Minutes of the 31st ARG Meeting

17-19 November 2006

Albuquerque, New Mexico, USA

Attendees: Steve Baird, John Barnes, Randy Brukardt, Gary Dismukes, Bibb Latting (except Friday morning), Pascal Leroy, Brad Moore, Erhard Ploedereder (except Saturday morning), Ed Schonberg (except Sunday), Tucker Taft, Bill Thomas (except Saturday afternoon and Sunday), Joyce Tokar (except Friday morning and late Friday aftermoon).

Observers: Greg Gicca (except Saturday afternoon and Sunday), Steve Michell (Friday only).

Meeting Summary

The meeting convened on 17 November 2006 at 09:15 hours and adjourned at 13:45 hours on 19 November 2006. The meeting was held in a conference room at the Hotel Albuquerque at Old Town in Albuquerque, New Mexico, USA. The meeting covered nearly the entire agenda (excepting a few Ada 2005 amendment AIs).

AI Summary

The following AIs were approved with editorial changes:

AI05-0007-1/02 Stream 'Read and private scalar types (9-0-1)
AI05-0008-1/03 General access values that might designate constrained objects (9-0-3)
AI05-0013-1/02 No_Nested_Finalization is difficult to enforce (11-0-1)
AI05-0015-1/02 Constant return objects (11-1-0)
AI05-0016-1/01 Others => <> can be used in place of null record (8-0-2)
AI05-0020-1/01 Operators of partial views of fixed point and access types (9-0-2)
AI05-0021-1/01 Issues with containers (10-0-1)
AI05-0025-1/01 Missing rules in 12.7 (10-0-2)

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

AI05-0002-1/01 Unconstrained arrays and C interfacing (4-0-7)
AI05-0009-1/02 Confirming rep. clauses and independence (11-0-1)
AI05-0017-1/01 Freezing and incomplete types (10-0-1)
AI05-0022-1/01 Container tampering should be checked for formal subprograms (10-0-1)
AI05-0023-1/01 'Read on records with variant parts (8-0-3)
AI05-0024-1/01 Run time accessibility checks (10-0-0)
AI05-0027-1/01 Finalized container objects (5-2-5)
AI05-0028-1/01 Problems with preelaboration (8-0-2)

The following AIs were discussed and assigned to an editor:

AI05-0018-1/01 Formal package matching rules
AI05-0019-1/01 Primitive subprograms are frozen with a tagged type
AI05-0026-1/01 Missing rules for Unchecked_Union
AI05-0029-1/01 Meaning of 12.5(8)
AI05-0032-1/00 Extended return statements for class-wide functions

The following AI was voted No Action:

AI95-00448-1/01 Extended range of Year_Number (7-0-5)

The following Amendment AIs were discussed and shelved until such time as we are considering new features for Ada:

AI05-0010-1/01 Revoking 11.6 permissions
AI05-0030-1/00 Requeue on synchronized interfaces
AI05-0031-1/00 Add a From parameter to Find_Token

The following SIs were approved with editorial changes:

SI99-0003-1/02 Support overriding indicators (10-0-0)
SI99-0011-1/02 Add A_Tagged_Incomplete_Type_Declaration to Declaration_Kinds (9-0-2)
SI99-0017-1/02 Trait_Kind is unclear for access definitions (12-0-0)
SI99-0018-1/01 Add new Attributes to Attribute_Kinds (12-0-0)

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

SI99-0004-1/02 Changes to Asis for changes to access types (9-0-1)
SI99-0012-1/03 Add support for null procedure declarations (11-0-0)
SI99-0016-1/01 Correct Corresponding_Type_Operators (12-0-0)
SI99-0022-1/01 Add Boolean queries to ease use of Trait_Kinds (9-0-1)
SI99-0023-1/01 Uses of subtypes Name and Name_List (10-0-2)

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
SI99-0009-1/02 Handle new aggregate features
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
SI99-0024-1/01 Provide a semantic model in addition to existing syntactic model

The following SI was voted No Action:

SI99-0020-1/01 Use "is null" defaults in Traverse_Element (12-0-0)

Detailed Minutes

Meeting Minutes

The minutes of the previous meeting were approved by acclamation.

Review of WG 9 decisions impacting the ARG

Pascal says that we didn't get assigned POSIX, rather a new PRG was created with Steve Michell as Rapporteur. Steve asks ARG members to join the PRG.

Date and Venue of the Next Meeting

We'll next meet in Geneva, June 29 afternoon and June 30, July 1.

Tucker announces that his wedding anniversary is the day before this, so he probably couldn't attend. Groans all around. Someone suggests that he give his wife a trip to Switzerland as an anniversary present. We discuss whether alternative dates before the meeting would be better for him. No conclusion is drawn.

Details of the ASIS revision

Randy and Pascal note that converting the ASIS standard to the WG 9 recommended format has turned up a number of problems. Randy would like some guidance on these issues.

Randy notes that it might be a good idea to reorganize the standard to more closely follow the current ISO drafting rules. One example would be creating a separate definitions section. He's asked what else ought to be changed. He doesn't have a detailed list; in general, the ASIS standard is closer to following the rules than the Ada standard is.

The group decides that more information is needed in order to provide guidance. He’s asked to create a list of possible changes and circulate it.

One problem is that the clause numbers will change as we insert new clauses for new entities. We don’t have the option of inventing new numbering schemes, as that is determined by the drafting rules for standards. The fact that the numbers will change frequently will make review of the draft standard very difficult.

After discussion, we decide to insert new clauses as “not a clause” for the time being to avoid numbering changes. The SIs should reference the clause numbers of the original Standard. Of course, at some point in the development of the Standard we’ll have to convert them to real clauses, but that can wait a year or more.

Many of the SIs have wording sections that are modeled on the existing Standard. But we’re changing that organization significantly. We decide that the wording sections of the SIs should match the new document format.

Should we make an AARM analog for the ASIS Standard? There is a lot of junk in the existing Standard. For instance, the “CR” items provide a sort of cross-reference between entities and operations. That’s mainly useful for the writers of the standard (unless it is summarized in an annex, anyway). Making an AARM analog seems like a good idea.

Tucker would like the tool to be able to produce full specifications for the packages. In particular, the private parts and the other stuff deleted from the Standard should be moved to the AASIS document and somehow used to generate Ada files. Randy will take an action item to investigate if that can be done (the tool and input formats were not designed for that purpose, so it might be hard to retrofit).

Randy notes that the ASIS Standard is full of “Application Notes”. He had converted these to regular notes, but the more he worked on the Standard, the more dubious he got. They tend to be examples of use, rather than notes at all. Perhaps these should be handled differently than regular notes, maybe even in the AASIS. Pascal says the less non-normative text in the Standard, the better. Yes, the group agrees that these should be moved to the AASIS.

Randy worries that there are very many simple (almost mechanical) changes made to the original ASIS Standard to create the new one. These are mostly marked as changes for “version 1” of the document [the “baseline” version of the Standard]; the functional changes that we’ll make will be marked as changes for “version 2” of the document. That means that they’ll be in different colors, and that should make it easier to look only at things that are important. But he didn’t mark spacing, formatting, and punctuation changes (it’s just too much work, and too trivial).

Do we need to record what changes where made to convert the document into its new format? A detailed record would take weeks to produce, and it’s hardly likely anyone would want to read it anyway. The group feels that it is important to have a simple record of these changes. It is suggested that Randy create a simple SI that describes in very general terms what reformatting and editorial changes were made to the Standard as part of its format conversion. Randy takes an action item to do this.

Tucker wonders why "=" is declared to be abstract throughout the Standard. Randy points out that this is described in the Rationale part of the Standard. The explanation there is not very satisfying. Why didn’t they just override these properly? Abstract "=" is a pain, and it can reemerge (it doesn’t prevent calling of "=" in generics). Pascal will take an action item to study this and possibly produce an SI to define equality properly for ASIS types.

