Final Minutes of 6th ARG Meeting
1-3 April 1998
Burlington, MA, USA

Attendance: John Barnes (from afternoon of first day), Ben Brosgol, Norm Cohen, Robert Dewar (first day through lunch), Gary Dismukes, Bob Duff, Robert Eachus, Kiyoshi Ishihata, Mike Kamrad, Pascal Leroy, Stephen Michell, Erhard Ploedereder, Tucker Taft

Meeting Summary

The meeting convened on 1 April at 9:15 a.m. at the Aonix Offices in Burlington, MA, and adjourned at 3:30 p.m on 3 April. The first half-day was spent on general issues and future ARG direction, the remainder of the meeting on the discussion of the AIs.

The first general issue was the role of the ARG in the context of Ada validation. The ARG is expressing some willingness towards taking responsibility for adjudicating the Ada validation tests and testing results as the DoD ends its funding for Ada validation, if no other alternative is found to act in the role of the AVO.

The second general issue was the question of language evolution, now that it is rather obvious that an Ada 0X will not be a project organized and financed by the US DoD. The ARG therefore would like to broaden its work by cautiously moving towards reviewing and preparing language extensions, as a step towards making this „the“ process for the next revision of the language.

Beyond general approval by WG9 and ISO, acceptance of both these responsibilities will require the cooperation of the ARA or any other sponsoring agency on the nature on this responsibility. It also will require the funded support of a coordinator and of an editor, since this workload cannot be carried on a purely voluntary basis. In terms of on-going work, it will require more judicious screening of AIs according to their importance, in order to free up the needed time. The next ARG meeting, tentatively scheduled for late August or early September, could provide an opportunity to explore how the ARG might actually handle the workload for these new responsibilities.

The third general issue was the production of the Standard Corrigendum. Examples of other Standard Corrigenda (thanks to Kiyoshi) showed that there was no uniform style used. It was decided to minimize the effort in completing the Standard Corrigendum by listing only the AI summaries and (possibly shortened) questions of the approved binding interpretations, sorted by chapter. This may not be the ideal presentation, but it is the most time- and cost-effective method. Inclusion of the often lengthy discussion and recommendation sections was rejected as politically inopportune. The production of actual wording changes was considered too time-consuming to be an option.

The ARG unanimously approved a motion thanking Ben Brosgol and Aonix for their generosity and graciousness as hosts for this meeting.

AI Summary

The AIs were discussed during the meeting in nearly the same order as listed in the meeting agenda and the whole agenda was covered.

The following AIs were previously **approved** and their latest drafts required no discussion:

AI-113 - Exception raised by Month, Day, Seconds Ada.Calendar? (11-0-0)
AI-116 - Elaboration of task with no task definition (13-0-0)
AI-156 - Polar implementation of complex exponentiation for negative exponents (12-0-1)
AI-165 - Recursive use of task attributes isn't considered (10-0-0)

The following AI was once again **approved** with no changes:

AI-150 - Uniqueness of Component Names (previously,13-0-0; now 10-0-1)

The following AI was **approved** to be handled as **No Action**:

AI-201 - Object subtype must statically match designated subtype (10-0-2)

The following AIs were **approved with small changes**:

AI-085 - Questions about Append_File mode (12-0-0)
AI-103 - Storage pools and access types designating task types (12-0-0)
AI-105 - Extra negative value (9-0-0)
AI-120 - What is the minimal upper bound of type Integer? (10-0-0)
AI-143 - Distinct names for Compilation Units (12-0-0)
AI-152 - Operators not inherited from root numeric types (11-0-0)
AI-153 - Picture String Grammar or Composition Rules Need Tightening (10-0-1)
AI-159 - Shared Variables in Shared Passive? (11-0-1)
AI-171 - Elaboration of subtype indications with per-object constraints (11-0-1)
AI-176 - Access_Check is performed for access discriminants (11-0-0)

The following approved AI was discussed and the request for a **letter ballot** confirmed:

AI-160 - Daylight savings and Ada.Calendar (10-1-2)

