Minutes of the 5th ARG Meeting
14-16 November 1997, St. Louis

Attendance: Erhard Ploedereder, Stephen Michell, Gary Dismukes, John Barnes, Bob Duff, Pascal Leroy, Vincent Cellier, Ted Baker, Norman Cohen, Robert Eachus, Kiyoshi Ishihata, Magnus Kempe, Mike Kamrad, Joyce Tokar (for the first hour)

Meeting Summary:

The meeting convened on the 14th at 2 p.m.

Erhard briefly summarizes the WG-9 resolutions on the AIs. He also notes that the funding problems show their effect; quite a few AIs have been resolved but not yet written up. He therefore proposes a mode in which AIs are assigned explicitly to ARG members for completion. Joyce reports that some funding for Bob Duff may be forthcoming from the ARA.

The AIs were discussed during the meeting in (almost) numerical order. The entire agenda was covered.

The following AIs were **approved without change**:

AI-76 - String literal constraint ramification is correct (11-0-0)
AI-84 - Questions about Generic_Elementary_Functions (10-0-1)
AI-96 - Sparse case statements (11-0-0)
AI-122 - Use_type_clause Where First Subtype is Not Directly Visible (11-0-2)
AI-129 - Discriminant Inheritance for Private Extensions (11-0-1)
AI-132 - Exception Raised at End of Stream (11-0-1)
AI-138 - Null String Literals when the Index Type is Modular (10-0-2)
AI-151 - Bounds of string returned by Ada.Strings.Maps.To_Range (13-0-0)

The following AIs were **approved as "No Action":**

AI-144 - Pragma List (11-1-0)
AI-146 - Controlled Components are Finalized (10-0-2)

The following AIs were **previously approved**, editorial review awaiting AI completion:

AI-105 - Extra negative value
AI-133 - Controlling bit ordering
AI-148 - Requeue of protected entry calls
AI-165 - Recursive use of task attributes isn't considered
AI-166 - Parameterless_Handler values designating default treatment

Letter ballots were requested on the following approved AIs:

AI-51 - Size and Alignment Clauses for Objects (8-0-2)
AI-109 - Size and Alignment Attributes for Subtypes (7-0-3)
AI-160 - Daylight savings and Ada.Calendar (10-1-2)

The following AIs were **approved with changes or "in principle":**

AI-113 - Exception raised by Month, Day, Seconds in Ada.Calendar? (11-0-0)
AI-116 - Elaboration of task with no task_definition (13-0-0)
AI-119 - Is Normal Termination an "External Interaction"?
(9-2-0 on reversal of summary)
AI-120 - What is the minimal upper bound of type Integer? (10-0-1)
AI-147 - Optimization of Controlled Types (10-1-2)
AI-150 - Uniqueness of Component Names (13-0-0)
AI-152 - Operators not inherited from root numeric types (12-0-1)
AI-156 - Polar implementation of complex exponentiation for neg. exponents (12-0-1)
AI-176 - Access_Check is performed for access discriminants (13-0-0)

The following AIs **were discussed**, votes await AI completion:

AI-58 - Accessibility Rules for Shared Passive Packages
AI-85 - Questions about Append_File mode
AI-100 - Truncation required if Machine_Rounds false?
AI-103 - Storage pools and access types designating task types
AI-117 - Calling conventions
AI-143 - Distinct names for Compilation Units
AI-153 - Picture String Grammar or Composition Rules Need Tightening
AI-159 - Shared Variables in Shared_Passive?
AI-162 - Anonymous allocators and tasks/finalization
AI-164 - Definition of remote access type
AI-167 - Erroneous scalar Unchecked_Conversion?
AI-172 - Optional main subprogram?

