles of subpools. Because these are immutable, this is easy.

Erhard complains that removing the access reachability checks just removes the safety and it is not that big of a change.

Steve notes that the reachability check would be the first check on an assignment that requires a deep check at every access value in the record (all the way down). Current Ada semantics only requires checks for top-level constraints on assignment (never any checks on components of composite objects).

Randy doesn't believe any of this reachability/reference stuff is worth having. Pointers are unsafe and they are going to stay that way. He believes that their best use is as a building block for better abstractions, and those abstractions can and ought to provide the safety (exhibit A is the unbounded containers of Ada).

Erhard notes that if you forget an Allow_Reference, you can get dangling pointers without using Unchecked_xxx. That seems to make things more unsafe than they currently are.

That argues for some reachability check. Perhaps we need another alternative containing minimal reachability checks.

Randy would rather go in the other direction, dropping Allow_Reference and any form of automated recovery. He says that in his model, the only way to cause a subpool to go away is to call Unchecked_Deallocate_Subpool. That would be exactly as safe as the current version.

Ed worries that a reachability check on a formal parameter might require passing information along with a parameter. Tucker says that is correct, you would have to pass this information: but only if you know that it has one of these parameters. But for class-wide types, you cannot know whether a component might need such a check (one could be added to a later extension), so there is a distributed overhead.

Jean-Pierre says that the goal is to get Mark/Release. That is very unsafe by itself; it seems weird to require it to be safe.

We either need to add reachability checks, or we need to eliminate the reference stuff and have no automatic deallocation.

We take a straw poll in hopes of deciding on a direction:

Add reachability checks: 4
Remove automatic deallocation: 3
No subpools: 1

Tucker and Randy will produce alternatives for both ideas and they will shoot it out next time.

AI05-0114-1/01 Conflicting definition of Letter

Randy wrote this up suggesting no change because of compatibility concerns. Changing the behavior of Ada.Characters.Handling and Ada.Strings.Maps would be a very bad idea.

Tucker suggests a user note:

“There are certain characters which are defined to be lower case letters by ISO 10646 and are therefore allowed in identifiers, but are not considered lower case letters by Ada.Characters.Handling or Ada.Strings.Maps.Constants.”

AARM Reason: This is to maintain compatibility with the Ada 95 definitions of these functions.

Approve AI with changes: 7-0-1.

AI05-0115-1/04 Aggregate with components that are not visible

We assign this to Gary, as Tucker, Randy, and Steve had tried for a long time to work this out and were unable to get it right.

AI05-0117-1/01 Memory barriers and volatile objects

Typo: The form of a pragma [CPU]{Coherent}

Question: does it make sense that it is with respect to other coherent objects only? Erhard suggests that that is relative to all other memory writes. Tucker says that it’s not all memory, it is relative to any external effects. Tucker says that what Erhard wants to do is synchronize external effects.

Tucker thinks that saying that coherent is an external effect is sufficient. Those already cannot be updated.

Coherent is strictly weaker than Volatile.

Integrate this with the other pragmas in C.6. Make it really clear what is different from Volatile. Volatile implies Coherent.

Randy says that he doesn’t believe that Volatile writes directly to memory. There is an Implementation Note that says that. But that is not normative at all. Someone asks what other languages mean by Volatile. C and Java use this term, but not in quite this way. Robert Dewar has noted that GNAT will pretty much have to use the definition of Volatile provided by the GCC back-end, and that does not require writes directly to memory (which requires barriers or other "extra" instructions).

Randy says that he thought “memory” includes hardware caches.

Tucker says that memory has got much more complex over the years; the thinking that led to these pragmas in the early 1990s is way too simplistic today.

So maybe we ought to update the definition of Volatile. Even for Atomic we don’t necessarily want to go directly to memory.

That seems like a more productive solution.

Tucker suggests C.6(16) be changed. It currently reads "For a volatile object, all reads and updates of the object as a whole are performed directly to memory." Randy says that he thinks this means “memory subsystem” (meaning that caches are included). Tucker suggests rewriting the Implementation Note might be sufficient, saying that coherent cache is OK.

It should ensure that all clients see the right order; beyond that we don’t care.

Keep Alan Busy: 10-0-0. (Keep alive).

AI05-0122-1/01 Private with and children of generics

Approve AI: 9-0-0.

AI05-0125-1/01 Nonoverridable operations of an ancestor

Randy notes that this is a real problem because there is no workaround. Dispatching to the wrong routine can't be programmed around without abandoning dispatching. Thus this seems to require some sort of fix. Randy notes that this problem occurred several times during the design of Claw, forcing a deeper child hierarchy than was really desired. It also causes horrible bugs, but overriding indicators have already fixed those (an error will occur if **overriding** is used and the routine does not override).

Tucker will try to write wording to make it work.

Jean-Pierre worries about the inconsistency implied by making it work. He wonders if there is any examples of using this "feature" in its current form. Ed and Steve also express concern about the inconsistency.

Randy suggests adding another overriding indicator - **force overriding**. Jean-Pierre jokes that we could use **overriding in all case** without any new keywords.

Tucker suggests just having **overriding** do this, currently that would be illegal, and now would work. It would be illegal if you didn't give any indicator. In that case, there is no inconsistency.

This would make **not overriding** significant. That's ugly, and might cause problems with some generic.

We could make an incompatiblity instead by making **not overriding** illegal as well. That seems like the best idea; we think that using this "feature" intentionally will be very rare.

