Minutes of the 50th ARG Meeting

15-17 November 2013

Pittsburgh, Pennsylvania, USA

Attendees: Steve Baird, John Barnes (except Friday), Geert Bosch, Randy Brukardt, Jeff Cousins, Gary Dismukes, Bob Duff, Brad Moore, Erhard Ploedereder (except the second half on Sunday), Ed Schonberg (except the last half-hour Sunday), Tucker Taft.

Observers: Bill Duff (Saturday afternoon) [Bill is a student at the nearby CMU, and is Bob's son], Steve Michell (except Sunday).

Meeting Summary

The meeting convened on Friday, 15 November 2013 at 13:40 hours and adjourned at 12:15 hours on Sunday, 17 November 2013. The meeting was held in the Boardroom at the Wyndham University Center in the Oakland neighborhood of Pittsburgh, Pennsylvania. The meeting covered all of the normal AIs and some amendment AIs on the agenda.

AI Summary

The following AIs were approved:

AI12-0081-1/01 Real-time aspects need to specify when they are evaluated (9-0-1)
AI12-0084-1/01 Box expressions in array aggregates (9-0-1)
AI12-0093-1/02 Iterator with indefinite cursor (11-0-0)

The following AIs were approved with editorial changes:

AI12-0031-1/03 All_Calls_Remote and non-direct calls (8-0-3)
AI12-0036-1/01 The actual for an untagged formal derived type cannot be tagged (9-0-2)
AI12-0042-1/07 Type invariants inherited by nonprivate extensions (6-0-2)
AI12-0052-1/02 Implicit objects are considered overlapping (7-0-4)
AI12-0059-1/01 Object_Size attribute (5-0-2)
AI12-0065-1/01 Descendants of incomplete types (6-0-5)
AI12-0071-1/04 Order of evaluation when multiple predicates apply (11-0-0)
AI12-0082-1/01 Definition of “dispatching domain” (8-0-2)
AI12-0085-1/01 Missing aspect cases for Remote_Types (10-0-0)
AI12-0088-1/02 UTF_Encoding.Conversions and overlong characters on input (5-0-5)
AI12-0089-1/01 A generic function is not a function (10-0-1)

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

AI12-0003-1/04 Default storage pool for storage pools (6-1-0)
AI12-0061-1/02 Index parameters in array aggregates (9-0-2)
AI12-0086-1/01 Aggregates and variant parts (7-1-0)
AI12-0094-1/00 access_to_subprogram_definition should be a declarative region (10-0-0)

The following AIs were discussed and assigned to an editor:

AI12-0038-1/03 Shared_Passive package restrictions
AI12-0055-1/03 All properties of a profile are defined by pragmas
AI12-0074-1/03 View conversions and scalar out parameters passed by copy
AI12-0090-1/02 Pre- and postconditions and requeues
AI12-0091-1/01 Add procedure Sin_Cos to Ada.Numerics.Generic_Elementary_Functions
AI12-0092-1/00 Soft Errors

The following AI was discussed and placed on hold status:

AI12-0083-1/01 Automatic creation of constructor functions (8-0-0)

The following AI was discussed and placed into promising status:

AI12-0041-1/01 Type_Invariant'Class for interface types (11-0-0)

Detailed Minutes

Previous Meeting Minutes

Approve minutes. 9-0-1. [Editor's note: In posting these final minutes, I noticed that the year was wrong (2012 instead of 2013) in the meeting summary.]

Date and Venue of the Next Meeting

The next meeting will follow the Ada-Europe conference in Paris. That will be the afternoon of June 27th (WG 9 has Friday morning as usual), all day on June 28th, and the morning of June 29th . The venue is TBD, as the conference venue may not be available on the weekend.

Thanks

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

Thanks to HILT and SIGAda for the excellent arrangements.

Thanks to the Rapporteur (Jeff Cousins) for his efforts to lead us.

ARG Procedures

We made a number of primarily editorial changes, listed in the e-mail of November 12th. There were enough such changes that it seemed reasonable to re-approve these procedures.

Erhard comments that the second sentence under Promising and Hold is overspecification. We could use these statuses even during a revision for things that we don't want in the current revision but might want to reconsider. Or for AIs that we don't know what to do with (we've had a few of these in the past). So delete the two sentences (“This status is used...”).

Approve procedures with changes: 8-0-2.

ASIS

WG 9 has put the work on ASIS on indefinite hold. The main reason for creating a standard is to encourage portability between implementations. But there is little evidence of demand to port ASIS applications; most are constructed for a single implementation and never moved. We also need more implementer input into any Standard update that we might propose, but today they aren't very interested in the formal standard preferring to add facilities as they need them informally.

As such, WG 9 believes that we need to wait for user demand for more portability and a standard. There isn't enough user demand today to encourage the implementers to follow a Standard even if we constructed it.

Parallel Processing

Steve Michell gives us a WG 9 recap. Canada had proposed a formal parallelism study group for SC 22, but that did not go anywhere. Various languages have proposals. Specifically, OpenMP – which has various language interfaces; CILK. C wants to put parallelism into the language (CPLEX).

Canada is concerned that these approaches will interfere with each other, making multilingual programming very difficult. SC 22 didn't agree.

The subcommittee is working on proposals for Ada. They think that just having work distribution mechanisms isn't enough, we also need mechanisms that real-time people can use.

Steve Michell suggests that we would use aspects to specify parallelism. That would require extending aspects to work on statements. [Editor's note: Since aspects are properties of entities, and a statement is not an entity, that would be quite an earthquake. Probably it would be best to use a similar syntax to specify these things, but avoid calling them aspects.]

