Minutes of the 42nd ARG Meeting

29-31 October 2010

Fairfax, Virginia, USA

Attendees: Steve Baird, Randy Brukardt, Gary Dismukes (except last half of Sunday), Bob Duff (except first part of Friday morning), Brad Moore, Erhard Ploedereder, Jean-Pierre Rosen, Ed Schonberg, Tucker Taft, Bill Thomas. Alan Burns attended via Skype for the discussion of AI05-0117-1 and AI05-0167-1 on Sunday.

Observers: Greg Gicca.

Meeting Summary

The meeting convened on 29 October 2010 at 9:15 hours and adjourned at 14:05 hours on 31 October 2010. The meeting was held in various rooms at the Hyatt Fair Lakes. As expected, the meeting covered about three quarters of the entire agenda, including all of the ready Amendment AIs.

AI Summary

The following AIs were approved :

AI05-0189-1/04 Restriction No_Standard_Storage_Pools_After_Elaboration (9-0-0)
AI05-0223-1/01 Wide_Maps needs finalization (6-0-3)

The following AIs were approved with editorial changes:

AI05-0051-1/09 Accessibility checks for class-wide types and return from nested-... (5-0-2)
AI05-0117-1/02 Memory barriers and Volatile objects (10-0-1)
AI05-0158-1/05 Generalizing membership tests (9-0-1)
AI05-0159-1/09 Queue containers (10-0-0)
AI05-0177-1/04 Expression functions (9-0-1)
AI05-0183-1/06 Aspect specifications (8-0-2)
AI05-0185-1/03 Wide_Character and Wide_Wide_Character classification and folding (5-0-3)
AI05-0188-1/09 Case expressions (10-0-0)
AI05-0191-1/03 Aliasing predicates (7-0-3)
AI05-0199-1/01 Record aggregates with components of anonymous access types (7-0-0)
AI05-0222-1/01 A completion of a primitive subprogram can occur after freezing (9-0-1)
AI05-0230-1/01 Inheritance of null procedures with preconditions (10-0-0)
AI05-0231-1/01 Issues in Ada.Directories (8-0-2)
AI05-0233-1/00 Questions about Locales (6-0-4)

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

AI05-0111-3/04 Subpools, allocators, and control of finalization (8-0-1)
AI05-0119-1/00 Package Calendar (8-0-2)
AI05-0125-1/02 Nonoverridable operations of an ancestor (9-0-0)
AI05-0139-2/07 Syntactic sugar for accessors, containers, and iterators (9-0-0)
AI05-0153-3/01 Subtype predicates (10-0-0)
AI05-0167-1/04 Managing affinities for programs executing on multiprocessors (9-0-0)
AI05-0176-1/09 Quantified expressions (9-0-1)
AI05-0190-1/05 Global storage pool controls (10-0-0)
AI05-0214-1/02 Default discriminants for limited tagged types (10-0-0)
AI05-0225-1/01 Call using constant protected objects (9-0-1)
AI05-0227-1/01 Identifier equivalence (6-0-3)

AI05-0228-1/01 Default initial values for types (9-0-1)
AI05-0229-1/01 Specifiable aspects (9-0-1)

The following AI was discussed and assigned to an editor:

AI05-0232-1/00 Hole in AI05-0067-1

The following AIs were discussed and voted No Action:

AI05-0111-2/01 Specifying a pool on an allocator (without reachability checks) (9-0-0)
AI05-0135-2/02 “Integrated” nested packages (8-0-1)
AI05-0153-1/07 Subtype predicates (10-0-0)
AI05-0154-1/00 Unconstrained 'Access on array components (7-1-2)
AI05-0161-1/02 Restrictions for default stream attributes of elementary types (8-0-2)
AI05-0226-1/01 Extended digits (7-1-1)

Detailed Minutes

Previous Meeting Minutes

Approve minutes by acclamation.

Date and Venue of the Next Meeting

Our next meeting will be in the Tampa Bay area February 18-20, 2011, with Greg hosting. New York is expensive and cold; we'd rather have expensive and warm.

We will plan on phone meetings in between. The first one will be November 19, 1PM EST. 3 hours maximum duration. Everyone is invited to this phone call.

ASIS

We previously decided to add ASIS clauses in AIs. We'll do that as the last regular clause, that is, we'll add !ASIS clauses between !ACATS Test and !appendix.

Jean-Pierre analyzed the AIs and most don't need work. He would like to discuss several of them however.

Jean-Pierre will make assignments of the AIs for each ARG member to work on. Open AIs ought to be assigned to their authors.

Randy suggests that the ASIS clauses are due after the February meeting, to give priority to the Amendment work. Jean-Pierre notes that new work on AIs ought to include these sections (he then mentions it for virtually every AI that we discuss during the meeting).

On Sunday, we discuss the individual AIs that Jean-Pierre noted required some additional input.

For AI05-0029-1, Jean-Pierre would like to know what is returned for an operation that exists but isn't declared. If this is a set of elements created out of thin air, what is the enclosing element? Probably it is supposed to appear where it would have been declared had it been declared. We also probably want a query Exists_But_Not_Declared. Jean-Pierre would rather have a special declaration kind. He will study that off-line.

For AI05-0071-1, Jean-Pierre wonders where the constructed operations are declared. He thinks that AI95-0158-1 also is not defined in the same way. Tucker suggests that for ASIS purposes, each class-wide type would also have a set of implicit declarations for all of the primitive operations.

For AI05-0015-1, constant and variable are usually different declaration kinds. There is no way to get this. Jean-Pierre says since ASIS 2005 was never standardized, we should add a new kind; the compatibility is not an issue. Moreover, GNAT has the wrong name for the kind, so they will need to change it anyway (so the compatibility issue will occur anyway). The kinds should be named Return_Variable_Specification and Return_Constant_Specification.

Tucker suggests switching the parts of these names; Jean-Pierre says that these are closer to the existing ones.

For AI05-0162-1, allowing completion by a private type. What should Corresponding_Type_Declaration return? It currently returns *the* completion, but now there may be more than one. Tucker suggests that for going back and forth, private ↔ full, and incomplete ↔ full for the current routine.

Then add routines Corresponding_Partial_Declaration, Corresponding_Incomplete_Declaration, Corresponding_Full_Declaration to get the specific views.

For AI05-0179-1, for a label without a statement. Jean-Pierre suggests adding a new statement kind "Implicit_Null_Statement". Tucker wonders if a trait or query on a null_statement would be better. Jean-Pierre says that adding an enumeration will make all of his cases fail and he'll know to update his code. That wouldn't be true for a new query.

Tucker suggests a subtype Null_Statement, with kinds Explicit_Null_Statement and Implicit_Null_Statement in that subtype. Then less code would need to change.

Thanks

Thanks to the host, SIGAda, for the fine accommodations for the meeting.

Thanks to Randy Brukardt for his efforts as note taker and editor.

Thanks to Ed Schonberg for his efforts as our leader.