AI-173 - Optimizations and the use of ‘Address
AI-184 - View conversion to an indefinite subtype
AI-185 - Branch cuts of inverse trigonometric and hyperbolic functions

The following AIs were discussed but are **controversial and undecided**:

AI-130 - Should no_Local_Allocators disallow nested instantiations?
AI-131 - Interface to C – passing records as parameters of mode "in"
AI-158 - T’Class as generic actual type
AI-161 - Default-initialized objects
AI-168 - Aliased objects can have discriminants modified
AI-186 - Range of root_integer (7-3-3)

The following AIs are still **unprocessed and most seek an AI editor**:

AI-169 - Exceptions raised by Adjust/Finalize -- Missing case
AI-170 - Can an attribute defined in an annex be set in an attribute definition clause?
AI-171 - Elaboration of subtype_indications with per-object constraints (Gary)
AI-174 - Are 'Read and 'Write guaranteed to be "inverses" for predef. types? (Robert E.)
AI-175 - Full conformance of expressions with attributes
AI-176 - Access_Check is performed for access discriminants
AI-178 - Which I/O operations are potentially blocking?
AI-187 - Task attribute operations are atomic but not sequential
AI-188 - The definition of setting a task base priority is too vague
AI-189 - The meaning of the terms "processor", "multiprocessor", and "processing node"
AI-190 - Compile-time vs. Run-time Errors
AI-191 - An OBJECTive View
AI-192 - A library subprgram_body should replace, not complete, an instance
AI-193 - Classwide Adjust and exceptions
AI-194 Typo in Standard_Error Definition

Standard Corrigendum:

Erhard reminds the group that a Corrigendum is required by ISO and that we should work towards a delivery date in 1998. He suggests to have it ready for the November meeting of WG-9, but has some doubts that this is realistic, unless somebody is funded to bring the document together. He considers assigning the AIs to individuals for "massaging" as counter-productive, but a possible last resort.

The Corrigendum will consist of only the binding interpretations, and possibly the presentation AI. Some discussion on format produces several ideas: Norm suggests to use the summary and wording organized by the sections/subsections of the ARM. What about references to the AIs in the Corrigendum, as footnotes to the reader to provide background? This is not the normal approach, but without these references it may be necessary to add more information to the Corrigenda entries.

The opinion is voiced that the real impact of the work of the ARG is the production of AIs and that the Corrigendum will largely be a checkbox to be filled and nothing more. We should minimize the effort to describe the content of the AIs. While everybody agrees on the latter point, there are different opinions on the importance of the Corrigendum, given that the AIs have no official standing by themselves.

It appears that official ISO documents permit referencing non-normative documents which should permit us to make references to individual AIs to minimize the size of the Corrigendum. How much effort to prepare the individual AIs and what information to include? The summary, of course, but how about the question and response or discussion? Are the current statements of questions sufficient to explain the motivation for the AI? If not, then it will take time to produce these questions. Erhard suggests to have some boilerplate statement there ("inconsistent rules in the RM", "missing rule in the RM", etc.) and point to the AI for detail. Action item for Erhard and Bob is to prototype a few AIs as a basis for determining at the next meeting how to handle the remainder of the AIs and for scheduling their completion for the Corrigendum.

"Ada-comment":

The members rightly complain that the ada-comment redistribution mechanism is broken. Erhard is tasked to see to establishing a mode of operation that avoids active involvement of Intermetrics in the redistribution of the comments. Erhard warns that such a system will be much more simplistic, since nobody is willing to transfer the arcane scripts from the Intermetrics site to another site.

Next meeting:

Time and location of the next meeting are briefly discussed. One possibility is to use the weekend before or after the Ada Europe Conference in Uppsala. However, June is deemed to be rather late for the next meeting. As an alternative, an East coast meeting in early April is proposed. Among the members present, none of these alternatives is superior in predicted attendance.

Erhard decides to postpone the decision until funding questions have been answered. He will announce time and location of the next meeting towards the end of 1997. He indicates that he favors a meeting in early April.

Action Items:

Completion of AIs was **assigned to ARG members** as follows:

Ted Baker: AI-119, AI-131, AI-166
John Barnes: AI-113, AI-120
Norm Cohen: AI-130, AI-167, AI-173
Gary Dismukes: AI-158, AI-171, AI-176
Bob Duff: AI-51, AI-58, AI-105, AI-109, AI-116, AI-133, AI-143, AI-150, AI-152, AI-156, AI-159
Robert Eachus: AI-100, AI-117, AI-153, AI-172, AI-174, AI-185, AI-186
Mike Kamrad: AI-162, AI-164
Pascal Leroy: AI-85, AI-161, AI-168, AI-184
Stephen Michell: AI-103, AI-148
Erhard Ploedereder: AI-147, AI-165

There are AIs received but not yet processed, namely: AI-169, AI-170, AI-171 (assigned to Gary), AI-174 (assigned to Robert E.), AI-175, AI-176, AI-178, AI-187-194. **ARG members are sought to volunteer for initial write-ups of these AIs.**

Additional action items:

Erhard : Contact Tuck to cooperate in simplifying/transferring the distribution/logging of ada-comment to sw-eng host.