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

AI-164 - Definition of remote access type (6-0-6)
AI-167 - Erroneous scalar Unchecked_Conversion? (9-1-2)
AI-168 - Aliased objects can have discriminants modified (10-0-2)

The following AIs require rewriting so they can be reviewed before the next meeting:

AI-051 - Size and Alignment Clauses for Objects
AI-058 - Accessibility Rules for Shared Passive Packages
AI-109 - Size and Alignment Attributes for Subtypes
AI-130 - Should no_Local_Allocators disallow nested instantiations?
AI-133 - Controlling bit ordering
AI-147 - Optimization of Controlled Types
AI-148 - Requeue of protected entry calls
AI-158 - T'Class as generic actual type
AI-161 - Default-initialized objects
AI-162 - Anonymous allocators and tasks/finalization
AI-173 - Optimizations and the use of 'Address
AI-184 - View conversion to an indefinite subtype
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-192 - A library subprgram_body should replace, not complete, an instance
AI-193 - Classwide Adjust and exceptions
AI-195 - Streams
AI-197 - Aggregates of a controlled type
AI-199 - Does pragma convention for a generic unit apply to instances?
AI-202 - Primitives of formal type derived from another formal type
AI-203 - T'Model_Mantissa{ if T'Machine_Radix is not 10, and T'Model_Mantissa otherwise}.

The following AIs were not discussed **awaiting their rewrite**:

AI-119 - Is Normal Termination an "External Interaction"?
AI-131 - Interface to C - passing records as parameters of mode "in"
AI-166 - Parameterless_Handler values designate default treatment

The following AIs were not discussed and are looking for editors to draft first versions:

AI-191 - An OBJECTive View
AI-194 - Typo in Standard_Error Definition
AI-196 - Assignment and tag-indeterminate calls with controlling results
AI-198 - Pragma Convention(Intrinsic) is not a completion
AI-200 - Generic formal subprograms as dispatching operations
AI-204 - Interfaces.Fortran must be absent, right?

Validation Process, Language Revision

Robert Dewar reported that the AJPO would cease operation soon and with it the support of the AdaIC. Validation support will also end and the AJPO is hoping that ARA will assume validation responsibilities.

The ARA intends to focus on a Web site and to support some participation of the AdaIC and the continuation of validation. The initial position of ARA was to make ARA the sole responsible authority for validation. This position would apparently exclude users and third-party product developers. Clearly this position suffers from a lack of objectivity that the current process exhibits, where validation authority lies outside of Ada vendors. Additionally, to date the ARA has not been representative of all Ada vendors (it does not include Green Hills, Eilers, TLD Systems, SGI, Concurrent). The ARA is reconsidering this position.

Robert Dewar suggested an alternative to ARA, having the ARG and WG9 adjudicate the tests and test challenges. Tuck was quick to point out that vendors can stuff the membership in ARG, leading to the same vendor bias. Many in the ARG also voiced concerned about the amount of labor that it would take for a volunteer group, which appears already resource-starved. Robert contends that most (70%) of the AIs are „junk“ and, if they were properly dismissed, it would give the ARG time to do this additional work. At the same time, Robert says that the timing is right for the ARG to consider language extensions (such as circular type dependencies across package specifications) in addition to the current language maintenance. As such, when the ARG would approve these extensions, it would do so with a set of tests for the purposes of both testing conformance and promoting understanding of the language extension. So the ARG would be in the business of creating tests for the new language extensions. To properly assume this responsibility the ARG/WG9 would need to be supported by paid positions for coordinating an FRT, for preparing and editing tests and AIs, similar to what Dan Lehman and Bob Duff were paid to do. It is estimated that it would require $50K per year to fund this support which may attract someone substantial. Finally, Robert proposed to the ARG to test his contention by reserving two half days at the next ARG meeting to addressing long range planning and experiment with language extensions by considering circular types. This would be done in conjunction by filtering out more junk AIs.