Tucker will attempt to word this version.

AI05-0127-1/02 Adding Locale capabilities

This alternative is too ambitious; we'll consider the other, simpler alternative.

No Action: 9-0-0.

AI05-0127-2/02 Adding Locale capabilities

The _Unknown constants should be in the other order to be consistent with everything else.

Brad would like to use 639-3 as the standard rather than instead of 639-2/T. That seems OK. Randy notes that the unreferenced standards should not be mentioned in the biblography.

Approve AI with changes: 9-0-0.

AI05-0137-2/01 String encoding packages

Jean-Pierre notes that Robert never sent the String version of the package. Jean-Pierre isn't sure that the String version is really useful. It seems harmless, consistent, and not completely trivial, so we ought to have it.

Naming: Randy had noted that there is an issue with the wide string version of this package appearing under Ada.Strings. This could go under Ada.Characters, or it could directly be a child of Ada. Trying to keep it in Ada.Strings doesn't work well, because the parent would have to be duplicated elsewhere.

Randy's suggestion was Ada.Characters.UTF_Encoding.String_Encoding, etc.

Tucker suggests making these plural and dropping the second Encoding:

Ada.Characters.UTF_Encodings.Strings
Ada.Characters.UTF_Encodings.Wide_Strings

Someone wonders if Randy's complaint is correct. After some studying of the Standard, Randy's memory of Ada.Strings proves wrong, the prefix of everything is Ada.Strings. He probably was confusing it with Ada.Characters. So there is not a major problem with using Ada.Strings, move it back there.

The double Encodings is too much (Ed). But the inner plural is weird.

Ada.Strings.UTF_Encoding.Conversions
Ada.Strings.UTF_Encoding.Strings
Ada.Strings.UTF_Encoding.Wide_Strings

Approve intent of AI: 7-0-2.

On Sunday, Jean-Pierre presented an AI updated with the changes previously recommended. We agree that the changes all have been made.

Approve AI (with changes presented, now version /02 of the AI): 6-0-3.

AI05-0139-2/06 Syntactic sugar for accessors, containers, and iterators

Steve Baird thinks that there is a matching issue for generic derived types for reference types. They are required to have only one discriminant in order to get the special semantics. He says that the corresponding type may have additional discriminants. He gives an example:

```
type T (D : access I) and Reference is tagged...

type NT (D : access I, D2 : Integer) is new T with null;

generic
   type FT is new T with null record;
package G is

    -- reference of FT used in body.

package I is new G (NT);
```

Tucker says that in the generic body, the use of a reference could only see the one visible discriminant. The added ones don't participate. So FT would be a reference type. Tucker says that the only purpose of this restriction is so that we know what discriminant to use, so it is not critical to enforce this in a generic body.

Moving on to indexing. Tucker explains how the choice between constant and variable indexing is determined. Essentially, constant is preferred unless a variable is required (assignment, **in out**, **renames**).

Tucker asks everyone to mentally remove the "(reversible)" from the two iterator object definitions. Also "(reverse)" from the iteratable object definition. These just seem to make the wording more confusing than it has to be; they'll be replaced by an additional sentence.

Ed says that Iteratable is a class in Java, with this spelling. So this name is OK.

Erhard wonders why Reference is not an aspect. We never thought about that; we had started with interfaces and only changed to aspects for indexing when the interfaces didn't work out. That seems like a better idea.

Tucker suggests aspect Implicit_Dereference => <name of access discriminant>

Then we don't need the exactly one discriminant rule (and the generic discussion above).

Tucker explains the rules for iterators.

Randy wonders if “next element” of an array is well-defined. He would have used the array index to describe that which is well-defined. Tucker thinks this is well-enough defined.

Steve Baird wonders if the loop parameter is finalized. Tucker says that this is a view of the value, so it doesn't need to be finalized. Randy objects, this is the return of a function value, we need to talk about when it is finalized. But it is a renames of the function result, not a copy of the function result; it follows the rules for renames vis-a-vis finalization.

John wonders why it only works for one dimensional arrays. He wanted multidimensional arrays. It is noted that implies that some order is defined – but in what order? He wonders if these are worthwhile at all.

Steve Baird says that we have the idea of implicit iteration over multidimensional arrays for streaming. That is a defined order. So we could use that to define an order for multidimensional arrays.

Straw vote on arrays: No arrays: 1; Single-dimension array: 3; Multidimensional array: 5.

So try the multidimensional array.

Randy suggests perhaps splitting the iterator text into various sections, because the “in the second form” stuff is rather confusing and there is very little shared text.

Try to describe “reverse” with a separate sentence, the parentheses are just too confusing.

Approve intent: 8-0-2.

AI05-0142-4/07 Explicitly aliased parameters

John would like this better if it said “by reference”. He then says it is getting too close to lunch and withdraws his comment.

Tucker points out that this is really about returning references to elements of containers, other capabilities sort of fell out and were not the main thrust.

Approve AI: 7-0-3.

AI05-0144-2/07 Detecting dangerous order dependences

Add “(see below)” after the first use of “known to be unvarying”.

"Known to be unvarying" is generating heartburn in some of the ARG.

Ed suggests just dropping renames from these rules. Jean-Pierre objects, saying that this is where this check is most valuable (the objects don't *look* the same, but are the same).

It is odd that "known to be unvarying" leaves out indexed_components. We could add it by saying it is known to unvarying index components. This does seem like a slippery slope of complexity.

