Minutes of the 35th ARG Meeting

20-22 June 2008

Venice, Italy

Attendees: Steve Baird, John Barnes, Randy Brukardt, Brad Moore, Jean-Pierre Rosen, Ed Schonberg, Tucker Taft, Tullio Vardanega.

Observers: Steve Michell (Friday afternoon), Greg Gicca (Friday afternoon, Saturday morning), Matt Heaney (Saturday morning).

Meeting Summary

The meeting convened on 20 June 2008 at 14:15 hours and adjourned at 11:20 hours on 22 June 2008. The meeting was held in a conference room at the Centro Culturale Don Orione in Venice Italy. The meeting covered the entire agenda.

AI Summary

The following AIs were approved:

AI05-0053-1/06 Aliased views of unaliased objects (8-0-0)
AI05-0091-1/01 An other_format character should not be allowed in an identifier (8-0-0)
AI05-0093-1/02 Additional rules that need to use "immutably limited" (7-0-1)
AI05-0104-1/01 Null exclusions are not allowed in uninitialized allocators (8-0-0)

The following AIs were approved with editorial changes:

AI05-0006-1/05 Nominal subtypes for all names (8-0-0)
AI05-0009-1/05 Confirming rep. clauses and independence (8-0-0)
AI05-0052-1/09 Coextensions and distributed overhead (8-0-0)
AI05-0063-1/04 Access discriminants on derived formal types (8-0-0)
AI05-0071-1/02 Class-wide operations for formal subprograms (6-0-2)
AI05-0083-1/02 Representation values of formal parameters (7-0-1)
AI05-0090-1/03 Ambiguities with prefixed views of synchronized primitives (7-0-1)
AI05-0094-1/01 Timing_Events should not require deadlock (8-0-0)
AI05-0096-1/02 Deriving from limited formal types (7-0-1)
AI05-0097-1/01 3.9.3(4) should include abstract null extensions (8-0-0)
AI05-0098-1/01 Incomplete type names can be used in anonymous access-to-subprogram types (8-0-0)
AI05-0105-1/01 Resolution of renames uses anonymousness (7-0-1)

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

AI05-0067-1/05 Build-in-place objects (7-0-1)
AI05-0095-1/01 Address of intrinsic subprograms (7-0-1)
AI05-0099-1/01 The tag, not the type, of an object determines if it is controlled (7-0-1)
AI05-0100-1/01 Placement of pragmas (8-0-0)
AI05-0102-1/01 Some implicit conversions ought to be illegal (8-0-0)

The following AIs were discussed and assigned to an editor:

AI05-0001-1/01 Bounded containers and other container issues
AI05-0050-1/03 Return permissions are not enough for build-in-place
AI05-0101-1/01 Remote functions must support external streaming
AI05-0103-1/01 Return statements should require static matching for all named access types

The following SIs were approved with editorial changes:

SI99-0030-1/02 Add definition section to the standard (8-0-0)
SI99-0035-1/02 Undefined capabilities of the Data_Decomposition package (8-0-0)
SI99-0036-1/01 Expressions from normalized expressions (6-0-1)

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

SI99-0024-1/09 Provide a semantic model in addition to existing syntactic model (8-0-0)

The following SIs were discussed and assigned to an editor for further work:

SI99-0007-1/02 Add support for new object oriented prefix notation
SI99-0019-1/01 Add query to check if a name is an implicit dereference
SI99-0021-1/01 Obtain representation clauses based on defining identifiers, not declarations

Detailed Minutes

Meeting Minutes

The minutes were approved unanimously with changes John sent to Randy.

Date and Venue of the Next Meeting

As approved last time, SIGAda in Portland OR, October 31-November 2, 2008. WG 9 is on the afternoon of the 30th. Mike Feldman suggested that we make any vacation plans before the meeting, as the weather can be rainy in Oregon in November.

Next Ada-Europe is in Brest, France; June 12-14, 2009 would be our meeting dates. No one yet has a conflict with these dates.

Since we've promised ASIS to WG 9 by Ada-Europe next year, we'll want a February meeting to finish off the work. Steve Michell suggesting having it in Tallahasse, Florida, February 20-22, 2009. (An alternative would be February 15-17, 2009.) This depends on Ted Baker being able to host.

Thanks

Unanimously, we thank Ada-Europe for the fine accommodations.

PRG report

Steve Michell (the rapporteur for the PRG) gives us an update on the PRG's activities.

The PRG would like to try to pass things off to Ada when that makes sense. That is, they believe that some capabilities of POSIX ought to be present in the Ada standard and thus available to all Ada users. The issue that they discussed specifically was Localization. For instance, there is no localization of Wide_Character (To_Upper, etc.) in Ada. Randy points out that there is no Ada.Wide_Character.Handling; we decided not do that in Ada 95 and Ada 2005 followed that. That means that there is no To_Upper for Wide_Characters. Brad will submit an AI on this topic.

They also would like to look at Time/Date formatting. Randy points out that we explicitly decided not to attempt a date/time formatting solution, as everyone (and every system) has a different idea of what is required.

Old Action Items

Old action items were completed except as noted below. Randy has not done the massive extension to Ada.Directories (AI05-0049-1). Bob Duff didn't do the ASIS example. Greg is almost finished looking for suspect wording (SI99-0037-1); he'll distribute the full version of that. Greg also didn't do the last minute ASIS Standard tasks (of course, it's not the last minute yet for ASIS.) Tucker forgot to merge AI05-0075-1 back into AI05-0051-1,

and did not work on the Amendment AI (AI05-0074-2). He also did not work on SI99-0007-1, SI99-0019-1, and SI99-0021-1 (those may be handled by the semantic subsystem).

