Minutes of the 46th ARG Meeting

24-26 February 2012

Kemah, Texas, USA

Attendees: Steve Baird, Geert Bosch, Randy Brukardt, Jeff Cousins, Gary Dismukes, Bob Duff, Brad Moore, Erhard Ploedereder, Ed Schonberg, Tucker Taft.

Observers: None.

Meeting Summary

The meeting convened on 24 February 2012 at 09:10 hours and adjourned at 12:30 hours on 26 February 2012. The meeting was held in the Boardroom of the Kemah Boardwalk Hotel on the boardwalk of Kemah, Texas. [Unfortunately, the Boardwalk's roller coaster was broken. The suggestion during the meeting was that it would have provided a perfect way to resolve disputes, first to scream loses. For me it symbolized the wild ups and downs of the process of creating a Standard. - Editor.] The meeting covered the entire agenda of Ada 2005 AIs.

AI Summary

The following AIs were approved with editorial changes:

AI05-0269-1/09 Editorial comments on Draft 14 (8-0-2)
AI05-0274-1/06 Side-effects during assertion evaluation (8-1-1)
AI05-0275-1/03 More issues with the definition of volatile (10-0-0)
AI05-0283-1/02 Stream_IO should be preelaborated (9-0-1)
AI05-0284-1/01 Accessibility of anonymous access returns (9-0-1)
AI05-0285-1/02 Defaulted environment variable queries (8-0-2)
AI05-0286-1/03 Internationalization of Ada (6-0-4)
AI05-0287-1/01 Some questions on subtype predicates (10-0-0)
AI05-0288-1/01 Conformance of formal access-to-subprogram types (10-0-0)
AI05-0289-1/01 Invariants and in-mode parameters whose value is changed (7-0-3)
AI05-0291-1/01 Target object needs to include synchronized interfaces (7-0-3)
AI05-0292-1/01 Terminology: “Indexable type” is confusing (10-0-0)
AI05-0293-1/01 Remove unnecessary hyphens (10-0-0)
AI05-0294-1/01 Update the glossary (9-0-1)
AI05-0295-1/02 Improve presentation of aspects (8-0-2)
AI05-0296-1/01 Freezing of actual subprograms which have parameters of formal incomplete types (7-0-3)
AI05-0297-1/01 First_Valid and Last_Valid attributes (8-1-1)

The intention of the following AI was approved but it requires a rewrite:

AI05-0290-1/01 Improved control over assertions (5-0-5)

Detailed Minutes

Previous Meeting Minutes

Randy applied all of the previously reported changes to the minutes (this is the currently posted "draft version 2").

Approve minutes as posted: 10-0-0.

Date and Venue of the Next Meeting

Next meeting will be at Ada Europe. Ed reports that finding a meeting venue is a problem for the weekend days. The dates are June 15-17, 2012. We probably only need a 2 day meeting.

The question is raised whether we'll need a June meeting at all? Revisit that on Sunday.

On Sunday, we decide to stick with the two day meeting plan. Many of us will need to attend the WG 9 meeting to ensure there are no surprises. It's unlikely that no bugs will be found in the interim, and we have several AI12s that probably ought to be considered soon (particularly the specification of exceptions for assertions). Since the first day of the meeting will only be a half day, it's unlikely that we could finish the work in that time. So we'll have a two day meeting, potentially ending at the normal stopping time for a second day.

Thanks

Thanks to Pat Rogers for arrangements for the meeting.

Thanks to the minature train and its frequent whistle for providing a unique atmosphere for the meeting. [The Boardwalk has a small train track that runs throughout the complex and a train with three cars that hold at least 30 people. This includes running past the windows of the meeting room. Entrances to restaurants and stores that cross the tracks (which is most of them) have miniature crossing gates and the train blows a whistle just before crossing them (like a real train). This turns out to be necessary; after dinner one night I nearly walked into the crossing gate and if it had not been there I probably would have walked in front of the oncoming train. It may be miniature, but I suspect that the effects of getting hit by it would not be pleasant. - Editor.]

Old Action Items

There are no known open action items.

New Action Items

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

Everyone:

- Participate in and vote on the letter ballot for the resolution of AI05-0290-1.

Randy Brukardt:

- Apply the changes made at this meeting to the draft Standard, along with editorial comments from the ARG and public. Deliver that draft as soon as possible to Joyce Tokar for WG 9 approval.

Editorial changes only:

- AI05-0269-1
- AI05-0274-1
- AI05-0275-1
- AI05-0283-1
- AI05-0284-1
- AI05-0285-1
- AI05-0286-1
- AI05-0287-1
- AI05-0288-1
- AI05-0289-1
- AI05-0291-1
- AI05-0292-1
- AI05-0293-1

- AI05-0294-1
- AI05-0295-1
- AI05-0296-1
- AI05-0297-1

Tucker Taft:

- AI05-0290-1

Detailed Review

The minutes for the detailed review of AIs and SIs are divided into ASIS Issues (SIs), Ada 2005 AIs (AI05s), and Ada 2012 AIs (AI12s), but no SIs or AI12s were considered at this meeting. The AIs and SIs are presented in numeric order, which is not necessarily the order in which they were discussed. Votes are recorded as "for"-"against"-"abstentions". For instance, a vote of 6-1-2 would have had six votes for, one vote against, and two abstentions.

If a paragraph number is identified as coming from the Ada 2012 Standard, the number refers to the text in Draft 15 of the Ada 2012 AARM. Paragraph numbers in earlier drafts may vary.

Detailed Review of Ada 2005 AIs

AI05-0269-1/09 Editorial comments on Draft 14

These are assigned to the group; each member other than Randy and Tucker reviews a chunk of this AI. (Randy and Tucker leave the room to work on AI05-0291-1.)

On Sunday, we consider the results of each review.