We turn to Tucker, since he is the one insisting on adding these checks (recall that the compromise for allowing **in out** parameters in functions is adding these checks). Thus, if Tucker is satisfied that they go far enough, the rest of us are as well.

After some thought, Tucker suggests just requiring renaming to require static expressions for index expressions, and dump known to be unvarying. Tucker goes on to say he would prefer to require the expressions to be "constant". But that brings up issues with access-to-constant pointing at access-to-variable. Tucker thinks such code is very tricky, and it is OK to reject it; that is, if a user has a "constant" which gets intentionally modified, it is OK to ignore that modification for the purpose of making checks – most readers would make the same inference.

Tucker says that “constant” by itself doesn’t work; there are other problems.

Randy worries that we've said nothing about how we are going to fix the dereference case. Saying "constant" doesn't work well there because constant really isn't constant for composite types. Tucker will rewrite this bullet, dropping known to be unvarying.

On Sunday, Tucker presents his new wording:

Add after 6.4.1(6):

Two names are *known to denote the same object* if:

- both names statically denote the same stand-alone object or parameter; or
- both names are selected_components, their prefixes are known to denote the same object, and their selector_names denote the same component; or
- both names are dereferences (implicit or explicit) and the dereferenced names are known to denote the same object; or
- both names are indexed_components, their prefixes are known to denote the same object, and each of the pairs of corresponding index values are either both static expressions with the same static value or both names which are known to denote the same object; or
- both names are slices, their prefixes are known to denote the same object, and the two slices have statically matching index constraints; or
- one of the two names statically denotes a renaming declaration whose renamed *object_*name is known to denote the same object as the other, and any expression within the renamed *object_*name contains no references to variables nor calls on non-static functions.

AARM Reason: This exposes known renamings of slices, indexing, and so on to this definition. In particular, if we have

```
C : Character renames S(1);
```

then C and S(1) are known to denote the same object.

We need the requirement for dereferences and index expressions to be "known to be unvarying" in renames in order to avoid problems from later changes to those parts of renamed names. Consider:

```
   type Ref is access Some_Type;
   Ptr : Ref := new Some_Type'(...);
   X : Some_Type renames Ptr.all;
begin
   Ptr := new Some_Type'(...);
   P (Func_With_Out_Params (Ptr.all), X);
```

X and Ptr.**all** should not be known to denote the same object, since they denote different allocated objects (and this is not an unreasonable thing to do).

We don't need a similar requirement for slices as the existing requirement for statically matching index constraints eliminates any problems (the index constraints either have to be static or declared by the same subtype declaration).

End AARM Reason.

== End new wording ==

It is noted that the AARM Reason needs revision. [Editor's note: This shows a problem with the revised wording. The example in the Reason is still valid, but the proposed wording would have X and Ptr.**all** be known to be the same object. We need to add some wording about prefixes of dereferences.]

The !discussion should show the false positives that could occur from an access-to-constant pointing at a variable (and those are too tricky to care about). [Editor's note: the existing AARM To Be Honest note will be retained and serves this purpose. It uses **in** parameters, which is an even more likely case in practice.]

Approve AI with changes: 9-0-0.

AI05-0146-1/04 Type invariants

!discussion – the last paragraph before Assertion Policy is misleading. The only hole is from passing 'Access of a non-externally visible subprogram out of the abstraction. The client cannot cause the hole (without help). Tucker will improve this paragraph to make it clear.

Should we have both type and subtype predicates (referring to this AI and AI05-0153-1)? These are reasonably different (although you could get a similar effect using subtype predicates carefully). Tucker suggests calling this aspect "type_invariant". Probably also call the other one "subtype_predicate" (AI05-0153-1).

Steve Baird asks what about stream attributes? They pass out results. We need to add text to cover that, to the bulleted list.

Steve Baird wonders if this applies to inherited subprograms. The wording talks about explicit subprograms, so it would not apply. Tucker notes that the model is that this is done in the body. He goes on to point out that the semantics of inherited routines includes a conversion as part of the call, so that would cover what is needed.

Steve Baird and Tucker will try to word a stream attribute addition.

On Sunday, Tucker presents updated wording; the stream bullet is new.

- After default initialization of an object of type T, on the new object;
- After conversion to type T, on the result of the conversion;
- After a call on the Read or Input stream attribute of the type T, on the object initialized by the stream attribute;
- Upon return from a call on any subprogram or entry that:
    - has a result of type T, or one or more **in out** or **out** parameters of type T,
    - is explicitly declared within the immediate scope of type T (or is a part of an instance of a generic unit declared within the immediate scope of type T), and
    - is visible outside the immediate scope of type T or overrides an operation that is visible outside the immediate scope of T,

  on the result object of type T, and on each **in out** or **out** actual parameter of type T.

Approve AI with changes: 8-0-1.

Randy abstains as he is concerned about confusion between AI05-0146-1 and AI05-0153-1.

AI05-0147-1/10 Conditional expressions

We discuss Steve's wording proposal in the second last message in !appendix. Tucker thinks this can be simplified, but others say he is wrong. He decides to drop his objection.

Erhard thinks this should be much simpler. The explanation is that it is as easy as we can make it, because the context is imposed in different ways.

Randy worries that the wording “single” type is wrong in the last bullet. Others disagree. Randy looks up the definition and points out that it is wrong for this usage.