New Action Items

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

Steve Baird:

- AI05-0103-1
- SI99-0024-1 (with Tucker Taft and Randy Brukardt)
- Create an AI to define names for the aspects of representation implied by the C.6 pragmas and to determine inheritance rules for them (see discussion of AI05-0009-1).

Randy Brukardt:

- AI05-0049-1
- AI05-0067-1
- AI05-0095-1
- AI05-0099-1
- AI05-0102-1
- SI99-0024-1 (with Tucker Taft and Steve Baird)
- Create an AI to prevent representation pragmas from being given in the formal part of a generic unit (see discussion of AI05-0009-1) [assigned AI05-0106-1 after the meeting]
- Create an AI to eliminate leaks when an allocator fails (see discussion on AI05-0050-1) [assigned AI05-0107-1 after the meeting]

Editorial changes only:

- AI05-0006-1
- AI05-0009-1
- AI05-0052-1
- AI05-0063-1
- AI05-0071-1
- AI05-0083-1
- AI05-0090-1
- AI05-0094-1
- AI05-0096-1
- AI05-0097-1
- AI05-0098-1
- AI05-0105-1
- SI99-0030-1
- SI99-0035-1
- SI99-0036-1

Bob Duff

- AI05-0050-1
- Create an ASIS example tool to find return-by-reference incompatibilities (preferably both using and avoiding the new semantic subsystem).

Greg Gicca:

- Add newly ARG approved SIs to the draft ASIS standard.
- SI99-0037-1
- Renumber clauses to eliminate so-called hanging paragraphs (at end of standard creation process, not now; see ASIS standard format in the Paris minutes)

Matthew Heaney:

- AI05-0001-1 – organize containers meeting for September 22-23, 2008

Brad Moore:

- AI05-0101-1
- Create AI(s) for topics that the PRG would like the ARG to do their work for them.

Jean-Pierre Rosen:

- SI99-0007-1
- SI99-0021-1
- Convert an existing ASIS example tool to use the semantic subsystem.

Tucker Taft:

- AI05-0051-1
- AI05-0074-2
- AI05-0075-1 (merge into AI05-0051-1)
- AI05-0100-1
- SI99-0019-1
- SI99-0024-1 (with Steve Baird and Randy Brukardt)
- Decide what ASIS should do if characters appear in the source that do not fit into Wide_Character and create an appropriate SI (see discussion of SI99-0024-1).

Detailed Review

The minutes for the detailed review of AIs and SIs are divided into ASIS Issues (SIs), Ada 2005 non-amendment AIs, and Ada 2005 amendment AIs. 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 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 ASIS Issues

SI99-0007-1/02 Add support for new object oriented prefix notation

Jean-Pierre wonders whether you get the object or not when in the syntactic view - which parameter list do you get? Ed notes that something must happen for protected subprogram calls, so this support must exist already (and the rules determined). But someone should check that the wording is adequate. Jean-Pierre is volunteered to investigate this. He also should fix the question to say that new capabilities aren't needed (surely we want to be able to know whether something is a prefix call!).

SI99-0019-1/01 Add query to check if a name is an implicit dereference

Our last discussion on this SI suggested that some syntactic interface is needed here. Tucker will keep this SI. Randy looks at the old minutes and Tucker says he thinks he understands the previous decision.

SI99-0021-1/01 Obtain representation clauses based on defining identifiers, not declarations

Jean-Pierre says that if there are multiple declarations, you can't get the individual declarations. Jean-Pierre wants to add the defining identifiers to the Corresponding_Representation_Clauses function. There is concern that that could be incompatible, and that the capabilities are already provided by the semantic subsystem. Jean-Pierre says that going into the semantic subsystem for pure text processing would be a pain, and the capabilities should be provided in the existing interface.

There is e-mail and note that Corresponding_Pragmas also has similar problems. Both functions should be fixed in the same way. Jean-Pierre will take this and propose a solution to both functions.

SI99-0024-1/09 Provide a semantic model in addition to existing syntactic model

Randy notes that the actual specification is not included in the wording in ASIS (because of potential copyright issues). The SI probably should be reorganized to reflect that.

Tucker notes that the Boolean queries in xxx.xx.1.1 are technically redundant, but he thinks they are worthwhile. Randy agrees.

Jean-Pierre asks why we are using the name Region_Part_Array; ASIS tends to use "_List" for such types. We ought to be consistent with the existing part of ASIS when possible.

Do the functions for Representation_Items look through privacy? The full view should have all of the representation items, but partial views should not. The wording should be "visible on that view" so that requesting information on a private view shows just what's known about that view. These queries do return pragmas as well as clauses (the wording says representation *items*); the return type name is misleading and should be changed.

Jean-Pierre comments that a lot of this is similar to existing ASIS facilities. Tucker says that is correct, because it seemed important that this is complete; leaving out some possibilities would be very annoying for users.

Steve Baird asks how derived types fit into the category functions, as well as the "additional operations" that can be declared if a component is private. The category could depend on the location of the view of the type. Tucker thinks that the functions should return the "most" that a view ever has.

Upon consideration, that seems wrong. Consider Is_Array in the following:

```
package P is

   type Priv is private;

   package Nested is
      type Der is new Priv;
   end Nested;

private
   type Priv is array (1..10) of Character;
end P;
```

In the body of Nested, Is_Array(Der) = True. Tucker thinks that Der would need more than one view in this case. Each particular reference has a location, and that should represent what is true at that point. And you might get a different subtype_view, even though the name of the type is the same. That implies that all views depend on the location of the original reference (not just the declaration denoted).

There should be some sort of Underlying_Type to get the *real* type. We don't want to have to walk though many types to find out if something is an integer. Jean-Pierre suggests "Ultimate_Type".