There was some discussion on the validation process in general. For the DoD requirement for Ada, it was necessary to conduct strict validation as the DoD market demanded it. Now as the market place extends beyond the DoD, vendors, like Intermetrics, are finding that strict validation is no longer a big deal. Most customers really want full language portability across all Ada implementations. As the ACVCs are currently structured, achieving 100% compliance is very expensive and may not be the best interest of vendor’s market. May-be it’s time to focus precious vendor resources on commonality over strict validation to add value to the Ada community. Simple minded growth of the ACVCs needs to stop. Instead, the goal should be to improve the existing ACVC suite (by removing unproductive tests and sharpening the remaining ones) and to carefully add tests to assure commonality of new language extensions. On the other side of the argument, some vendors (like ACT) feel that validation is a major hallmark and market advantage of Ada, not to be watered down considerably or dismissed entirely. Also, the European community, which is currently registering an increase in Ada activity, feels rather strongly about the

benefits of validation.

The possibility of the ARG considering language extensions has lots of appeal. First, without any formal language revision process (like Ada9X) in sight, having ARG/WG9 consider and approve language extension would be „the“ process for Ada0X. Second, it would promote vitality in the language, as Ada would be seen to be naturally evolving. Of course, with language extension would come the responsibility of eventually creating new tests and validation. The general consensus at the meeting was that the ARG is open to handling the validation suite approval role assuming there is funded support. If Jim Moore finds out that ISO rules permit these new roles for WG9/ARG, Robert Dewar will report to ARA on this possibility.

Standard Corrigendum

Erhard reports little progress. He mentioned that Phil Brashear has submitted a summary of approved AIs for publication in the next Ada Letters and he was going to bring this as example of Corrigendum. Instead, courtesy of Kiyoshi, he showed the Corrigendum for C, which is very skimpy. Many at the meeting question the value of doing anything but a summary of the AIs, just to check off an ISO procedure box. Erhard is concerned about the size of the document. It is suggested that including only summary and questions of binding interpretations, sorted by chapter, would keep the size small as would making good use of formatting to eliminate white spaces.

„ada-comment“

Dealing with the problems of the ada-comment mailbox will wait until the deposition of the sw-eng host is known. (Note: Erhard disclosed that the old ajpo address also still works!!)

Next Meeting

A meeting in conjunction with Ada-Europe in June is too early and a meeting in conjunction with the SIGAda Conference in November is too late. Consequently, it was decided that the next meeting should take place in late August or early September in either Paris or Stuttgart. (Tuck takes an action item to draft an AI on the "circular type dependency issue" for this meeting) The ARG will decide where and when by the Ada-Europe conference.

Action Items

Action items were reviewed. Most have been accomplished. **Old action items** that remain on the list are:

Ted Baker: AI-119, AI-131, AI-166 (from the previous meeting)
Robert Eachus: AI-100, AI-117, AI-153, AI-172, AI-174, AI-185, AI-186
Erhard and Bob: Prototype a few AIs to determine how to handle the remainder of the AIs in Standard Corrigendum form.

New action items on the drafting or completion of AIs were assigned to the following ARG members:

John Barnes: AI-120, AI-192, AI-203
Ben Brosgol: AI-153
Norm Cohen: AI-133, AI-167, AI-173
Gary Dismukes: AI-158, AI-171, AI-176, AI-197, AI-202
Bob Duff: AI-51, AI-58, AI-105, AI-109, AI-143, AI-152, AI-159
Mike Kamrad: AI-162, AI-164
Pascal Leroy: AI-85, AI-161 (and "extension" AI), AI-168, AI-184
Stephen Michell: AI-103, AI-148, AI-187, AI-189
Erhard Ploedereder: AI-147, AI-190
Tucker Taft: AI-130, AI-162, AI-188, AI-193, AI-195, AI-199

The following action items were also assigned:

Erhard: Set the data and place of the next ARG meeting by the time of the Ada-Europe conference.
Erhard: Brief WG9 on new role for ARG on validation and language extensions.
Robert Dewar: Brief the ARG on the new role for ARG on validation and language extensions
Tucker Taft: Produce a draft AI on a language extension to allow for mutually dependent types in separate package specifications

Details of the AI Review

AI-51

Tuck is concerned that the alignment issues have been overlooked in the AI. More specifically he is concerned that an implementation might be required to support alignments that are greater than the size of an object, such as a 4 byte object being aligned on 16 bytes. This is painful for compilers. Instead it is expected that the user would accomplish this by the use of a record. He proposes that the required support be that the alignment evenly divide the size of an object (in bytes). A special case should be made for alignment of zero, which means no alignment. Paragraph 13.3(43) addresses the size part of the issue but there needs to be an alignment part added to this paragraph. Bob Duff will make the changes.