Brad has been working on parallel libraries for Ada (known as Paraffin). These are available on Soureforge (http://paraffin.sourceforge.net/).

Tucker notes that the first step probably should be an API for OpenMP or the like.

Brad Moore, Steve Michell, Miguel Pinho, and Tucker Taft is the subcommittee.

Ed notes this is two tasks: one is features for Ada; the other is outreach with other languages. Tucker doesn't think we want to separate those much.

Jeff asks that links to the papers and Brad's SourceForge be passed around to the ARG.

Steve Michell notes that machines are moving beyond cache-coherent systems to “networks on a chip”. The subcommittee is trying to look forward to future architectures as well as current ones.

Tucker notes that it would be nice to support atomic operations (possibly defined in terms of protected operations), like compare-and-swap. Geert notes that GNAT compiles appropriate protected objects into that.

Old Action Items

Steve Baird says that the paper for AI12-0016-1 is going to be done “someday”. He also hasn't recently worked on AI12-0020-1 (an Amendment), he doesn't think there is a rush.

Brad Moore says that he was too busy to work on AI12-0031-1 (and he forgot that he needed to do so). He says that he'll try to finish it tonight (which he did, see AI12-0031-1 discussion below).

Ed Schonberg has no explanation for not working on AI12-0002-1 (he explained it as “a dog ate my homework”!). Van Snyder isn't available right now, so that was his excuse for not working on AI12-0058-1 (also with Tucker Taft).

Tucker Taft says that he was overwhelmed with work related to being the Program Chair for the HILT conference. It was such a burden that he signed up to do it again next year! Anyway, he updated AI12-0071-1 last night (Thursday), so we can discuss that during the meeting even though it was left off of the agenda since it wasn't done by the deadline. He says that he will try to do AI12-0068-1 tonight (Friday) -- but he worked on meeting assignments rather than that AI on Friday and Saturday nights.

Randy notes that a number of people didn't do their ACATS test homework by the deadline of September 30. Those can be added to the Ada 2012 ACATS even after its initial release, so those items will remain on the action item list.

Current Action Items

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

Steve Baird:

- AI12-0016-1
- AI12-0020-1
- AI12-0061-1
- AI12-0074-1
- AI12-0090-1

John Barnes:

- 4 ACATS tests (see separate assignments).

Geert Bosch:

- AI12-0055-1 (split into two AIs, see discussion)

Randy Brukardt:

- AI12-0059-1
- 4 ACATS tests (see separate assignments).

Editorial changes only:

- AI12-0031-1
- AI12-0036-1
- AI12-0042-1
- AI12-0052-1
- AI12-0065-1
- AI12-0071-1
- AI12-0082-1
- AI12-0085-1
- AI12-0088-1
- AI12-0089-1
- AI12-0093-1

Alan Burns:

- 4 ACATS tests (see separate assignments).

Jeff Cousins:

- AI12-0091-1

Bob Duff:

- AI12-0092-1
- AI12-0094-1
- 3 ACATS tests (see separate assignments).

Steve Michell:

- Work on solutions for parallel programming for Ada (with Brad Moore and Tucker Taft).

Brad Moore:

- AI12-0003-1
- Work on solutions for parallel programming for Ada (with Steve Michell and Tucker Taft).

Erhard Ploedereder:

- AI to fix A(3) for parameters of access types (such as the stream operations), see discussion of AI12-0052-1.
- 4 ACATS tests (see separate assignments).

Ed Schonberg:

- AI12-0002-1
- AI12-0058-1 (with Van Snyder and Tucker Taft)
- 4 ACATS tests (see separate assignments).

Van Snyder:

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

Tucker Taft:

- AI12-0038-1
- AI12-0058-1 (with Ed Schonberg and Van Snyder)
- AI12-0068-1
- Create an AI to describe what happens for class-wide preconditions and postconditions if the parameter names are changed. (See AI12-0090-1/01).
- 2 ACATS tests (see separate assignments).
- Work on solutions for parallel programming for Ada (with Brad Moore and Steve Michell).

Tullio Vardanega:

- 2 ACATS tests (see separate assignments).

Detailed Review

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

If a paragraph number is identified as coming from the working Ada 202x AARM, the number refers to the text in draft 2 of the Ada 202x AARM. Paragraph numbers in other drafts may vary. Other paragraph numbers come from the final Ada 2012 AARM; again the paragraph numbers in the many drafts may vary.

Detailed Review of Ada 2012 AIs

AI12-0003-1/04 Default storage pool for storage pools

“The immediate scope of a pragma {when not used}[which is not used] as a configuration pragma ...”

Bob would prefer to give a name to the standard storage pool.

That doesn't seem any simpler than this; we'd need a package and come up with a name for both the package and pool.

Randy suggests that we could use an "identifier specific to a pragma" in pragma Default_Storage_Pool, rather than defining a new, similar pragma. Randy notes that **null** already is playing this role in the pragma, we just need another such name. No other reserved word seems appropriate. Tucker suggests "Standard". This is a third way to do this (the way it is written up is one, Bob's standard pool object is another, this is a third).

Tucker suggests using Standard (as in package Standard) so that it could not possibly conflict with a user-defined pool name. It seems weird to allow a package name in this pragma, but probably not any weirder than an identifier specific to a pragma.

Bob says this is just a less capable way to doing what he is suggesting. That's intentional; Steve notes that the default storage pool is a strategy, not necessarily a pool object.

Straw poll: As written - 0; define a standard storage pool somewhere - 1; use Standard (Tucker suggests that it has to denote package Standard) - 6.