The informal text at the top of the SI has some issues, it probably ought to be removed (particularly the list of subprograms, as the names have been changed in some cases). It is suggested that it be placed in the appendix.

The exception Caffeine_Error is raised; a break is taken.

When we return from the break, we begin to discuss the function Ultimate_Ancestor. What exactly does it do? Do you look through derivation? Yes. It would unwind any number of derivations and (incomplete or partial) views to get at the "real" properties of the type.

Questions: what about private extensions? What about discriminants of a derived type (where they change from those of the original type)?

Steve Baird suggests "Ultimate_Ancestor of the first interface", he doesn't like the non-determinism of an unspecified interface. Is "first" unambiguous in this suggestion? A private type can be completed with interfaces in some other order (or even a different list). Clearly, this is bad. So we leave Tucker's "unspecified" wording. But it should say "unspecified interface" rather than "unspecified one".

Task and protected types with progenitors are not derived, Is_Derived is False. That's the way the Ada standard defines derived types.

We can get progenitors of a type from a child package (defined later).

Full_View is a bit messy if a limited with is involved. If you have a limited view of a private type, Full_View(Complete_View(T)) goes to the full type; Complete_View(Full_View(T)) goes nowhere. A bit of a trap. But you could use Ultimate_View to get there. Tucker thinks that we should just define Full_View(T) to return the full view of the completion for an incomplete type. (The Standard doesn't necessarily define that.) On the other hand, Complete_View does not look through privacy.

ASIS.Identifier does not work for aspects; they cannot be created out of thin air. (They must exist somewhere in the program source). Program_Text would work; but it's probably better to just make the list of aspects an enumeration.

Jean-Pierre notes that Address is missing from the list of aspects. Tucker explains that these are only for types.

Randy thinks that we need a better name than XXX_Error. Tucker says it was just a placeholder. It should be ASIS_Inappropriate_<some word>, probably ASIS_Inappropriate_View. It probably should be added to the Asis.Exceptions package.

Another of Randy's e-mailed questions was about xx.x.3.2. Functions Is_Base_Subtype and Base_Subtype should be eliminated from xx.x.3.2 - they only apply to elementary types. First_Subtype should be documented here. Steve Baird wonders if this is true; the base subtype doesn't have a name, but surely it exists. That's the unconstrained type for a constrained first subtype.

Progenitors should not be in Subtype_Views, and the forward definition for Ultimate_Ancestor should have a forward reference.

Jean-Pierre wonders if we should have Is_Variable rather than Is_Constant, so that it is more logical. A value isn't a constant, at least not as defined by the language, so Is_Constant = True for values is misleading. Since a value is not a variable, either, Is_Variable = False would not be as misleading. Tucker thinks such a change would sweep the issue under the rug. It is suggested "Is_Constant_View" would be a better name for this function.

Jean-Pierre asks if a renaming of a component is a component?? What do the queries return for an object renaming? If you have Obj : **renames** A.**all**; is Is_Dereference true for Obj? And so on.

For static analysis purposes, Jean-Pierre says that renames are a pain.

Steve Baird notes that if you are looking for all of the places that raise Constraint_Error from dereferencing a null value, the renaming is such a place, and the use of Obj is **not** such a place. So clearly you can't always walk through renames. That means that we probably need a query and function to walk through them.

On to xx.x.4.3. The accessibility level shouldn't be defined based on type Natural. Use instead:

```
type Static_Accessibility_Level is range 0 .. <implementation-defined>;
```

Is_Implicit_Dereference returns true only for access-to-object; there should be similar routines in for access-to-subprogram for Callable_Views. Note that the type is different and disjoint so that there is no confusion.

Moving to xx.x.4.4. "Longest_Integer" needs to be defined somewhere. The first sentence of the wording should say "…given view is a view of the value…".

Jean-Pierre asks about static expressions of enumerated types? Seems like you should be able get the values for those.

Tucker suggests calling it Static_Discrete (rather than Static_Integer). Steve Baird would like to have an infinite precision package here. This leads to a long discussion, most people think that is overkill. We eventually converge on the idea that is overkill. But should we even return numbers? They rarely work perfectly, but still would be useful for enumerations and other bounded types.

Jean-Pierre thinks that returning String here is funny, ASIS *always* uses Wide_String. Tucker thinks these are special, but they aren't that special.

Aside: what does ASIS do for giant characters (beyond Wide_Character)? It always returns Wide_Strings, so those characters don't fit. Humm, someone needs to decide that. Tucker will take an action item to try to decide what to do here (surely this is allowed in string literals and comments, whether it is allowed in Ids is not known).

Use ASIS_Natural rather than Natural in Object_Size, as it may have a larger size than Natural.

Tucker asks for volunteers to help with the wording. Randy and Steve volunteer. Tucker will take the types, Randy and Steve will divvy up the rest.

Jean-Pierre will try to convert an example ASIS application of his to use this interface.

Steve Baird wonders if there is a way to distinguish a limited view from a regular incomplete type. It seems like there ought to be such a mechanism.

Approve intent: 8-0-0.

SI99-0030-1/02 Add definition section to the standard

Randy notes that the standard reference needs to include the Corrigendum (ISO/IEC 8652:1995/COR1:2001) and Amendment (ISO/IEC 8652:1995/AMD.1:2007). That is agreed.

Approve SI with changes: 8-0-0.

SI99-0035-1/02 Undefined capabilities of the Data_Decomposition package

Wording: superceded should be superseded.

Move all of the stream stuff to Annex F. That's all of 23, and all of the routines with parameters or results of Data_Decomposition.Portable_Data. (See the !discussion of the AI.)

"logging and delogging" is junk in third paragraph of 23, change to "marshalling and unmarshalling".