Erhard: Set the dates and place of the next ARG meeting by the first of next year.

Erhard and Bob: Prototype a few AIs to determine how to handle the remainder of the AIs in Standard Corrigendum form.

The ARG adjourned on the 16th at 4 p.m.

===

Details of the AI review:

In these minutes, „Robert“ always refers to Robert Eachus and „Bob“ to Bob Duff.

AI-58

Bob Duff is assigned to complete this AI as per the Henley meeting.

AI-051

On this AI as well as on AI-109, the group reviewed the history of both to determine why the votes were cast (there have been two) and how the current AIs respond to these votes. It appears that the current AIs respond to all comments made at the Vermont meeting.

Robert thinks that the AI fails to cover non-decimal fixed point objects and believes there are significant customers (such as Boeing on AWACS) that care. It appears this could be handled by changing the last bullet to eliminate non-decimal fixed point types. After much discussion on an example concerning 7-bit fixed point values in special RAM, it was determined that these values should be handled by an array of components and not by stand-alone objects (a.k.a. the St.Louis RAM argument). Therefore no changes need to be made for fixed point.

This led to determining how component sizes are covered by the ARM or the AI. Paragraph 13.5.1(19) sufficiently covers array components but many viewed paragraph 13.3(73) to be not sufficiently clear to cover record components. Hence, add a parenthetical clause to the head paragraph on item 3 stating that it also applies for components. But adding this clause then breaks the rules for stand-alone objects in bullet four. The fix is to make the fourth bullet into a new fourth item applying only to stand-alone objects.

Approved 8-0-2. A letter ballot was requested.

AI-76

Is this AI too trivial to approve, especially since it deals with the AARM, strictly speaking?

Approved 11-0-0. Future examples of this type of comment will not be turned into an AI but put into the AARM category as confirmations.

AI-84

Approved 10-0-1.

AI-85

One goal for adding the append capability was to be able to use the Unix append. Due to the unique nature of Stream_IO, the append will be difficult, since the semantics of

Set_Index cannot be implemented easily (by a seek) on Unix streams. After examining Tuck's response which reflects the way Unix append works, people agreed it works for non-Stream_IO.

Pascal will produce the write-up, including the problems with Stream_IO, since the group felt the issues to be too related to be dealt with in separate AIs, after all.

AI-96

Approved 11-0-0.

AI-100

Assigned to Robert Eachus, who should coordinate with Ken Dritz.

AI-103

There was some discussion about the summary not matching the question. Stephen will update the AI.

AI-105

A straw vote approved this AI at the last meeting. Bob will complete the AI according to the minutes of the last meeting. It will then go out for letter ballot.

AI-109

Erhard starts with an example to illustrate the issue of the 2. paragraph:

```
type first is new integer range 1..10;
for first'size use 8;

subtype second is first'base range 1..100;  -- second'Size = 7
subtype third is second range 1..10;  -- third'Size = 8
subtype fourth is second range 1..9;  -- fourth'size = 4
subtype fifth is first range 1..10;    -- fifth'size = 4
subtype sixth is first'base range 1..10; -- sixth'Size = 8
```

The difference between Third, Forth (and possibly Fifth and Sixth) is a bit of a surprise. (Although Erhard agrees that it is a necessary consequence of other semantics that are based on statically matching subtypes and that size clauses for subtypes would then be disastrous, he wonders whether there isn't a better alternative.)

Bob feels that the main significance of this AI is to make it clear how packing should work. Unlike in Ada83, the AI interpretation of the Pack pragma does not require the explicit setting of component type sizes to perform packing. Erhard spent time determining that the Pack pragma does not do a "deep" pack, namely, it only packs the selected array type and does not pack the component type. Bob explains that the main purpose of the AI part on packing is to mandate that the subtype Size, not the object Size of components guides packing. I.e., a four-element array of 4-bit packed nibbles or of 4-bit integers fits into a word.