Erhard argues that conversions need to work. Tucker suggests that we define that the type of the conditional expression is the target type, and then each operand is converted.

Steve and Tucker will reword this.

On Sunday, Tucker presents the revised wording.

Name Resolution Rules

If a conditional_expression is expected to be of a type *T*, then each *dependent*_expression of the conditional_expression is expected to be of type *T*. Similarly, if a conditional_expression is expected to be of some class of types, then each *dependent*_expression of the conditional_expression is subject to the same expectation. If a conditional_expression shall resolve to be of a type *T*, then each *dependent*_expression shall resolve to be of type *T*.

The possible types of the conditional expression are further determined as follows:

- If a conditional expression is the operand of a type conversion, the type of the conditional expression is the target type of the conversion; otherwise
- If all of the dependent expressions are of the same type, the type of the conditional expression is that type; otherwise
- If a dependent expression is of an elementary type, the type of the conditional expression shall be covered by that type; otherwise
- If the conditional_expression is expected to be of type *T* or shall resolve to type *T*, then the conditional expression is of type *T*.

Legality Rules

All of the dependent expressions shall be convertible to the type of the conditional expression.

Dynamic Semantics

All of the dependent expressions are converted to the type of the conditional expression.

== End of new wording ==

These rules ought to be consistently worded in terms of syntax entities (or the reverse), not half-and-half.

The lead-in to the bullets should be “a conditional expression”, the first bullet should be “the conditional expression”.

Jean-Pierre asks about fixed point: if X is a fixed type object, is (**if** F **then** X **else** X*X) legal? Yes, if the context identifies a unique type.

[Editor's note: This dynamic semantics rule doesn't make sense in context; see mail of July 26, 2010 for details and the solution used.]

Approve AI with changes: 9-0-0.

AI05-0151-1/07 Allow incomplete types as parameter and result types

The previously approved wording was incompatible with Ada 2005. We also decided to make access-to-subprogram consistent with direct subprogram_declarations. This simplified the wording.

Approve AI: 10-0-0.

AI05-0153-1/06 Subtype predicates

The changes are relatively minor.

John wonders if we shouldn't restrict this to elementary as that fixes the holes. Randy wonders if that would cause a generic contract problem. We probably could fix that.

Randy notes that we could safely allow access to the discriminants and bounds. Those would be the only things visible in the predicate expression. (Or the object itself when elementary.)

Steve and Tucker are considering adding a requirement to recheck the predicate after an assignment to a component, including via parameter passing (**in out**). They would want to take that offline.

We take a straw poll as to whether to keep the general facility with holes, or a restricted facility without holes:

0	Drop AI
2	Only elementary
5	Only elementary and bounds and discriminants
1	Anything, with extra checks

So we need wording to add rules for bounds and discriminants. Steve and Tucker will do wording.

On Sunday, Tucker presents his proposed wording:

3.2.4 Subtype Predicates

For any subtype, the following language-defined aspect may be specified:

Subtype_Predicate

> This aspect shall be specified by an expression. The expected type for the expression is any boolean type. A Subtype_Predicate may be specified on a type_declaration or a subtype_declaration; if none is given, an implicit "**with** Subtype_Predicate => True" is assumed.

Legality Rules

Within the expression of a Subtype_Predicate aspect_specification for a composite type, the only components of the type referenced shall be discriminants, and a name that denotes the current instance of the (sub)type shall be used only as a prefix of a selected_component for a discriminant, or as a prefix of an attribute_reference with attribute_designator being Length, First, Last, or Range.

Static Semantics

The *predicate of a subtype* is defined as follows:

- For a (first) subtype defined by a derived_type_declaration, the specified Subtype_Predicate, and-ed with the predicate of the parent subtype, and-ed with the predicates of any progenitor subtypes.
- For a (first) subtype defined by a non-derived type declaration, the specified Subtype_Predicate.
- For a subtype created by a subtype_declaration, the specified Subtype_Predicate, and-ed with the predicate of the subtype denoted by the subtype_mark.
- For a subtype created by a subtype_indication that is not that of a subtype_declaration, the predicate of the subtype denoted by the subtype_mark.
- For a base subtype, True.

[Editor's note: The predicate has no effect on the static or dynamic semantics of the subtype indication except as noted here. In particular, it has no effect on the range of scalar subtypes.]

=== End of proposed wording ===

Jean-Pierre asks for an AARM note explaining the reasons for this restriction.

Randy objects to having different rules for composite and access-to-composite. John agrees.

Tucker says that he would prefer to restrict this to not accessing variables.

Steve complains that would cause people to call a function to do global stuff.

Ed argues that we don’t need any restrictions.

Jean-Pierre says that the issue with variable components is where these things are enforced. That only affects the current instance, not global variables. [Editor's note: I believe that this comment is mis-recorded; neither I nor any reviewers (including Jean-Pierre himself) understand why this would be the case.]

Tucker argues that a collection is like a global array indexed, which is going to be allowed. Randy would rather that it isn't allowed. It is noted that such a ban wouldn't be very effective, since it always could be done via a function call.

Tucker gives in and says that he will just change to composite and access-to-composite in the rule.

Approve intent: 7-0-2.

AI05-0155-1/03 ‘Size clause on type with nonstatic bounds

Steve Baird points out that this doesn’t mention the upper bound on sizes. Tucker thinks that’s overkill.

Legal? should be followed by (No.) in the example of the !question.

Approve AI with changes: 9-0-1.