22.22, 22.23 drop the (See rules 6.A and 6.B above.)

Approve SI with corrections: 8-0-0.

SI99-0036-1/01 Expressions from normalized expressions

Remove the colon and bullet from the third item. The second item should not have a bullet on paragraph 8.

Approve SI with changes: 6-0-1.

Detailed Review of Ada 2005 regular AIs

AI05-0001-1/01 Bounded containers and other container issues

Matt Heaney gives us a status report. He hasn't done too much. WG 9 has shown sudden interest in the status and progress of this project, so we need to give it a higher priority.

He says the requirements are not that clear.

For instance, the tasking forms can be implemented in three different ways. Which one is appropriate? Some people on comp.lang.ada (real-time practitioners, in Randy's opinion at least) noted that locking a single container is not useful most of the time, you need to lock a set of containers that make up your abstraction. Do we want to allow simultaneous reading? The group thinks that simple, completely task-safe containers are useful enough of the time (especially in prototyping) that we should probably provide them.

Matt goes on to ask about restricted forms (that avoid exception handlers and the like to be used in high-integrity applications). AdaCore has some experience in this area. It's not clear that users actually have much interest in these forms; they've mostly been used for in-house applications. There aren't SPARK forms at this point, but most users seem to prefer to write their own. Randy wonders why there would need to be much difference for such forms, as we don't specify the implementation of these containers.

Tucker asks if it is possible to abstract some of the properties of the containers. Such abstract properties could be provided in all forms. Matt is dubious that this works.

Matt wonders whether finalization should be specified differently for bounded forms. He comments that the place of finalization would likely be different. Randy notes that the current rules were intended to be loose enough to allow a wide variation in implementations, including large allocations that would be essentially the same as a bounded form. However, we might want to specify more determinism for finalization in the bounded forms.

Matt says that he would like to have a schedule. Ed suggests having a container meeting in New York.

What is the list of containers that are of interest here? Bounded forms is always included, other types of containers?? Protected containers? Tucker says that would be useful. What about the iteration callbacks? Do we allow concurrent reads? Yes. Tucker suggests following Java, but they don't allow concurrent reads.

Should the callbacks be protected access-to-subprogram? Matt doesn't think that would be helpful, it wouldn't lock the right thing. The protected object that synchronizes the container is not visible.

Randy suggests a radical solution: use an instance of a protected interface to make the lock available to the user. That would allow multiple containers to share a lock, and would reduce the objections of the real-time folks.

Ed suggests a meeting at AdaCore in September 22nd-23rd just for container discussion. Then we'll work on the second day of the Portland meeting (November 1st).

Randy suggests adding a few additional people to the working group (mailing list) to get a few more opinions.

AI05-0006-1/05 Nominal subtypes for all names

Randy explains that Pascal noted the previous definition was incompatible with that for the stream attributes; that was not intended. (Indeed, AI said that it was intended to be compatible!) So this was pulled and fixed.

Jean-Pierre wonders how an implementation can specify some other nominal subtype; Randy notes that the wording says "unless otherwise specified". So there has to be wording in the definition of the attribute.

The discussion needs updating to match the wording: it shouldn't say "first subtype" for untagged composite types.

Approve AI with changes: 8-0-0.

AI05-0009-1/05 Confirming rep. clauses and independence

We start with 13.3(3). Randy explains that "valid" has a defined meaning in Ada, and we need to cover cases where the value is valid but it is inappropriate for the entity or the way the entity is used. Tucker would like to change the wording slightly to "and appropriate for the use of the entity". That seems to leave out code addresses for data items. Steve Baird suggests "and appropriate for the entity and its use".

Typo in the summary: "Two new pragmas {are introduced} to."

Tucker suggests changing the last sentence of 9.10(1) to a wording that was not recorded. Steve Baird has a better suggestion:

"It is unspecified whether two nonoverlapping parts of the same composite object for which a non-confirming representation item is used to specify packing, record layout, Component_Size, or convention are independently addressable."

Tucker says that this doesn't quite work, so we leave the last sentence of 9.10(1) alone.

Jean-Pierre is confused by the summary. It talks about types being independently addressable, while this concept is only defined for objects. "...components {of an object} of the type are independently..."

C.6(9) is missing the curly brackets for an insertion for the last sentence. Remove the editor's note from this paragraph, we agree that anonymous array objects should be added.

C.6(13): "...from [all]{every} other component[s]..."

Steve wonders whether Independent is an aspect of representation. Tucker thinks that is an interesting question that we haven't considered. C.6(14) says that these are representation pragmas. But we don't have names for these aspects of representation; there should be sentences about that (Atomic*, Volatile*, Independent*). Tucker suggests a separate AI to answer this question. Steve Baird has the short stick. Also, we should decide if all of these are inherited for tagged types. Independent_Components applies to extension components of a descendant (not to the parent components).

We discuss the generic formal case. Randy notes that there is a similar unanswered question for pragma Pack. After discussion, we decided that we do not want representation pragmas in a generic formal part, and we need a rule to that effect. Randy will write an AI to add this rule.

With this upcoming AI, we do not need any rule here about generic formals, so delete that rule and its AARM note.

Ed suggests that he was previously wrong; we don't want inheritance for these pragmas. Having that would prevent giving later representation clauses.

Steve Baird worries that `type T is new S;` would potentially be different without inheritance.

We cannot decide on how or whether these pragmas are inherited; Steve is directed to write up what makes sense to him and we'll throw stones at his solution.

Tucker worries that the legality rules are written such that they would not support inheritance very well.

"It is illegal for a pragma Independent to apply to a component if it conflicts with other representation items that apply to the component. It is illegal to apply pragma Independent_Components to a type if it conflicts with other representation items that apply to the component."