Tucker's work on an ACATS test uncovered a bug in the description of Multiway_Trees.Append_Child. The third parameter should be No_Element. Add this as a new item in this AI.

Also add the item from the National Body Review responses (see the end of these minutes).

Looking at items of interest (the item number is given first):

(44) Ed wonders why the parameters weren't lined up (13.11.4). Randy says that this is the same as Storage_Pool, from which this is derived. He could change those, but didn't because they were not modified. This needs to be mentioned in the AI somewhere.

(42) The question was raised as to why the original suggestion to use a bulleted list was not used. Randy explains that this was discussed in e-mail, and no better suggestion has emerged. The bulleted list suggestion would have turned this one paragraph into eight paragraphs, and it doesn't improve the wording enough for the complications. He should document this somewhere in the AI.

(55) Geert notes D.16.1 has wrap-around problems in the RM-Final.pdf. Those should be fixed.

(56) Geert says that 9.5(2/3) has a dot in it. (This doesn't really have anything to do with this item.) Randy notes that wording changes of AI05-0291-1 rewrote this paragraph.

(29) Gary notes a typo in the AARM Note: "...b[y]{e} used...".

(1) Bob notes this wasn't actually applied to the Standard. The change should be applied to the Standard.

(17) Brad notes "...[are] provide..." (the "are" should have been deleted as well).

(16) Brad wonders if this wording is clear enough. Bob made the original comment and he agrees it is clear enough.

(23) Jeff wonders why “nonzero” rather than “positive”. Randy says that is how we wrote existing Alignment wording (13.3(26.3/2)), so it seemed best to copy that rather than invent new.

(25) Jeff notes two unrecorded other places that needed changes and were changed. A.18.6(94.4/3) and A.18.9(113.4/3).

(26) Jeff notes that most “including <something> itself” are in parens, but three are not: A.18.10(175/3, 187/3, 191/3) [In Draft 14 these are one less (174/3, etc.)]

[Editor's note: in these three paragraphs, the wording is similar to "all of the descendants of Parent other than Parent itself." (this is the wording of A.18.10(174/3), the others are similar). This is the reverse sense of the parenthetical remarks "including <something> itself" and certainly is not parenthetical. For instance, we don't want Delete_Child (174/3) to delete Parent, but Parent is a descendant of itself. Therefore there is no change made here.]

Approve AI with changes: 8-0-2.

AI05-0274-1/05 Side-effects during assertion evaluation

This has to be well-defined and allow memo functions. The previous wording was too vague; it was not possible to answer the question of what the intent was vis-a-vis the example discussed at the last meeting.

Tucker suggests “affect any part of an object".

Randy suggests his last wording. Steve says that two calls to the same memo function would trigger that wording.

Tucker suggests talking about “immediate reevaluation”; if the result would be different, the bounded error would occur.

Erhard notes that this would be a cheap check. Bob disagrees; he doesn't think it would be cheap.

Typo: 23.3/3 “callable [entry]{entity} C”

The Bounded Error is a problem. The possibility of raising Program_Error is problematical – it makes programs less safe, especially when porting code to a new compiler.

It is suggested to use an Implementation Permission instead: the implementation would be allowed to reject such expressions. Specifically, the implementation can reject an expression if reevaluating the expression “could change” the result.

Tucker volunteers to write this wording.

Steve worries that the bounded error for protected operations is too broad. It prevents a potentially blocking operation from being called from a normal assertion. This bounded error should only apply to assertions that belong to protected operations.

Steve notes that expression is misspelled in the problem statement.

AI05-0274-1/06 Side-effects during assertion evaluation

On Saturday, we consider Tucker's revised version of this AI.

Tucker tightened up the (first) Bounded Error to only apply to protected operations.

The other bounded errors have been turned into a permission to reject offending expressions.

“...expession...”

Erhard is concerned that “associated with a call” is too vague.

“evaluated as part of a call” is suggested.

Erhard suggests that it is “checked as part of a call on or return from a callable entity...”.

Add an AARM note: “This allows an implementation to reject such assertions. To maximize portability, assertions should not include expressions that contain these sorts of side-effects.”

This will potentially reject assertions that use side-effects.

“side effect” should not have a hyphen unless it is used as an adjective. There are other occurrences (6 clauses) that should be changed.

Approve AI with changes: 8-1-1.

Bob votes against, he says he doesn't see the point of this. But he then refuses to discuss his position further despite repeated prodding, so we don't really understand why.

AI05-0275-1/02 More issues with the definition of volatile

Geert had objected to the !discussion of the AI. Randy wanted to make sure that we all agree on the model and that any problems were just his misunderstandings rather than a real problem.

The important thing is that two tasks that evaluate volatile variables “see the same order” of evaluation. Tucker notes that these can be piecemeal. Geert notes that seeing the piecemeal order is not possible because any such update would be erroneous by 9.10(11), unless there is some other thing which makes it sequential. And anything that would make the updates sequential would necessarily do so between entire updates, not pieces.

Erhard worries about multicore programming and cache flushes. Geert says that does not belong in a programming language, it matters at the microarchitecture level. So a programmer shouldn't care; the compiler has to know how to guarantee the needed consistency. Perhaps an implementation would have a scheme for specifying what memory is assigned to which tasks (in order to minimize costs). [He also gave some examples which I didn't record – apologies to anyone that would like more information on this – Editor.]

Geert will rewrite the discussion of this AI. This should explain “why” we are not explaining more.

AI05-0275-1/03 More issues with the definition of volatile

On Saturday, Geert sent a revised !discussion

> !discussion
>
> From 9.10 it follows that (in non-erroneous programs) accesses to variables, including those shared by multiple tasks, are always sequential. This guarantees that no task will ever see partial updates of any variable. For volatile variables, C.6(16) additionally specifies that all tasks see the same order of updates.
>
> If for a shared variable X, a read of X occurs sequentially after an update of X, then the read will return the updated value if X is volatile, but may or may not return the updated value if X is non-volatile. For non-volatile accesses, a signaling action is needed in order to share the updated value.
>
> Because accesses to the same atomic variable by different tasks establishes a sequential order between the actions of those tasks, implementations may be required to emit memory barriers around such updates or use atomic instructions that imply such barriers.