We agree that this needs to be changed to use a special name (either as Tucker suggested, or just as an identifier specific to a pragma).

Brad will take this one.

Approve intent of AI: 6-1-0. Bob is still opposed as he thinks a real object would be better.

Tucker notes in passing that a standard storage pool object could not be defined in a pure or preelaborated package as it is a variable. That would restrict its usage (especially in preelaborated packages). The special name approach does not have that problem.

[Editor's note: This is not quite right. I think I recorded Tucker's comment correctly, as we talked specifically about how that's not much of a limitation for pure packages (which can't declare non-local access types anyway). As such, it's only interesting for preelaborable packages. However, a preelaborabable package can have a variable so long as the type of that variable has preelaborable initialization. Since Root_Storage_Pool does have that kind of initialization, it's possible to have a storage pool variable in a preelaborated package. But I don't think we would want to insist that the standard storage pool has preelaborable initialization; such types typically need complex initialization that's unlikely to be preelaborable. In any case, it's not quite as clear cut as the comment above suggests.]

[Unrelated editor's note: By making this an identifier specific to a pragma, this AI could (and should, IMHO) become a Binding Interpretation. Brad ought to consider that and possibly rewrite the AI in the form of a BI.]

AI12-0031-1/02 All_Calls_Remote and indirect calls

Brad would like some guidance before updating this.

We agreed that local calls would not use the PCS. The problem is that access-to-subprogram don't know whether it is local or not, as that depends on both the target of the call and the location of the call. Option 1, all such calls are direct (not through PCS). Option 2, all such calls should be though the PCS.

All_Calls_Remote is for debugging the system when the system isn't really partitioned. So it is only important that calls that **could** be partitioned be remote; not ones that could never be remote.

For a remote access-to-subprogram type, that should be a remote call. But Bob points out that the runtime could bypass the PCS if the partition id is the same for the caller and callee. That runtime optimization has to be turned off when All_Calls_Remote is set; that's true for any sort of remote call (so this is not a new issue). For a normal (not remote) access-to-subprogram, then it is a local call (no use of PCS). It has nothing to do with the designated subprogram.

All_Calls_Remote prevents any optimization (compile-time or runtime) of remote calls, and ensures direct calls that could be remote are remote.

Steve asks about remote dispatching calls (thru remote-access-to-class-wide). They also should have optimization prevented.

Brad will update the AI this way.

AI12-0031-1/03 All_Calls_Remote and non-direct calls

Remote subprogram calls are defined in E.4. Probably should have a (see E.4) for that. (ultimately, we decided to effectively copy the definition instead).

E.2.3(19/3): Tucker thinks the wording is still unclear. Try instead

> If aspect All_Calls_Remote is True for a given RCI library unit, then the implementation shall route any of the following calls through the Partition Communication Subsystem (PCS); see E.5:
> - A direct call to a subprogram of the RCI unit from outside the declarative region of the unit;
> - An indirect call through a remote access-to-subprogram type;
> - An dispatching call with a controlling operand designated by a value of a remote access-to-class-wide type.
>
> Other calls are defined to be local and shall not go through the PCS.

Redo AARM note: 19.b:

> There is no point to force local {direct} calls ({including}[or] calls from children) to go through the PCS, since on the target system[,] these calls are always local, and all the units are in the same active partition. {However, i}ndirect and dispatching calls {through remote access values} [however] go through the PCS because the extra complications in determining whether the {caller}[denoted subprogram] is local to the All_Calls_Remote RCI unit was deemed to be [too] excessive.

!discussion

> The goal of the All_Calls_Remote aspect is to force calls from outside the declarative region of the RCI unit to go through the PCS. Calls that are local to the {RCI unit}[PCS] should always be local and never go through the PCS.
> This ideal [is] can be implemented easily enough for direct calls, but there are excessive complications for the case of indirect or dispatching calls. In particular, it would be difficult for the implementation to determine whether {a call}[if a remote access-to-subprogram value or a remote access-to-class-wide type value denoted a subprogram that] was local to the RCI unit, and thus whether the call should go through the PCS.

Bob thinks that its silly for both the discussion and the AARM to say the same thing. Drop the second sentence of AARM 19.b.

After discussing some other topic, we return to the wording of E.2.3.(19), which Randy e-mailed.

Tucker thinks the wording is still unclear. It is revised to:

> If aspect All_Calls_Remote is True for a given RCI library unit, then the implementation shall route any of the following calls through the Partition Communication Subsystem (PCS); see E.5:
> - A direct call to a subprogram of the RCI unit from outside the declarative region of the unit;
> - An indirect call through a {value of a} remote access-to-subprogram type;
> - An dispatching call with a controlling operand designated by a value of a remote access-to-class-wide type.
>
> Other calls are defined to be local and shall not go through the PCS.

Tucker notes that the rule should only apply to designated subprograms from the All_Calls_Remote unit.

- An indirect call through a remote access-to-subprogram [type]{value that designates a subprogram of the RCI unit};
- An dispatching call with a controlling operand designated by [a value of] a remote access-to-class-wide [type]{value whose tag identifies a type declared in the RCI unit}.

Drop the “other calls” sentence. It is not true without qualification, which we don't want to have. (Calls to other RCI units have to be remote, even if All_Calls_Remote does not apply to them.)

Randy suggests simplifying the recommendation and using it as the summary, since it explains the problem in two sentences. The recommendation then becomes “see summary.”

Randy mailed the entire wording section of the AI. There is a glitch in the E.2.3(19): “A[n] dispatching...”