Better to copy C.6(10-11):

"It is illegal to apply either an Independent or Independent_Components pragma to a component, object, or type if the implementation cannot provide the independent addressability required by the pragma (see 9.10).

It is illegal to specify a representation aspect for a component, object, or type to which pragma Independent or Independent_Components applies, in a way that prevents the implementation from providing the independent addressability required by the pragma."

Approve AI with changes: 8-0-0.

AI05-0050-1/03 Return permissions are not enough for build-in-place

In the Summary, change "should be permitted" to "is permitted".

Second bullet of the wording: drop Otherwise and parenthesis, add "If" ("If the result subtype of the function is unconstrained, then if..."

"match" is weird; this isn't static matching, and there is no concept of dynamic matching. "Satisfies" is the right term, but we need sliding. After much pondering, Tucker suggests "is not convertible to" (first bullet).

...and the result subtype is not convertible to..."

Steve thinks that suggests the legality of conversion rather than the dynamic semantics. 4.6(4) defines "convertible" as legality.

Darn! The language does not have an existing term that has the right meaning. We probably need to use "dynamically match". Then we need to define that term somewhere.

So define this term in 4.6, after paragraph 58.

"A constraint dynamically matches another constraint:

- For a constrained index constraint, if the length of each dimension of the value of the operand equals the length of the corresponding dimension of the target subtype;

- Otherwise, if the constraint satisfies the other constraint."

That seems heavy and error-prone and ungrammatical (it wouldn't get updated automatically as the language is maintained – a likely trap).

Tucker suggests just adding the term to 4.6(57): "If conversion succeeds, that value is said to dynamically match the subtype." That seems pretty weird.

We decide to take this off-line.

Steve Baird objects to the location that the exception is raised in. He's concerned that the exception is moving. Randy points out that even whether the exception is raised at all can be changed by this permission - see the example in his recent e-mail. That's surely worse than the exception handler changing, but it cannot be helped. Tucker convinces Steve with additional examples that weren't recorded and Steve withdraws his objection.

Randy objects to this permission about the storage pool. Before he can explain why, Steve points out that there could be multiple calls on Allocate, so saying "the" call is junk. Someone is confused, so Randy and Steve explain that there can be multiple calls for non-contiguous objects and/or coextensions. Should be "any associated calls" or something like that.

Randy would like a more general permission to clean up unreachable stuff, this can happen for allocators that are abandoned for any reason. There doesn't seem to be any reason to fix this problem for *just* extended return statements. Create a new AI on allocators that fail, surely we need a permission to call Deallocate in that case to avoid a leak, Randy claims we also need a permission for early finalization that we do not currently have. Tucker and

Steve disagree, they claim the object is not allocated until after initialization is successful. Randy doesn't buy this; he's worried about any controlled part, not just top-level ones.

Randy notes that his previous attempts to fix this issue have died because of opposition to enabling garbage collection. That's a larger issue, and we ought to fix the smaller problem in any case. Try to avoid discussing garbage collection, we're only looking at failed allocators. Move the Deallocate stuff into this new AI. Randy would like the Allocate stuff to be in this new AI as well.

Give this back to Bob to create the definition of "dynamically matching".

Steve and Tucker start discussing "at any time", which seems to be too many places. It should be either before the function, or at the point of the return statement, not some random place. We surely don't want it occurring in some nested function call – or in a local exception handler that doesn't enclose the return statement. Randy points out that it could potentially be raised in both places in the same function, depending on the form of the call. Bob will need to fix this, too.

AI05-0052-1/09 Coextensions and distributed overhead

This one was previously approved, but Pascal noted that calling limited private types immutably limited was a bad idea. He also noted that having immutably limited parts making a type immutably limited is an idea that the designers of Ada 95 rejected. Randy researched the latter change and determined that there was no need for it, so it should be dropped.

Thus, the definition of immutably limited was adjusted to eliminate these problems.

Typo: discussion in paragraph "Note that this rule... [in]{is} incompatible...".

Approve AI with changes: 8-0-0.

Randy later notes that a Tucker action item was covered here. 3.7(10) needed to allow some limited private types to be immutably limited (otherwise we have an unintentional incompatibility with Ada 95). He determined it was best to adjust the definition of immutably limited rather than changing 3.7(10).

The last bullet should be "A type derived from an immutably limited type" (so this does not include the type itself – that would be circular).

The AARM Ramification should say a "non-synchronized limited interface".

Approve AI with more changes: 8-0-0.

AI05-0053-1/06 Aliased views of unaliased objects

This one depends on AI05-0052-1, which was sent back, so we have to consider it again.

Approve AI: 8-0-0.

AI05-0054-2/04 Variable views of constant objects

This isn't in the ZIP file of the AIs that was distributed before the meeting.

Erhard wanted to talk about this (and he couldn't be here for family reasons), and we don't have the AI text, so we'll defer it again to next time.

AI05-0063-1/04 Access discriminants on derived formal types

This one depends on AI05-0052-1, which was sent back, so we have to consider it again.

The summary should say "are presumed".

Approve AI with changes: 8-0-0.

AI05-0067-1/05 Build-in-place objects

Steve Baird does not like "morphs". Tucker suggests "mutates". "Transforms" is also suggested. Randy suggests that the word should be "weird", because this is an unusual semantic, if it sounds normal it will be confusing. We agree on "mutate".

But then John comments that might be confusing with "immutably". Ed says that mutating an immutably limited type is not confusing.

Ed proposes the formulation: “an object of an immutably limited type is mutable” as a reminder that such objects in general have state. Should this mantra appear somewhere in the AARM?

Steve says that the dictionary definition of "mutate" seems appropriate.