AI-58

Review of editorial changes points out that three requested changes were missing.
1. Program_Error should replace Constraint_Error in paragraph 3 of the Summary and just ahead of Example 4 in the Discussion.
2. Example 1 needs to reference P1.X'Access in the statement of Main.

New editorial changes were requested:
1. „normal“ -> „not a shared passive“ in para. 3 of the Summary.
2. make sure that it is clear in para. 3 that this is „checked at run-time“.
3. change Example 3 comments, ditto for Example 4.
4. Redo discussion to make sure of usage of „erroneous“.

Tuck questions the wording of the last paragraph in the Summary regarding the "deeper than" relationship of non-dependent shared passive packages. It is hard to find better wording for this relationship that doesn't evolve into lots of double negatives. (One possibility is the phrase "statically known to be not deeper than")

Tuck also questions why the semantic dependency relationship used in the AI isn't extended to the bodies of the shared passive packages. (As an example: body of p2 withs p1 and therefore can be assured that p1 "lives" longer than p2 and permits the safe access of pointers in p1)

Bob will revise the AI and, based on the result, the AI will either be submitted to letter ballot or to a vote at the next meeting.

AI-85

Discussion reviewed the history of the decisions for the AI. Pascal confirmed that the AI was written with the Unix Append mode in mind.

The wording in the first sentence of the second paragraph of the Discussion "goes back to the append point" is confusing and it should be "goes after the last element of the external file".

The category of this AI should be changed to a confirmation since the summary can be deduced from the manual even without a Note. (This also implies a change to the format of the AI.)

Also, there were two wording changes. It was noted that in A.12.1(35), it says that "Set_Mode changes the mode of the file." The word "changes" is rather inappropriate, since it is possible (although not terribly useful) to call Set_Mode with the parameter Mode equal to the current mode of the file. So it should say "Set Mode sets the mode of the file." And in item 2 of the !discussion the word "external" was added in the sentence: "For a file that is Reset to mode In_File, reading restarts at the beginning of the external file."

Approved (with changes) 12-0-0. Pascal will make the changes.

AI-103

Approved with minor editorial changes, 12-0-0. Stephen Michell will make changes.

AI-105

Apparently no official vote has been taken in the past. Add "see summary" in the response section.

Approved 9-0-0. Bob Duff will make the changes.

AI-109

The same consideration (regarding alignment) from AI-51 should also be applied to this AI. Bob Duff will make the changes.

AI-113

No further discussion.

AI-116

No further discussion.

AI-119

Tabled until Ted Baker produces a write-up.

AI-120

Robert Dewar points out that "(and not Storage_Error)" is wrong and therefore should be removed. John proposes a politically more correct summary.
Approved with these editorial changes 10-0-0. John Barnes will make the changes.

AI-130

Tuck objects to this AI because of the implementation-defined semantics used in this write-up. He argues for a static check with regard to the No_Task_Hierarchy pragma. Others point to the problems caused by default initializations, involving task creation, and the like (see minutes of previous meetings).

Tuck also objects to the mention of generics in No_Local_Allocators, which is inconsistent with other Restriction pragmas. He argues that, in the expansion approach (the most widely implemented method for generic instantiations), the checks can be done as part of the expansion process. For a code sharing approach, the checks can be performed at link-time. The compiler can pass the responsibility to the linker, since the Restriction Pragma semantics has always been seen as a post-compilation rule. Others argue that the Restriction pragma can affect the contract model of generics (as well as of packages), a model that is not only manifest in some compilers for checking its rules, but more deeply ingrained in the design of the compiler.