Should this be an AARM note? Tucker claims this is the clearest explanation of Volatile that he's ever seen. So it should go in the AARM.

“For volatile variables {(including atomic variables)}, C.6(16)...”

“...return the updated value if X is volatile {or atomic}...”

"...establish[es] a sequential..."

"...non[-]volatile..." (see AI05-0293).

Approve AI with changes: 10-0-0.

AI05-0283-1/02 Stream_IO should be preelaborated

Typo in summary: "is [be] preelaborated".

Approve AI with changes: 9-0-1.

AI05-0284-1/01 Accessibility of anonymous access returns

Steve volunteers to come up with wording for (1). We agree that it needs to be done.

For (2), Tucker worries that we shouldn't be talking about the object, we should be talking about the type. No, this is talking about the "result object", which has nothing to do with the type (in general).

Tucker thinks this shouldn't be a bullet at all; it "finishes" the thought. So it is indented the same as 10.1, but is not a bullet.

On Saturday, we take up Steve's proposed wording changes:

> Append to 3.10.2(28/3):
>
> If A is an access result type, then the accessibility level of the view shall not be statically deeper than that of the master that elaborated the associated function body.
>
> Append to 3.10.2(32/3):
>
> If S is an access result type, then the accessibility level of P shall not be statically deeper than that of the master that elaborated the associated function body.
>
> Add (as another bulleted list item) after 6.5(5.8/3):
>
> If the result type of the function is an access result type then the accessibility level of the type of the expression (if any) shall not be statically deeper than that of the master that elaborated the function body.
>
> ====
>
> The first of these three changes is needed for the example in AI05-0284:

```
function Bad return access Some_Type is
   Local : aliased Some_Type;
begin
   return Local'access; -- should be illegal
end Bad;
```

> The second handles the analogous access-to-subprogram case:

```
function Bad2 return access procedure is
   procedure Local is ... ;
begin
   return Local'access; -- should be illegal
end Bad2;
```

> The third is needed for something like

```
function Bad3 return access ... is
    type Local is access ... ;
begin
   return Local'(...); -- should be illegal
end Bad3;
```

Steve explains that the examples provide the justification for the new rules.

This change is wrong; this is a conversion. It's implicit, but 8.6(27.1/3) says it is conversion. So 4.6(24.21/2) would need a change (similar to the SAOAAT rule in 4.6(24.17/2).

Tucker digs up 3.10.2(19.3/3), which defines the statically deeper relation for an anonymous access in a return statement. So we don't need any new rules here.

This is not obvious to all. But most are eventually convinced.

So answer question (1), "see 3.10.2(19.3/3)".

Question (2) remains as is.

Randy will clean this up to reflect the fact that there is no hole for (1).

Approve AI with changes: 9-0-1.

AI05-0285-1/01 Defaulted environment variable queries

Should this raise Program_Error if there are no environment variables? No, you want to use the default in any case.

Bob will rewrite this.

AI05-0285-1/02 Defaulted environment variable queries

On Saturday, we consider the wording that Bob sent:

> Replace this:
>
> If the external execution environment supports environment variables, then Value returns the value of the environment variable with the given name. If no environment variable with the given name exists, then Default is returned. If the execution environment does not support environment variables, then Program_Error is propagated.
>
> with:
>
> If an environment variable with the given name exists, returns its value. Otherwise, returns Default.
>
> At editor's discretion, add:
>
> AARM Note: On an implementation that does not support environment variables, the "Otherwise" above will apply. Duh.
>
> Add to !discussion:
>
> Not only is the two-step process inconvenient, it could cause a race condition.

We need the AARM Note. It should start with “For”, not “On”. Drop “Duh”.

Tucker and Geert both want this explicit in the normative wording. Tucker points at A.17(13/2).

> If the external execution environment supports environment variables and an environment variable with the given name currently exists, then Value returns its value; otherwise, it returns Default.

Approve AI with changes: 8-0-2.

AI05-0286-1/02 Internationalization of Ada

Tucker would like the CR to be optional, so that Linux/OSX systems can directly create the files. So we want to reword this to say that both <CR><LF> and <LF> alone work (as a single line end).

Every code point represents itself. <CR><LF> represents one line ending. This should be a separate sentence.

Randy will reword.

Should we get rid of 2.1(18)? Yes, and get rid of the annotation as well.

Tucker suggests @Redundant[Other source representations are permitted.]

Bob wonders why this isn't a requirement (as Randy suggests)? Tucker worries that it is going too far, it might make some implementation cause problems. Ed notes that it is rather late to impose new requirements.

Erhard worries that advice is insufficient to make anybody happy. It's not enough for the Swiss comment, and those who don't want any UTF-8 will not be happy with the advice, either, so making it a requirement changes nothing for them. The group is swayed by this argument.

Make this an Implementation Requirement. Use “shall”.

Moving on to (2):

The Implementation Advice is annoying, the BOM requirement in particular is a problem. In particular, some systems allow passing UTF-8 to their existing interfaces and this "just works". Adding any interpretation of the string would cause problems on such systems. On the other hand, doing that on Windows will not work at all (the implementation has to use the 'W' API calls rather than the 'A' calls to support wide characters – Windows having two disjoint sets of API calls for "normal" strings and "wide" strings).

Randy notes that code that expects plain Latin-1 results may or may not work with Name returning unmarked UTF-8. That's especially a problem with Ada.Directories (where there can be no prior knowledge of what to expect).