John comments that the question says "build-in-place type", but that there are only objects that are build-in-place. It needs to say "type whose objects require build-in-place". Tucker thinks that surely we have the idea of a build-in-place type: a type for which all objects are required to be built-in-place. We decide to let John suggest a rewording of this question later.

Brad notes that the term defined is "built in place", while we normally have written "build-in-place". The wording for 4.3.2(5) uses "build-in-place". Tucker says we use hyphens when this is used as an adjective. We don't do that in this wording.

Reword 4.3.2(5) to say: "If the ancestor_part is a function call and the result of the function call is required to be built in place (see 7.6), …"

Randy is annoyed that the AARM will require extensive revisions for this term change.

Fix the problem Randy noted previously in e-mail:

"If the ancestor_part is a function call and the result of the function call is required to be built in place (see 7.6), then the ancestor_part shall have a constrained nominal subtype unless there are no components needed in the record_component_association_list."

Tucker notes that the next rule needs similar wording. Randy notes the assume the worst rule is seriously messed up: it doesn't handle functions of generic formals, for instance.

Tucker notes that this is a legality rule, and that cannot depend on a dynamic semantics rule for build-in-place. Dynamic semantics looks through privacy, and surely we don't want to be breaking privacy in legality rules.

Steve Baird notes that this has to be a tagged type. Tucker says that this should be just be a "limited type". In that case, we're automatically assuming the worst (everywhere), so no recheck rules or special assume-the-worst is needed. And this is not an Ada 95 incompatibility, because all limited aggregates were illegal in Ada 95.

"If the ancestor_part is a function call and the type of the ancestor_part is limited, then the ancestor_part shall have a constrained nominal subtype unless there are no components needed in the record_component_association_list."

The second RM paragraph and AARM Reason is deleted.

Steve says he has a minor point. Coextensions are just changing ownership, this is true here too (this is not mutation). We should use the same wording we did for coextensions of object declarations (see AI05-0066-1) "becomes" instead of "morph into" in the last bullet of the new text to 7.6(17).

"performaned" should be "performed".

We should put these in change order so that we can review better.

Steve notes a problem that Randy had previously noted (and lost): the aggregate requirement needs to apply to parts, so that a deferred constant of an arrays of private which is controlled (for instance) does not have the elaboration problem.

"In the case of an aggregate, if the {full} type {of any part} of the newly-created object is controlled, the anonymous object is built in place."

Tucker doesn't like the change to 7.6(16). Randy explains the reasons (the canonical semantics apply to limited and nonlimited), Tucker would prefer no change at all. But Randy disagrees, and Ed eventually stops the discussion as non-productive. The rest of the group doesn't seem to have an opinion on the issue, so Randy and Tuck are directed to resolve this between themselves.

Steve is worried about the implied change in accessibility level here. He thinks that the accessibility check would be wrong. Tucker claims that the checks occur when the object is created, but based on the ultimate accessibility level. We check 6.5, and the checks are against the master that elaborated the function. So there is no change in accessibility level.

Randy will take it to do the editing (it's not that much).

Approve intent of AI: 7-0-1.

AI05-0071-1/02 Class-wide operations for formal subprograms

Tucker explains the wording.

Randy wonders if name is correct, or should it be defining_identifier here? Actually, it probably should be defining_designator or maybe just "designator". Tucker says that "defining name" is defined for just such a use. The group is skeptical, but we look it up and that term is indeed defined in the Standard (3.1(10)).

"default subprogram name" should be default_name.

So the wording becomes:

If a generic unit has a subprogram_default specified by a box, and the corresponding actual parameter is omitted, then it is equivalent to an explicit actual parameter that is a usage name identical to the defining name of the formal. {If a subtype_mark in the profile of the formal_subprogram_declaration denotes a formal private or formal derived type, and the actual type for this formal type is a class-wide type T'Class, then for the purposes of resolving this default_name at the point of the instantiation, for each primitive subprogram of T that has a matching defining name and that is directly visible at the point of the instantiation, a corresponding subprogram with the same defining name is directly visible, but with T systematically replaced by T'Class in the types of its profile. The body of such a subprogram and the rules for handling the case when there are no controlling parameters are as defined in 12.5.1 for primitive subprograms of a formal type when the actual type is class-wide.}

Should this be a Binding Interpretation? Tucker says it should be the same as the other 12.5.1 changes (which belong to the Amendment (and was a Binding Interpretation - AI95-00158)).

Ed asks how this works for functions with controlling results. The answer is the same was as defined by AI95-00158 in the Amendment. There is not supposed to be anything new (dynamically) here.

The important point to remember here is that the identical operation could be imported implicitly if it is a primitive operation of a generic formal derived type. This would also require conjuring the actual operation. This just allows it to be imported more explicitly as a formal subprogram as well as via a primitive operation.

Tucker would like to make it clearer that call rules also apply. He adds text that appears above. But formal subprograms are never dispatching. So we reword the paragraph yet again.

If a generic unit has a subprogram_default specified by a box, and the corresponding actual parameter is omitted, then it is equivalent to an explicit actual parameter that is a usage name identical to the defining name of the formal. {If a subtype_mark in the profile of the formal_subprogram_declaration denotes a formal private or formal derived type, and the actual type for this formal type is a class-wide type T'Class, then for the purposes of resolving this default_name at the point of the instantiation, for each primitive subprogram of T that has a matching defining name and that is directly visible at the point of the instantiation, a corresponding subprogram with the same defining name is directly visible, but with T systematically replaced by T'Class in the types of its profile. If there are no controlling formal parameters, the body always raises Program_Error; otherwise the body of such a subprogram is as defined in 12.5.1 for primitive subprograms of a formal type when the actual type is class-wide.}

Randy wonders if we shouldn't just make the instantiation illegal in this case. Raising Program_Error seems to be hiding an error. We can't do that in the AI95-00158 case as a call could appear in the body, but these can only appear in the instance's parameter list.