Pascal states that the second paragraph under Recommended Level of Support is really a language rule and should be so stated in the summary. Also add the phrase "for integer types" to the third-to-last paragraph of the discussion.

Approved 7-0-3. A letter ballot was requested.

AI-113

There is no reason why the functions in question shouldn't conform to how Split handles this issue, namely, by raising Time_Error. And, there is good reason to change to these semantics, since the implementation model of a separation of the time component values may not, and often will not, apply, in which case it is difficult to see how the functions in question can yield defined results (the leap year problem). Change the class to be a binding interpretation. Assigned to John (according to his recommended wording) for write-up.

Approved 11-0-0. The AI will go out for editorial review.

AI-116

Bob will change the class to binding interpretation.

Approved by consensus (13-0-0).

AI-117

Assigned to Erhard to finish the work from the Henley meeting.

AI-119

The Henley meeting voted (9-2-0) to reverse this AI. Rewriting this AI is more work than just removing the "not" in the summary. Assigned to Ted.

AI-120

Bob points out that the C standard requires support for integer types with at least the 32-bit range, wheras Ada does not. It is embarrassing for Ada that not all compilers are required to support „type T is range 1..1_000_000“. This made no points with the meeting. It was agreed that the summary should state that an implementation "need not provide more" than 3.5.4(21) and that it should only raise Constraint_Error not Storage_Error in case of a large index range that exceeds the range of the largest supported integer type.

Approved with these corrections, 10-0-1. Assigned to John. The AI will then go out for editorial review.

AI-122

The issue is the "transitive import" semantics that the use type clause provides, since it creates direct visibility to the operators found in the Types package which is not even "with"ed by the package at hand. While this seems to be a somewhat surprising consequence of the language rules, which takes a long argument to prove, the group nevertheless felt that this is what the users wanted.

Approved 11-0-2 on the next day.

AI-129

Approved 11-0-1, despite Pascal thinking it deserved no-action.

AI-130

Erhard argues, in a separate e-mail message, against the current position of the AI because it breaks the contract model and requires too much link time mechanism to enforce the restriction. Instead enforcing the currently stated restriction of only library-level instantiations will enforce the contract model by doing a compile-time check.

Pascal and Robert then noted that there is a much deeper problem here. Record components with default initialization, protected objects with initialization of their state, default values for parameters all share the property that allocations might happen that are not apparent at source level. The current semantics of No_Local_Allocation does not cover these cases at all. Since these types can be private, the privateness principle will need to be broken for compile-time checks. The group could not come up with "circumstantial" restrictions that would otherwise ensure the absence of such implicit allocation without being extremely and unacceptably over-restrictive. In passing, it was noted that the strict predicate of No_Local_Allocators is actually value-dependent (and hence undecidable), since a default-initialized component might be located in a non-existent variant of a constrained record object.

There is a concern that this sets a precedence for exposing more implementation information about types, which violates the spirit of abstract data type information hiding (and will cause problems in some compilers). But the Restrictions facility is inherently implementation focused (especially restrictions like this one that deal with nested usage, No_Nested_Finalization and No_Task_Hierarchy being the other ones) and therefore some exposure of the implementation details of imported types (such as Text_IO that uses allocated data for file_type) must be made in order to enforce the Restrictions. This will affect semantic analysis involving private types because now uses of private types will need to check that they adhere to restrictions relating to previously hidden features of the private type.

The No_Task_Hierachy and No_Nested_Finalization restrictions share the same problem, since components might be tasks or of controlled type. It is not clear what the language can do about the portability of these three pragmas and still be effective.

It was even suggested that these three restrictions should not be in the language, after all, since the benefits do not justify the costs. ASIS-based analysis tools were mentioned as an alternative restriction-enforcing mechanism.

Norm then suggested to take a step back and reformulate the semantics of the pragma from the viewpoint of intended usage, and leave the formulation of secondary restrictions to the implementation. He will rewrite to emphasize usefulness over portability; he will coordinate with Erhard on the write-up.