Implementations can support some mechanism, but we can't define a mechanism (the problem is too hard for our very limited time).

The requirements for Windows and OSX and other operating systems are different. Tucker finds an article on-line that says Windows uses three different encodings for different things.

We can put in an IA that something is done, but Randy and others argue that is silly. The value of IA is to encourage commonality, and just saying that something should be done has no effect on commonality. "Goodness" is not the job of the Standard; one presumes that implementers will provide functionality that their customers need. So no IA at all for (2).

(3) Bob notes that Exception_Information should include the exception name, so there is a problem here. Randy notes that since we already added Wide_Exception_Name, it would make sense to do the same for Exception_Information.

Exception_Message doesn't need encoding; whatever is passed in is passed out. There even are people who encode binary in these messages. Yikes. Trying to impose an encoding could break such binary encodings, among other tricks. As with IO, any interpretation of these strings is likely to lead to trouble.

We decide that it is too hard. Current practice is to use UTF-8 implicitly here (and often in other cases); but anything we do here might break this. We also have to avoid breaking existing programs that use Latin-1 strings.

Exception_Information includes the Exception_Message, so something that universally will work is impossible. Wide_Exception_Information would be OK for the implementation-defined parts, but how to convert the Exception_Message is unknowable. We would need more information as to the representation of the underlying string of the Exception_Message in order to do this (a common problem throughout the Standard), so a proper fix will have to wait for a future Ada standard.

So do nothing here, either.

(4) So do nothing here as well; the same sorts of arguments apply.

(5) A.4.9(14.3/2) has the wrong version should be (14.3/3).

Tucker thinks we should have a “hash” version as well. He also thinks these should be defined in Ada.Strings; these are very similar to the "Case_Insensitive" routines that were previously defined.

Fix A.4.7(29/2):

“For each of the packages Strings.Fixed, Strings.Bounded, Strings.Unbounded, and Strings.Maps.Constants, and for {library} functions Strings.Hash, Strings.Fixed.Hash, Strings.Bounded.Hash, and Strings.Unbounded.Hash, the corresponding wide string package {or function} has the same contents except that”

Probably also need to do this in A.4.8 somewhere.

Tucker would like to change the definition of Equal_Case_Insensitive to use case folding. We think that is the same (but we'll need to check the Unicode tables to be sure). Then we can “just” add these to A.4.7 and A.4.8 in the normal way.

Approve Intent: 6-0-4.

AI05-0286-1/03 Internationalization of Ada

A.4.10(3/3):

Returns True if the strings [are the same, that is if they] consist of the same sequence of characters after applying locale-independent simple case folding,...

In 2.1(16), add the “(CHARACTER TABULATION)” after the 16#09#.

Robert sent a private remark about files using record endings being not allowed. We don't think this is a serious concern, there will be some way to import standard files (such as the ACATS).

The third paragraph of the summary should say “we considered recommending”. “But we were unable to come up with specific advice...”

!proposal “We recommend” ==> “The Swiss National Body...” Better to mention this in the first paragraph, but certainly get rid of “we”.

We should open an AI12 on this topic, and mention that in the discussion. [Assigned AI12-0021-1 – Editor.]

“...a[n extremely misguided] suggestion...”; don't be so negative.

Update the !ACATS test section. The requirement is trivially met by running the ACATS, nothing further can be tested. C-Tests are needed for Wide_Equal_Case_Insensitive and Wide_Hash_Case_Insensitive.

Approve AI with changes: 6-0-4.

AI05-0287-1/01 Some questions on subtype predicates

For problem (1), we agree that we need to do this. The wording is OK.

For problem (2), there is some confusion about whether this is really a problem. In the example (in the AARM Note) the subtype is declared before the type and aspect_specification, so does it really have the predicate? Yes, the subtype does indeed have the predicate (the subtype has any constraints, and we would not want predicates to work differently).

On the wording, is the use of "mention" OK? This is a term that is defined by the Standard, but only for context clauses. So the English use of "mention" is OK, since context clauses aren't involved. The wording is OK as a whole.

The restriction proposed for problem (3) is needed because a subtype can have dynamic constraints, and we don't want to deal with that in loops. Case doesn't have the problem because those have to be static. Bob says this was the intent. We agree with this wording as well.

For problem (4), some sort of 'Min, 'Max attribute is needed. This is easy unless the predicate is never be true for a given subtype. Thus there needs to be some sort of allowance for a null range.

Steve suggests having something like 'Base that keeps the bounds. Then 'First and 'Last could still work. But Steve wants to ignore the predicate; that doesn't work in general:

```
subtype Set is Integer with Set in 1 | 3 | 5;
```

Here Set'Range_Base'First would be Integer'First; Set'Min would be 1. Big difference.

Someone suggests that the attribute could be illegal if the range is null. It would have to be if it is on a one-element enumeration type.

The attributes would only work on static subtypes with a (static) predicate.

Steve suggested 'First_Valid and 'Last_Valid. Any static subtype could use it. The attributes would be illegal if there is no valid values for the subtype.

In !proposal, fix “memberhips”.

Bob wonders if we need 'Range_Valid. Ed and Tucker say this would be confusing, because it would not be holey. Forget this one.

Straw poll on defining these attributes: 4-5-1.

This could be added later without adding any incompatibilities, and implementers can implement them now. It seems like a good idea, but many people feel that it is too late now for new “good ideas”.

Steve will write-up an AI12 on this topic. Drop number 4 from this AI. [This eventually turned back into an Ada 2005 AI: AI05-0297-1 – see below – the ARG changes its mind as often as a blonde bombshell – Editor.]

Approve AI with changes: 10-0-0.

AI05-0288-1/01 Conformance of formal access-to-subprogram types

Fix typos:

!question: There is no [any] language rule...

Is a fix [is] needed? (Yes.)