So we won't conjure up an operation in this case (which will make it illegal, since there will be no matching operation). So yet another rewording:

If a generic unit has a subprogram_default specified by a box, and the corresponding actual parameter is omitted, then it is equivalent to an explicit actual parameter that is a usage name identical to the defining name of the formal. {If a subtype_mark in the profile of the formal_subprogram_declaration denotes a formal private or formal derived type, and the actual type for this formal type is a class-wide type T'Class, then for the purposes of resolving this default_name at the point of the instantiation, for each primitive subprogram of T that has a matching defining name, that is directly visible at the point of the instantiation, and that has at least one controlling formal parameter, a corresponding subprogram with the same defining name is directly visible, but with T systematically replaced by T'Class in the types of its profile. The body of such a subprogram is as defined in 12.5.1 for primitive subprograms of a formal type when the actual type is class-wide.}

Approve AI with changes: 6-0-2.

AI05-0083-1/02 Representation values of formal parameters

Add (Yes.) to the end of the first sentence of the question.

Jean-Pierre wonders if this is good. He is eventually convinced that the overhead of the alternative is prohibitive.

Ed says that the difficulty seems to be Medium (since we've talked about if for a long time at two meetings).

"This is necessary to maintain the language design principle that Alignments are always known at compile time."

Approve AI with changes: 7-0-1.

AI05-0090-1/03 Ambiguities with prefixed views of synchronized primitives

"Synch_Inteface" should be "Synch_Interface" in the specification of Yet_Another_Op.

In the wording 9.4(11.4/2), replace "defining_identifier or defining_operator_symbol" with "defining name". (Since we noticed this definition earlier.) Do the same in 9.1(9.5/2).

Randy notes that there is no way to call entry Yet_Another_Op: it is always ambigious. It would have to have its name changed in order to be able to call it. This is annoying, but no one has any idea of a solution that isn't worse than the problem.

The wording change for 4.1.3(9.2/2) should be shown in the full paragraph.

Approve AI with changes: 7-0-1.

AI05-0091-1/01 An other_format character should not be allowed in an identifier

We discuss this, and agree that the Unicode people probably are the experts in character sets and what should be in identifiers. (After all, the current definition was based on assuming that Unicode knows best.). And this change simplifies the language, and it will stop people from doing the silly thing of using these characters in identifiers. Leaving the change to the distant future will just increase the compatibility problem.

Approve AI: 8-0-0.

AI05-0093-1/02 Additional rules that need to use "immutably limited"

Approve AI: 7-0-1.

AI05-0094-1/01 Timing_Events should not require deadlock

Steve objects to "as soon as possible". Tucker looks at D.5.1(10/2), which says "immediately at the first point when *T* is outside the execution of a protected action".

Tucker claims that "as soon as possible" does not preclude deadlock.

"as soon as possible {after the completion of the call of Set_Handler}." That makes it more like paragraph 13.

Approve AI with changes: 8-0-0.

AI05-0095-1/01 Address of intrinsic subprograms

Tucker would like to fix the definition of program unit to make it clear that renaming a non-program unit does not make a program unit.

He notes that option 1 breaks the contract model. The Implementation Permissions also does in some sense.

Tucker offers always raising Program_Error in this case, and using the permission of 1.1.5(6) to report programs that will unconditionally raise Program_Error.

Randy complains that this is just a warning. He would like a real error, at least when the error can be detected without destruction of the contract model.

Tucker suggests the wording:

"The prefix of X'Address shall not statically denote a subprogram that has convention Intrinsic."

"X'Address raises Program_Error if X denotes a subprogram that has convention Intrinsic."

Tucker would like an AARM note that a rename of a non-program unit is also not a program unit.

Approve intent of AI: 7-0-1.

AI05-0096-1/02 Deriving from limited formal types

Tucker suggests "...{then} in addition...". First "a" should be "an" in the AARM note.

Answer the question: "(Yes if untagged.)"

There is concern about the question not mentioning tagged. Hopefully, the answer given above will be enough.

Add a tagged example to the discussion.

The last example in the discussion needs "`...is {limited} new P.LP;`".

Approve AI with changes: 7-0-1.

AI05-0097-1/01 3.9.3(4) should include abstract null extensions

"...rules [which]{that} ordinarily make function{s}...".

Steve notes an error in the new AARM 3.9.3(4.a/3) [that note is for AI05-0068-1]. "That is necessary {to} preserve privacy."

Tucker and Jean-Pierre would like to improve the wording of this whole set of bullets. They make some suggestions, but eventually they are stopped as is too big a job for the meeting. There is just too much risk of breaking something. Tucker can try rewording these bullets if he wants to.

Approve AI with changes: 8-0-0.

AI05-0098-1/01 Incomplete type names can be used in anonymous access-to-subprogram types

Tucker wonders why null exclusion isn't allowed on an incomplete view. Randy replies that an incomplete view is not an access type. (It's impossible for it to have index or range constraints for the same reason: it is not an array or scalar type, either.)

Second paragraph of question: "3/10/1" should be "3.10.1".

Parenthesize the change to 3.10.1(6):

as the subtype_mark in the subtype_indication of an access_to_object_definition; the only form of constraint allowed in this subtype_indication is a discriminant_constraint{ [(no null_exclusion is allowed)]};

That is redundancy brackets, not a deletion above.

Approve AI with changes: 8-0-0.

AI05-0099-1/01 The tag, not the type, of an object determines if it is controlled

Tucker wonders about privacy, this wording doesn't cover that. An untagged private type can be completed with a tagged type. Randy tries to point out that dynamic semantics ignore privacy. But then Tucker says that there is nothing wrong with the original wording.