Randy wonders what we’re going to do with obsolete functions and types. They’ll need to be marked somehow; we’ve made many of these but haven’t discussed what form that marking will take. Perhaps we need an annex like the one in the Ada Standard? Yes, we need something like Annex J for ASIS. As with Annex J, the idea is that new code will avoid these features, while existing code can continue to use them.

Thanks

By acclamation, the ARG thanks our host, SIGAda, for the accommodations for the meeting.

Old Action Items

Bibb did not do AI05-0012. Tucker did not do the lower priority items AI05-0003, AI05-0006. Randy did not quite complete the conversion of the ASIS standard (it proved to be a lot more work that anticipated). All other items were completed.

New Action Items

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

Steve Baird:

- AI05-0002-1
- AI05-0018-1
- AI05-0023-1
- AI05-0026-1

Randy Brukardt:

- SI99-0016-1
- Complete converting the existing ASIS standard into the format of our standard-creating tools. Create an AASIS, moving non-normative stuff out of the descriptions of the subprograms. (See “Details of the ASIS Revision”.)
- Create a simple SI describing in general terms the editorial and formatting changes that were applied to the entire ASIS Standard to create our baseline version. (See “Details of the ASIS Revision”.)
- Create and circulate a list of changes needed to bring the ASIS standard up to the current ISO standard drafting rules.
- Investigate enhancing our standards tools to allow automatic generation of the complete package specifications for the ASIS packages. (See “Details of the ASIS Revision”.)

Editorial changes only:

- AI05-0007-1
- AI05-0008-1
- AI05-0013-1
- AI05-0015-1
- AI05-0016-1
- AI05-0020-1
- AI05-0021-1
- AI05-0025-1
- SI99-0003-1
- SI99-0011-1
- SI99-0017-1
- SI99-0018-1

Greg Gicca:

- Add ARG approved SIs to the draft ASIS standard.

Bibb Latting:

- AI05-0009-1
- AI05-0012-1

Pascal Leroy:

- AI05-0017-1
- AI05-0019-1
- AI05-0028-1

- AI05-0029-1
- SI99-0007-1 (move to the semantic domain)
- SI99-0019-1
- SI99-0021-1 (move to the semantic domain)
- SI99-0023-1
- Study whether it is possible to properly define "=" for types in ASIS rather than declaring "=" abstract — create an SI to fix this if possible. (See “Details of the ASIS Revision”.)

Tucker Taft:

- AI05-0003-1 (lower priority)
- AI05-0006-1 (lower priority)
- AI05-0022-1
- AI05-0024-1
- AI05-0027-1
- AI05-0032-1 (lower priority)
- SI99-0004-1
- SI99-0024-1

Bill Thomas:

- SI99-0009-1
- SI99-0012-1
- SI99-0022-1
- Query ASIS users on whether changing the names of the parameters to some ASIS routines (specifically, Attribute_Kind, Name_Image, Prefix, Index_Expressions and Slice_Range) would cause hardship (see discussion of SI99-0023-1)

Detailed Review

The minutes for the detailed review of AIs and SIs are divided into Ada 95 AIs, 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 Ada 95 AIs

AI95-00448-1/01 Extended range of Year_Number

Randy explains the problem: the change to the range of Year_Number makes the transition from Ada 95 to Ada 2005 difficult for implementers: it is very complicated to change the bounds on types, and maintaining multiple copies of the runtime is too expensive. He wrote up the AI as a requirement as that was easier, but he doesn’t think it really matters whether it is a permission or requirement.

Requiring it is annoying; Pascal says he could live with a permission. So this should be an Implementation Permission that upper bound is 2399.

John would prefer to not give this permission — this would be a non-portability and we don’t need it.

Gary wonders how this permission could cause trouble. Maybe someone used 'Last or forced the year to fit into the minimum number of bits.

Ed says that their implementation already has made this change and it is unlikely to be changed back; and they would like the permission. Gary says that AdaCore is generally very concerned about incompatibility — but not this one.

The discussion does not seem to be changing anyone’s mind. We take a vote:

Approve AI as a permission: 5-3-3. This is not conclusive. Erhard voted against because he wants it binding. John and Pascal are against because of the non-portability.

Straw vote on making it a requirement: 4-4-3. That is even less conclusive.

Pascal threatens to send this to WG 9 for a resolution. Someone suggests that we vote it No Action instead, since it seems unlikely that we will reach consensus. We vote on that:

No action: 7-0-5. At last, a conclusive vote.

Detailed Review of ASIS Issues

SI99-0002-1/03 Add A_With_Clause to the list of expected kinds for Asis.Elements.Trait_Kind

This just allows querying traits on with clauses.

Approve SI: 10-0-0.

Later in the meeting, we decided to merge this SI with SI99-0022-1, and delete it completely. See the discussion of SI99-0017-1.

SI99-0003-1/02 Support overriding indicators

Erhard does not like the enumeration, because there is only a ‘t’ difference between two of the literals. That’s likely to be confusing. Pascal dislikes An_Overriding_Indicator, because that could be confused with the syntax entity overriding_indicator, which is present on every subprogram (the literal is supposed to mean just the keyword **overriding**).

Bill suggests Not_An_Overriding_Indicator; he says this is consistent with other ASIS literals. Tucker suggests An_Indicator_of_Overriding and An_Indicator_of_Not_Overriding. No one is very excited about this idea, but it is consistent with other ASIS literals, and it is not confusing when used. No one seems to have a better idea.

Bill notes that the type should be Overriding_Indicator_Kinds to be consistent with the other types in ASIS, and the function should be Overriding_Indicator_Kind.

Gary notes that the question (and subject) has a misspelling of “overriding”

Approve SI with changes: 10-0-0.

SI99-0004-1/02 Changes to Asis for changes to access types

Tucker combined three SIs into 1. There were two routines that determined the subtype of an object, one returning the subtype mark and one returning the subtype indication. These were combined into one, as there isn’t much value to having these separate with the expansion of anonymous access types.

Tucker also says he made a mistake and Has_Null_Exclusion has the wrong description.

Pascal wonders why we need Result_Has_Null_Exclusion. This is called with the whole function; there might be null exclusions elsewhere in the function (in the parameters), but this applies only to the result.

Pascal suggests that we have a function that returns the return subtype (that should exist, right?), then we can apply Has_Null_Exclusion to that subtype.

Existing function Result_Profile returns a subtype mark. That's not good enough any more, some SI should have fixed that. Ummm, that would be **this** SI!! Tucker wonders if he forgot something.

Pascal points out that all of the queries that returned a subtype_mark in Ada 95 (that is, locations that only allowed subtype_mark) that now allow anonymous access and/or null exclusion need something changed.

Tucker thinks those other places were handled, but clearly Result_Profile should also be obsolete, and a new function Result_Subtype should be defined to return a subtype. Then Result_Has_Null_Exclusion isn't needed and can be removed.

Pascal wonders if there is enough stuff here to analyze anonymous access-to-subprograms. He wonders if there is a way to get the designated profile for such types. That seems to be covered by existing ASIS functions for regular access-to-subprogram subtypes.

Tucker says that Access_To_Function_Result_Profile (defined in 16.20) also needs to be obsolete and replaced in the same way as Result_Profile.

Approve intent of SI: 9-0-1.

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

The SI says that existing queries (defined in 17.6 and 17.9) cover this use. These queries might need updating to allow this usage. "Might" is not an appropriate wording change!

Pascal says that Corresponding_Name_Declaration (17.6) is useless for implicit declarations (since it probably returns Nil), and this is an implicit declaration. So that doesn't work well. We discuss whether prefix notation is really in the semantic domain. But that seems wrong, too, as there is syntax here.

This is only a tiny corner of what is broken in OOP in ASIS. Dispatching calls are defined to return Nil. Class-wide subtypes are represented as Nil. It's hard to get concerned about fixing just this corner — there is a lot wrong here — and that why we want to introduce the new semantic domain model.