Steve then announces: In addition to the places that these Thanks normally apply, they also apply in the private part of this meeting. [Editor's note: If you don't get his joke, you need to spend more quality time with your copy of the Standard. :-)]

Old Action Items

Pascal didn't do his task. Randy didn't do AI05-0119-1 (assigned by the Amendment subcommittee). Tucker didn't do AI05-0125-1, AI05-0189-1, AI05-0202-1. Randy didn't update AI05-0158-1 with the wording Steve had sent (it got misplaced).

Tucker will try to address AI05-0125-1 and AI05-0189-1 before the end of the meeting (he did, and those versions were discussed).

New Action Items

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

Steve Baird:

- AI05-0232-1 (with Randy Brukardt)

Randy Brukardt:

- AI05-0111-3
- AI05-0119-1
- AI05-0227-1 (with help from Tucker Taft)
- AI05-0228-1
- AI05-0232-1 (with Steve Baird)

Editorial changes only:

- AI05-0051-1
- AI05-0117-1
- AI05-0158-1

- AI05-0159-1
- AI05-0177-1
- AI05-0183-1
- AI05-0185-1
- AI05-0188-1
- AI05-0191-1
- AI05-0199-1
- AI05-0222-1
- AI05-0223-1
- AI05-0230-1
- AI05-0231-1
- AI05-0233-1

Bob Duff

- AI05-0153-3
- AI05-0167-1
- AI05-0190-1
- AI05-0225-1
- AI05-0229-1 (with a few topics given to Tucker Taft)

Pascal Leroy:

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

Jean-Pierre Rosen:

- Assign Ada 2012 AIs to ARG members to create ASIS changes that support the AI changes.
- Create an !ASIS section for AI05-0188-1.

Ed Schonberg:

- AI05-0176-1

Tucker Taft:

- AI05-0125-1
- AI05-0202-1
- AI05-0214-1
- AI05-0227-1 (help Randy Brukardt as requested)
- AI05-0229-1 (specific topics)

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

AI05-0051-1/09 Accessibility checks for class-wide types and return from nested-extension primitives

Steve explains the wording change by showing an example.

```
type T is tagged ...

type Dt (D : access Q) is new T with ...

type TC_Ptr is access T'Class;

function Foo return T'Class is
   Local : aliased Q;
   V2 : DT (D : Local'access);
   P : TC_Ptr;
begin
    P := new T'Class'(V2); - Has to be caught.
    P := new T'Class'(D => Local'Access, ...); -- Error
    ...
```

Tucker would say that if there are access discriminants then the object accessibility would be used. We could add a flag to the tag in order to determine whether the check is needed. That sounds messy.

Tucker would prefer to assume that all class-wides always have an access discriminant. But that would be very incompatible, it would mean that returning local objects would never be allowed, even when there are no access discriminants in sight.

We decide to revert to the previous version of this wording, and create a new AI [now AI05-0234-1 – Editor] to address this specific problem. (Especially as AI05-0142-4 also modified this paragraph.)

Approve AI with changes: 5-0-2.

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

Tucker has abandoned this version in favor of AI05-0111-3.

No Action: 9-0-0.

AI05-0111-3/02 Subpools, allocators, and control of finalization

The WG 9 meeting finished early, so a subset of the ARG that was present spent an enjoyable couple of hours (OK, maybe not enjoyable) discussing Randy's "super-simple subpools" AI.

Randy started by explaining how it works and why.

Tucker suggests adding “Unchecked” into the name Deallocate_Subpool. No, the user-callable one has that name; the existing one in the pool is called Deallocate, so the name Deallocate_Subpool is right.

“Reset” probably should be “Set”. Most of the similar examples in the standard use Set, not Reset. (For instance, Set_Length in the containers.)

Root_Subpool should be an abstract type.

Tucker suggests declaring a subtype of Subpool_Handle in the examples. This would mean that users of these example pools would not need to **with** the subpool package.

In the example, Deallocate_Subpool should reset the memory location first, then pop the stack.

We discuss the accessibility level rule. It seems unnecessary, because we are not protecting you against subpools going away. Tucker thinks that just requiring an accessibility check on the pool would be enough to prevent the problems that Randy was worried about. That would mean that the pool has to be at the same level as the access type (it can't be more nested as it needs to be visible to the access type). That works because types are not finalized; they continue to exist until the master that created them is left.

Tucker is wondering about class-wide cases. Access-to-class-wide with extensions that are more nested would be a problem. But those can't be allocated anyway (we have such a check on allocators).

Why is the handle an access-to-class-wide? That makes it easier for the pool implementor. There is nothing that the client can do with it, as the designated type does not need to be available to clients, the type is limited, and there is no reason for a constructor to be made visible to the client.

Randy then updates the AI creating version /03.

AI05-0111-3/03 Subpools, allocators, and control of finalization

Steve notes that the precondition on Allocate_from_Subpool does not match the comments. We decide to delete the comment, as the default subpool is used when Allocate is called.

Pool_of_Subpool should say **not null**. Indeed, virtually all of the parameters should have that. Only Deallocate should not have **not null**.

Bob suggests Optional_Subpool_Handle being the existing type. Subpool_Handle would then be a subtype that is not null. Steve shows a preference for that, Tucker says he is neutral. [Editor's note: Didn't do this because it complicates the user model, especially the subtype that we hope pool creators include.]

Randy thinks that this could be used as the basis for safer abstractions. Steve notes that wrapping subpool handles for reference counts makes the objects limited or constrained, which limits what can be done that way.

The Note under “Create_Subpool” ought to talk about storage pools with subpools, not “subpool manager”.

Steve wonders why this isn't a child of Storage_Pools, as this is a specialization of it. Tucker notes that would allow this to see private data of the root implementation, which might be of value. So make this change.

Why is the parameter of Create_Subpool “aliased”? Randy notes that he wanted to return access to subcomponents. But Tucker notes that implementations have to use 'Unchecked_Access to do that in any case, so this doesn't buy anything. So remove the “aliased” and AI05-0142-4 from the Editor's note.

Brad asks if this can be preelaborated? System.Storage_Pools is. So this should be as well.

Bob notes that System.Subpools.Unchecked_Deallocate_Subpool should be Ada.Unchecked_Subpool_Deallocation to be as close as possible to the existing routine. Since this is for the client, it should be in Ada.

Tucker says that this needs to be a verb, the existing one is a generic. So move this to Ada, and leave the name alone.

Brad notes a typo: “be being reused” in the specification of Set_Pool_Of_Subpool.

In the example, Brad suggests using overriding indicators as needed.

There is some confusion in the example, as the client only needs to see the subpool operations. Probably, there ought to be a separate client package that only contains Mark and Release.

Tucker suggests making the pool a private type (not an extension). The full type would be an extension, with the overridings in the private part. Then only the client view would be in the visible part. [Editor's note: Making this not

an extension doesn't work, because it then wouldn't be a visible storage pool and couldn't be used to specify Storage_Pool for an access type.]