AI-131

Erhard cites the WG-9 resolution of the subject of interfacing AIs and once again questions the summary of this AI because it doesn’t follow the C language standard. Someone reminded the meeting that Robert Dewar has influenced the current decision because of the large base of users for GNAT. Erhard argues that this being an Implementation Advice, GNAT would be free to ignore it, but he believes that the current advice is plainly wrong for an ISO standard and should be corrected. Bob replies that we should not make rulings that we believe compiler vendors will ignore, and the ACVC will fail to enforce. Ted suggests that the user should be encouraged to use a pragma, like the one described in the AI, for forcing the convention for parameter passing. Gary proposes that this AI which deals with both implementation advice and language interfaces may be best handled by the newly proposed WG-9 rapporteur group dealing with interfacing issues. There were objections to the currently described pragma C_Pass_By_Copy which focuses on data size; instead there is a preference for making the pragma convention-based, making it apply to all usages of the Convention C. Ted will rewrite this AI to support the Convention approach.

AI-132

Approved "as is" 10-0-1 in lieu of a letter ballot.

AI-133

Previously approved at Henley but the rewrite was not completed. Bob will complete it according to the directions of the Henley meeting. Norm intends to write an opposing position.

AI-138

Everyone agrees with the conclusion but there is some discussion about the relevance of the use of the term "predecessor" and its (lacking) definition in the RM.

Eventually approved as written, 10-0-2.

AI-143

The language goal is maximum freedom for vendors to implement their libraries. Apparent agreement is that this restriction is a post-compilation rule which may also be enforced during compilation. Due to the arduous reasoning of the exchange, the AI class is changed to a ramification. Bob will complete the write-up.

AI-144

Classified as no action, since the issue was felt to be pathological and rather unimportant, 11-1-0.

AI-146

Classified as no action, 10-0-2, because the commentor has received sufficent response via ada-comment and no real issue has been substantiated.

AI-147

Erhard distributed a revised AI. He explains his reasoning.

Gary and Pascal point out that his reasoning is flawed with respect to initialization, since side-effects of default-initializations of record components are not allowed to be optimized out of existence, so that there is a significant difference between a variable declared without initializing expression and subsequent assignment, and a variable explicitly initialized. Consequently it would be strange that initialization (for records) would differ from initialization of non-limited controlled types in that controlled initialization can be optimized away while the record initialization can't. Eliminating the initialize-finalize pair will lose those side-effects. There is strong feeling that these side-effects shouldn't be lost and therefore Initialize-Finalize pair shouldn't be eliminated.

This leaves optimization that eliminate Adjust-Finalize pairs. The user can count on the Initialize side-effects but not the Adjust effects. The summary is revised to remove the first Initialize from the third paragraph. John comments that the identifier "O" should not be used in AI examples, because it is easily confused with "0".

Approved 10-1-2. (Stephen opposed with his HRG hat on.) This significantly reduces the write-up which Erhard will revise.

A follow-on discussion brings up Randy's recent comment on the strangeness of the last sentence of the Implementation Permission (which appears to allow elimination of an adjustment without also eliminating a finalization). Erhard will reflect this comment in the write-up. (This last sentence is indeed rather ambigously formulated and will simply be subsumed by the AI.)

AI-148

No discussion, assigned to Stephen for completion per Henley results.

AI-150

The illegality is caused by the private region in the child; at that point both components become visible and break the rule in 8.3(26). The example in the AI needs to be fixed to have the type derivation in the private part of the child unit. The AI fixes an anomaly that pertains to just child packages. The AI should probably be explicit that it pertains to this anomaly. The summary is stated too strongly; it shouldn't disallow homographic functions in generic instances. Robert tries to come up with an example to show that the problem might also apply in generic instantiations. His example is a sequence of multiple instantiations, each mixing in a homographic visible component to the results of the previous instantiation. The second such instantiation is already illegal by existing rules. The root cause of this AI - late visibility of the full type declaration overlapping the visibility of the added component -- truly arises only for child packages. It does not arise for nested packages, since the rules are such that the derivation from a private type forever "hides" its private components. It is suggested that the current summary is really a design principle; additional wording should be added to be more narrowly applied to record components and to derived types in child packages. Norm recommends an additional reference to paragraph 3.4(14) to add more depth to the reasoning.