Text_IO.Put_Line (Natural{'Image} (N));

GNAT does not accept this example (a compiler bug). Thus the incompatibility is even less likely. (Later, Ed reports he's not sure why.) Tucker reports that his compiler does in fact do this as specified in the Standard.

Approve AI with changes: 10-0-0.

AI05-0289-1/01 Invariants and in-mode parameters whose value is changed

Invariants and **in** mode changes are completely under control of the programmer of the abstraction. Randy thought that a Note to this effect is valuable.

Erhard asks "What is so special about **in**?" We know that people can change values, so why are these special?

Erhard suggests that most **in** parameters could eliminate the check easily. We disagree, because if the parameter is an index into a global table, that table could have been changed.

It is noted that Text_IO is an example of a package that uses a lot of **in** parameters. It's too late to convince people to change their programming style.

Tucker announces that he's become convinced by Erhard. We generally err on the side of safety when making checks. And the Text_IO model is common. Moreover, **in out** is not very compatible (can't pass functions). Tucker thinks that at least some invariants can be eliminated, because of hiding.

Straw vote: checks on in: 6; no checks on in: 2; abstain: 2.

So invariants will be checked on all parameters (on the way out). Drop "**in out** or **out**" from 7.3.2(19/3). And we don't need the proposed note (7.3.2(25/3)).

Approve AI with changes: 7-0-3.

AI05-0290-1/02 Improved control over assertions

Erhard suggests that Assertion_Policy be made to work similarly to Suppress (including an Unsuppress form). Ignore = Suppress; Check = Unsuppress.

Randy had suggested that assertions are checks, and Assertion_Policy (Ignore) is equivalent to Suppress (Assertion_Check);

Bob wonders why you have to make Suppress always be erroneous. He doesn't see a reason that some checks cannot be "Ignore". Range checks, for instance, have a well-defined semantics if they are not made and they would fail (the value is invalid).

We agree that we need suppress-like semantics; and we need check names. Suppress All_Checks also turns off assertions (in any case).

Geert suggests that for verification purposes, he wants to force all assertions off; he doesn't want to have to verify them. That requires a real ignore requirement, as opposed to the "permission" to ignore that we have for Suppress.

Randy suddenly remembers that Suppress scoping only works because it is a permission, not a requirement. There are cases where Unsuppress would require an implementation model not used by any compilers (but it works so long as Suppress and Unsuppress use the same model). [A solution would be to drop the requirement of Suppress "propagating" from a package specification – but that would require a lot of new wording.

Proposal #1 is OK. In addition, Invariants should be checked based on the policy of the type (they must be if we add scoped policies).

Proposal #2: Tucker thinks this is unnecessary if the check is based on the type. Randy is not so sure, because of Invariant'Class.

Erhard suggests that for Post'Class with a policy of Check, that postcondition should be enforced on any derived routine. Dispatching should still check this, even if the derived routines themselves have a policy of Ignore.

More generally, whatever you inherit that is required to be checked, cannot be ignored. This at least has to be true for any 'Class assertion. Then we don't need this unspecified at all (any version).

Proposals #3 and #4 are about how checks are suppressed/ignored; we've previously discussed them. For Proposal #5, Randy notes that you might want to change some of the container library specifications to use predicates, but you can't change the exceptions.

Tucker suggests an aspect “raises_exception” rather than new syntax.

What about cases where there are multiple checks that raise different exceptions? Predicates can be stacked with multiple subtypes. This doesn't work for Preconditions. We would need to allow multiple preconditions. That makes this a harder problem.

We can do this compatibily in the future. It ought to be given a high priority for Ada 2012. So make this into an AI12. [Assigned number AI12-0022-1 – Editor.]

It appears that if Assertion_Policy is Check on a compilation unit, that would be sufficient to define libraries where the client checks (Predicate, Pre) are always on. But we need to verify that “partition-wide” options are not given priority over “compilation” (at least in the standard mode).

Erhard will try to make a new proposal.

Tucker suggests using the check names as part of the pragma:

```
pragma Assertion_Policy (Pre => Check, Post => Ignore, ...);
```

AI05-0290-1/03 Improved control over assertions

On Saturday, we study the proposal that Erhard has sent.

Discussion occurs on a rule that Erhard copied from 11.5(7.2/2). (The second paragraph of "Static Semantics") “generic instantation” means syntax “generic_instantiation”. Add “entire” before instance. Fix both this and the new one.

“suppresses the check” is a bad idea, because that makes it optional.

If the {relevant} assertion policy in effect at the point of a subprogram or entry declaration is Check,

This occurs in a bunch of places.

We consider moving the “Ignore” text to the Assertion_Policy section from the various assertion definitions. Add something about Implementation-defined here.

Tucker suggests:

> If required by the assertion policy (see 11.4.2) in effect at the point of a subprogram or entry declaration,

For every case “is Check”.

Erhard wants “explicit” declaration:

> If required by the assertion policy (see 11.4.2) in effect at the point of an explicit subprogram or entry declaration,

We discuss the idea that ancestor's checks are always performed on descendants if the policy is Check for the ancestor. That does not follow naturally from the wording. Tucker thinks that “applies” might work if ignored class-wide preconditions don't “apply”. We also need to get rid of “enabled”, which is only used once anyway. 6.1.1(19/3).

Steve worries the Inline rule has a problem for Unsuppress. Randy suggests that the rule applies only to truly inlined subprogram bodies. “...also applies to the subprogram body, if inlined.”

Steve comments that it is too hard to implement this rule. If the body has dropped the checks, it's impossible to put them back later.

Tuck suggests this only apply to Suppress. Randy worries that if Unsuppress applies to the body, and Suppress applies to the call, the Unsuppress will lose, and this is wrong.

It is better that we just drop this permission and make Inline semantically neutral as based on whether or not inlining is performed. (That is, the body determines, always.) So drop the second sentence of 11.5(7.2/3).