That leads us to several "big picture" questions. To this point, we've looked at the semantic domain model as some non-critical fixes for ASIS that we could drop if it got too tough. But if we expect it to provide basic new functionality (such as identifying prefix notation when used), then it is not an optional part.

Should the semantic domain model get handled by patching up the existing routines or by adding a new semantic set of routines? Tucker suggests we do both. But that is a huge amount of work; the semantic domain model is complex enough without duplicating it (poorly). Moreover, OOP in ASIS is essentially impossible for the reasons mentioned above; trying to patch that up is not worth the effort (and it would be less compatible than building new routines for that purpose). So we need to do at least part of the semantic interface to provide proper handing of OOP. That implies that at least that part of the semantic interface is not optional; we ought to concentrate on that part first.

With this conclusion, we can return to the SI itself. This means that Corresponding_Name_Declaration either returns Nil or a real prefixed view; the implementation gets to decide.

The wording section should be !discussion. We should add something to say that these routines either will return nil or a prefixed view. The summary should be changed (no new functions are needed for the syntactic domain).

Can some normalizations be allowed and non-reversible? Randy says that most normalizations are optional, controlled by user-specified parameters. Greg thinks that static expressions can be normalized away. Ed says that may be, but customers want direct source recoverability. Even if the standard gives permission for more flexibility, the users don't want their implementations to take advantage of that flexibility. So why even have it in the standard??

Should we have an Is_Prefix_View? That should be addressed as part of the semantic aspects (its not syntactically different than other selected notation).

Pascal will take an action item to move this SI to the semantic domain. There is no need to change the syntax analysis.

SI99-0009-1/02 Handle new aggregate features

What does the second Application Note mean? It is talking about the meaning when the aggregate is normalized. This is not an application note (which are generally examples of use); this is just ordinary semantics. The note should be rewritten as such.

Bill will fix this SI.

No vote was taken on this SI.

SI99-0011-1/02 Add A_Tagged_Incomplete_Type_Declaration to Declaration_Kinds

The title of the SI is wrong, and the last sentence of the question should say Definition_Kinds, and the discussion should not reference Sergey. Try to get some of the information from the minutes into the discussion.

Bill thinks there is a hole in the SI; he thinks that there might be an issue with discriminant_parts. No, we don't need a change there.

"Add support for tagged incomplete types" should be the title.

There are number of errors in the wording section: The name of the type defined in 3.9.9 should be Definition_Kinds. The SI says to modify Corresponding_Type_Kinds, but 15.8 is Type_Definition_View (and that appears to be intended by the June minutes). English wording like that for private types needs to be added to 15.8; we have to say this because all of the other cases are covered. " For an incomplete_type_declaration, returns the definition element representing the incomplete declaration view.". (Remember that in ASIS terminology a "view" is whatever follows the word **is** in a type declaration — entirely unrelated to the language's notion of view.)

Approve SI with changes: 9-0-2.

SI99-0012-1/03 Add support for null procedure declarations

The discussion shouldn't reference Sergey and should mention that this needs to be handled like abstract subprograms. The first paragraph of the discussion is true, but it is not very interesting. So the discussion should just say that all of the queries for procedures apply since we decided that this is a trait.

The last sentence of the wording should say "..a Nil_Element ...".

The summary should be rewritten to reflect the actual wording.

Approve SI with changes: 12-0-0.

Later in the meeting, we decided to make traits obsolescent; see the discussion of SI99-0017-1. We then reconsidered this SI.

This should define a Has_Null query. It should be defined for procedures and formal procedures.

Approve (revised) intent of SI: 11-0-0.

Bill will redo this SI.

SI99-0016-1/01 Correct Corresponding_Type_Operators

There is general agreement with the narrow question and solution posed by the SI.

The more general discussion probably should be put off until the semantic interfaces are defined.

There is some concern that the expanded body tree is “weird”, in that you couldn’t write it in Ada. There doesn’t seem to be any alternative to that; the semantic interface will have a similar problem.

Fix the subject to “Correct Corresponding_Body”.

Approve Intent of SI: 12-0-0.

SI99-0017-1/02 Trait_Kind is unclear for access definitions

Tucker points out that the **access** trait is obsolete. Randy points out that we can’t make an enumeration literal obsolete.

The topic and question should avoid talking about **access**.

Erhard notes that the comment for An_Ordinary_Trait should be replaced. Best to use something generic like “None of the following traits are present”.

The best way to handle the obsolete **access** trait is to add application notes to 3.9.5 and 13.10 to suggest that there is a better way to do this.

Someone suggests that perhaps the whole thing should be made obsolete. It could be replaced by the queries proposed in SI-0022.

This gets general agreement.

Tucker moved that we move everything having to do with Trait_Kinds to be obsolete, and replacement them with Has_xxx. Gary seconds. We vote on the motion: 11-0-1.

What to do with this SI? Randy points out that the wording is still broken, even if this is obsolete. Remove the new wording about “null” (as we’ll remove that trait as well).

The SI should mention that this is becoming obsolete.

Approve SI with changes: 12-0-0.

Randy points out that we’ve approved two SIs with traits this morning. We look back at those SIs.

SI-0012 should define a Has_Null query. It should be defined for procedures and formal procedures.

What about SI-0002? We should just merge it with SI-0022 (the routines will all be defined there), and delete SI-0002.

SI99-0018-1/01 Add new Attributes to Attribute_Kinds

Question should say “(Yes.)”. Recommendation should be (See summary.). Drop the first two sentences of the discussion. Add “But they are currently listed alphabetically, and any other order seemed ugly”.

Approve SI with changes: 12-0-0.

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

Fix the name of the function: “refence” is something you do when you are securing your border.

The discussion doesn't make sense: the prefix of a selected_component should be a name. It seems that he is saying that it should take more things, but similar queries don't allow that. Perhaps that's because they're syntactic. This is semantic: it doesn't belong in the syntactic interface. Perhaps this should be moved to the semantic domain.

Pascal will take this one and handle it in the semantic domain.

On Saturday evening, Tucker asks to discuss this issue further. He wants to make the implicit dereference an explicit node in the logical ASIS tree that is marked as implicit.

That seems to violate the explicit syntax only meta-rule for an ASIS tree. Tucker argues that all compilers make implicit dereferences explicit at some point. None of the implementers present object to that statement. But Randy comments that such nodes are added deep in semantic analysis, and would be hard to support as part of the syntax (which is long since discarded and is only loosely related). There does not appear to be much sympathy for Randy's position.

There is general agreement that making implicit dereferences explicit is a better model.

SI99-0020-1/01 Use "is null" defaults in Traverse_Element

Tucker wondered if we have **null** <>. No, we don't, because that would be two different defaults at the same time. The semantics would have to be that if there is a matching routine, it is used, and otherwise a null routine is used, but that would be awfully confusing, and in any case it is too late to add that now.

The problem here is that this ASIS change is not compatible with ASIS 99. Moreover, this is a silent change: working instances would still compile, but not do the right thing (replacing some subprogram's actions with nothing).

We could have a second generic with **is null** defaults, but that seems like overkill. The discussion says that the incompatibility is not an issue. We look at the arguments in the SI. Erhard does not believe the second point: doing nothing might not cause obvious trouble, just something will not happen. Perhaps some construct will not be checked for violations of a rule. Or memory might not be freed. It does not seem certain that the change in behavior would be detected.

The justification for this SI is the extra procedures that need to be written, but note that these are just a single line in Ada 2005: procedure Nothing (...) is null; So the clutter is substantially less than it was in Ada 95.

No action: 12-0-0.

SI99-0021-1/01 Obtain representation clauses based on defining identifiers

This appears to be in the semantic domain; it's the semantic entity (a type or subprogram) that is interesting, not the syntax. This function should really be returning an Aspect_Item_List. So this should also be postponed to integrate with the semantic interface. Probably this should take an entity.