Approve AI with changes: 8-0-3.

AI12-0036-1/01 The actual for an untagged formal derived type cannot be tagged

We need an additional rule to prevent tagged ancestors from being used as actuals for untagged derived formal types. Tucker wonders if the category already requires that. Randy notes that 12.5.1(1) defines the category, and it allows this case. So we need a rule.

Ed reports that GNAT compiles this program without complaint, which isn't going to work.

Tucker thinks that the reverse problem is possible. He shows an example:

```
generic
   type N is tagged private;
   type NC is new N with private;
package GG is
   ...
end GG;

package P is
   type T1 is private;
private
   type T1 is tagged ...
end P;

with P;
package Q is
   type T2 is new P.T1;
end Q;

with Q;
package body P is
   package G is new GG (T1, T2);
   ...
end P;
```

T1 is tagged in the instantiation, T2 is not tagged and is not declared with an extension (but we know it is descended from a tagged type).

So replace the existing wording change with:

> The actual type for a formal derived type shall be tagged if and only if the formal derived type is a private extension.

This is incompatible, but only in cases where the derived type is inaccurately describing the actual type (as tagged vs. untagged).

Approve AI with changes: 9-0-2.

AI12-0038-1/03 Shared_Passive package restrictions

Tucker is confused as to what he was trying to do, the two restrictions seem identical. Randy notes that we needed to deal with hidden access types (parts of private types).

Steve wonders why parts of task and protected types aren't included in the existing E.2.1(7) rule. Tucker thinks they probably ought to be covered.

Tucker will try to figure out how this works and we can talk about it tomorrow.

[Editor's note: He didn't do that and we didn't talk about it again at this meeting.]

AI12-0041-1/01 Type_Invariant'Class for interface types

Randy notes that the dynamic rules already work so there is no wording other than allowing them.

Steve wonders about null procedures being inherited from multiple types, but Randy believes that we already have (and solved) such problems. [The new rule of AI12-0042-1 (7.3.2(6.1/4)) should eliminate any problems by making inheritance illegal in that case.]

Set status promising: 11-0-0.

AI12-0042-1/05 Type invariants cannot be inherited by non-private extensions

The subject needs to be changed to reflect the summary.

The original issue was Type_Invariant'Class inherited into record extensions (with visible components). A similar issue was identified for adding invariants to private extensions of record types (with visible components). In both of these cases, privacy could be broken such that the invariant would not hold after a change (that is, in such cases, the places where invariants are checked are insufficient to prevent problems). We decided its not worth preventing such things, even with the possibility of misuse. The above should be added to the start of the discussion.

The problem that is being solved occurs when private operations are inherited visibly. Tucker says that replacing "inherited" with "visible" (at the point of the declaration) is necessary. Steve worries about other declarations; Tucker points out that this is "visible", not "directly visible".

A case where the problem would exist is:

```
package P is
   type T is private with Type_Invariant'Class => World_is_Good(T);
private
   function Build_World return T; -- Does not check T's invariant.
end P;

private package P.Child is
   type NT is new T with private;
   -- Build_World is inherited here. This one checks Type_Invariant'Class. Must be overridden.
end P.Child;
```

If this is a public child (no word **private**), then the rule doesn't trigger, because Build_World would be inherited in the private part and thus not be a visible subprogram that invariants apply to.

Tucker says it would be simpler to say that "if the private operation is visible at the point of type NT".

Perhaps we need to say "inherited and visible".

Erhard suggests: 7.3.2(6/3): If a private operation of some ancestor is inherited at the point of the private extension, and ...

We move to the 7.3.2(18/3). We look at the example in Steve's e-mail from August. The problem is for T3.

```
package Type_Invariant_Pkg is
   type T1 is tagged private with Type_Invariant'Class => Is_Ok (T1);
   function Is_Ok (X1 : T1) return Boolean;
   procedure Op (X : in out T1);
private
   type T1 is tagged record F1, F2 : Natural := 0; end record;
   function Is_Ok (X1 : T1) return Boolean is (X1.F1 <= X1.F2);
end Type_Invariant_Pkg;

package Type_Invariant_Pkg.Child is
   type T2 is new T1 with null record;
private
   overriding procedure Op (X : in out T2);
   type T3 is new T2 with null record;
   overriding procedure Op (X : in out T3);
end;
```

Tucker does not believe this is the best way to describe this. Dynamically callable is the bad idea. He thinks that the issue is the nearest ancestor that is in a visible package specification with a private part (and therefore could have invariants). He claims that this rule was only for specific invariants; Randy disagrees, he thinks this latter part was added for these sorts of cases (specifically, ones like T2).

We can't convince Tucker that the wording is good enough. He wins the chance to rewrite this.

We agree on the intent that checks (all invariant checks) are made in cases like this, but we haven't agreed on how to describe this formally.

Keep alive: 7-0-3.

AI12-0042-1/06 Type invariants inherited by non-private extensions

The subject should say “nonprivate” (no hyphen).

In summary, “... to which {this}[the same] invariant applies.”

Delete “Modify 7.3.2(16)” and “Modify 7.3.2(18)” and the associated text; those were a previous false start that were not removed..

Steve says that Tucker's new wording causes checks on a case where we don't want them.

```
package P is
   type T1 is tagged private
      with Type_Invariant'Class => ...
private
   procedure Op (P : T1);

   package Nest is
      type T2 is new T1 with private...
         -- (2) All the operations have the invariants apply here. So overriding is required.
   end Nest;

   type T3 is new T2 with record...
       -- (1) Op would require checking here because the ancestor is visible.
end P;
```

Steve makes comment (1), Tucker replies with comment (2). Thus there is not a problem.