"smallest enclosing" should be "innermost enclosing".

Drop the hyphen "-" from "implementation defined".

Bob notes that Pre'Class is not allowed as a pragma argument identifier. Erhard suggests Pre_Class.

All_Checks should be exactly the same as Assertion_Policy (Ignore) at the point of the pragma. Steve objects that suppress should always be a permission.

Give back to Erhard and Tucker.

AI05-0290-1/04 Improved control over assertions

On Sunday, Tucker explains the latest proposal.

Steve points out that the 3.2.4 change breaks Membership/Valid. The semantics of those operations should not be changed by the assertion policy.

Erhard suggests that we just leave the predicate as it is; and then check based on the policy at the point of the subtype. Randy notes this works differently than Pre'Class, where we want to inherit.

Tucker wonders about "`subtype T is S;`" Does this change the Assertion_Policy?

Tucker thinks that we really do want the inheritance. Which means that we need to have both...

Ed complains that this is defining a subset of a type, and it doesn't make sense for it to be different. So it doesn't seem like it makes sense to turn it off.

Steve mentions that if you turn off a predicate check, you could have a qualified expression neither raise an exception nor be in the type:

```
A := Qual'(Foo);
if A not in Qual then
   Put_Line ("True");
end if;
```

Straw vote.

(1) Membership/Valid/case/looping uses one predicate, checking potentially uses a different predicate.

(2) One predicate, on or off checks. The final subtype with a predicate controls the checking.

(3) One predicate, on or off checks. The final subtype controls the checking (always).

(4) No assertion policy for predicates; suppress for predicates (including erroneous).

(5) No assertion policy for predicates; non-erroneous suppress for predicates (but this requires one of (1), (2), (3)).

(6) No assertion policy for predicates.

Erhard complains that (5) is a problem: if you have a case statement that doesn't have the choices, what happens if there is no check? That is required to be checked 5.4(13) – note that is not a suppressable check.

There is some support for predicates to work like constraints. Erhard and Randy object, because it makes it impossible to use predicates to check a library's required conditions. Using only preconditions is more complicated.

Tucker suggests another option: the checking is based on use of the subtype names, (such as in the parameters).

It would be on the use of the subtype name (in some declaration: object, component, subprogram, type conversion).

Steve wonders if there is a conformance problem if the policies differ for a subprogram. So this would need to affect subtype conformance.

6.1.1(31/3) Randy wonders how this wording gives us Pre'Class checking that depends on the preconditions of the declared called subprogram, rather than the invoked subprogram.

Tucker says this is about the "explicit" aspect_specification. We need a note here to explain.

Randy asks why "assertion kind"? This should be "*assertion*_aspect_mark" to match Tucker's syntax.

Randy asks what happens if the "innermost declarative region" has two policies. Which one wins? The last one. "Region" has the right definition, so we should use that.

> If multiple Assertion_Policy pragmas apply to a given construct for a given assertion kind, the assertion policy is determined by the one in the innermost enclosing region for a pragma Assertion_Policy specifying a policy for the assertion aspect. If no such Assertion_Policy pragma exists, the policy is implementation defined.
>
> ...
>
> The *policy*_identifier Check requires checking that assertion expressions of the given aspect be checked that they evaluate to True at the points specified for the given aspect; ...

Randy notes 4.6(51/3), 4.6(57) also need work for predicates; a predicate is a check and that needs to be reflected in the wording.

Approve intent: 5-0-5.

Tucker will rewrite another draft; we'll approve this AI by letter ballot when it is done.

AI05-0291-1/01 Target object needs to include synchronized interfaces

Typo in note following 9.4(11.9/2):

"and considered {to be} of mode **in**, so we have to disallow access-to-variable rather than allow access-to-constant."

Randy notes that there is a minor compatibility problem: **access** T would have been allowed in Ada 2005. Adding "constant" would fix that. This should mentioned in the !discussion.

The main paragraph is 9.5(2/3), not 9.4(2/3).

In 9.5(2/3): "...a name denoting {a prefixed view of} such a..."

In addition, a primitive subprogram of a limited interface can be implemented by an entry or a protected subprogram, and a name denoting such a subprogram also identifies a target object.

Randy and Tucker get into a discussion of the proper wording. Everyone else seems to have gone to sleep. Randy and Tucker will take this off-line.

9.7.2(4/3) should be 3.3/3. (A paragraph number bust.)

Gary asks about the wording of 9.5(6.1/3): "...subprogram renam[e]{ing{, the name of the renamed {entity}[item]..."

AI05-0291-1/02 Target object needs to include synchronized interfaces

To Be Honest: [Thie]{This}

In AARM 9.4(11.9/3) "...so we have {to} disallow..."

In bracketed note: ...timed_entry{_}[ ]calls, and...

...non[-]prefixed... Tucker objects because this is "non" "prefixed view" (which is a term). Can this be reworded?

We try for a while without conclusion. Gary withdraws his suggestion. (Add the "non-prefixed" to the list of exceptions in the hyphen AI, AI05-0293-1.)

We should add an AARM note after 9.5(7.1/3) to mention that this rule does not apply to calls that are not to a prefixed view (of a limited interface's primitive operations). In that case, the normal parameter passing mode checks prevent passing constant protected objects.

Drop the "Randy Humm" note.

The discussion that starts with "We have to clarify" should be edited slightly since we didn't clarify anything.

Approve AI with changes: 7-0-3.

AI05-0292-1/01 Terminology: "Indexable type" is confusing

Erhard would like to delete the second syntax production, then merge the 3/3 wording into a single sentence ("The expected type for the *iterable*_name is any array type or any iterable container type.") Then "The second form of iterator_specification is called an array component iterator if the type of the *iterable*_name is an array type. The iterator_specification is called an container element iterator otherwise."