Robert Dewar points out that, in the RM rule for No_Local_Allocators, allocator is a syntactic notion and not a runtime notion, which is backed up with the type font in the RM. This being clearly wrong, an AI is needed to resolve the issue. Certain allocators are deemed desirable, usually at library level and at elaboration time. What the pragma should do is permit these allocators and not others. But how to word this? (There is a similar dilemma between merging static and dynamic semantics that is found in the preelaboration feature.)

The meeting slipped into a side discussion of lower and upper bounds to the semantics of this pragma. The lower bound is that it prohibits the static use of allocators. The question is where is the upper bound? The AI is written to allow the implementation to define the upper bound, such as anything that "smells" of local allocation, i.e., allocation after elaboration of the program. Is this so difficult to specify that it needs to be totally left to implementations? This isn't very portable. Tension is between the simplicity and portability of limiting the upper bound to a rule enforceable by static semantics versus the flexibility of expanding the semantics to permitting runtime execution that would not violate the intended restriction.

Robert Dewar proposes as an upper bound that there be no allocators in default expressions, default initializations, generics.

Tuck points out that freezing rules seems to statically handle the restriction for default expressions.

Getting back to the generics issue, Tuck points out that those users interested in the respective restrictions will never use generics in the abstract manner of only applying a contract model. They will analyze every instantiation and so the contract model is meaningless to them. Preserving the contract model in the presence of these restrictions is not profitable.

Discussion of implicit heap allocation brought us to the No_Implicit_Heap_Allocation restriction whose semantics are implementation defined. This is at one end of the implementation spectrum while a totally static implementation is at the other end. Robert Dewar calls for one solution or the other but nothing in-between. It appears the current AI supports the former; Tuck volunteers to propose the other.

(There is a typo in the current AI: document{ation} requirement in !wording.)

As a trial, Tuck will produce a rewrite based on a model of static enforcement of the restrictions.

AI-131

Tabled until Ted Baker produces a write-up.

AI-133

Norm is concerned about maintaining portability regarding Endianness on machines with scalar (his term for the load size of the machine) capability which is a multiple of 8. He has documented his concerns in a paper that he submitted. (Note: Norm has previously published a paper in Ada Letters containing formulas that makes the positioning for bit ranges of components portable in the presence of Endian differences.)

Tuck draws several examples of positioning and ranges with Big and Little Endian which prove the difficulty of all of this. It was decided to table further discussion until the last day, giving attendees the time to review Norm's paper.

The discussion on the last day established that Norm's paper provides an excellent presentation on endian-ness. Does it better address the issues raised by the AI? Concensus was converging rapidly to accept the solution proposed in his paper when the question of 'position, 'first_bit and 'last_bit were considered. After examining the details of these attributes, Norm's paper seems to cover the issues for default or non-default bit ordering as long as the default bit-ordering is used as the basis for the reporting by these attributes. Norm does provide some schemes for allowing the user to specify bit layout that can handle default reporting on either endian machine.

13.5.3(6) needs to be changed to refer to the default bit-ordering of the System. Also, there needs to be a legality rule preventing offset overlaps in component clauses of a record representation clause.

The intent of Norm's paper is approved for the AI, 8-0-3. Norm will write it up.

AI-143

Why is this AI a ramification as opposed to a confirmation? Tuck proposes that ramification are distinct from confirmation by the fact that ramifications require a note. This AI doesn't meet that distinction and it will be changed to a confirmation.

"The language does not specify how to resolve the ambiguity." is added at the end of the summary to more explicitly respond to the second sentence of the Question.

Approved with changes, 12-0-0. Bob Duff will make the changes.