Steve thinks that there might be a solution using access discriminants. He wants to avoid writing 'Unchecked_Access. Tucker and Randy worry that such a solution would make it harder to use the result. Randy notes that components of subpool handle types are likely to be used by clients.

Bob notes that the Unchecked_Accesses are living in the implementation of the storage pool. Storage pool implementers are doing low-level unsafe stuff anyway. 'Unchecked_Access is no worse than the 'Address that is an integral part of writing a storage pool.

There is a long discussion about the model of reuse of Subpool objects. Tucker explains that the only time that the Ada implementation cares about the subpool object is when the pool is created and the connection made. What its lifetime is outside of that doesn't matter.

Tucker notes that when you are doing storage management, you have to be really careful. You can't make this safe without making it useless.

Steve notes that you could make your subpool objects at the library level (and reuse them), you then can avoid using 'Unchecked_Access. Tucker notes that 'Address is required to be used in storage pools. How can 'Unchecked_Access be more dangerous?

Tucker would like the allocator to be able to take a derived handle type.

Jean-Pierre notes that an ASIS clause is needed for the new allocator syntax.

Approve intent: 8-0-2.

Randy will update as soon as possible.

AI05-0111-3/04 Subpools, allocators, and control of finalization

On Sunday, Randy presents his latest draft of this AI.

“is a descendant” → “is any descendant”.

There is some junk in the example after the initial type declaration. Delete those two lines.

In the existing example, we may need to change the calls to Mark and Release.

Steve has a nit: “A subpool_handle_name is expected to be of any descendant...”

Erhard notes that the Redundant text in the third last paragraph of the static semantics is not redundant.

Probably 7.6.1 ought to have an indication that there are other kinds of masters.

We wonder about the definition of "explicitly finalized". There needs to be some mechanism to wait for left-over tasks; these can't run forever. Steve notes that masters are nested.

Steve suggests that we delete all mention of masters from here, and just leave the semantics. Then task waiting would be essentially the same as that for the collection (which has to be the same level as the pool object).

Doing that would technically require the objects to be finalized earlier than when the pool object is finalized (as that happens in reverse order). So handle that by adding an implementation permission to finalize objects allocated in a subpool “late”, when the storage pool is finalized. Need to use the words “instead of at” in that wording, or something to that effect.

Tucker has a nit: He doesn't like the second “explicitly finalized” in the last paragraph of Static Semantics. Randy notes that that is a technical term. In any case, it is going to go away with the rest of the master wording.

Steve worries about the task termination semantics, as it seems to break waiting behaviors that have existed for a long time. Randy argues that any access to remaining running tasks is going to be erroneous outside (the pointer is dangling). So a server task can't do anything interesting after the object that contains it is deallocated.

Steve worries about nested tasks. There might be an issue there; Tucker thinks this is the right semantics.

Tucker will work with Randy and Bob and Steve to work out the task termination issues.

Approve intent: 8-0-1.

AI05-0117-1/02 Memory barriers and Volatile objects

Should the implementation note be Implementation Advice?

Erhard would prefer a replacement for C.6(16).

"Any read or update of a volatile object needs to ensure that all tasks (on all processors) see the same order of updates to volatile variables."

Try again:

“All tasks of the program (on all processors) that read or update a volatile variable see the same order of updates to the variable.” replacing C.6(16).

Bob would like to say something in the AARM about memory barriers; Alan agrees.

"AARM Implementation Note: To ensure this, on a multiprocessor, any read or update of a volatile object should involve the use of an appropriate memory barrier."

Alan notes that the plural is important here. The order of access to two or more such variables matters; they need to have the same order of changes when viewed from any task.

“All tasks of the program (on all processors) that read or update volatile variables see the same order of updates to the variables.” replacing C.6(16).

Approve AI with changes: 10-0-1.

AI05-0119-1/00 Package Calendar

Do we effectively require GMT as an implementation? That would affect not only UTC_Offset, but also math and comparison operations.

We're afraid that this will take too much time to answer properly.

No action doesn't work, however. The question needs to be answered in some way (at least in relation to UTC_Offset). Thus, we will say that it is implementation-defined what happens (including to UTC) when the underlying time changes.

That seems to be adequately explained normatively in the existing standard (9.6(23) says that the time base is implementation defined), but an AARM note would be a good thing.

We need wording to clarify the sign of UTC_Offset. We need to make sure that matches what GNAT does.

Make this a Binding Interpretation.

There is no impact on ASIS; add a clause to that effect.

Approve intent: 8-0-2.

AI05-0125-1/02 Nonoverridable operations of an ancestor

The requirement for the **overriding** keyword ensures that there is no silent change of behavior.

Tucker says that there is a problem for task/protected operations. This wording doesn't handle implemented-by operations.

Steve worries about the case where there is an explicit declaration visible as well; it would not be visible there and it has a different slot.

Probably should be illegal to override two different things, as that would reintroduce the inconsistency. Steve shows an example:

```
package A1 is

   type Base is abstract tagged private;

   procedure P1 (X : Base);

private

   type Base is tagged null record;

   procedure P2 (X : Base);

end A1;

package A1_A2 is

   type Child is new Base with private;

   procedure P1 (X : Child);
   procedure P2 (X : Child);

private

   type Child is new Base with null record;

end A1_A2;

with A1_A2;
package A1.A3 is

   type Grandchild is new A1_A2.Child with private;

   procedure P1 (X : Grandchild);

private

   type Grandchild is new A1.A2.Child with null record;

   overriding
   procedure P2 (X : Grandchild);   -- Illegal, overrides two different P2s.

end A1.A3;
```

There is an issue with a private null procedure of an interface. We wonder if that is even legal. We decide that it is. [Editor's note: But the issue itself was not recorded, unfortunately.]

Anyway, there needs to be some wording to handle entries (implemented-by).

Approve intent: 9-0-0.

AI05-0135-2/02 “Integrated” nested packages

This is the only solution for the instantiation problem, but the workarounds are not terrible and a whole new visibility mechanism is a major undertaking.

No Action: 8-0-1.

AI05-0139-2/07 Syntactic sugar for accessors, containers, and iterators

The last sentence of the second paragraph of 4.1.5's static semantics should be in redundant brackets as it adds no new semantics.

Convention Fortran should change the order that elements are visited in an iteration. It's important that the order be defined, as it is possible to exit from the loop and compatibility would suffer if the order could change. But the defined order is really bad for Fortran (it could have very bad caching effects, for instance). Streaming has a similar problem, and should also be fixed in the same way.

The discussion and examples either need to be updated or removed.

Inside of a generic (and instance), the general dereference is the one defined for the formal type (if any). Perhaps have an AARM note or discussion on this topic.

In the 4.1.5 example, change “**with null record**” to “**with private**”.

Steve wonders if we need to use something like “available” instead of “specified” for Variable_Indexing. He wonders what happens if you have Variable_Indexing specified in the private part.