Change all of the *array*_names to *iterable*_name.

The reason Erhard would like to make this change is that there are two identical productions; they're only different because of resolution rules. Tucker thinks this is a bad idea. Much argument ensues. Tucker says that there is plenty of ambiguity in the grammar already. Erhard argues that while there is ambiguity in the existing grammar, it isn't in adjacent productions. And he notes that the changes needed are quite small. Tucker is still dubious.

Erhard agrees to write it up so that the amount of change is obvious.

There is a brief break while another proposal is considered. After that, Erhard has sent a proposal:

> One alternative is enough and the Grammar and RM-verbiage can be simplified:
>
> | defining_identifier [: subtype_indication] **of [reverse]** *iterable*_name
>
> The first form of iterator_specification is called a *generalized iterator*. The expected type for the *iterator*_name is any iterator type. In the second form of iterator_specification, the expected type for the *iterable*_name is any array or indexable container type. If the *iterable*_name denotes an array object, the iterator_specification is called an *array component iterator*; otherwise it is called a *conntainer element iterator*.
>
> Change *array*_name to *iterable*_name everywhere.

Minor improvement to the first sentence and bug fix:

> For the first form of iterator_specification, called a *generalized iterator*, the expected type for the *iterator*_name is any iterator type. For the second form of iterator_specification, the expected type for the *iterable*_name is any array or iterable container type. If the *iterable*_name denotes an array object, the iterator_specification is called an *array component iterator*; otherwise it is called a *conntainer element iterator*.

Approve AI with changes: 10-0-0.

AI05-0293-1/01 Remove unnecessary hyphens

received 11-xx-xx should be 11-11-29.

Approve AI with changes: 10-0-0.

AI05-0294-1/01 Update the glossary

Remove “that gives a boolean expression” in all four assertions (predicates, invariants, preconditions, postconditions).

Invariant. A invariant is an assertion [that gives a boolean expression] that is expected to be True for all objects of a given private type when viewed from outside the {defining}[enclosing] package.

Assertion. An assertion is a boolean expression that appears in any of the following: a pragma Assert, a predicate, a precondition, a postcondition, an invariant, a constraint, or a null exclusion. An assertion is expected to be True at run time at certain specified places.

Aspect. An aspect is {a specifiable property of an entity}[any of a miscellaneous set of properties of entities]. An aspect may be specified by an aspect_specification on the declaration of the entity. Some aspects may be queried via attributes.

Iterator. An iterator {is a construct that is} [may be] used to loop over the elements of an array or container. [Generalized] iterators may be user defined, and may perform arbitrary computations {to access elements from a container}.

Container. A container {is an object that} [type is one whose objects] contain other objects {all of the same type, which could be class-wide}. Several predefined container types are provided by the children of package Ada.Containers (see A.18.1).

”Indexable {container} type”.

Steve asks about additional terms. “storage pool”, “stream”.

Tucker would like “iterable container type”.

Steve will provide these later.

Saturday afternoon, we considered Steve's extra definitions for the glossary:

> Storage pool.
> Each access-to-object type has an associated storage pool object. The storage for an object created by an allocator comes from the storage pool of the type of the allocator.
>
> Stream.
> A stream is a sequence of modular integer values which can be used, along with the stream-oriented attributes, to support marshalling and unmarshalling values of most types.
>
> Iterable container type.
> An iterable container type is one that has user-defined semantics for iteration, via the Default_Iterator and Iterator_Element aspects.
>
> ====
>
> The wording for the "Iterable container type" entry is deliberately parallel with the existing wording for the "Indexable container type" entry.

Storage pool: is this too vague? No, this is how it is described in the normative standard.

Add a sentence about subpools:

Some storage pools may be partitioned into subpools in order to support finer-grained storage management.

A stream is a sequence of elements that can be used, along with the stream-oriented attributes, to support marshalling and unmarshalling of values of most types.

“semantics” is funny; change to “behavior” (in all of indexable container type, iterable container type, and reference type).

Approve AI with changes: 9-0-1.

AI05-0295-1/02 Improve presentation of aspects

Improve the changed wording (in particular, the phrase "nonconfirming representation value" was disliked, so the wording is rearranged to avoid it):

Modify 9.10(1/3):

> Otherwise, two nonoverlapping objects are independently addressable except when they are both parts of one composite object for which a nonconfirming [representation] value is specified for any of the following {representation} aspects: Layout, Component_Size, Pack, Atomic, or Convention; in this case, it is unspecified whether the parts are independently addressable.

Modify 13.1(13.1/2):

> A type_declaration is illegal if it has one or more progenitors, and a nonconfirming [representation] value was specified for {a representation aspect of}an ancestor, and this [representation aspect] conflicts with the representation of some other ancestor. The cases that cause conflicts are implementation defined.

Modify 13.1(0.1/3)

> ...determine [to] other properties...
>
> [Editor's note: This error occurred only in the !wording of the AI; the Standard was correct.]

Modify 13.1(0.1/3) [2nd paragraph]

> ...of {a}[an] representation item...

Replace 13.1(13/1) with:

> If a specification of a representation or operational aspect is not supported by the implementation, it is illegal or raises an exception at run time.

Modify 13.1(18.a/1): “a[n] representation aspect”.

Modify 13.1(18.2/2): “A{n} aspect_specification...”

The second 13.1(20.a) should be 13.1(20.b).

Modify 13.1(21.a.1/2):

> ... representation {aspect} value ...

13.1(22): Tucker claims that “containing” is well-defined for a “specification of a representation aspect”. So this can be simplified a lot. The “specification” is some syntactic construct. Randy is dubious but he is overruled.

> An implementation need not support the specification for a representation aspect that contains nonstatic expressions, unless each nonstatic expression is a name that statically denotes a constant declared before the entity.

Approve AI with changes: 8-0-2.