Pascal will take this SI along with the others that need semantic integration.

SI99-0022-1/01 Add Boolean queries to ease use of Trait_Kinds

The functions need to be specific to each kind of thing: for example, Has_Limited should only allow constructs that might in fact have the keyword **limited**. The function descriptions should not refer to traits and need to be complete (able to stand on their own). New functions Has_Reverse, Has_Aliased, and Has_Synchronized need to be added. Add info about limited derived types to Has_Limited (we failed to do that elsewhere).

Add text to say Trait_Kinds and Trait_Kind are obsolescent.

Recategorize the SI as a binding interpretation.

Bill says that Has_Limited should work on limited interfaces just to be consistent.

Bibb asks about **protected** and **task** interfaces, types, and the like. Pascal says that we can't have protected type extensions, and protected types have their own hierarchy. Interfaces have their own "kind" (see SI99-0006-1, which was approved last time).

Bill will redo this SI.

Approve intent: 9-0-1.

SI99-0023-1/01 Uses of subtypes Name and Name_List

Add the missing status line so this shows up on the agenda.

Pascal says that changing the names of the parameters is too incompatible, and "Expression : Name" is ugly. Tucker suggests renames, but those are not going to resolve for positional notation (there would be two homographs). Moving the renames to child packages only works for people who have use clauses, and even then the context clause would need changing..

There are only 5 such routines. Erhard would like to fix these. Ed says that he would not commit to forcing people to change parameter names. Erhard suggests asking the ASIS user community.

Tucker says that Name_Image and Prefix are just too goofy and should be fixed. Bill will take an action to send such a query to the ASIS mailing list.

Changing the return types of functions is harmless.

Approve the intent of changing the return types: 10-0-2.

SI99-0024-1/01 Provide a semantic model in addition to existing syntactic model

After we wait for Tucker to return, he describes his proposal.

There is some discussion about his "parts". Tucker implies that it is possible to do a lookup in a particular context. Ed and Pascal think it would be very hard to make that work. Compilers are designed only to be able to do lookups in the current symbol table state, which varies as a program is compiled. Randy points out that it would certainly be beyond the original scope of ASIS as described in the ASIS Rationale. We agree that we don't want to try to do lookups at random locations.

Tucker moves to the specific queries that would be supported.

Ed wonders about Logical_Position_in_Region. That is the position in the declarative region; the idea is that it can be compared for ordering, but might be sparse.

If Has_Declaration is true, there is an explicit declaration and then you can get back to the syntactic domain (Entity_Declaration, Entity_Identifier). If it is implicit, we don't want to allow that (we don't want the non-determancy). Is_Overloadable is redundant (you can tell from the kind of entity).

Tucker wonders if Region_Parts are worth doing now, or whether they should be lower priority. It would be useful to know where implicit entities are declared (visible, private, etc.) This seems like the way to do that.

Child_Part handles children; Tuck notes that public children and private children probably need different parts. Steve Baird says that you usually don't need that; A.B is the name of a unit, not that B is something in A. The declarative region of a package should not include children.

A generic package has Current_Instance which gets the "normal" package entity inside of the formal part.

First_Subtype and the ones following them are not supposed be indented; they apply to all subtypes. Pascal notes that Is_Elementary is redundant with the Category, but it might be easier to use. Is_Access is also indented wrong, it should be the same as Is_Scalar.

Typed_View applies to anything that has a type. Ed wonders about slices; they appear to be missing.

Ed worries that defining these will be as hard as defining a programming language. Hopefully these will be the same as the existing programming language; we should depend on the definitions in the Ada Standard to define these routines.

It is unclear what Family_Index_Definition should be. Perhaps it should be a range? Or maybe it is syntactic (and thus doesn't belong here).

Conventions are annoying because implementations can create their own. So Other_Convention includes an identifier, so implementation-defined conventions can be queried.

Pascal asks about getting back and forth between the syntactic and semantic domains. That could be very messy, especially getting back to syntax from semantic interface.

Ed says other than the implementation cost, this is very attractive.

Tucker will create another version, with more detail. It would be good to write the definition of some of these so we can see them. Subtype_View, Typed_View, and Callable_View are probably the most important to work on first.

Implementers are requested to look at these and comment on the implementability.

Detailed Review of Ada 2005 regular AIs

AI05-0002-1/01 Unconstrained arrays and C interfacing

Tucker says that the SofCheck compiler allows unconstrained arrays in these contexts for C convention subprograms. Steve Baird says that it is still allowed to support unconstrained arrays in these contexts, it's just not required. Pascal notes that the IBM Rational compiler does not support these. Randy agrees; Janus/Ada doesn't support these either, pragma Convention/Import/Export is rejected for things that don't make sense.

Pascal suggests that we just say that you can't touch the bounds, like Unchecked_Union. Steve Baird replies that there isn't much that you can do without the bounds (you can't do assignment, for instance); it isn't very worthwhile to go through this exercise.

The rule is similar for access-to-unconstrained-array. This sets off a strong argument. Tucker says that this has to work, and A'Last is Max. Steve Baird doesn't buy that; if you do this all in Ada (calling from Ada code a C convention function defined in Ada that returns an unconstrained array), the bounds suddenly change from whatever is defined in the function to Max. That's very ugly.

Tucker comments that B.1(38.1) allows this sort of change when conventions are used. Several object to that interpretation. The discussion devolves into a discussion of the intent of B.1(38.1). Pascal says that it doesn't mean "anything goes"; Tucker claims that it does. His view is that the presence of any interfacing pragma whatsoever makes the program erroneous. Pascal says that this view was not his understanding of the intent of AI95-00320 (the AI that added B.1(38.1)) otherwise he would have voted against that AI.

Randy thinks that B.1(38.1) is irrelevant in this case; irrespective of whether the language *allows* an implementation to exhibit unfriendly behavior (and this is clearly unfriendly), *requiring* unfriendly behavior is something that the Standard should never do.

Tucker worries about the non-portability caused by not requiring support for returning unconstrained arrays. But then he notes that two compilers already don't support this (RRS, IBM), so we already have the non-portability. It would be better that the Standard makes the non-portability clear, rather than hiding it. So we return to looking at the AI's solution to the problem.

Pascal would like to drop is “or L is Ada”. This is unusual, none of the other cases have such a statement, and if we wanted such a statement, it should be at a higher level. Also drop “either”.

There is no “recommended level of support” for conventions, so this should not use those words.

Pascal is concerned that this allows convention C on unconstrained array. Steve Baird says that intended, so that we can have constrained subtypes of an unconstrained array. The rules require the array type to have the convention in order for the constrained subtypes to have the convention. Other rules prevent using that unconstrained array in bad ways.

Pascal suggests writing the new B.1(39) text as an implementation permission.

“An implementation need not support an interfacing pragma specifying a convention other than Ada or Intrinsic that:”

Tucker claims that B.1(41) actually covers this. “Presuming the language has corresponding features”. Steve says that this is not an implementer’s choice; there seems to be a difference of opinion what is a corresponding feature. For instance, it is unclear if C arrays (which carry no bounds information) “correspond” to Ada arrays (which do). For C, we have the correspondence specified.

Perhaps we simply need to say something about C, and not about other languages. B.3(70) should say “constrained array subtype”. Someone complains that in that case the example wouldn’t work. Steve Baird says that the restriction to constrained wasn’t intended to apply to parameters of Imported functions, dropping dope on the floor is easy.

Randy notes that the C mapping advice (B.3(63-71.1)) does not describe the mapping of function results at all. He suggests that some wording be added to cover what can be used as function results. Steve will add this to the AI; the B.1(39) wording will be dropped. The B.1(17) change will be kept (access-to-unconstrained requires dope that cannot be easily dropped). Perhaps some wording about Imported/Exported C objects is also needed.

Approve intent of AI: 4-0-7.

AI05-0007-1/02 Stream 'Read and private scalar types

Tucker explains the problem, and proposes allowing either the first subtype or the base subtype for elementary types.