Tucker notes that this is likely to be a problem for other aspects as well. Randy notes that Legality Rules and Static Semantics rules tend to need to depend on views, but aspects are not based on views. That probably belongs in AI05-0183-1.

In 5.5.1, the routines all need to be declared abstract.

Jean-Pierre notes that a !ASIS section needed. He suggests a query to get the discriminant of an implicit dereference; if it Null_Element, it is a “regular” implicit dereference.

Approve intent: 9-0-0.

AI05-0153-1/07 Subtype predicates

We'll abandon this version in favor of the version with static predicates.

No action: 10-0-0.

AI05-0153-3/01 Subtype predicates

Ed would like “arbitrary” to be removed from the summary. Bob says that this means that any boolean expression can be used.

Bob's version does not include any restrictions on the boolean expression. He doesn't think that the restrictions proposed in AI-0153-1 do much because of function calls, which can depend on global values.

We call this the "anything goes" version: all subtypes can have predicates, and the predicates can be any arbitrary expression.

Someone asks to explain where the "hole" in the predicate model appears. The “hole” is where a value of a subtype that has a predicate passes the predicate when it is checked, but it no longer passes the predicate when it is read. This

can happen when non-discriminant components are modified individually or when the predicate depends on global variables (of any kind).

Straw poll: Anything goes: 5
Restrictions: 4
Undecided: 1

Jean-Pierre worries that calling these predicates is worrisome if they can become false after being checked. Users will expect these to work like constraints (and thus they remain true).

Tucker speculates on whether expression functions could help. They can be checked to avoid the use of global variables.

Steve and Bob think that doesn't work; it doesn't work for solving the node problem in GNAT. That is a global variable.

Randy notes that the static predicate does not have the "hole". Bob notes that the static and "normal" predicates are very different semantically.

Tucker reraises Randy's old idea of having two predicates: static_predicate and dynamic_predicate.

Erhard suggests that use of global variables is inherent to the Ada language.

Jean-Pierre wonders if something similar to 'Old would work. You could evaluate these values and save them. Bob says that doesn't make much sense, because the check wouldn't reflect reality: the clock would be stopped (thinking of the awful predicate containing a call to Ada.Calendar.Clock).

Jean-Pierre suggests “volatile_predicate”.

In favor of two different names: 8-1-1. (Gary opposes as he thinks it is too confusing).

Moving on: Should the First and Last attributes be allowed for static predicates? They make no sense for dynamic predicates. The problem is that people might try to use S'First..S'Last. We're allowing case and loops on S, so this isn't necessary (and would produce the wrong results).

So we think we'll leave 'First and 'Last illegal for all predicates.

Approve intent: 10-0-0.

AI05-0154-1/00 Unconstrained 'Access on array components

Erhard would like to solve this problem in the future (not necessarily in Ada 2012).

No action: 7-1-2. Erhard votes against.

AI05-0158-1/05 Generalizing membership tests

We worked from Steve's wording changes as Randy hadn't integrated it into the AI at the time of the meeting.

“shall be determined” should be “is determined”.

Randy wonders what happened to the subtype_mark case. It has been folded into the list.

This requires these all to be the same type. That is OK; all of the elements of the list need to be the same type even in the class-wide cases. The left-hand side of course can be a different type, so the normal cases still work.

Steve says that this original wording is based on the conditional expression wording. Randy suggests that an “otherwise” would help.

“If the type of an element of the membership choice list is elementary, then the tested type of the membership test shall be covered by that type. Otherwise, all of the elements of the membership choice list shall resolve to the same type, and the tested type of the membership test is that type. ”

Tucker would like to improve this wording, he will look at it tonight and tell us if there is anything better tomorrow.

In 4.4(3): “choice_relation” => “choice_expression”.

Tomorrow has come early: Tucker has new wording (in a couple of minutes).

“All elements of a membership choice list shall resolve to the same type, which is the tested type; or each element shall be of an elementary type, and the tested type shall be covered by each of these elementary types.”

In X in 1|2|3 then the tested type is universal integer. X would then have the expected type of universal integer. That seems OK.

Approve AI with changes: 9-0-1.

AI05-0159-1/09 Queue containers

Randy notes that the problem is that “implemented-by” is not a completion, it can only be an overriding. Thus, we can't just hide the protected type in the private part (unless we want to redo how "implemented-by" works).

Randy also studied whether we could allow types in the private part of a protected type. He thinks that would work reasonably well, but since it is so late in the process, and this change might have a substantial impact on compilers, it is probably too late to seriously attempt to relax this limitation.

So he notes that the best solution is the "empty specification" approach, but that eliminates an important property: users will have to understand interfaces to be able to use the packages.

Tucker suggests adding a subpackage “Implementation” before the protected type. That would look like:

```
package Ada.Containers.Unbounded_Synchronized_Queues is

   pragma Preelaborate(Unbounded_Synchronized_Queues);

   package Implementation is
       -- Not specified by the language
   end Implementation;

   protected type Queue (
      Ceiling: System.Any_Priority := Default_Ceiling) is
         new Queue_Interfaces.Queue with
      pragma Priority(Ceiling);
```

This solution is the least messy, so let's go with this.

Add “No impact” ASIS section.

Approve AI with changes: 10-0-0.

AI05-0161-1/02 Restrictions for default stream attributes of elementary types

We look at the static checking problems in the current draft (see the e-mail in the AI). This is not going to be easy to deal with.

Probably this needs to be a dynamic check. Then the restriction needs to be partition-wide; probably it would be implemented by checking a global flag in appropriate places.

Randy notes that would be a distributed overhead. Tucker says that the compiler could recompile everything – including the runtime system – in order to avoid that overhead. That's beyond what is normally done.

This doesn't seem important enough for that solution.

Tucker has wanted to have a run-time way to replace elementary stream attributes. That seems like a better idea, but it is too late to do that properly. (The e-mail on the topic didn't seem to converge to a solution.)

Tucker has a radical idea: how about restricting against use of predefined types? No Boolean or String??

Tucker notes that it is relatively easy to check that all user-defined types have stream attributes specified. So all you need to prevent is usage of predefined types. But that includes types declared in the run-time. It would be impossible to use Text_IO in such an environment, for instance.

No action: 8-0-2.

AI05-0167-1/04 Managing affinities for programs executing on multiprocessors

"not defined by the language" should be “not specified by the language” in the private part of the package definition..

Change "A task will activate on the dispatching domain of its activating task. A task without a Dispatching_Domain pragma will activate and execute on the same dispatching domain as its activating task."

To: "Every task will activate on the dispatching domain of its activating task. A task without a Dispatching_Domain pragma will also execute on this domain."

This seems weird. The reason is that the task doing the activating has to wait. The idea is that there is no dependence between dispatching domains.

Bob wonders why you can only use half of the CPUs for activations (presuming you have split the CPUs into two domains).

Erhard worries that he couldn't activate tasks in parallel. But such tasks would have a barrier at the end, so they should be in a single dispatching domain (with probably all of the available CPUs). In that case there is no issue.