If T2 is **with record**, then it's visibility is not considered. So again there is no problem.

Tucker does worry that “visible” is too ambiguous. He doesn't have a suggestion.

Randy worries that this definition means that new primitive operations of a record extension are not checked against an inherited class-wide type invariant. If the programmer later derives a private extension from that type, the operation would suddenly get checking. That's similar to the private case where we require overriding. Tucker claims that there is no issue, since there is no privacy breaking going on in this case. [Editor's note: He's since been convinced that there is an issue, and this AI has been reopened to address it.]

In the discussion, get rid of the history. “This AI discusses two issues...”

The ACATS test should include C-Tests.

Tucker will update this during the break.

During the discussion of AI12-0059-1, Tucker asks if “visible outside of the immediate scope of type T” is sufficient. This is previous wording used elsewhere.

Tucker will e-mail this, because it's too hard to figure out what he's talking about.

AI12-0042-1/07 Type invariants inherited by nonprivate extensions

Tucker's e-mail finally arrives after 17 minutes.

The ACATS section still needs to say that both B and C tests are needed.

Approve AI with changes: 6-0-2.

AI12-0052-1/02 Implicit objects are considered overlapping

The second paragraph after A(3) is completely new, it should have curly brackets around it.

Glitch in the A(3) AARM note: “But concurrent calls {operating} on overlapping objects...”

Geert wonders why we don't require concurrent reads. We don't know whether an **in** mode parameter is only read or whether it might be written (if it is a handle, for instance, like many of the File_Type operations in Text_IO). Going through the Standard to figure that out on a case-by-case basis would be a massive amount of work of dubious value. This is a weak requirement added because Ada 83 vendors sometimes did preposterous things (like using global variables in Calendar). Vendors certainly are allowed to go further.

Erhard worries that access parameters aren't properly handled here. They're by copy, and they could have the same value (designating the same object). That can't be expected to work. This applies to stream operations, for instance. That has been wrong since Ada 95, and there is some sentiment not to fix it at all. Erhard is asked to create an AI to explore/fix this issue.

Typo in the AARM note for A.10.3(21): "...with the file give{n} to the..."

"...guarentee..." is spelled wrong in the discussion.

Approve AI with changes: 7-0-4.

AI12-0055-1/03 All properties of a profile are defined by pragmas

Change "profile" to "usage profile" in the subject.

This was defined in such a way that is incompatible with the existing semantics in the standard. That seems to be a problem; we just wanted the existing text defined with semantics. Certainly, we don't want to change the semantics in the absence of a bug, that would require an Amendment. [Editor's note: We do need to make clear that the Ravenscar rules have the potential to practically invalidate the rules about scheduling such as FIFO_within_Priorities, because CPU assignments override those.]

So we want a pragma to codify the current extra rules. Possible names suggested include No_Task_Migration or Static_Task_Assignment. We do not decide on a name for the pragma.

The AI should be split. D.13(8/3) should be defined as a pragma (possibly with one of the names suggested previously) which is part of the Ravenscar profile, as above, and then a new AI with additional pragmas that aren't implied by Ravenscar (as defined in the existing AI).

Geert will take these AIs.

Keep Alive: 9-1-0. Bob votes against; he does not think making into pragmas is important enough, and he's not convinced that new functionality is needed.

Geert says that having these pragmas would benefit compiler-writers for testing.

AI12-0059-1/01 Object_Size attribute

Randy provided the last version of AI95-0319-1, which he had updated with the comments from the last time it was considered, even though it was killed at that meeting. This was killed mainly because of opposition from Rational, they didn't like it participating in static matching.

"...choose for a stand-alone aliased {variable}[object] of subtype S." One can imagine constants taking advantage of discriminants not changing.

Here we had a break and then discussed some other AIs. We then returned to 59.

There is a discussion as to whether this is well-defined when not specified. Tucker is convinced that there is a unique value for non-limited aliased variables. Steve disagrees. Steve and Tucker will take this off-line.

Randy asks whether we want to make these always subtype-specifiable. The consensus is yes. Remove all of the implementation-defined stuff about that.

Randy wonders whether this should be mainly defined as an aspect. Leave it described as an attribute, as this will be next to Size and it would look weird to talk only about aspects in a list of attributes.

Ed announces he just got an alert that his flight is canceled, and after some disruption (caused by many people checking and/or rebooking their flights), Ed and Geert leave to go immediately to the airport to catch alternative flights. (This discussion occurred relatively late on the last day of the meeting.)

Back to the AI. Steve worries that there isn't a unique value in some cases. We will make such cases implementation-defined. We need to say that the value for limited types is implementation-defined. [Steve didn't explain what other cases are involved.]

We don't want to force GNAT to change anything, thus implementation-defined where needed (**not** illegal).

Someone objects that the static matching rule isn't implemented in GNAT. GNAT has those cases as being erroneous, and certainly a static check is better than erroneous.

This will go back to Randy.

Approve intent of AI: 5-0-2.

AI12-0061-1/02 Index parameters in array aggregates

We discuss the wording.

We talk about the nominal subtype of the loop_parameter. This is not well-defined for **for** loops, either. Steve would like to propose that there is an implicit static predicate. That seems like overkill. Instead, we should just use First_Valid to Last_Valid (if the bounds are static). There is no mention of predicate. Tucker would like to add the predicate, or use the index subtype.