Typo: “uses make use of” (drop “uses”), drop the editor’s note (we’re not fixing Ada 95 anymore).

Pascal points out that this rule has confused more than one user. Tucker says that includes him.

Steve Baird says that the model is that the attribute has the profile stated in the Standard, and it calls the user-defined one with a potentially different profile. Do we need to change some wording to define this model? Tucker doesn’t think so. Because the convention is Intrinsic, anything that requires subtype conformance must fail. So it isn’t possible to see the profile difference; it will only matter to calls through the attributes (which by definition are calling the user-defined routine).

Typo in question: “Presume that We”, lower case “W”.

Approve AI with changes: 9-0-1.

AI05-0008-1/03 General access values that might designate constrained objects

This defines “known to be constrained”, and was discussed extensively last time.

Tucker notes that the third bullet would be privacy breaking. No, because this depends on the view; it is not known to be constrained if the view is limited private.

We don’t need to include tagged in this definition because they must be either constrained or indefinite. (A tagged type cannot have defaulted discriminants.)

Use “explicitly limited record” in the third bullet, this is the official term for this.

Tucker says that this definition makes him nervous, because the term seems generally useful and thus might be used in the future, but not all cases are covered. At the very least, there should be an AARM note.

Tucker would like to include partial views as not known to be constrained (they might have hidden discriminants). That has no effect on the current rules, because they have no visible components which can be renamed or used as the prefix to 'Access, there is no incompatibility. Tucker will give us wording for this later.

The erroneous case should mention pass by reference, because there no problem if the actual is passed by copy. Tucker suggests dropping the plural. “i.e.” is bad, used “that is”. The paragraph now reads:

“If the nominal subtype of a formal parameter with discriminants is constrained, and the parameter is passed by reference, then the execution of the call is erroneous if the value of any discriminant of the actual is changed while the formal parameter exists (that is, before leaving the corresponding callable construct).”

Tucker returns to the bullets.

- its nominal subtype is constrained, and is not an untagged partial view;

He doesn’t want this one first. But Steve objects as this is the most common case. We want the most common cases first.

The third bullet describes the “really limited type” (something we never defined a term for, but we say this the same way in many places). We don’t need to add “tagged type” here, as previously discussed — they’re either constrained or indefinite and thus fall under other bullets. Also, the new first bullet includes “untagged”, tagged private views are covered by it.

Tucker would prefer to say “view of a composite object”; this is a property of the view. Pascal says that this gives the impression that this one is different; “view” is often implied. Steve Baird says we could put it in parenthesis. John says that this seems particularly non-obvious that we are talking about a view. Tucker notes that all of the surrounding text uses view. That carries the day.

“A view of a composite object...”

Approve AI with changes: 9-0-3.

[Editor's note: A problem discovered after the meeting will send this AI back for further consideration.]

AI05-0009-1/02 Confirming rep. clauses and independence

Bibb describes what he did. He did not think that independent addressability was defined very well, so he added Implementation Advice to describe that.

Pascal doesn’t understand the new Implementation Advice. He thinks that it should always be the case that the implementation can make objects large enough to be independently addressable. Bibb says that 9.10(1) is not true. Several people object to that statement; this is the definition of independent addressability — its true by definition!

Bibb explains that he is worried about two tasks accessing the same object. That isn’t what independent addressability is talking about (that’s what atomic is talking about, a totally different concept). It is talking about two different objects accessed at the same time by two different tasks.

There is no need for the new Implementation Advice; it should be dropped.

Pascal wonders if we should apply the new pragma Independent_Components to type String. Tucker says that forcing the representation of String to change now would be awful; some machines would have to change to full word size for each character in order to guarantee independent addressability of each character.

Why does the pragma include components? We wanted it to apply to record components last time. Pascal says that it would be best to model this after Atomic_Components. The added record component pragma should be separate: Independent_Components that would be given within the record type declaration.

Randy notes that “nonconfirming” still needs to added to 9.10(1). We don’t want a confirming representation item to have an effect. There should be no change to 13.1(21.1/2). We also need to add convention to 9.10(1):

“However, if a nonconfirming representation item is used to specify packing, record layout, Component_Size, or convention for a given composite object...”

The pragma should look like Atomic. “Independent” for objects and components; “Independent_Components” for records and arrays.

Should this go in Annex C? Tucker says it should. Randy objects, these pragmas should apply to all implementations while Annex C does not. Tucker argues that this is fairly low level; if you don’t have Annex C support, you don’t have low level concerns. You can only get problems if you use representation items (which are only usefully required by Annex C).

Should Independent include objects? Tucker says no, because there is no such object. (See the definition in 9.10(1)). Randy complains that it is possible to write address clauses in some implementations for objects that wouldn’t be independent. Tucker argues that that would be a user’s bug. 13.3(13) says that an programmer has to ensure that the address is valid... This isn’t the technical term “valid”, so it is vague enough to cover bad addresses. So, we don’t want Independent on stand-alone objects.

Tucker would like the new last sentence of 9.10(1) combined with the previous sentence, otherwise it not true.

Bibb wonders if “implementation-defined” in 9.10(1) should be “unspecified”; no one documents it. Pascal says this is impossible to document in any useful way, since it depends on the interactions between objects and representation items. We agree to change to “unspecified”.

Independent_Components could apply to any composite type. (It might make sense for a protected type, if games are being played.) Randy complains that this change and the one for objects plus the extra pragma for record components mean that the supposed similarity to Atomic doesn’t exist: “It should work just like Atomic except that it is completely different”. Everybody laughs but it seems that it needs to be true.

Approve intent of AI: 11-0-1.

AI05-0013-1/02 No_Nested_Finalization is difficult to enforce

In the recommendation, aggregates should be allocators. The !corrigendum has a stray “such objects”.

Pascal wonders if this is always statically known. This is a post-compilation check, like all restrictions.

Question: “must therefore been”.

Approve AI with changes: 11-0-1.

AI05-0015-1/02 Constant return objects

Pascal wonders if this change isn’t too large for the Corrigendum mode that we’re in. John says that it is OK, it seems like an oversight. Changing syntax is annoying, but it is better to do it now than wait.

3.3(21/2) should be 3.3(21).

Approve AI with changes: 11-1-0

Erhard voted against because he thinks this isn’t important enough; it’s mainly to suppress an error message in Gnat.

AI05-0016-1/01 Others => <> can be used in place of null record

Tucker doesn't like the wording, he suggests:

"The reserved words **null record** shall appear only if there are no components needed in a given record_component_association_list."

Approve AI with changes: 8-0-2.

AI05-0017-1/01 Freezing and incomplete types

After a couple of false starts on the discussion, we start from the beginning to derive the problem and a solution.

Freezing of an incomplete type (view) freezes the completion. This is not a problem for normal incomplete types (the completion is right there, and it shouldn't be used in contexts that require freezing), and limited withs are already frozen. But Taft Amendment types are frozen at the start of the body — and the completion cannot be available yet.

The first idea would be that freezing of incomplete types has no effect. But that doesn't work: Pascal gave an example that would cause trouble in his e-mail (it's attached to the AI).

We might be able to ignore freezing of Taft Amendment (TA) types, however. Steve Baird would prefer that. Pascal is concerned, so is Randy. Randy worries that if TA types don't freeze in the body, you could have a premature use that doesn't cause freezing. Indeed, it would seem easy to write an example similar to Pascal's using TA types: just move the object declaration to the body before the completion. The compiler would have to know the location of the tag before the type was frozen.

It is suggested that the TA types become normal with respect to freezing when you reach the **is** of the package body. The freezing points cannot occur anywhere else (because of the occurrence of a nested body, for instance).

Freezing a deferred incomplete view has no effect (it doesn't even freeze itself); once you reach the body it becomes a normal incomplete view.

Pascal will try to write wording to this effect.

Approve intent: 10-0-1.