Alan notes that priority inheritance only works in a single domain, so it makes no sense to block something in a different domain.

What can happen across different dispatching domains? Activation, explicit communication, task termination.

This rule does not apply to CPUs; that is, when a CPU is assigned, that does not make any requirements on the CPU of activated tasks.

Alan notes that when he says “executes” in here, he means “execution after activation is completed”.

Erhard wonders why the “as soon as practical” for Assign_Task is open-ended. IRTAW didn't think anything more specific was possible.

Probably adding a documentation requirement here for the immediacy of Assign_Task would be a good idea.

Assign_Task is a task dispatching point for the caller. The same is true for Set_CPU.

Steve notes that the precondition for Assign_Task is that you can only assign a CPU if the task is currently in the System_Dispatching_Domain.

Tucker suggests rewording:

“The exception Dispatching_Domain_Error is propagated if T is already assigned to a Dispatching_Domain other than System_Dispatching_Domain, or if P is not one of the processors of DD (and is not Not_A_Specific_CPU); assigning a task to the System_Dispatching_Domain has no effect.” And drop the last sentence.

“environmental task” should be “environment task”.

What happens if you call Create from other than the environment task?? Alan says: falling over in a heap. We don't think that is a technical description.

Calling Create from other tasks and after the activation should raise Program_Error; wording should make this clear.

Brad notes that the package should be Preelaborated.

Should DD have Preelaborable_Initialization? No, because it can only be explicitly initialized.

Erhard comments that the Documentation Requirement should say “set” rather than “range”, because it is not necessarily a range.

Bob will update this AI.

Brad wonders about the one letter parameter names. Tucker says that T is traditional for one parameter routines, but we should use CPU instead of P. There is an objection that CPU is a subtype, but this is not a real problem. Similarly, we should use Domain instead of DD.

Approve intent: 9-0-0.

AI05-0176-1/09 Quantified expressions

Should we change the syntax to “=>”?? Randy worries about the possible aggregate syntax noted in the last e-mail. Tucker notes that the aggregate has no “all” or “some”, so there isn't a problem.

So we'll use “=>”.

Ed will edit this AI, and add the needed ASIS section.

The order is that specified for the loop specification. “reverse” is allowed. This bends the brains of some, but it's more work to disallow it (it's in the syntax) than allow it, and it seems harmless.

The example doesn't work for an array with zero or one elements. ('Pred could fail for such arrays.) It is sorted in that case, so ('Length < 1) **or else** ... the rest would work.

Approve intent: 9-0-1.

AI05-0177-1/04 Expression functions

Can expression functions appear in a protected body? No, because it isn't allowed syntactically.

Tucker suggests modifying 3.11.1 to drop “an”:

A *body* is a body, an entry_body, {a null_procedure_declaration or expression_function_declaration that completes another declaration}, or a renaming-as-body (see 8.5.4).

Add that null procedures can be a completion into summary.

In 6.7(2/3), delete the wording up to the “if”.

6.1(30/2). “defined” => “declared”, both in the new sentence and in the null_procedure_declaration case.

Gary does not like ending with “not”, but the existing sentences do that as well, so leave it.

6.7(5/2) “expression function” should be “null procedure” in this paragraph.

6.8 ... to declare a function whose body consists of a single return statement.

Erhard does not like the prefix “parameterized_”. Tucker thinks that we don't need the prefix, we should just drop it.

First legality rule, drop everything up to “if”.

Add “of the expression function” after the end of first paragraph of Dynamic Semantics. “that returns” => “whose return expression is”. But there is no such term as "return expression".

...that of a function body containing only a simple_return_statement whose expression is that of the expression function.

13.14(3/3) would include expression functions that are completions. We don't want that. Probably, we ought to be using syntactic “body” here, and then we don't need the renames-as-body exception either, so drop that as well. Do we need to include entry_body? No, because there are no declarations around an entry_body.

Taking 'access of a one-part expression function yields a pointer that could then be called before the expression is frozen. So 'access of an expression function causes freezing. Add wording to that effect.

That doesn't solve it for a completion case – the elaboration check would succeed. So if it is called within the window after the completion but before the expression is frozen, bad things will happen.

[Editor's note: I couldn't figure out what this meant exactly, so I created an example:

```
package Pack is

   function Foo (A : in Natural) return Natural;

   type Bar is access function Foo (A : in Natural) return Natural;

   P : Bar := Foo'Access; -- Freezes Bar and Foo.

   type Flub is range 0 .. 100;

   function Foo (A : in Natural) return Natural is
      (A + Flub'Size); -- The expression is not frozen here.

   Val : Natural := P.all(5); -- (!)

end Pack;
```

The call to P.all(5) passes the elaboration check, but still references an unfrozen type Flub, as the expression itself is not frozen. We didn't have this example at the meeting, but it illustrates the problem well.]

So when these are used as a completion, they must freeze. Therefore, leave 13.14(3/3) alone.

There needs to be an ASIS section.

Approve AI with changes: 9-0-1.

AI05-0183-1/06 Aspect specifications

Add “or aspect_specification” to the 13.1(9.1/1) wording (in both places, not just one).

Freezing question: do we have something that freezes these expressions at the right place. If they are frozen as part of the declaration, that is too early – they aren't even resolved yet. And there is no wording to do it later.

So there needs to be wording to do that. Something like the default expressions case so they are frozen at occurrence, and a rule to freeze them at the freezing point of the entity.

“At the freezing point of the entity associated with an aspect_specification, any expressions or names within the aspect_specification cause freezing.”

We talk about the new legality rule. It should be "same declaration" rather than "same entity", so as to match conformance of expressions.

Randy and Tucker will review the changes. Tucker will write the ASIS clause. There should be a query that returns all aspects of an entity.

Approve AI with changes: 8-0-2.

AI05-0185-1/03 Wide_Character and Wide_Wide_Character classification and folding

Randy notes that we need to explain "simple upper case mapping" better. This is defined in the Unicode standard. Tucker would prefer to directly refer to Unicode here.

It's not clear that we can directly reference Unicode without causing problems. (We've heard stories of standards being rejected for that reason; Unicode is not a standard to ISO.) So reuse the wording from 1.1.4(14.2/2).

“as defined by documents referenced in the note in section 1 of ISO/IEC 10646:2003”.

A.4.11(8/3) says 106046, should 10646.

The lower bound should be 1 for To_Lower and To_Upper, to be consistent with Ada.Characters.Handling.

Make the name changes that Randy proposed.

Randy asks whether we ought to change A.3.1(7/2). Tucker suggests dropping “case mapping” and “character classification” from this paragraph (since we are providing some of those). Bob thinks this was a political note and no longer is needed. But it seems harmless and we don't provide collating operations.

Add the routines to Ada.Characters.Handling that aren't there, and define them in terms of specific characters and ranges.

Approve AI with changes: 5-0-3.