It's much easier to just use index subtype. Using a case expression inside of an aggregate is not very likely (it would be better to distribute the choices). [Editor's note: But using a case statement inside of a **for** loop is more likely, and we would expect to use the same definition for those. I hope!]

Simple option: 7; complex option: 1; abstain: 3; so use the index subtype.

parameterized_array_component_association, would better as iterated_component_association, even though it doesn't iterate in any particular order.

Erhard wonders why this is a choice rather than just a kind of an aggregate. Allowing only one of something when there isn't a lot of extra work to allow many is a bad idea. Compilers already have to deal with static discontiguous aggregate choices, and this is defined similarly, so this shouldn't be a major disruption to compilers.

The AI will go back to Steve for updating.

Approve intent: 9-0-2.

AI12-0065-1/00 Descendants of incomplete views

In the AARM example, T3 is visibly derived (indirectly) from Integer, but it doesn't have numeric characteristics. “this situation” should be explained better. Explain which is the ancestor, and the descendant.

At least part of the formal text of 7.3.1(5.2/3) is not redundant. Erhard wonders about the type conversion rule that allows to convert to your grandparent's type. He doesn't find it clear as to what allows that.

Tucker says that Gary has convinced him that the first sentence of 7.3.1(5.2/3) is broken. He thinks the first sentence essentially says that an A is not an A.

Give this back to Tucker to straighten this out.

Keep alive: 10-0-1.

AI12-0065-1/01 Descendants of incomplete types

The “{Furthermore, it} is“ is missing the closing insertion bracket, which belongs as shown here.

ACATS: No additional tests are needed.

Drop the TBD from !wording.

This should be a !ramification rather than a !binding interpretation, because it makes no language change.

We discuss that T3 would match a formal derived (type T is new Integer). The characteristics “exist” but are never visible, but they can emerge inside a generic. We had a case like this for equality previously. Steve notes with wonder that while T3 would not match a formal integer type, it would match a formal derived, and that would match a formal integer type.

Approve AI with changes: 6-0-5.

AI12-0071-1/03 Order of evaluation when multiple predicates apply

Erhard wonders about the idea of progenitor subtypes. These are the subtypes named for the progenitors of the declaration. But this isn't well-defined, we need to add a bit of wording in 3.4.

Tucker notes that he (re)wrote these as “tests”, because they are not necessarily part of a check (they could be part of a membership or Valid).

Steve asks that the rule about evaluating the Predicate_Failure be marked as redundant. Tucker argues that it is insufficiently obvious to be redundant. Straw poll: Redundant: 1; Not redundant: 4.

Tucker will look for the cases where the terminology needs changing (“BIG NOTE”).

Tucker worries about S'Base. It's not a first subtype. We throw away all of the constraints (including those of parent), so we don't want it to have any predicates, either. Even from the parent. Geert notes that S'Base is used in generic elementary functions, and we don't want any predicates to reappear there (or anywhere else).

So S'Base has no predicates. Tucker should craft a user note to this effect.

There is a question about the wording change for the Valid attribute. The rules for Volatile do not allow the introduction of reads; thus we have to say that X'Valid reads X so that it can be implemented. (How the value could be tested without reading it from memory is hard to imagine.)

Approve intent of AI: 9-0-1.

Tucker will try to update the AI (including the summary and discussion).

AI12-0071-1/04 Order of evaluation when multiple predicates apply

Erhard: 3.2.4(30/3): “...until one of the tests fail{s}...”

Steve: grammar “the value is first tested whether it satisfies...” It seems like something was missing.

“the value is first tested {to determine} whether it satisfies...” Various places in 3.2.4(30-33/3) (Tucker says there are four such places.)

In the NOTE: “No predicates apply to the base subtype of a scalar type{;}[, so] every value...

There is an extra blank line in the summary.

!ACATS Test section should be updated to get rid of “whether AI-54-2 does” since we know what it does.

Approve AI with changes: 11-0-0.

AI12-0074-1/03 View conversions and scalar out parameters passed by copy

Steve explains the sort of problems that can happen in the access cases.

The possible fixes are: (1) statically illegal (incompatible, there is an easy workaround to the incompatibility); (2) pass in null (which is inconsistent, but even less likely to be noticeable); (3) make the type checks on input (also is inconsistent).

Steve also notes that the static solution has the advantage of being the same as scalars. It is also better to find out at compile time rather than at run-time.

Straw vote: Statically illegal 8-3-1.

Bob argues for a warning, along with declaring this case erroneous (or implementation-defined).

Bob points out that most subprograms do not read the object.

Steve will write this up as a statically illegal case.

Geert suggests that if this happens and we're allowing it (possibly via a suppressible error, see the discussion of AI12-0092-1), it should be defined to be abnormal. In that case, it would only be erroneous if read, which is what we would want.

Keep alive: 11-0-0.

AI12-0081-1/01 Real-time aspects need to specify when they are evaluated

Approve AI: 9-0-1.

AI12-0082-1/01 Definition of “dispatching domain”

Why does D.16.1(16/3) describe these as a “series” of processors? D.16.1(16/3) should say “set” rather than “series”. (That really ought to have been done as part of AI12-0033-1, but we can fix it here.)

The last two sentences of D.16.1(16/3) are confusing. (These were not touched by this AI previously). Modify them as follows:

> System_Dispatching_Domain {identifies a domain that} contains the processor or processors on which the environment task executes. At program start-up all processors are contained within {this domain} [System_Dispatching_Domain].

In discussion, paragraph 3: change “identify” to “identifies”.

Approve AI with changes: 8-0-2.

AI12-0083-1/01 Automatic creation of constructor functions

The problem is that a mix-in generic can't be used if the type has extension components and a contructor function.