AI-147

Tuck disputes the conclusion of the AI that the last sentence in 7.6(21) is incorrect. Erhard provides an example to show why this is so. The gist of the example is that the condition of no aliased subcomponents is necessary, but not sufficient, to make omission of the adjustment in the final target location possible. The missing condition is that the object as a whole must not be referenced directly by a subcomponent or indirectly by some other object accessible via a subcomponent of the object at hand -- a situation, for which the Adjust has been written to „track“ the object as it „moves about“ as part of assignments. If the Adjust is done only on the temporary but not the final destination of the object, the reference to the temporary remains. This is a clear contradiction to the canonical semantics with disastrous consequences.

```
X: CRType := something;
begin
  X:= something_else;
  X.foo := 7;
  print (X.comp.foo);
  print (X.comp2.all.sons(3).all.comp.foo);
end ;
```

If X.comp is made to point to X during the Adjust then the Adjust in the final destination X cannot be eliminated. If X.comp2.all.sons(3).all.comp is made to point to X by the Adjust, the same applies. (Here, the .alls are intended to

indicate that the back-reference can actually be anywhere is some complicated graph structure.)

Unfortunately, the situation is not easily recognizable by static analysis (it is, in fact, undecidable). While the aliasedness of subcomponents indicates the potential for a problem (and hence is already a prohibitor for the optimization), there is nothing to narrow the situation for the objects themselves, since tagged objects are aliased by definition. All this is not a problem as long as no 'Access is taken on the object or any of its components in the Adjust operation or any operation it invokes. However, this property is neither locally decidable nor even statically decidable by a global analysis. There was a lot of discussion to convince ourselves of these point.

There was a short side discussion on whether a user writing an Adjust routine that prints the address of the adjusted object has any right to expect the addresses that the canonical semantics would produce. The answer was „No“.

Bob and Tuck strongly advise that the summary be restated in a way that does not explain semantics in terms of a permission for the compiler to assume falsehood. Bob points to the semantics of „Pure“ as a good example of how to do it right, RM 10.2.1(18).

In an attempt to summarize the discussion, Erhard asked for recommendations for the AI.
1. restate a) in summary by removing the assumption wording
2. state that 7.6(21), last sentence must also consider references to the object as a whole

Erhard will reword the AI.

AI-148

Stephen is concerned that the wording in 9.5.4(7) will permit the releasing of the lock on the entry body (the source of the requeue) before initiating the start of the requeue onto the new entry. Tuck and Bob contend that wording for protected action execution and completion (9.5.1(3-7), 9.5.3(11,18)) make that interpretation not possible. They described the sequences of steps that the original protected action must follow in doing a requeue. With a very nice diagram, they showed that there isn't any other reasonable way for the sequence of steps to occur other than doing the requeue (as a nested protected action) before the original protected action is completed (and its lock is released).

Stephen Michell will go back to the drawing board to record these points.

AI-150

Ben questioned the need for this AI because Java has the capability that is prevented by this AI. It seems to be useful to the Java community. He argues for leaving an ambiguity here. His argument does not win.

The minutes from the St. Louis meeting were incorrect regarding the last editorial change.

„the parent“ needs to read as „the ancestor“. Approved 10-0-1.

AI-152

The discussion uncovered existing wording (3.2.1(9) and others) that plugs the hole that Norm originally pointed out. Some editorial change in the summary is asked for, namely that the first sentence should read: "Predefined operators, including those on root numeric types, are not inherited. Instead, ..."

Approved with editorial change, 11-0-0. Bob Duff will make the changes.

AI-153

Despite overwhelming approval (10-0-1), Robert Eachus requests editorial review by Ben, which Ben will make.

AI-156

No discussion.

AI-158

Tuck provided an example [on the chart] that describes the issues again with respect to the renaming of implicitly declared primitives of 'Class actual type in generics.

```
generic
   type T(<>) is new T2 with private;
pkg GP is
 procedure P1_ren(X:T) renames P1;
 procedure P2_ren(X:T, Y:T) renames P2;
 function Empty_Ren return T renames Empty;
end GP;

pkg I is new GP(T'Class);

X: T'Class := ...;
I.P1_ren(Empty)   -- <tag-indetermin.> calls wrapper, dispatches
I.P2_ren(Empty, X) -- currently no dispatching rules, but dispatches
X := I.Empty_ren;  -- assignment may raise exception
```

A few issues were raised:
The possible problem of convention is mentioned, but it's intrinsic already by ARM.
What if T2 is abstract ? cannot be, respectively is explicitly so.
What if Empty is abstract ? need a rule that prevents the renaming then.
How to distinguish a class-wide user-defined overload vs. the wrapper ? No immediate answer.