The intent is approved 13-0-0, pending the actual rewrite by Bob.

AI-151

After some brief discussion to orient ourselves and digest the rather difficult summary, approved 13-0-0.

AI-152

If the ARM is taken too literally, obviously illegal cases become legal. Replace "by fiat" in the summary with "by specific language rules" to be more "politically correct".

With this change the AI is approved 12-0-1. The write-up is assigned to Norm.

AI-153

Assigned to Robert E. for producing a binding interpretation (or uninformed musing).

AI-156

This AI corrects an obvious cock-up, regarding the exponentiation. The phrase "the same" in the summary is oblique and needs to be tightened. It was decided to move the wording in the wording section to the summary to replace the body of the sentence and to place a reference to the summary in the wording section.

Approved by 12-0-1 with this editorial change. The write-up is assigned to Bob.

AI-158

There was a lengthy discussion to (re)discover the rules that currently apply to generics and their instantiations with or without class-wide types.

Erhard discovered that the last statements in his write-up (distributed to ARG, but as yet missing from the !appendix) about the need for ‘Class in the generic formal subprogram parameters was not a compelling necessity (since substitution by the actual type will cause this to happen anyway).

A couple of examples sketched out at the blackboard to illustrate the issues were the following:

```
generic
 type t(<>) is new xxx with private;
 with function create return t is <>;
            -- under current rules, this is not a primitive (dispatching) operation
            -- but under the new rules this could be matched by a primitive operation with
            -- dispatching semantics for calls on create if the actual for t is class-wide

package P is
  procedure foo (x: in out t'class);
  …
end P;

package body P is
  procedure foo (x: in out t'class) is
  begin x:= create; end foo;
            -- under current rules, this cannot dispatch
            -- but under new rules this could be dispatching and probably should be always
            -- handled as a dispatch to avoid contract problems during instantiations.
end P;
...
type xxx is …;
type nxxx is new xxx;
package z is new P(nxxx);


generic
  type t(<>) is private;
  with function constr return t;  -- constr cannot be dispatching
package g is
  procedure test;
end g;
```

```
package body g is
 procedure test is
   x: t := constr;  -- can never be dispatching
 begin
   if x = x then
         -- surprise!! In the presence of a class-wide actual this could be dispatching.
         -- this is nasty, since the generic formal type didn't tell us about it
 end;
```

It was noted that the impact on access-to-subprogram semantics need also be explored.

Assigned to Gary with consultation from Bob per the instructions from Henley meeting for presenting alternatives.

AI-159

Previously discussed at Henley but it needs to be completed. Bob accepts reluctantly.

AI-160

Locale and operating system types play an instrumental role in handling time changes; hence handling this issue will be inherently implementation dependent. Important mission critical applications should depend on GMT and not on this package.

The summary is just focused on the specific question about daylight savings time boundaries and is not intended to handle all implementation dependent issues.

Discussion then turned to the problems that raising Time_Error would cause the user as opposed to returning some implementation-defined value. The discussion focused on what that value should be and tried to determine if an implementation independent semantics could apply. It failed. The only choice was to leave the AI as is.

Approved 10-1-2. A letter ballot was requested.

AI-161

Pascal and Gary point out the problems related to this issue in using the Unbounded Strings package; you can't create a null-string constant because the type is private and the package is preelaborated. This is not an isolated case. Users can run into this annoyance on their own.

Compliance with the Preelaboration requirement precludes any default initialization that results in calls to user-defined subprograms. To know if a private type has default initialization requires more information than is available in the public part. It may mean penetration of the private part, which breaks the encapsulation design principle. For those who look for the optimization benefit this may be a small price to pay. For those who look for the other benefits, such as distributed systems, elimination of elaboration checks, protection against Program_Error, etc., this price is too high.

One solution is to ensure that deferred constants of private type are static. But this fails to handle controlled types initialization. There is no consensus on a fix, particularly since all the proposed models seemingly also impact Pure packages.

Pascal suggests that a pragma on the private type might help to get around the breach of privateness.