Tucker had suggested that making the type abstract would allow a later derivation. Steve says that doesn't work if the user needs to declare an object in the generic.[Editor's note: It doesn't work very well if the generic type has private extension components, as then it's not possible to write an extension aggregate in any user-written constructor function. As such, an explicit call on some sort of exported initialization procedure would have to be used, which is error-prone.]

Randy tries to explain that he proposes here extending the concept of automatic creation of constructor functions to types that have an aspect specifying that their default initialization of extension components is sufficient initialization. In that case, the same mechanism that currently allows null extensions to automatically create constructors can be extended to all types with that aspect. Randy notes that this would greatly reduce the maintenance hazard related to that feature (his long-standing complaint about the feature).

The group agrees that the problem is worth solving, but this solution isn't well liked. The main complaint seems to be that the need to declare that default initialization is sufficient seems unusual to most. It's necessary, of course, to prevent problems when default initialization is not sufficient; perhaps it would be better to have an aspect to declare that. (Such a solution would still be compatible as this is a new capability either way.) We certainly would not want to give up compile-time checking of constructors in the case where they really do need to be written explicitly.

Since this problem has existed in Ada since Ada 95, we think we need to see more evidence that the problem is significant before we try to proceed to find a solution.

Hold AI: 8-0-0.

AI12-0084-1/01 Box expressions in array aggregates

Approve AI: 9-0-1.

AI12-0085-1/01 Missing aspect cases for Remote_Types

Gary asks that we add a hyphen to access-to-class{-}wide. Brad notes there is an occurrence of class{-}wide in the wording.

Approve AI with changes: 10-0-0.

AI12-0086-1/01 Aggregates and variant parts

This AI provides a small extension to the rules for aggregates to allow some non-static discriminants, so long as the discriminant has a static subtype that can select exactly one one variant.

Tucker would like to simplify the wording. Steve and Randy both have tried to simplify it without success, so he needs to make that attempt himself.

Tucker will take this and make an attempt to simplify the wording.

Approve intent of AI: 7-1-0. Jeff doesn't think it is worth bothering with.

AI12-0088-1/02 UTF_Encoding.Conversions and overlong characters on input

Randy and Robert agreed on the meaning during the e-mail discussion. Randy notes that Robert's argument about usability convinced him. Someone says that's unusual, so this must be the correct answer. Someone else suggests watching out for flying pigs [Editor's note: I hope it's not *that* unusual that we agree!].

Randy notes that the AI doesn't reflect this meaning. The AI wording should be changed to drop the Redundant part. Add a AARM ramification to say that overlong encodings are not considered invalid on input but never generated (it does not depend on the character set standard).

Change the answer to the question. Change the summary.

The last sentence of the second paragraph of the discussion needs to be changed.

“You cannot optimize away a conversion from UTF-8 to UTF-8 because there must be a check...”

Approve AI with changes: 5-0-5.

AI12-0089-1/01 A generic function is not a function

Randy explains that John asked this question during his editorial review, and it made him look over some of the wording to see if it made sense for generic functions. This fixes some that clearly doesn't.

For 6.5, put a To Be Honest after 6.5(5.2/3) to say that “function” also means “generic function”. This sort of follows from the fact that a “function body” includes generic function bodies, and the lead-in text thus includes generic functions. Thus the rest of the text must also include generic functions. This could be a ramification.

6.5(4/3) only talks about callable constructs, and a generic function or procedure is not a callable construct. Yuck. The To Be Honest needs to cover that.

Approve with changes: 10-0-1.

AI12-0090-1/01 Pre- and postconditions and requeues

Steve suggests that we have two options. One is that a requeue can't have any postconditions. Alternatively, it can have a postcondition only if it conforms (is the same), and it cannot contain 'Old.

Preconditions have to be evaluated on requeue. This is not a call, so we'll need new wording to accomplish that.

There is no need to require the preconditions to be the same, just that both get evaluated.

Ed asks what happens if the parameter names change on the requeue. Tucker notes that there is a similar problem with class-wide preconditions and postconditions. That should be a separate AI. Bob thinks this isn't worth writing an AI for; Tucker and Steve think it's necessary. Tucker will take this AI for class-wide preconditions and postconditions. [Editor's note: Is there a similar problem with the name representing the current instance in class-wide type invariants, and with any inherited subtype predicate? Those also will have different names than used in the assertion expression. Or is the fact that it is the current instance enough? Perhaps this could be addressed in AI12-0068-1.]

Steve volunteered to take this AI before the last question came up, and now it's too late to back out.

Approve intent of AI: 9-0-1.

AI12-0090-1/02 Pre- and postconditions and requeues

“roughly speaking” had better be in the AARM.

Tucker brings up type invariants. What if an interface or a protected type has a type invariant? It's possible to have an interface with an operation that is implemented-by an entry. We can requeue on that. If we have a private type that uses such an interface as a progenitor, then we can have a type invariant that applies to an operation that we requeue on. What are the rules for that? Steve will take this problem off-line.

Randy notes that the wording needs to allow requeuing on procedures that are implemented by an entry. Steve gets various other comments on the wording. (He says he made notes and that it wasn't necessary for the editor to record them all – hope he's right because I didn't record them - Editor).

“aforementioned” is bad as the text is 5 paragraphs. Say something else.

Back to invariants. A type invariant that applies to (the type of) E1 has to apply to (the type of) E2. Tucker says that the parameters have to have the same subtype, so we don't have to worry about invariants for parameters. Randy notes that E2 doesn't need to have parameters. If the invariants are implemented in the body (someone confirms this is how GNAT implements invariants), how does that work?