AI05-0159-1/08 Queue containers

Reword A.18.32 to say “...except that the generic formal Element_Type is indefinite.”

A.18.30: Erhard asks what Minimum_Priority is. It is the parameter to the routine in question.

Tucker thinks that the name Minimum_Priority is wrong since Before is “>”. Call the parameter Low_Priority.

The fact that the profiles of the routines differ from the interface is confusing. This is how Ada works, so there isn't much we can do about it. But we can make it clear that these routines are overriding by adding overriding indicators. So, add **overriding** to all of the operations that are overriding and **not overriding** to the others (Dequeue_Only_High_Priority).

Approve AI with changes: 10-0-0.

AI05-0163-1/01 Pragmas in place of null

Erhard and Tucker object strongly to removing the Design Principles and Implementation Advice.

Steve says that 1.a needs to remove “and is strictly true at the syntactic level”. Tucker agrees.

Erhard says the next sentence is not true either.

Steve suggests replacing “erased” by “replaced by unrecognized pragmas”, because then the syntax is not involved.

The discussion needs to be rewritten. Tucker will do that.

Jean-Pierre would rather restrict what pragmas appear in place of null. He would like to see only “meaningful” pragmas. But that doesn’t handle implementation-defined pragmas or unknown pragmas.

Approve AI with changes: 8-0-1.

AI05-0165-1/01 Inheriting non-conformant homographs

This has a lot of discussion, which follows the same lines as described in the AI's discussion section. Eventually, the position explained in the AI (this is way too much work to make it illegal, given there is no known semantic problem) wins out.

Drop the !qualifier, since there is no omission.

Approve AI with changes: 9-0-0.

AI05-0167-1/03 Managing affinities for programs executing on multiprocessors

We need a dynamic semantics rule that the pragma expression is evaluated.

Tucker thinks that the type Dispatching_Domain ought to be limited.

Erhard says first sentence in the Name Resolution Rules has the wrong type name: System.Multiprocessors.Dispatching_Domains.Dispatching_Domain is right.

Probably the CPU parameters should default to Not_A_Specific_CPU.

Should the parameters of Assign_Task and Set_CPU be ordered in different order? No, that would be confusing because not all of the routines would be changed. In addition, the default would then not make sense as the other parameters are defaulted.

There should be space after the comma in lists of parameters (in function Create, for instance).

Dynamic Semantics, third paragraph. Erhard wonders about “initially”; this has nothing to do with initialize. Tucker suggests “currently”.

The second paragraph of the Dynamic Semantics: what does “failed” mean? Is it like an abort? These are about the pragma. The text needs to say that (or ordering might provide it, but probably best to write it out).

Calls to Create can be restricted to running in the environment task before the start of the main program. There needs to be some wording to say that.

Jean-Pierre asks if exceptions are raised or propagated? Randy says that library routines “propagate” exceptions, checks “raise” exceptions. So this should be “propagated”. (But the standard isn’t very consistent about that.)

Tucker asks how these will be used; he wonders about initialization on declaration. Could Create be a constructor function? Alan thinks that would be good, then there couldn’t be any uninitialized objects. (Would need (<>) on the type.)

Alternatively, we could have discriminants that are First and Last CPU. But that would force the type to be controlled so that other things could be done (such as setting up some table).

Should Assign be able to assign a task to the System dispatching domain? Is this a no-op, or are later assignments illegal? Alan will consider this in the next draft.

System_Domain is a deferred constant.

If parallel assignment is to be supported, it needs to be said somewhere.

Jean-Pierre asks if predefined dispatching domains should be required to be in some particular package. There doesn't seem to be any value to that. Not sure if there is any point to this documentation requirement: implementers will have to document any such things as they are implementation-defined. Otherwise, no one could use them!

Randy notes that since you can't share CPUs, predefined dispatching domains would take up the CPUs and prevent further changes. In that case, calling Create is always going to fail. Arguably, having such dispatching domains is a bad idea. We will let Alan decide what to do here. Jean-Pierre says that perhaps an AARM note is appropriate.

Approve intent of AI: 10-0-0.

AI05-0170-1/03 Monitoring the time spent in interrupt handlers

Jean-Pierre wonders why the package is optional; if the package doesn't exist, the Separate_Interrupt_Clocks_Supported is pointless. You'd have to with the package to use it (which would be an error if it didn't exist). So the package shouldn't be optional.

Raise an exception if the Booleans are False, since it's bizarre to call these when they aren't implemented. Use Program_Error for this.

Wrong spelling of initialize.

Approve AI with changes: 10-0-0.

AI05-0171-1/04 Pragma CPU and Ravenscar profile

Dynamic Semantics "given immediately within the declarative_part of a subprogram_body has no effect unless it appears in the main subprogram".

The position of pragma is not a Name Resolution Rule, it should be a Legality Rule. Neither is the rule about the expression being static. The Legality Rules heading is missing after the first paragraph.

Steve Michell wonders why this is defined in the Real-Time Annex rather than the core (section 9). It is less fundamental than Priorities, which is not in the core, so surely it does not belong in the core either.

Put the package first, then the pragma, then everything else. Randy suggests putting all of the package rules first (Tucker says there is only one) and then everything else.

We need a dynamic semantics rule that the expression of the pragma is evaluated.

Approve AI with changes: 8-0-2.