AI05-0018-1/01 Formal package matching rules

Tucker proposed in e-mail that "tweaking" (rounding and the like) is applied to the values of static expressions when those value are assigned to objects. In a context where the expression value is used directly, tweaking is not applied.

Pascal does not like Tucker's model. He says that it makes it unclear when the tweaked value is used and when the untweaked value is used. He does not know how this solves the original problem. Tucker points to 12.7(8.1), which says that the formal is replaced by the actual expression.

Pascal says that this would be a big change for their implementation. It might be necessary to keep both values around in some contexts and be careful about what value to use.

Steve Baird thinks that this example ought to work, but differs with Tucker on how to accomplish that. He would prefer to tweak (bad pun) the matching rules.

Tucker points out that the rules making something hard to implement is not a good reason for changing the rules.

Do we agree that this example should work? Everyone agrees on this point; having 3.14 /= 3.14 is just too weird and unfriendly.

Erhard says that sounds like Steve wants to make instances use machine arithmetic rather than static arithmetic. Pascal says that this is still static infinite precision arithmetic, but it might be the machine value that is being compared as opposed to the original value.

Does fixed point have this problem? Perhaps; no one is sure.

Steve Baird will create a proposal.

AI05-0019-1/01 Primitive subprograms are frozen with a tagged type

We discuss the problem. The solution seems to be to partially freeze the subprogram when the type is frozen. That is, freeze the subprogram but not the parameters. Randy suggests that we might separate freezing the subprogram and freezing the profile. (The profile would be frozen by calls, etc.) 13.14(14) says that the callable entity freezes its profile; we'd need to break that. He was only half serious, but Tucker and Pascal seemed to think it was the solution instantly. Sometimes you look like a genius without having a clue...

The profiles would have to be frozen at the end of the unit (in the normal way).

What bad would happen if we did that? No one has any real problems. We would need new rules for when the profile is frozen.

Ed comments that this would be a new complication for the freezing rules.

Tucker suddenly announces that we don't need any rules to freeze the profile: he thinks the types will be frozen by a call or instantiation or default expression.

Pascal says that he prefers this idea to his earlier proposal.

Ed would like to run an experiment to see if disabling that rule causes much of anything to break. Pascal says that until very recently they did not actually implement 13.14(14).

John notes the example in the discussion is wrong. The full types should be:

```
      type T1 is new Parent with null record;
      type T2 is tagged null record;
```

Erhard asks that the question be changed to "Is this incompatibility intended?".

Pascal will take this AI.

AI05-0020-1/01 Operators of partial views of fixed point and access types

The paragraph in the question should be 4.5.2(9.1-9.4/2).

The curly brackets to describe an insertion are missing from the first wording.

There is a typo in the question: "...has [to] the effect of..."

Fix Pascal's name in the first appendix item.

Subject has "Partial" capitalized for no reason, drop "...are considered".

Steve Baird asks if this could happen for an incomplete type. He says that you could declare an access primitive of an untagged type.

```
   function "=" (Left, Right : access Incomp) return Boolean;
```

So change 4.5.2(9.3/2) to say "partial or incomplete view".

We don't need to do that for fixed point, because you can't declare "*" (direct use of an incomplete type is prohibited).

Add some discussion to this effect.

Subject should be “Universal operators of fixed point and access types”.

Replace the summary with the following.

“Operators of partial views of fixed point types are considered when determining if the predefined universal_fixed operators can be used. Similarly, if equality is defined for an anonymous access type with a designated type of a partial or incomplete view, then that equality operator is considered when determining if the predefined universal_access operator can be used.”

Approve AI with changes: 9-0-2.

AI05-0021-1/01 Issues with containers

Typo in (3) of summary: “raise{s}[d]”.

(3) in the discussion section is empty, add a couple of sentences there.

Approve AI with changes: 10-0-1.

AI05-0022-1/01 Container tampering should be checked for formal subprograms

The question says “finalized”, but that confuses people. The problem is broader than that. It would be better to say “delete an element from”; that’s really the classic example of tampering.

Tucker will take this AI and provide wording for this.

Approve intent of AI: 9-0-2.

AI05-0023-1/01 'Read on records with variant parts

Someone claims that 13.13.2(27/2) discusses this. No, that’s 'Input, the question is about 'Read.

Pascal says it would be simpler to say that an anonymous object is created for Read and that is assigned to the out parameter. Then the rules are obvious. Otherwise, we have a case where discriminants are just changed, and that way lies madness.

Randy objects that this model doesn’t work for limited types. That doesn’t matter; this case can happen only for a nonlimited type. That’s because tagged types cannot have defaults, and the attributes of limited untagged types aren’t available unless the attribute is user-defined. So we don’t have to worry about the default behavior of 'Read for limited types with defaulted discriminants, which are the only kind that can possibly change the discriminants.

Does this anonymous object have default initialization? That would be visible to the user if there are any controlled components or default expressions that call functions. But you can’t pass garbage to a user-defined 'Read, which might be needed to handle a component. You would be allowed to do the normal optimization to remove pairs of Initialize/Finalize.

Erhard wonders about 13.13.2(55); why does it apply to discriminants only? We can’t allow trashing of discriminants, but it is too expensive to prevent trashing of the rest of it. It seems that implementing that rule requires some sort of temporary object (it’s not allowed to directly read discriminants into the object that was passed to Read).

Tucker would like the anonymous object to be created only if it has to be (that is, only when the object is unconstrained with defaults). Steve Baird suggests an implementation permission to allow that. We don’t want to rules to be too complex. On the other hand, Tucker worries that just adding defaults might cause the behavior to change (because of side-effect of initialization). A permission still seems like the best option.

7.6.1(21) is the similar permission for assignment.

Steve Baird will create wording for this AI to describe the anonymous object model and for the permission to omit this object.

Predefined Input calling predefined Read might create two temporaries: we might consider an extra permission to allow eliminating one of these temporaries.

Approve intent: 8-0-3.

AI05-0024-1/01 Run time accessibility checks

Pascal explains the problem as it being unclear what programs should fail static checks, and which programs check at runtime. He says that the rules seem to require too much analysis.

Pascal says that the first example (question 1) came from writing static checks. He doesn't think the rules are clear here. Tucker and Steve both object: there is nothing in this example that could possibly cause the allocator to be rejected — there is only one type.

Tucker thinks that there is a terminology problem. "not deeper than" implies a full ordering, which we don't have. It really ought to be "not deeper than and not incomparable". Yikes! "not deeper than and comparable" is better. Tucker suggests that the wording should go back to masters: "this guy's master is the master directly or indirectly of the other guy".

Steve Baird suggests that the "immediate master of one is a master of the other". Or "every master of A is a master of B". Randy worries that this change will make the accessibility wording even more impenetrable to users. Tucker replies that he doesn't think that is possible, getting a general laugh.

Change 4.8(10.1/2) to:

"For any allocator, if the designated type of the type of the allocator is class-wide, then a check is made that every master of the type determined by the subtype_indication, or by the tag of the value of the qualified_expression, is also a master of the type of the allocator."

It would be better to use "innermost" master:

"For any allocator, if the designated type of the type of the allocator is class-wide, then a check is made that the innermost master of the type determined by the subtype_indication, or by the tag of the value of the qualified_expression, is also a master of the type of the allocator."

We check Gary's example (as given in e-mail): the revised rule works there as well.

Pascal notes that there are many occurrences of *deeper than* in the Standard, it seems like any of them could be broken in this way. Steve Baird comments that we certainly need to change return checks [6.5(8/2).].

Tucker is worried that we don't define *a master* of something. The wording seems to define only *the master* of an entity. Pascal wonders why we say "innermost master" if there is only one master. Tucker says this really means "innermost master of the elaboration of the declaration of the entity". Anyway, change the wording of 4.8(10.1/2) to:

"For any allocator, if the designated type of the type of the allocator is class-wide, then a check is made that the master of the type determined by the subtype_indication, or by the tag of the value of the qualified_expression, includes the elaboration of the type of the allocator."