Steve will retry.

AI12-0091-1/01 Add procedure Sin_Cos to Ada.Numerics.Generic_Elementary_Functions

Geert says that having the instruction isn't relevant, the issue is more that the argument reduction would be shared, and argument reduction is expensive. GCC actually recognizes cases like this, so there is no need for an additional routine in GNAT.

This is an optimization that compilers can do. Tucker thinks this is a slippery slope, in there are many other combinations that make sense and often appear together. We don't want to open a large can of worms.

Erhard says that users won't believe that optimization happens. He thinks that common machine instructions should have first-class interfaces.

Geert says that an AARM note that argument reduction should be done only once would be appropriate. But in which cases?

If we have this, we should have a Cycle version. Jeff would like to take this AI and provide wording.

Do other languages provide this? It's not in the C or C++ standards now, but that might change in 5 years. It is in some C implementations (GCC C, Visual Studio).

Keep alive: 10-1-0.

AI12-0092-1/00 Soft Errors

Tucker worries about the criteria for choosing between a Soft Error vs. a Hard Error. Bob primarily plans this be for a compatibility issues. Tucker does not like coming to a language and seeing stuff specifically for compatibility checks. He doesn't want to see the line between old and new stuff.

Erhard tries to give a more general definition for soft errors. It confuses everyone, including the note takers. [Editor's note: I think he meant a definition something like the following, but this is mostly my invention: “A soft error is used when the compiler can detect a dubious construct, but the construct can have a legitimate use in limited circumstances.”]

Bob had suggested that compilers would have a mode where soft errors would reject the program. ACATS would include tests that would require that (as well as test what happens when the error is suppressed).

Steve worries that having soft errors would make it necessary to go back in the standard and find cases were we can statically detect bad code. That seems like a can of worms. (for instance, Pos := -1;) Bob says that we aren't required to do that.

Tucker suggests identifying errors as how new they are. He suggests that it would be sort of like Suppress, where certain errors would could be ignored by a compiler. It could be controlled by a configuration pragma.

Brad suggests that we call those suppressible errors.

What is the granularity of this? This would just be a global setting, mainly for those who can't modify the source. If they can modify the source, they should get rid of the bug, not suppress the error, so local control (like pragma Suppress) isn't needed.

Straw poll: 7-keep alive suppressible error; 2-kill suppressible error (UK); 3-abstain

The AI goes back to Bob for a full write-up.

AI12-0093-1/01 Iterator with discriminated cursors

Randy explains that the original idea was that Cursor was definite.

Tucker thinks that the First and Next are functions, so that the idea of an indefinite cursor works, with a separate object.

Steve worries that Next takes the *previous* cursor as a parameter; so we need to have the object last that long. So we effectively need two objects to exist at the point at the call to Next.

You could use a Holder to work around the problem (wrap the indefinite type in a Holder container, then use the Holder type as the cursor type).

We think that fixing this would be insane because of the need for two objects. Thus, we will leave the semantics as they are, especially as the workaround is easy.

No action: 9-1-1.

Randy and Steve note that we need an AARM note to say that the accessibility of the loop object is that of of the loop statement, so the object is finalized when the loop is left. Secondly, the object is constrained by its initial value if the cursor type is indefinite. Steve says that the tag if it is class-wide has to be covered. Tucker says that you can't ever change the tag of an object during its lifetime. This object is a variable with a constant view.

So this should be a ramification. Randy will take this.

Approve intent of AI: 10-0-1.

AI12-0093-1/02 Iterator with indefinite cursor

The answer to the question should be Yes., as opposed to No.

Behavior has the British spelling in the question.

Approve AI with changes: 11-0-0.

AI12-0094-1/00 access_to_subprogram_definition should be a declarative region

We think this is OK. Bob Duff will take the AI. He ought to check that declarative region is not used in any way that would cause problems with static accessibility checks.

Does this apply to named access-to-subprogram declarations as well? Steve worries about the current instance of the type in the declaration (if it had the same name as one of the parameters). Tucker says that would just be hidden, it's not a problem. Tucker thinks that it is more natural for this to be a declarative region, it's probably more work for a compiler to make it not one. Randy recalls some Ada 83 ACATS tests with this effect.

We'll try to make both named and anonymous access types work the same. That seems more natural.

What about use of the parameter names in default expressions? An example is shown:

```
type T is access function
      (A : Integer;
       B : access function (C : Integer := A) return Boolean)
   return Boolean;
```

Someone suggests that hidden from visibility rule saves us here. 8.3(16) says that declarations (the parameters here) are not visible until the end of the declaration, but that is the ending semicolon. It's not clear that's the right rule.

6.1(21) says that you can't use a parameter name in the same formal_part. But that allows the parameters to be used in the result part. Until we added anonymous access-to-subprogram, that wasn't a possibility anyway. But now it is, for example:

```
type T is access function (A : Integer)
   return access function (B : Integer := A) return Boolean;
```

We probably need to change 6.1(21) to say *profile* rather than formal_part.

We ask Bob to summarize the intent of the AI. He says that access_to_subprogram_definition is a declarative region. Randy notes that you also need to say access_definition is a declarative region. Steve says that it could be "the profile of an access_definition and access_to_subprogram_definition". We worry about that wording making sense in the anonymous access-to-object case (which is syntactically part of access_definition). Tucker

thinks it is harmless to have that. Returning to the intent of the AI, Bob is going to fix 6.1(21) to say profile or something like that.

Approve intent of AI: 10-0-0.