AI05-0173-1/01 Testing if tags represent abstract types

Identifies_Abstract_Type would be a more accurate name. Most however would just prefer Is_Abstract. Tucker says that the word "type" doesn't appear here and Is_Descendant is similarly referencing the type.

Change the discussion to reflect this.

Tucker wonders about Size and Alignment; those could be added here. We think we'll wait until there is a clear usage.

Approve AI with changes: 6-0-1.

AI05-0174-1/03 Implement Task barriers in Ada

The clause title should be Synchronous Barriers (Ada has another meaning for barriers, and we shouldn't confuse people into referencing the wrong clause).

Steve Baird notes that there is no private part here, but as there is a limited private type there ought to be one (using the normal boilerplate for that).

This type “needs finalization”, like a protected object. Add some wording to that effect.

The [selection]{mechanism} for determining...

Steve wonders what happens if one of these is used in ATC. He notes that it would be best if ATC and abort work the same way. The wording needs to be expanded to include ATC. Tucker suggests saying “When a call on Wait_for_Release is aborted...” [Editor's note: This wording doesn't make sense in the context of the existing wording; as of this writing, there is an ongoing discussion on the proper form of the wording.]

Make the package Preelaborate. Should this be Remote_Types? It isn’t useful directly, but they might want it. Task operations like rendezvous generally aren’t allowed between partitions. So don’t make this remote types.

Steve objects to “Released_Last”, because that implies a temporal ordering that we explicitly disallow. Someone suggests “Custodian”; “Distinguished” is separate. “Steward”, “Chosen”, “Lucky”, “Blessed”. Alan suggests “Notified”. That gets some agreement.

So change “Released_Last” to “Notified”.

Approve AI with changes: 10-0-0.

AI05-0176-1/07 Quantified expressions

Tucker suggests that we make **some** a reserved word.

Ed suggests **for others** rather than defining a new reserved word (with all of the incompatibities that implies). Tucker says yuck, we’d be sacrificing readability for expediency. John says that SPARK uses **for some**.

Given the guidance from WG 9, we think we have to make **some** a reserved word (no other choice makes sense). The AI will be changed to reflect that.

[Editor's note: After the meeting, it was noted that the AI doesn't include the proper syntax to include the container iterators (based on the current draft of AI05-0139-2). This was corrected.]

Approve AI with changes: 7-0-3.

AI05-0182-1/01 Preciseness of S’Value

Tucker points out that there is a table for Hex_ that ‘Wide_Image uses (that is how it determines whether to return the character or the Hex_xxxxxxxx name), so it would not be a hardship to require the check.

OTOH, Wide_Character'Wide_Value is pretty useless, so requiring a lot of work is silly. And the intended result is well-defined.

Tucker would prefer that we make this an implementation permission. We ought to allow the easy implementation, but we don’t want to force implementations to change what they currently have as this attribute is nearly useless. We agree with Tucker's suggestion.

Approve intent: 7-0-1.

AI05-0183-1/04 Aspect Specifications

Jean-Pierre asks about where the names are resolved. At the first freezing point or the end of the declaration list?

Steve Baird notes that this causes a form of Beaujolais as simple change could change the resolution of the expression. Adding a declaration that freezes could change the name. Several are dubious.

Steve continues by claiming that you could get a different declaration and still be legal. If you have a use clause and a later explicit declaration with the same name, you could have both expressions be legal but mean different things.

Tucker suggests requiring that the interpretation at the freezing point and the interpretation at the end of the declaration list be identical.

Could we wait until the end of the declaration list? No, because we need things like Size at the freezing point.

Is this case a pathology? Not really, if you have inherited routines for an untagged type that are overridden, the body executed and the default expressions could change. That's awful.

Jean-Pierre asks about visible operations at the point of the aspect clause. These are irrelevant, it is what is visible later that matters..

We need an example of this problem in the discussion of the AI.

Typo: "the aspect_mark identifies an [entity]{aspect} that denotes..."

Erhard would prefer a different organization of the Name Resolution section. There are four sentences starting with "if the aspect_mark..."; these items should be in a separate paragraph; there are two items in the first paragraph and two in the second. Split the first paragraph, separating the items from the rest, possibly use bullets for the items.

The parameters of a subprogram need to be "directly visible".

Steve Baird wonders about a multiple declaration list:

```
X, Y : T with Size => Expr;
```

Expr is evaluated twice, not sure if the wording says that.

Erhard would like to say less about where aspects are evaluated, as this already says that it depends on the particular aspect.

Randy suggests that we say "unless otherwise specified, the expression is evaluated here", because we don't want to have to write this explicitly for ordinary aspects like Size.

Tucker would like to know how to introduce aspects. He suggests a blanket rule for specifiable attributes, and then have separate rules for pragmas and other things.

Jean-Pierre wonders if we want to make it possible to avoid using the existing clauses. Should we allow a enumeration representation clause, for instance. It is the "coding" aspect, an aggregate would work.

For components, each component would be specified individually (First_Pos, Last_Pos, Position).

Currently, the aspect is layout, and it applies to the entire record. We probably won't mess with this; we won't allow specification of components.

Tucker would like to make a separate AI to handle changes to individual aspects. But add a general statement about specifiable attributes in this AI.

Wording needs to be added to 13.1(9-9.1) to include aspect clauses (to prevent specifying one aspect in multiple ways).

Tucker says that he tried to use some common wording in AI-139-2, 4.1.6.

This would then be used to create an annex similar to the attribute one.