Change 6.5(8/2) to:

"A check is made that the master of the type identified by the tag of the result includes the elaboration of the function body. If this check fails, Program_Error is raised."

Pascal says that there are 38 occurrences of *deeper than* in the Standard (and more in the AARM). Do we need to check all of these? Yes, we should. We begin a systematic look at every use of *deeper than*.

We start with the definition of *deeper than* in 3.10.2(3/2). The text is confusing: “An accessibility level is *deeper than* another if it is more deeply nested at run time.” Replace that sentence with “An accessibility level is *deeper than* the accessibility level of another master if it is more deeply nested at run time.” This needs to be a partial ordering; is that clear? Erhard says that this wording is clear that this is a partial ordering; maybe an AARM note to say that explicitly would be useful.

3.10.2(3-4/2) are marked as redundant, but they are defining terms. Only the part after “For example” in paragraph 3 is redundant. Paragraph 4 does not actually define “statically deeper”, paragraph 17 does that; this occurrence should not have been in italics. That means that paragraph 4 really is redundant.

Turning to uses of *deeper than*, we look at 3.10.2(29/2). We don’t have the comparability problem unless there is an anonymous access type. We don’t have access parameters on entries, so you can’t cross tasks this way. So we don’t have the same problem. Tucker suggests that we add an AARM note that we believe that because there are no access parameters of entries, the accessibility levels are always comparable. That way, if we ever try to eliminate that restriction, we’ll at least know that we need to change this bullet.

The AARM note following 4.6(39.1/2) mentions access parameters, but those can never be an array (they’re access types, silly), so that part of the note should be dropped. The check is equivalent to X.**all**'Access, so there should not be a problem here.

We covered the first sentence of 4.8(10.1/2), but we need to look at the second sentence. The accessibility level of a discriminant is either that of the value of the discriminants or that of the object (the allocator). The value of the discriminants is essentially equivalent to 'Access; that of the object is that being allocated and that cannot fail. So there is no issue here with the second sentence.

Steve Baird wonders if there is any effect from Unchecked_Access. Tucker doesn’t think there is.

6.5(22/2) is the same as 4.8(10.1/2) so we think it is OK.

9.5.4(6) is unusual: there is only a static check. Everywhere else there is a pairing of a static and dynamic check: Why is there no dynamic check? Tucker claims that this is only to prevent requeuing on nested stuff. It’s always possible to tell if something is nested inside of yourself.

13.13.2(31/2) is about streaming. Subtype S is the root of the tree; every extension has to be comparable to that of S. (They might not be comparable to each other, but we aren’t trying to do that.) Steve Baird worries that the accessibility level of S is what is used, and S could be declared at a more nested level than the root type T. It’s the root type’s accessibility that we’re interested in. So we have a different bug; this should say “T” rather than “S”. Otherwise this is OK. But we could have said “...different than T” to be more like 3.9(12.1/2) — this is really an equality check. Tucker says that we never say *different* in Standard wording. So we should say “...not the same as T”.

We’ve reached the end of all of the interesting uses of *deeper than*.

We also need a fix for question 2A. 3.10.2(14.1/2) probably should say “or in a subtype_indication of a derived_type_definition or private_extension_declaration...”.

We look at question 2B. This is an amazingly stupid example: you could never assign anything other than a null to the object. On the other hand, we can’t leave it undefined; it has to mean *something*. We certainly don’t want to add this case to 3.10.2(14.3) as that defines a coextension.

That means this stupid example needs its own bullet:

- for an allocator used to define the constraint for an access subtype in an object_declaration, the level of the object_declaration;

Steve Baird notes that you could also declare a useless type like this useless object. Tucker says that we need to look at all subtype_indications and make sure they’re covered. We decide to let the AI author do that.

Tucker will update the AI.

Approve intent: 10-0-0.

AI05-0025-1/01 Missing rules in 12.7

Randy explains that the syntax change caused the rules in 12.3(9) and 12.3(10) not to apply to <> associations.

Pascal suggests adding "generic_association or formal_package_association" in 12.3(9). The other approach is just to copy these two rules into 12.7 with appropriate replacements.

12.7(4.1) seems to be a copy of 12.3(10), so we don't need to copy that.

So we should copy 12.3(9) in front of 12.7(4.1), replacing generic_association with formal_package_association and changing "generic unit being instantiated" to "template".

"The *generic_formal_parameter*_selector_name of a formal_package_association shall denote a generic_formal_parameter_declaration of the template. If two or more formal subprograms of the template have the same defining name, then named associations are not allowed for the corresponding actuals."

The subject should be "Missing legality rules for formal_package_association".

Approve AI with changes: 10-0-2.

AI05-0026-1/01 Missing rules for Unchecked_Union

Question (1): B.3.3(10/2) is privacy breaking. Pascal suggests assuming the worst here.

Tucker says that this is a representation item; they are always privacy breaking. Pascal doesn't want more rules that break privacy.

Steve Baird suggests using "needs finalization" for this, as we have this predicate.

Randy worries that we decided that predicate wouldn't work for a coextension. Tucker says that a component cannot have a coextension.

Steve suggests that the discriminants could come from the top level. We laugh about the notion of an Unchecked Coextension.

So we'll change the terminology to "needs finalization" in B.3.3(10/2). Steve wonders how this applies for generics, we would need an assume-the-worst rule for bodies (it would be rechecked in the specification).

Question (2): The rule that says that it is illegal to reference a discriminant of an unchecked union in a component_clause needs to be added. What about confirming rep. clauses? Explicitly saying that it has zero bits would be weird. That rule would be added after B.3.3(12).

Question (3): Pascal decides that here he was confused; this question should be deleted.

Steve Baird will take this AI and create the needed wording.

AI05-0027-1/01 Finalized container objects

Tucker worries that this requires an extra check on each operation. Pascal says that it can be combined with the tampering check (Matt also suggested that in e-mail.). Tucker thinks such a tampering check would not need to be done on the operations, he thinks there is some way to do it in Iterate and similar routines. Pascal and Randy don't believe this; they think that such a delay would cause the container to be corrupted before the check was made.

But the location of the check doesn't matter: we cannot allow inserting into a finalized container. Otherwise we'll have storage leaks or worse. Tucker changes his focus to note that the read-only operations don't need a tampering check. He thinks that a finalized container can read as if it is empty. Randy comments that we don't want to require implementations to allow reading of finalized containers — that could be very complex in some cases. We should be able to raise an exception.

That implies that it should be a bounded error to access a finalized container, either Program_Error is raised or the container acts as if it is empty. However, if the operation tampers with the container or elements, it is not a bounded error, it just raises Program_Error.

Pascal doesn't understand how come there isn't an explicit check anyway. Consider a vector: the pointer at the array is **null** and that requires an explicit check. Tucker admits that is true, but his vector container initializes to an empty one with the pointer at the array being **null**. He doesn't want extra overhead to be able to distinguish between a newly initialized empty container and a finalized one. That doesn't seem worth the effort, as it is very hard to create a program which accesses a finalized container. It would require access from a Finalize routine declared in the same master as the container object; this is unlikely to occur in practice.

Tucker will take this AI to write the wording.

Approve intent of AI: 5-2-5.

Pascal votes against approval because he wants Program_Error to be raised all the time. John agrees with Pascal.

AI05-0028-1/01 Problems with preelaboration

The discussion is from some other AI; ignore it.

Question (1) 10.2.1(10.1/2) should apply only if pragma Preelaborable_Initialization does not apply to the formal type.

Steve Baird finds it weird that a generic specification is not checked for problems, but that follows from rules. You get a second chance with the specification (rechecking the instance), but you don't get a second chance with the body.