After examining the example with variations for instances of either type derivations or 'Class of the parent type and the addition of a formal subprogram which is a primitive of the parent type, the solution proposed by Tuck as augmented by these new items/constraints appears to fix the problem:

- the concept of wrappers for the implicit primitives of the actual class wide parameters
- a rule to protect against improper instantiations with abstract type derivations

Gary will write this up with help from Bob and Tuck.

AI-159

Several small editorial changes were made. Approved 11-0-1. Bob Duff will make the changes.

AI-160

Stephen could not remember his objection from St. Louis meeting and maintains the request for a letter ballot.

AI-161

Each of the restricted types in 10.2.1(9) was examined in detail. It appears that only task types and protected types should be covered by this categorization pragma.

For private and private extension type, it is suggested that an additional "preelaboratable-default adjective" (whose

description is to be covered by a separate language extension AI -- see below) be added to tell the implementation that the private type is defined in a way that adheres to the rules of preelaboration. Erhard asks why not open up the private part of the package to get the information a client needs to determine the "preelaboratable default"-ness of the private type? This is definitely naughty and put us on a slippery slope. The "preelaboratable-default"-ness of the private type is part of a contract to clients and would be better served by an adjective/pragma that documents this restriction of the contract. The pragma approach is preferred over opening the private part by 7-2-3. (Note: Such a pragma could be generalized to cover other promises about types in the contract to clients.)

Certain predefined types will be defined to have "preelaboratable default". The default for all types in predefined Pure and Preelaborate packages is "preelaboratable default" until shown otherwise.

For controlled types, the restriction on their use is moved to the list of restrictions in C.4 of the ARM.

(Note: One of the purposes for preelaboratable non-shared passive packages is to eliminate the elaboration check. For shared passive packages it is used to help create a load image (for ROM) suitable to be shared between two different remote runtime systems.)

Tabled until Pascal rewrites this AI. Pascal will also prepare a draft of an extension AI for the new pragma and check the predefined types for application of the pragma.

AI-162

The following changes were directed as a result of the discussion:
1. The AI needs to be expanded to include the finalization of controlled type objects.
2. Replace statement construct as the potential master to an "oosiwah" (another construct) which can appear in a compound statement, declaration or barrier condition.
3. Examine the special needs of barrier conditions.
4. Redo the definition of master so that it may more specifically deal with finalization of temporaries and task dependency.
5. Explore combining the handling of anonymous objects and tasks created by anonymous allocators.
6. Find a way to permit implementation freedom to use either innermost or outermost "oosiwah"s or something in-between as the master. (This example illustrates the issue:

   ```
   P(F(new T), G(new T));
   ```
   where F would be defined as
   ```
   function F(Q: access T) return T is
   begin
     return Q.all;
   end F;
   ```
   The identification of master(s) should be flexible enough to make the call of P a master or both F and G masters.)

7. Apply new version numbers to the AI
8. Simplify the example and add a controlled type component to the record, dropping the string component.
9. Fix typo „veiw“ in !wording
10. The title should refer to „anonymous access types“
11. The summary needs to say „statement containing a subprogram call“

This is assigned to Mike for rewriting with assistance from Tuck and Bob.

AI-164

The following changes are directed as a result of the discussion:
1. Restore the category of the AI to binding interpretation because it does require a wording change.
2. Update summary to reflect the binding interpretation.
3. Complete the wording section by adding a phrase like the following to E.2.2(9), " or any number of private

extensions"
4. Reverse the decision of Henley to drop the private from E.2.2(9). By keeping the private constraint in this paragraphs ensures the least surprise to developers when non-distributed applications are made distributed.
5. in the summary: „of {a} limited type“

Vote on the intention, 6-0-6. Mike Kamrad will make the changes.

AI-165

No discussion since the Henley meeting previously approved this AI with a vote of 10-0-0.

AI-166

Tabled until Ted Baker produces a write-up.