We are electing Bob to do this new AI, using the AI05-0139-2 wording as a model. He will need to "fix" the wording from AI05-0145-2 into this format. [Editor's note: I did this for the wording proposed for AI05-0146-1 and in these minutes for AI05-0153-1].

Approve intent of AI: 9-0-1.

AI05-0184-1/03 Compatibility of streaming of containers

We need to add multiway trees here. Probably in the list of changes.

Approve AI with changes: 8-0-0.

AI05-0185-1/01 Wide_Character and Wide_Wide_Character classification and folding

Drop all of the pragma Inline occurrences.

!proposal "This proposal {is} to create..."

"False" should be capitalized in the wording as "True" is, and we better be consistent.

Approve AI with changes: 8-0-0.

AI05-0186-1/05 Global-in and global-out annotations

This is too immature; it's not going to be ready for standardization in time for this Amendment. (This doesn't mean it is a bad idea, just not ready.)

No action: 9-0-1.

AI05-0188-1/05 Case expressions

The new Legality and Dynamic Semantics rules just added by Tucker to AI05-0147-1 need to be shared (not changed to if_expression). [Editor's note: see the comment about the Dynamic Semantics rule in the minutes for AI05-0147-1. It makes more sense to make it part of the existing Dynamic Semantics rule, which is not shared.]

Add a sentence saying that other resolution rules are found in 4.5.7. Also get rid of the AARM To Be Honest.

Get rid of the two legality rules "If the expected_type of the case_expression..."; those will be shared as well.

John wonders if the AIs need to be merged into a single one. We also would merge the entire clauses.

We're not sure. We ought to create an alternative that merges the two and we can then decide which we prefer. Randy will do this.

[Editor's note: After the meeting, I studied this particular idea and concluded it was not worthwhile. About half of the rules are not shared and thus would be confusingly mingled together (all of the Dynamic Semantics are in this category); most of the shared rules are in other clauses and have no effect on this one. I found I needed to make a decision for the AARM draft; it's getting too late in the process to do lots of experiments. So this homework item has been canceled.]

Approve intent: 9-0-0.

AI05-0189-1/02 Restriction No_Allocators_After_Elaboration

This could be implemented as a global flag that is checked at each allocator. Jean-Pierre notes that each storage pool could do this, that might be cheaper. But user storage pools would not have such a check.

Perhaps this would be better called No_Default_Allocators_After_Elaboration, then the default pool could do the check simply, and there would be no requirement for user-defined pools to check.

If we did want user-defined pools to check, we would add a distributed overhead because most compilers directly call the user operations. That wouldn't be possible for allocations in order to enforce this restriction.

Jean-Pierre suggests adding a function to test whether the main subprogram has started. Randy notes that some similar questions (about completion of the main subprogram) can be answered via Task_Identification. Tucker suggests adding a function to determine whether a task has completed its activation. That would also get the start of the main program by asking the question about the environment task.

Tucker wonders whether this should be no allocators after the begin of the current task. The idea is that any task can allocate memory during its activation, but not later.

Tucker also suggests adding an Environment_Task_Id function to Task_Identification; currently, the id has to be saved using Current_Task during the elaboration of some library package. That can be a pain.

The restriction should mean that there will be no allocators from the default storage pool if the current task has completed its activation. User-defined pools could write such a check themselves, using the proposed facilities in Task_Identification. Those facilities also should be part of this AI.

Approve intent: 8-0-0.

AI05-0190-1/03 Global storage pool controls

Tucker worries that this would cause distributed overhead, but that seems wrong. This just changes the pool used by declarations of access types when no pool is explicitly specified.

Steve wonders about instances. Consider a pragma Default_Storage_Pool that encloses a generic unit, and a different pragma Default_Storage_Pool that encloses an instance of that unit. Which one is used? Pragma Suppress has such a model, but we don’t know if it is right. Bob ought to look at this and come up with an answer (preferably without making a mess out of shared generics).

Tucker thinks the wording needs to say that the pool that is in effect at the point of the definition of an access-to-object type is the one used.

Steve worries about confirming Storage_Size clauses changing the behavior of a type. Can Storage_Size be confirming? Tucker says that it could be if it is set to 0, it could be confirming if the default pool is null. But does that change any behavior? It doesn't seem to.

Why is there text about coextensions here? The coextension text is Implementation Advice. We're not interested in changing this to a requirement. Probably Bob is saying that if a default pool is in effect, that this should apply. That needs to be clearer.

Randy points out that the point of the access discriminant definition is not the point that we want to determine the pool for a coextension. So that rule is wrong anyway; some exception from the rules is needed.

Tucker makes the claim that access parameters should also be excepted from the rules about the default pool. Randy says that the point was to get user control over all allocators. Tucker claims that these items are on the stack, and should never be allocated anywhere else. Randy says this is not what Franco and others wanted; they want all allocators to come from a particular pool so that the values can be moved to other types with longer lifetimes. Randy thinks that writing stack allocation as an allocator is misleading to readers anyway – if you want stack allocation, declare an object! Not surprisingly, Tucker strongly disagrees and no consensus is reached.

Keep alive: 6-0-2.

AI05-0191-1/01 Aliasing predicates

Tucker doesn't understand why arbitrary types are needed. Erhard explains that he needs to compare a part of an object to the object as a whole, these are different types.

Jean-Pierre clarifies by noting that you might have a subprogram with two parameters, where one is a record and the other is something else. You may want to know if the "something else" is a part of the record.