AI05-0188-1/08 Case expressions

In problem, “contruct” should be “construct”.

Accessibility: Tucker would prefer to distribute the static checks to the separate dependent expressions. Randy and Steve say that this would be hard.

6.5(5.4/3) needs distribution, both of the types and the accessibility. So we need a meta rule because doing that everywhere would be a mess.

Case expression prefix should be “selecting_”. Do update 5.4 with that, it is less confusing.

“the other kind” as opposed to “an other kind” for the general note on conditional.

The title of 4.5.7 probably ought to be "If expressions and case expressions"

Erhard really wants these two sections merged. We also discuss other possibilities.

Erhard will try to draft a version that merges the sections during lunch.

AI05-0188-1/09 Case expressions

After lunch, we look at Erhard's proposed merger.

Erhard simplified the rules for case expressions, depending mostly on the case statement rules. This is an unfortunate forward reference, but we're not going to move the entirety of 5.4.

Improve the last sentence of the first paragraph of Dynamic Semantics. “Otherwise (when there is no else clause),”

We agree that this combined version is better. The clause title should be “Conditional expressions”.

We're still waiting for Steve's wording. We'll look at that later.

On Saturday afternoon, we look at Steve's new wording. “conditional accessibility” should be “distributed accessibility”. That still isn't a great term. Tucker suggests “Multiple accessibility”. Erhard suggests “Multiple levels of accessibility”. Steve suggests “Multiple sources of accessibility”. We'll go with “distributed accessibility”.

Drop the first half of the wording, up to the “in other words”. Change “also imposed” to “instead”. Perhaps a version of that first part ought to be an AARM note.

Bob says that Steve ought to draft AARM wording that confirms that there is no problem with view conversions having their type ignored. This can only happen for tagged types because a conditional expression is a constant and a view conversion of an untagged type can only occur as in **in out** parameter (i.e. a variable).

We'll use Erhard's reorganization of the wording.

Jean-Pierre will do the ASIS section for this AI.

Approve AI with changes: 10-0-0.

AI05-0189-1/03 Restriction No_Default_Storage_Pools_After_Elaboration

Steve wonders if default storage pool is well-defined. Bob notes that pragma Default_Storage_Pool changes the default pool, and we don't want to deal with that here.

We look in 13.11, and determine that the term is “a standard storage pool”.

So change “default” to “standard” everywhere in here.

The text is inconsistent as to the name of the function Activation_Is_Complete(d). We'll drop the 'd': Activation_Is_Complete.

The restriction should have separate paragraphs for the dynamic check and the post-compilation checks.

“For the purposes of this rule, an access type derived from a formal access type does not use a standard storage pool.”

Put the post-compilation check first. Start the second paragraph with “Furthermore, “.

Tucker will send us this version once he has finished wordsmithing it.

AI05-0189-1/04 Restriction No_Standard_Storage_Pools_After_Elaboration

ASIS impact: There is none (restrictions aren't special to ASIS).

Approve AI: 9-0-0.

AI05-0190-1/05 Global storage pool controls

Tucker explains the latest compromise. The “however” would be deleted from the null case, and an Implementation Permission would be added that the stack can be used rather than the specified pool for anonymous allocators.

If you say no non-specified pool, that's really what you want. OTOH, forcing the use of a pool that requires a storage leak makes no sense, thus we make an exception when a pool is specified. If you don't want to use the stack, you can use the No_Anonymous_Allocators restriction.

Erhard wonders why derived access types are not mentioned. That's because they always inherit the pool from their parent; that is not modified here. Perhaps some (redundant) wording should be added to explain that.

Most of the second paragraph of the Legality Rules is really Static Semantics.

Only the first sentence should be here. Randy suggests moving the Legality Rules after Static Semantics.

Bob doesn't like that.

Tucker suggests moving the last sentence of this text and merging it with the first sentence of the Static Semantics. Bob agrees that this is an improvement, we'll make this change.

Steve thinks that if there are two pragmas in a row in the same scope, we want the second one to apply. He doesn't think that happens with the existing wording.

“another pragma” => “a later pragma” (4th sentence of 2nd paragraph of legality rules). Another is unordered, we need an ordering here. Bob doesn't believe this. Everyone else believes this, so we'll make the change.

ASIS clause: this needs an enumeration literal to be added to the type pragma_kinds for the new pragma.

Approve intent: 10-0-0.

AI05-0191-1/02 Aliasing predicates

The note and change involving external_tag should be dropped.

The new wording goes after 13.3(73).

“must” => “shall”.

Is_Same means that{, if the representation is contiguous,} the objects sit at the same address and occupy the same length of memory.

“memory space” is not well defined. Use “bits” instead.

Its value is True if the representation of the object denoted by A occupies exactly the same bits as the representation of the object denoted by X; it is False otherwise.

Its value is True if the representation of the object denoted by A shares at least one bit with representation of the object denoted by X; it is False otherwise.

Gary complains about this description. The syntax for attributes has “static_expression”. We either have to change that or we have to make this a magic function.

This doesn't seem like a function, as the parameter passing would be wrong (we don't want to evaluate the value). It's nasty.

Changing the syntax doesn't seem very helpful.

Tucker suggests that maybe we want a type Extent that describes the memory use of the representation of an object. There would be an attribute that would create that. And then primitive operations to compare to Extents (Is_Same, Is_Overlapped).

Such a type would be limited.

Erhard notes that he would hope that this could be calculated at compile-time (practically).

We'd introduce a new package System.Representations. Introducing a new package is messy.

Extent would be hard to represent (it would be a list of ranges in general), and would get messy if it was passed as a parameter or renamed.

Perhaps it is OK to say that these are a function. The function would have an italicized type “any type”. We would have to define the evaluation of the parameter in some special way.

Hand-waving doesn't seem that bad.

So use the function version with an unusual parameter type and then evaluated. The parameter could be aliased in italics, as none of the legality rules on aliased should be enforced, but the by-reference passing would be right. It's an evaluation of a name.

Gary wonders why changing the syntax is considered so bad. Tucker says that it would have an impact on name resolution. No one understands that; the type is Boolean here, and the name would be resolved without context.

Bob will write an AARM note disclaiming any lack of sanity implied by this definition, explaining that there is no good way to describe this.

'Val is probably the best existing example of such a magic function (but a lot of this is new).

ASIS needs to add two literals to Attribute_Kinds.

Approve intent: 8-0-2.

Later in the meeting, Erhard sent a new AI05-0191-1.

Various typos are found and fixed; Erhard resends the AI with these fixes.

AI05-0191-1/03 Aliasing predicates

Gary complains that the evaluation of the name is not explained.

“The actual parameter shall be a name that denotes an object.”

“Evaluation of attribute evaluates the name of the object provided as the actual parameter.” Ugh.

“This function evaluates the names of the object, and returns true...”

Do something similar with Overlaps.

Node => Note.

Approve AI with changes: 7-0-3.