AI05-0296-1/01 Freezing of actual subprograms which have parameters of formal incomplete types

[Aside: Index “heart of darkness” for the accessibility rules.]

Randy had worried about this applying to a child instance. For instance:

```
generic
   type Inc;
package Gen is
   ...
end Gen;

generic
   with function Is_Inc (A : Inc) return Boolean;
package Gen.Child is
   ...
end Gen.Child;

package Inst is new Gen (Foo);

package Inst2 is new Gen.Child (Bar); -- Should not freeze Bar.
```

Tucker notes that the same should apply to a nested generic as well.

Tucker says just drop the “of the same generic”, as it can't be “formal” outside of the scope of the generic.

Tucker and Randy start wondering about “tagged incomplete”. It's possible to call such a formal subprogram (so long as the object comes from outside the generic), and that requires it to be frozen.

We have to make the rule apply only to “untagged formal incomplete types”.

```
generic
   type Inc is tagged;
   with function Is_Valid (O : Inc) return Boolean;
package Gen is
   procedure Do_It (O : Inc);
end Gen;

package body Gen is

   procedure Do_It (O : Inc) is
   begin
      if Is_Valid (O) then -- Legal, better be frozen.
        ...
      end if;
   end Do_It;

end Gen;
```

Fix summary to drop “of the same generic”. Randy will add some discussion on the above topics.

Approve AI with changes: 7-0-3.

AI05-0297-1/01 First_Valid and Last_Valid attributes

[The AI12 split from AI05-0287-1 eventually changed back into an AI05. - Editor.]

On Sunday, we start discussing the AI12 proposed by Steve.

Robert Dewar has indicated that he intends to implement these in GNAT soon. Gary comments that he doesn't like “First_Valid” and “Last_Valid”. He would prefer “First_Member” and “Last_Member”. The names are more important if implementers are going to copy the (unfinished) proposal; there will be pressure in the future to use these names.

Ed suggests “First_Value” and “Last_Value”. Bob had previously suggested “Predicate_First” and “Predicate_Last”.

Steve complains “First_Value” is too much like “First”, the difference isn't obvious.

Randy comments that we have this problem because we're unwilling to use “First”. It's thought to be confusing (two possibilities – first range value, first predicate value). So there is still no interest in doing that.

Tucker says that he does not like Bob's idea because it works on static subtypes that don't have a predicate.

Probably should put these after 'First in 3.5.

We need to settle the names somehow. Let's have a "can live with" straw poll.

> “First_Valid”: 10
> “First_Member”: 3
> “First_Value”: 7
> “Predicate_First”: 4

Some members are not happy with this outcome and want to add more choices. So we start discussing “Min” and “Max”. Randy had complained that this would be a sort of overloading not allowed for user code. Tucker says that we could define these as functions and then all would be well definitionally.

Tucker says that indeed, he'd use both at the same time. “S'Min(S'Min, 7). He's overplayed his hand -- Randy and Steve gag on this suggestion.

We try another straw poll: "dislike" a name.

> “First_Valid”: 3 half votes
> “Min”: 4
> “Min_Val”: 3

Straw poll for preference:

> “First_Valid”: 6
> “Min_Val”: 4

Geert says that it is appealing that “Valid” is true for “First_Valid”.

Five people vote to close this discussion of the attribute names (nothing new is being added).

Perhaps we should add these to the language now. The only contentious issue was the names of the attributes, and if we're going to have to live with these names anyway, there doesn't seem to be any important reason not to put them in now. Straw poll shows support for that position. [I didn't record the vote of this straw poll - Editor]

Where to put these attributes: 3.2.4, 3.5, or 13.9.2?

Another straw poll, preference for location of definition:

3.2.4:2
3.5: 7
13.9.2:1.

Put into 3.5.

These are defined only for static subtypes, so that dynamic predicate have been left out.

This is the most restrictive version of these attributes, so we can expand this in the future if that proves necessary.

Tucker proposes that we put it in 3.2.4, because the restrictions are all about predicates. Steve points out that there is no mention of predicates in the description. [But that's a bug – Editor.]

It fits best in 3.5.5 (operations of discrete types).

These are just the (static) subtypes that you can loop over.

Tucker would like an AARM note in 3.2.4, to mention that these exist.

Approve AI with changes as an AI05: 8-1-1.

Bob opposes thinking about "new features" created at the last moment, much less adding them to the Standard.

Responses to National Body Reviews

Add to the Swiss National Body response that AI12-0021-1 was opened for the remaining issues; so that they can be addressed in a future version of Ada.

Brad asks whether the response to Denmark (in the Ada-Europe review) is questionable. Shouldn't the other arrays also have a specified Component_Size rather than Pack?

Randy notes that Pack, or for that matter any specified representation, is dubious for these types because they're required to be compatible with that used by the C-compiler. Obviously, the Ada implementation will have to do whatever is necessary for that.

There is concern that this one type is inconsistent with the others. Randy argues that this is already true, since this type has a Component_Size clause and no one is suggesting removing it. So it will never be consistent with the others. There is also a concern that a machine with an unusual byte size will have problems without the Pack [but it is just as likely to have problems *with* the Pack – Editor.]

Put the Pack back B.3(23/3). But leave the Component_Size. Change the response to the Denmark comment to reflect this change. This change will be made in AI05-0269-1, as item (62).

Jeff would like the response to the German Ada-Europe comment to include a reference to 13.1(9.2/3) and 13.1.1(17/3) and especially 13.1(9.d/3) [which explains the model].

Brad asked that no answer be developed for 3.8.1 comments in the Canadian comment, as these are obviously wrong. He indicates these were his personal comments, and he (and therefore Canada) does not need a formal response.

Tucker will write a short answer to the SIGAda comment. Basically, we've stayed to the original intent of improving containers and assertions. In addition, WG 9 has approved the scope and timing of Ada 2012.