So most feel that a Ramification is more appropriate. Rewrite the AI as a ramification with just a To Be Honest AARM note.

Approve intent of AI: 7-0-1.

AI05-0100-1/01 Placement of pragmas

Adam noted that there are weird cases where pragmas appear to be technically illegal. We need someone to fix this.

Tucker suggests (after much discussion) adding "item" to the list of syntax entities. We carefully go through the list, and do not find any problems with adding that.

But that doesn't address the "not in place of", and that is the point of confusion.

Tucker says that he thinks that the meta-rule for pragma placement is that the program is still legal (syntactically) if the pragmas are erased. The group agrees that that seems to be an appropriate principle.

Based on this meta-rule, Tucker doesn't think that we can improve upon "not in place of"; he thinks it doesn't apply when the syntax allows nothing at all for a listed entity (as in the list of entries in the question).

Not everyone agrees with this resolution, but no one has a better suggestion (or perhaps are afraid that the AI will be assigned to them if they speak up).

Tucker will write this wording. Add "item" to this list, and then add an AARM note explaining that "not in place of" does not apply to lists of zero items. (And also add notes explaining the meta-rule for pragma placement.)

Approve intent: 8-0-0.

AI05-0101-1/01 Remote functions must support external streaming

Randy notes that he fixed the obvious problem. But he doesn't understand the secondary problem, and thus made no attempt to "fix" it.

Tucker says that such calls (that don't return a remote access type) are illegal. But a lengthy search didn't turn up any such rules. Tucker wonders if that was simply disallowed by definition of remote types. But AI05-0060-1 removes anonymous access from this consideration, so they are now allowed as parameters and return types.

Therefore, Tucker says that AI05-0060-1 may have indirectly had the effect of removing the restrictions that should not have been removed. He says that remote access parameters are supposed to have the same restrictions as remote access types.

This will need to be researched by someone with knowledge of this area.

Brad will take this AI and try to figure out what (if anything) is wrong with AI05-0060-1 and add any needed fixes to this AI. Also, he will try to figure out if controlling access results are also remote access types (as Thomas Quinot seems to think.)

Keep alive: 7-0-1.

AI05-0102-1/01 Some implicit conversions ought to be illegal

Tucker believes that cases like this could be written in Ada 95. But he doesn't think that the problem existed. We look at 8.6(25), which is where implicit conversion is (implicitly) defined. In Ada 95, this says "access-to-variable", so there was no problem (access-to-constant would not resolve). But we didn't want the name resolution to be that specific; we would want a general resolution rule and a more specific legality rule.

Perhaps we should add a legality rule into 8.6 (it would be after 8.6(32)). Steve Baird complains that this would be very tricky.

Tucker suggests:

"If the expected type is *T1* and the actual type of the construct is *T2*, then *T2* shall be convertible to *T1* (see 4.6)."

An AARM note should be added to explain, and this should be indexed to "implicit conversion".

"unconfortable" in the discussion should be "uncomfortable".

Tucker notes rules about convertible in 3.7, 3.7.1, and 6.4.1. (4.5.2 also uses the term.) These apparently were an (incomplete) Ada 95 solution to the problem. These rules could be marked as redundant, with a pointer to 8.6.

Approve intent: 8-0-0.

AI05-0103-1/01 Return statements should require static matching for all named access types

The bracketed text needs to include a "shall".

"...as previous{ly} modified..."

"the result subtype of the function is {an access {sub}type..."

Tucker worries that this is too restrictive.

Steve would prefer to use "static compatibility". He had previously suggested that and we rejected it.

We could also just make this case illegal directly. "If the function subtype excludes null, the result subtype shall exclude null."

We consider other alternatives:

Ed would prefer static matching everywhere.

Randy would prefer static matching for elementary and static compatibility for composite. He doesn't believe that encouraging the use of access subtypes is worthwhile, usually you will give the bounds /discriminants in an allocator, not in a subtype. And these things have caused no end of semantic trouble in Ada.

Tucker would prefer just static compatibility. He thinks the model should be that it is allowed anytime that there is no check on return.

Steve Baird will take this one AI and make some sort of proposal.

Keep alive: 8-0-0.

AI05-0104-1/01 Null exclusions are not allowed in uninitialized allocators

John wonders why we don't have a similar rule for object declarations (requiring an initializer). That's more complicated, not worth it now.

Approve AI: 8-0-0

AI05-0105-1/01 Resolution of renames uses anonymousness

Randy notes that this renames is the only way to make such a call.

```
type Acc_Int is access all Integer;

function Foo return Acc_Int; -- (1)

function Foo return access Integer; -- (2)

Obj : access Integer renames Foo; -- OK, renames a call on (2)

procedure P (A : access Integer);

P(Foo); -- Ambigious.

P (Acc_Int'(Foo)); -- OK, calls (1).

P (Obj); -- OK.
```

The function (1) returning a named access type is callable with effort, but the one returning an anonymous access type (2) cannot be called other than via the renames.

Tucker says that he would not expect to be able to rename a named access object as an anonymous access object, so the use of a function call shouldn't change that fact. No one disagrees with this.

Ed is concerned that this rule is more specific that we usually use for resolution. He fears that this is a camel's nose situation, where similar rules will start popping up all over.

Randy queries again on the importance of this test case in the ACATS. Tucker suggests a test case using two functions that return components (one anonymous and one named), that is a better test than the functions alone. There is general agreement that the rule should be tested.

Typo: misspelled anonymous in !response, first line (n instead m).

Ed reluctantly agrees to this rule so long as it doesn't spread past renames.

Approve AI with changes: 7-0-1.