Tucker suggests that we could have an attribute that returns an abstract representation of a memory representation. This would work much like 'Address. Then we could have normal function to compare these.

Erhard thinks this is too low level. He doesn't want to talk about memory.

Erhard would like a magic package, which is uniformly considered disgusting.

He then suggests a pair of attributes:

Obj'Is_Same(<any object>)

Obj'Is_Overlapping(<any object>)

These are not functions, the argument can have any type.

Jean-Pierre asks what the answer is for non-Independent objects in the same byte. Perhaps this should be Is_Independent.

Tucker would like to see these as Is_Not_Same and Is_Not_Independent.

Steve wonders what the answer is for zero-sized objects. That should be clear (maybe an AARM note is enough).

On reflection, Tucker worries about Independence. For a sequential program, it might not matter; the semantics don't change. Arguably both attributes (Independent and Overlapping) are needed.

Tucker thinks that Is_Same could be limited to the same type (or a type that membership would return True). He hopes this would simplify the implementation by allowing the comparison of the address of the object (only).

Keep alive: 6-0-3.

AI05-0195-1/01 Uninitialized scalars

There is no consensus about the nature of the problem or whether it needs fixing. After much discussion we agreed on the following:

The answer to the question should be (Yes.), and the fact that people were confused should not require changes in the text. In particular, a compiler could treat these components as not known to be valid, and thus make a check at any use of X.F1 where a valid value is required (such as array indexing). Or it could check earlier if it wants to ensure these are valid, and not have to make a check at usages. What's not allowed is failing to check either at the assignment or uses like array indexing.

No action: 6-0-2.

AI05-0200-1/00 Mismatches in formal package declarations

Ed will take this one.

AI05-0202-1/01 Task_Termination and Exceptions raised during finalization

The notion of a normal termination followed by an exception during finalization being still called "normal" is too weird. Randy says that the alternative is misleading. Tucker prefers this to report "exception"; he argues that it is consistent with the handling of a declare block.

Tucker will take this AI and attempt a consistent definition taking into account failed termination.

AI05-0206-1/01 Remote_Types packages should be able to depend on Preelaborated packages

This idea seems OK; no one can think of any reason that there would be a problem (especially as this is no different than the body of a Remote_Types unit). Brad will attempt to create wording to have this effect.

Change the subject to "Remote_Types packages may depend on Preelaborated packages".

Record is missing in the example in the definition of type W. Also, package X needs to be Preelaborated (so that it meets the requirements of the proposed rule).

The last line of the first paragraph of the discussion has a typo: "...[ans]{and} shouldn't be able to open up new holes."

The paragraph to consider deleting is A(5), not A(4).

Approve intent: 8-0-0.

AI05-0209-1/01 Universal operators of fixed point types (again)

Randy explains that AI05-0020-1 said that we do not need the word incomplete here because untagged incomplete types cannot be parameters; but AI05-0151-1 has made that legal. Thus we do need "incomplete".

Approve AI: 7-0-3.

AI05-0210-1/01 Correct Timing_Events metric

What is "this time"? "During this interval" would be better.

Erhard suggests that the older time be mentioned first.

"That is, the maximum time between the time specified for the event and when the handler is actually invoked assuming no other handler or task is executing during this interval."

Approve AI with changes: 10-0-0.

AI05-0211-1/01 No_Relative_Delay should not allow relative timing events

Steve asks that we say Timing_Events.Set_Handler in this wording.

We don't want the "no calls" wording, because they could be renamed or used as the prefix of an attribute ('Address and 'Access).

"There are no delay_relative_statements{, and there is no use of a name that denotes the Timing_Events.Set_Handler subprogram that has a Time_Span parameter}."

There are other cases that should be fixed likewise: D.7(10/2), D.7(10.7/2), D.7(5). Add those to this AI.

Approve AI with changes: 10-0-0.

AI05-0213-1/01 Formal incomplete types

This doesn’t solve enough problems to include; it is mostly useful for signature generics.

No action: 8-0-2.

AI05-0214-1/01 Default discriminants for limited tagged types

Randy will reword to say that ‘Constrained is True for a prefix of a tagged type, and drop the attempt to specify "constrained by initial value". It's too messy.

Approve intent: 10-0-0.

Jean-Pierre notes a typo at the end of the problem “[such] {should} be eliminated”

AI05-0215-1/00 Errors in AI05-0030-2

Tucker volunteers for this one.

AI05-0216-1/01 No_Task_Hierarchy is still wrong

post-compilatation (fix spelling).

Tucker would like to say “library-level master”.

“No task depends on a master other than the library-level master.”

Approve AI with changes: 8-0-1.

AI05-0217-1/01 Record extensions and “immutably limited”

In 2nd paragraph of discussion, fix spelling of “progenitor”.

Approve AI with change: 10-0-0.

AI05-0218-1/00 Generics and volatility

It was Steve’s question, so he gets it.

AI05-0219-1/01 Pure permissions and limited parameters

The summary says “limited”, it should say “immutably limited”.

The last e-mail message is from Gary Dismukes, not Robert Dewar.

Remove the redundant markers from the last sentence in the wording – it is confusing.

Approve AI with changes: 8-0-1.

AI05-0220-1/00 Definition of “needed component”

Steve will take this one.

Future AI to extend the base on literals to 36

Erhard notes that Germany asks to extend the base on literals to 36. John says that sounds like fun, and he gets the AI to draft.