Question (2) 10.2.1(10.1/2) should mention formal untagged derived types, because they can have different discriminants and thus the formal says nothing at all about the properties of the actual. This is only necessary if the formal type has discriminants; Tucker says that this would be a compatibility issue if we covered all formal derived types. There is no problem with new Integer. Steve Baird suggests "discriminated formal derived type". In fact, it only matters if the type has defaulted discriminants; you can't evaluate the defaults in any other case (even if they are there). And you can always work around any incompatibility with pragma Preelaborable_Initialization.

Question (3) Why does 10.2.1(10.2/2) not detect this case? It already says assume nonstatic, so that seems to cover the case where the actual is nonstatic. Pascal cannot remember what he was thinking. There is no problem here.

But Steve Baird won't leave the topic alone. He thinks that 10.2.1(10.2) and 10.2.1(10.3) might not be needed, because you couldn't actually write an instance that actually passed a nonstatic subtype or object. Pascal says the items in question are nonstatic anyway (by definition in 4.9), and removing these rules would have no effect. You could change the rules to say assume static, but that is a change from Ada 95. Tucker does not agree with that, a preelaborable generic doesn't have to be instantiated as a preelaborable unit. The language would need to say "an instance in a preelaborable context." This seems to be insufficiently broken.

Question (4) 10.2.1(11.4/2) should allow overriding with null procedures. We can drop "user-defined" then and just say "a controlled type with an Initialize procedure that is not a null procedure".

Question (5) 10.2.1(11.4/2) is privacy breaking. You should be able to apply pragma Preelaborable_Initialization to type T.

But Tucker complains that this solution is incompatible. We would have to make any extension of Controlled not have preelaborable initialization. You'd have to give pragma Preelaborable_Initialization on every controlled type. But this is fixing a hole in Ada 95, it was allowed there because it was a bug.

Allow the pragma Preelaborable_Initialization on any composite type. Any controlled type that appears in a package specification would have to not have preelaborable initialization. We wouldn't want to depend on the presence or absence of a private part.

Question (6) This would have weird effects. For the untagged case, it is perfectly possible to have a parent without preelaborable initialization and a child type with preelaborable initialization. Arguably, 10.2.1(11.4) should allow preelaborable initialization even when the parent does not have it. But the parent would need to be able to have pragma Preelaborable_Initialization applied.

We could do this with another kind of type that's between not having preelaborable initialization and having preelaborable initialization. But that seems like overkill for this relatively unlikely case. We joke about types having "potentially preelaborable initialization".

We decide this question is bogus.

Pascal will update this AI.

Approve intent: 8-0-2.

AI05-0029-1/01 Meaning of 12.5(8)

Pascal describes how the "=" for the formal type "emerges". Note that the generic is correct by a corrigendum resolution. You can't make the call to "=" illegal in the instance, the call could be in the body. You can't repeal the reference to 7.3.1 in 12.5(8), that would be way too incompatible.

So it appears that there is no way to make this case illegal. But Pascal would like a consistent model. Steve Baird suggests that there is a phantom "=" operator that always exists, but it becomes visible in certain circumstances.

Tucker suggests adding an AARM note after 7.3.1(5) saying that there are cases where an operator is never declared but still have be used ("re-emerge") in a generic unit.

This could also happen for derived types: if the generic has more visibility on the ancestor than the derived type, the generic might have inherited operations that wouldn't be visibly inherited by the derived type. We agree this must work.

Pascal wants some normative text, because he doesn't believe the meaning of the instantiation is clear, and he doesn't want to depend on the Dewar rule. Tucker thinks that there may be nothing wrong.

There is a declaration of "=" in the instance (after **private**). But where does the matching one come from?

Tucker suggests adding "... even if this operator has been overridden for the actual type {or even if the operator is never declared}" to 12.5(8). 12.5.1(21) also should get this extra text. But Gary and Pascal object to this, because the operator never exists. Tucker says that "never declared" is different than "never exists". That seems like hairsplitting — but this seems to be a case where hairsplitting is justified. Pascal would like something added to 7.3.1 to say that they exist in the twilight zone but are never declared. Then those existing operations can be used to match the operations declared in the generic as described in 12.5.

That seems like the best approach. Pascal will attempt to write this wording.

Detailed Review of Ada 2005 amendment AIs

AI05-0010-1/01 Revoking 11.6 permissions

It is suggested that the title be "Suppressing 11.6 permissions". Pascal doesn't want to discuss this one further at this time.

AI05-0030-1/00 Requeue on synchronized interfaces

[Editor's note: the AI number was assigned after the meeting.]

Alan had proposed that we allow requeues on procedures of synchronized interfaces (essentially dispatching requeue). Pascal wants to know what people think about this. Randy had pointed out that we had made a conscious decision to not support this (because what a requeue on something that is not an entry should do is unclear).

Alan had suggested an exception if you were to requeue on a real procedure. Pascal says that seems rather inconsistent with the rest of the handling of entry calls on real procedures.

Alan never sent an example to the list, but for some reason Tucker has it. Tucker mumbles about the example, but since he is the only one who has seen it the discussion doesn't go very far.

Pascal says that raising an exception for a requeue to a procedure would be fragile. If the entry was replaced by a procedure that called the entry, the requeue would work dramatically differently. Steve Baird says that already happens for a timed entry call.

Erhard would not want to allow a requeue to a procedure. That prevents using a synchronized interface; we have no way to force a procedure to be implemented by an entry.

Replacing requeue with a procedure call would be weird: especially if the requeue is non-blocking (without **with abort**). Blocking when you say no blocking is nasty. Steve Baird again says that timed calls have similar problems. There is some concern about the semantics for conditional and timed entry calls — perhaps those too shouldn't allow procedures. We might have made a mistake in allowing them in the Amendment.

Tucker suggests a pragma specifying that a primitive procedure of a synchronized interface (or maybe even a formal subprogram) is an entry; that would be a restriction checked by the compiler. Then we could raise an exception if a non-entry is used in an entry context like requeue (you'd have a way to eliminate the risk of an exception).

John suggests sending it to the International Real-Time Ada Workshop. He also would like to wait for more experience with the use of synchronized interfaces — it's hard to tell how they'll be used.

AI05-0031-1/00 Add a From parameter to Find_Token

[Editor's note: the AI number was assigned after the meeting.]

The proposal was to add a From parameter variant similar to the Index ones.

Randy worries about the incompatibility — he says that it isn't too likely, but this is very similar to allowing Ada 95 implementations to add Index with From. And we didn't allow that in the end.

Pascal says that Randy's concern is minor. But he worries about the portability issue; some compilers would have it, and others would not. The standard surely won't have it until the next Corrigendum is issued, so neither implementers nor users would know about the added routines.

Silence fills the room. No one feels strongly enough about this to speak in favor of it.

Tucker says that it would be best to tell the implementer to create a child unit, and not change the language-defined units.

Erhard wonders if we should give some guidance as to what such a child unit would look like. Informal work seems pointless here — it would have the same problems as an adopted change, only with even less visibility.

AI05-0032-1/00 Extended return statements for class-wide functions

[Editor's note: the AI number was assigned after the meeting.]

Tucker ran into a problem when writing a presentation for the SIGAda conference

He imagined device drivers that register a constructor function when they are loaded. The constructor function returned Device'Class. Device is a limited type.

Consider how you would use an extended return to write the SCSI device constructor:

```
return Obj : <specific type> do -- Not legal
```

This isn't legal because we require the type here to be the same as that of the function [defined in 6.5(5.2/2)]. Instead, you have to write:

```
return Obj : Device'Class := Func do
```

where Func is a dispatching call.

Tucker thinks that the first case really should be legal. Randy finds the example mildly bogus; a constructor should be a dispatching function, not something returning a classwide type. He thinks you could write a factory this way, but the actual constructs should still be dispatching routines. Others remind him that arguments from usage patterns are weak. Tucker argues that this is somewhat similar to the unconstrained case, where we allow giving constraints. That does seem reasonable.

Tucker will create an AI proposal.