Pascal gets the (un)enviable opportunity to write this AI, probably expanding its scope to encompass the other preelaboration problems.

AI-162

What happens when the subexpression that requires one of these finalizations occurs in a barrier expression of an entry (of a protected object) ? These expressions might be evaluated multiple times. When does the finalization need to happen ? An example could include the comparison of unbounded strings! The consensus is that it needs to happen "soon". Assigned to Mike for write-up according to the Henley resolutions and this newly discovered issue.

AI-164

This needs a write-up to be supplied by Mike.

AI-165

Approved at Henley but needs a write-up which is assigned to Erhard; he will also determine the approval vote tally for the consensus vote at Henley. (It was 10-0-0).

AI-166

Approved at Henley but needs a write-up which is assigned to Ted.

AI-167

The group revisits the major points already discussed at the Henley meeting. The discussion centered on a model that makes the conversion in question a bounded error, possibly rendering (for conversion to sparse enumeration) the result to be a value with invalid representation, eligible to testing by 'Valid. Staying with erroneousness, there is no opportunity to do anything reasonable, formally speaking. The sentiment is voiced that that's fine because we know that, in reality, this conversion and follow-on 'Valid query would work. The counter-argument is that users want more assurance. This AI applies to Unchecked_Conversion and imports.

Bob recalls that the erroneous execution for this case was originally selected for optimization reasons. It is certainly strange that unchecked conversion of a wrapper record is the way around this erroneous behavior or any related optimization.

John recommends that the most straightforward way out of this dilemma is to simply state by fiat than 'Valid when applied [immediately] after an Unchecked_Conversion whose target is a sparse enumerated type produces a useable result and not an erroneous execution. Alternatively, Bob recommends that 13.9.1(12) be changed to returning a value with invalid representation (which can be subsequently checked with 'Valid applied to the object with this value).

Norm will produce the write-up of this AI.

AI-168

Pascal did provide a series of comments per direction at Henley but distribution of these comments to ada-comment failed once again. Pascal will repost these comments again and the AI is tabled until the next meeting.

Note: There is still a persistent problem in distributing ada-comments to all members of the ARG. It is apparent that the redistribution mechanism at Intermetrics is not working. Action item for Erhard to contact Tuck to cooperate in simplifying/transferring the distribution/logging of ada-comment to the sw-eng host.

AI-172

By and large, the discussion of the Henley meeting was reiterated. (Erhard stating that a rule that imposes a restriction on the implementation-defined method of linking a partition makes no sense, Bob opposing this view.)

Erhard finds the second sentence of the summary even more ominous that the RM statement on this issue, since it might be read as prohibiting a linker command syntax that doesn’t distinguish a "root package" from a main program. It appears that the second paragraph is unnecessary to the confirmation expressed by this AI but it might be stated in the response.

Change "with no" in the first sentence to "without a".

How to handle the incorrect ACVC test? It could survive as a test for a different test objective, namely that there must be way to link this test and execute the elaboration code.

Robert will do the write-up per the instructions in the last three paragraphs.

AI-173

There was discussion about the optimization capabilities of C and FORTRAN compilers in handling address operators. Arguably compilers effectively optimize around dereferencing of such address values. Ada should allow this as well.

Bob, who wrote the paragraph 13.3(16), said that the user was on her own if she applied ‘Address to an object that was not aliased. First, the object might not be addressable at all. Second, optimizations should be allowed to keep non-aliased objects in registers thereby making ‘Address undefined, or cache their value, rendering the use of the obtained address rather useless. It would take much more effort by the compiler to determine non-aliased objects to which ‘Address is applied in order to avoid optimizations that involve this object. (In fact, such determination is impossible for global variables and their components). The wording is currently loose enough to permit the continued use of ‘Address in legacy Ada83 code, while still not preventing optimizations.

Tuck is proposing some additional restrictions beyond 13.3(16) to enable subprograms to apply ‘Address to a formal parameter, that may be a part of legacy code. It’s too late to permit the user specify a formal parameter with the aliased keyword. Hmmm…how about creating a calling convention where formal parameters are aliased??? Not really.