AI05-0199-1/01 Record aggregates with components of anonymous access types

We want an easy AI to finish the meeting. [Editor's note: This was the last AI that we worked on, on Sunday afternoon.]

“{shall} statically match”.

“, or all of them shall be of anonymous access types that statically match”

“, or all of them shall be of anonymous access types and their subtypes statically match”

Bob notes that this isn't quite the same as the existing rule, as it doesn't talk about subtype.

Tucker claims that this rule is only about getting the same name resolution. So it is closest to type conformance. That would be “corresponding designated types are the same, or corresponding designated profiles are type conformant.”

“, or all of them shall be of anonymous access types and whose designated types are the same, or whose designated profiles are type conformant.”

So we've spent 25 minutes on an “easy” AI.

Approve AI with changes: 7-0-0.

AI05-0214-1/02 Default discriminants for limited tagged types

Add “define” to the last paragraph of the !discussion.

Tucker wonders about renaming of components (not allowed because it is not constrained by its initial value); that is covered by "known to be constrained".

Gary and Steve note that this has new implementation cases for tagged types, but that they are not different than what happens for untagged types.

Tucker suggests that 'Constrained be defined to be True if the object is known-to-be-constrained. Randy worries that there might be an inconsistency introduced that way.

Tucker doesn't think that can happen after much thought.

So change to “or is known to be constrained”.

Gary explains a problem he has. Consider the case of a limited private type with visible discriminants with defaults (not visibly tagged). In this case, the 'Constrained value is not defined to be True; it would allow changes in the value if the full type is untagged but not if the type is tagged. This is not a semantics problem, as 'Constrained is dynamic semantics (so looking through privacy is OK).

Gary worries that this would mean that an object that looks unconstrained would have 'Constrained = True.

Tucker is dubious that compilers actually do this right (given that limited objects cannot have their discriminants changed after initialization). He would prefer to add “objects of immutably limited types are constrained” to 3.3.1(9). He claims we don't care where the constraint comes from. Then no change is needed in 3.7(2).

Steve notes that could be inconsistent; 'Constrained would now return True in some cases where it currently would return False. These aren't useful cases, but it would be a change.

Tucker would like to see what GNAT does in this case. Gary creates an example and runs it. It apparently returns True, so it does this wrong.

It still has to be documented as an inconsistency, but as the attribute is not useful and GNAT and AdaMagic do it “right” already, it isn't a huge problem.

Still we need to look at how constrained object is used in the RM before we make a change. So we need to look at them (Bob says there are 592 occurrences in the AARM). Tucker will do this, and propose wording.

Randy recalls that the problem with *constrained by its initial value* was that it triggered various private type rules that we probably don't want involved. Excluding those rules made the change messy.

ASIS no impact, but user may be affected as they need to check for defaults on tagged types.

Approve intent: 10-0-0.

[Editor's note: After the meeting, it was determined that the test program that was run was incorrect. Following is the correct test program:

```
with Ada.Text_IO; use Ada.Text_IO;
procedure AI05_214_Test is

   package Pkg is
      type Ltd_Priv (B : Boolean := True) is limited private;
      procedure Proc (L : in out Ltd_Priv);
   private
      type Ltd_Priv (B : Boolean := True) is limited null record;
   end Pkg;

   package body Pkg is

      procedure Proc (L : in out Ltd_Priv) is
      begin
         if not L'Constrained then
            Put_Line ("Proc.L'Constrained = False");
         else
            Put_Line ("Proc.L'Constrained = True");
         end if;
      end Proc;

   end Pkg;

   use Pkg;

   procedure Outside_Proc (L : in out Ltd_Priv) is
   begin
      if not L'Constrained then
         Put_Line ("Outside_Proc.L'Constrained = False");
      else
         Put_Line ("Outside_Proc.L'Constrained = True");
      end if;
   end Outside_Proc;

   LP_Obj : Ltd_Priv;

begin

   Proc (LP_Obj);
   Outside_Proc (LP_Obj);

end AI05_214_Test;
```

The results defined by the language would be:

```
Proc.L'Constrained = False
Outside_Proc.L'Constrained = False
```

GNAT and Janus/Ada both get these results for this program.

Thus, the reasoning about the inconsistency above is incorrect.

Inconsistencies are the most dangerous type of language difference. This one is not justified, as we either can revert to the original rule (and let Gary's oddity be just that) or we could simply make the 'Constrained attribute illegal for objects of a limited type (using an assume-the-worst rule for generic bodies). Since the attribute is not useful for a limited type, that in theory would not impact actual programs, it probably would eliminate some overhead from programs (the overhead needed to get the "right" answer for the above program), it would not expose existing Ada programs to the danger of a change of behavior, and it would require little language wording (just adding the word "non-limited" to 3.7.2(2)).

In the editor's opinion, either of the above would be preferable to an inconsistency and changing lots of language wording.

End Editor's Note.]

AI05-0222-1/01 A completion of a primitive subprogram can occur after freezing

Add a To Be Honest note: and make this is a Ramification. Make no wording changes.

Approve AI with changes: 9-0-1.

AI05-0223-1/01 Wide_Maps needs finalization

Add an ASIS section, no impact.

Approve AI: 6-0-3.

AI05-0225-1/01 Call using constant protected objects

Steve worries that this wording does not work, because this does not cover the implicit local calls. Tucker notes that you could take 'access of a protected procedure in a protected function. Assume a local access-to-protected-procedure. Then you could have this same bug.

So we need to extend the current rule to cover this, as opposed to changing name.

However, the current wording “target object of a call” does not work, because it only works for a call.

Tucker would rather change “target object of a call” to “target object of a name”. Bob does not think this is a target in cases like 'Access.

Steve suggests “For any name denoting a protected entry and protected procedure, the associated protected object shall be a variable.”

AARM Note: The point is to prevent any calls to such a name, directly, or via an access value, renames, or generic formal subprogram.

There is a complaint that this wording would prevent using P'Count within a function.

Bob suggests listing the 4 uses that we need to prevent: a call, prefix of Access, renames, or actual of a generic formal subprogram.

Bob will write this wording.

Approve intent: 9-0-1.

AI05-0226-1/01 Extended digits

Considered confusing (see e-mail), and not important enough.

No Action: 7-1-1.

Erhard votes against out of respect for the requester.

AI05-0227-1/01 Identifier equivalence

Randy explains that he read the Unicode documentation; they have extensive discussions of this issue directly in the Unicode 5.0 standard. The most important point is that case-insensitive comparisons of strings and case mapping of strings are different operations. Comparison is guaranteed to be stable – that is, it will work the same in all future versions of Unicode. OTOH, case mapping is not stable and might change in future versions.

In addition, all of the compatibility problems with Ada 95 that were discussed occurred because of confusion between "full upper case mapping" and "full case folding". Full case folding converts to a lower case version, so none of the examples with Latin1 characters can occur.

Unicode recommends full case folding for string comparisons. The problem is that it is possible (even probable) that multiple distinct identifiers would have the same upper case mapping. That's trouble for 'Image and 'Value.