The group is leaning towards no action or a confirming paragraph 13.3(16), except Tuck’s point that using ‘Address provides a language-defined method for violating the type rules. Several people feel that this hole should be highlighted as implementation-defined. Namely, the user who accesses an object both through ‘Address and through its name is courting trouble and must be disciplined to "poke” the object through ‘Address in a type safe manner.

It was observed that "volatile" assures the actuality of values in memory and will defeat all optimizations regarding the object. "Aliased" should have the semantics that the compiler should at least be allowed to assume type-safeness, i.e., an assignment to a dereferenced "access to T" can only affect objects of type T, no matter how the access value was obtained. For any other conversions of addresses to access types, the user is guaranteed nothing.

Norm will dig into this AI and complete a write-up that handles the optimization and type safety concerns from the meeting.

AI-176

The intent was approved 13-0-0 and the AI assigned to Gary for completion. It was felt necessary that this AI be a binding interpretation to make sure that the check can be suppressed under the category of an Access_Check.

AI-184

Pascal and Bob point out that there are several more comments in the queue that expose more problems with definiteness and type derivation, and that go beyond the problem in this AI. Bob and Pascal say that are several other similar problems related to definiteness and type derivation. The claim is that switching between definiteness and indefiniteness in either direction as part of derivation kills type safety.

Pascal points out another example, where the defaults are removed:

```
type T(D: Integer :=0) is record…;

generic
  type nt is new t;
package p  is ...

package body P is
  x: nt;  -- legal or not ?  If the actual is unconstrained, it better be illegal
end p;

type actual (d2: integer) is new t(d=>d2);

package I is new p(actual);
```

Making derivation between definite and indefinite untagged type illegal would be a rather stringent binding interpretation to fix all these problems. The alternative is creating individual binding interpretations to plug each of these holes (for view conversions, for generics, for ...).

Erhard questions the problem pointed out by this AI and thereby the reasoning that there is a large number of holes that require such a large binding interpretation. He argues that the example in the AI should cause a run-time check for discriminant equality on the assignment, since the "view" seems to require that. Pascal counters that the current rules address the constraint properties of the actual, not those of the formal and thus, there would be no constraint check. Erhard agrees (reluctantly).

Robert and Erhard then point to 12.5.1(6) that will make the instantiation in the above generic example illegal but it appears that this does not plug the hole caused by the AI example (the renaming of a subsequently non-existent component). The conclusion is that the problem is valid.

The Rationale points out a useful example for deriving an indefinite type from a definite type:

```
type matrix (n, m: integer :=1) is record …;

package squares is
  type square_matrix (n: integer) is private;
private
  type square_matrix (n: integer) is new matrix (n, n);
end squares;
```

A possible additional language rule might be that, if one discriminant is defaulted, so must all the others. (This, in effect, is a quite similar rule to the rule that derivation must maintain the definiteness that already applies to the parent type.)

Pascal will do the write-up. It was pointed out that this AI is loosely related to AI-168 and should be discussed together with it.

AI-185

Kiyoshi asks if this problem also is found in the Ada83 secondary numerics standard. It appears that both GNAT and APEX compilers are implementing this feature similarly. Does this match the way the rest of the world handles this? Robert volunteers to solicit assistance from Ken Dritz, to explain how the Ada83 secondary numerics standard and this Annex handle this, and compare with how the rest of the world does it. The non-complex versions of the interfaces suggest that the function should be continuous and in the I and II quadrants. Robert will complete the write-up.

AI-186

The issue is that the root_integer type is not large enough to handle all modular types. In this case, T'Modulus can't normally be represented in the root_integer range, when T is the largest modular type. Forcing implementations to provide this will force extra length integer manipulation that unnecessarily exceed the applications' need and capability. This is an inherent problem with modular and integer types (it also applies to computations involving 'Pos of integer types).

It appears that the best we can expect is for implementations to do the right thing.

A vote on the summary as stated yields 7-3-3. Kiyoshi wants the guarantee that the cited generics with modular formal types will universally be valid, e.g., by providing something like root_modular to cover this problem. Norm said he would look at this in more detail. 3.5.4(14) is really broken if root_integer does not cover all modular types. Pascal prefers an "include" rather than "is" statement of the range of root_integer. Steve wants clearer citation of the implementation permissions.

Robert will do the write-up.