Randy doesn't have a recommended solution, as he couldn't figure out how to reconcile the 'Image values (which we can't change for compatibility reasons) with the case folding.

Tucker thinks that we ought to follow the Unicode recommendations as much as possible. This requires dealing with the 'Image problem.

Tucker suggests that we make enumeration types illegal if there are two identifiers that have the same upper case mapping. That would eliminate any conflict in 'Image.

Otherwise, simple upper case mapping seems correct where that is needed, as that is what we have been doing in the past. We believe that this would not make any inconsistency or incompatibility with Ada 95.

We'll continue to use full case folding for identifier equivalence. We'll need to redo the wording to reflect that, as it surely has nothing to do with "convert to upper case"! We definitely do not want to be inventing our own case equivalence rules – we want to use the ones created by other experts.

Tucker notes that he would like to generalize his previous rule, saying that two identifiers are illegal immediately within the same declarative region if they have the same upper case mapping. He's concerned about External_Tag strings, as well as conflicts between debugging images of identifiers. Randy worries that this would be very expensive to implement (requiring a search through an entire scope).

Randy will write this up with help from Tucker.

Approve intent: 6-0-3.

AI05-0228-1/01 Default initial values for types

The expression should be evaluated when used (like other default expressions). The expression would be checked in the normal way.

If we allow this on subtypes, it would have to affect static matching. That would be OK in that predicates work the same way.

If it is subtype related, there would be a lot of holes, because there are too many anonymous subtypes (which would not be initialized). So we will make this type-related.

Tucker wonders if this expression allows **others** within the expression. That would seem necessary for unconstrained array types. But what is the applicable index constraint? It would seem to vary depending upon the constraints of the object. That's unpleasant.

It would be more unpleasant if the value had a constraint that did not necessarily match the constraint of the object. Then the legality checks might fail.

Tucker wonders if we should limit this to scalar types. Steve notes that it is weird to allow this for record types; they already have a way to get default initialized. It's also weird to allow this for access types; initializing to an allocator or 'Access seems weird, and **null** is already the default anyway.

Jean-Pierre suggests Default_Component_Value for arrays (of scalar). That would avoid needing to use an aggregate and eliminate the uncomfortable questions about the applicable index constraint.

So we agree to have two aspects and restrict them to scalar types.

Tucker wonders if we should restrict this to a static expression. Steve notes that that would be inconsistent with records.

Randy notes that the argument that this is easy to implement is that it would use the same mechanism as null access values. That's fine if the expression is static (and scalar); if it is dynamic, we would need to evaluate functions (and handle finalization, tasks, etc.).

Bob objects; he would like to use a function that would always raise an exception.

Jean-Pierre notes that there is still a range check of the value, even if it is static. So that could be used in place of Bob's function in any case where T'Base has a value out of the range of T.

Straw poll:

Static expression only: 8
Arbitrary expressions evaluated once: 0
Arbitrary expressions evaluated once per object: 2
Abstain: 1

This is pretty clear; we'll require the expression to be static.

ASIS will need two new aspects (however that will be done).

Steve will check if there are other places that would cause problems. There seems to be an issue with 13.13.2(35); these should be checked even there. That will need new wording.

We need 6.4.1(12 and later) to be triggered. “Implicit initial values” covers it, and that wording was updated. Probably we should use that term for streaming.

6.4.1(12) needs a bullet for scalar types (or combining with the access types).

Returning to the streaming case. Randy wonders if the intent was to exclude access checks for the streaming check. But that seems bogus: streaming null for a not null subtype is bad and should be caught. So redo the streams wording to use “implicit initial values”.

Tucker and Steve can't find any other problems.

Approve intent: 9-0-1.

AI05-0229-1/01 Specifiable aspects

The group feels that it is important to be able to specify as much as possible via aspect clauses. Bob will look at every pragma and make a decision on them. (Convention, Priority, CPU are especially important).

Convention would have a problem with a type. We could add an enumeration, but then users would have to **with** and **use** System (and there would be name pollution). B.3(11.b).

Perhaps we want to add wording that would allow identifiers as in pragmas. There is controversy about how hard that would be to add wording to aspect clauses.

Tucker will investigate the difficulty of adding pragma-specific identifiers to aspect clauses.

Bob will figure out the wording needed to allow Priority and CPU as aspects.

Tucker will also look at ways to include Import and Export.

Turning off Atomic on a derived type: If you have a generic that takes formal that has an ancestor that is atomic, it could be passed a non-atomic type. Which would be bad if the generic body assumed the type was atomic (say two tasks are beating on it).

So we probably want a restriction to prevent turning off any of the Atomic/Volatile/Independent and similar aspects on a derived type if the parent has it on.

We have a lot of properties that are “class” properties that are always inherited by derived types. A generic can always assume that at least the properties of the ancestor type apply. We don't want that to change.

So we'll adopt a rule that turning these off for derived types is not allowed unless otherwise specified. That probably belongs in AI05-0183-1.

Approve intent: 9-0-1.

AI05-0230-1/01 Inheritance of null procedures with preconditions

Proposal should say (See wording.)

The wording should say something about Pre'Class and Post'Class being allowed. Actually, that should be mentioned in the summary (since it can't be empty).

Steve asks whether a better term than “ ineffective procedure”. Tucker says that this definition is too hard.

We discuss the possible uses of Pre and Post on null procedures. It seems that these are mostly hacks. Just make them illegal on all null procedures.

So we just add “or null procedure” to the first paragraph. Drop everything else.

ASIS clause says no ASIS effect.

Approve AI with changes: 10-0-0.

AI05-0231-1/01 Issues in Ada.Directories

Typo: “avialable” in 3rd line of question 2.

In A.16(125/2), “Name_error” should be “Name_Error”.

Add ASIS section of no impact.

Approve AI with changes: 8-0-2.

AI05-0232-1/00 Hole in AI05-0067-1

Steve says that the language might be OK, but implementors will want to chose a build-in-place calling convention.

Build-in-place is dynamic semantics, and it depends solely on a particular execution of a return statement returning a particular object. So there is no problem here. An implementation can chose a conservative strategy of always building-in-place for this call.

Steve and Randy will redraft this as a confirmation. The answer is No. (No fix needed.)

AI05-0233-1/00 Questions about Locales

(1) it is implementation-defined. Brad will write an AARM implementation note suggesting that it is associated with the current user, unless it's running on a server with no user.

[Editor's note: This really should say "unspecified", since we're not adding any language requiring documentation.]

(2) Brad says that this is preelaborated so that it can be Remote_Types.

Tucker doesn't think that there is any requirement to get the same answer in different partitions. So that isn't a real problem.

Steve notes A.19(8/3) says “the active partition”, which doesn't make sense because there may be more than one. Bob suggests adding “current” here. Tucker doesn't think we ever say that elsewhere. He would rather say “partition of the current task”.

Approve AI with changes: 6-0-4.