AI-167

All the previous points were revisited during the discussion. Concensus appears that the AI should make use of 'Valid meaningful, in the presence of checks for invalid values. The user should be able to do an Unchecked_Conversion and assigned the result to a variable without checks so that the user can apply 'Valid on the variable to test its validity. This means that the implementation must be able to copy abnormal values without exceptions to support this. (Note: This feature is aimed at checking for faulty enumeration type values.) Avoiding checks will mean for example that compilers must avoid using floating point registers to avoid the machine generated traps. Currently 13.9.1(12) makes such Unchecked_Conversion erroneous and needs to be changed for this goal to be reached. The AI as currently written (i.e., invalid results) would require the checks upon assignment, and hence won't solve the problem short of difficult magic words around 'Valid. The solution seems to be to create appropriate wording/semantics to admit abnormal values through Unchecked_Conversion or imported functions but not trash the type integrity for uses of the value.

Recommended changes:
1. Unchecked_Conversions or imported functions may produce abnormal values.
2. Assignment of abnormal values is a bounded error.
3. Any other use of an abnormal value is erroneous.
4. 'Valid can be applied to objects holding abnormal values
5. 13.9.1(12) is rescinded; Unchecked_Conversion on scalars is never erroneous.

Intent of changes were approved 9-1-2. Norm Cohen will do the rewrite.

AI-168

There is agreement on the proposed restriction, but there is a solution for the second problem missing. The Subject of the AI should change „can“ to „can not“. The "Henley example" is wrong, but easily fixed; it should read:

```
type Ptr is access all T;
PT : Ptr (0) := ...;
```

The problem is the assignment, since the array-assignment does not check the constraints of the components. Requiring such checks was rejected as too big a change.

An additional legality rule is recommended, namely to forbid access constraints (subtype_indication) on general access types whose designated type is a private type whose partial view has no discriminants. It addresses the second issue raised in the Discussion.

Intent of the changes were approved, 10-0-2. Pascal will rewrite.

AI-171

Discussion produced some minor editorial changes. Approved 11-0-1. Gary Dismukes will make the changes.

AI-173

Tuck says that the AI doesn't address the 'Address issue which was the point of his original message. Tuck gives an example where a subprogram will accept an actual parameter using a by-reference type and then change the value of variable by the 'Address of the variable.

The issue is whether and when optimized code can reuse storage in the presence of 'Address.

It was observed that the summary (and possibly other parts of the AI) should definitely be in terms of volatile views rather than the volatileness of the objects (which is not a statically decidable property).

As to the dangers, Erhard provides an example:

```
procedure Z is
  X: rec;
begin
  …
  P (X'Address, X.foo, expression that needs memory);
  -- no further use of X in Z
```

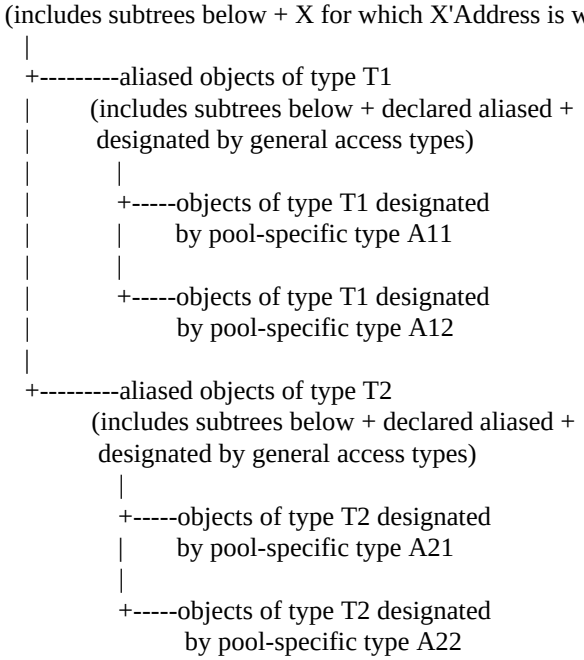
Erhard states that there is a danger that the optimizer may reuse the storage of X to make room for the expression evaluation. Tuck contends that the optimizer would be wrong because of the existence of the X'Address. Erhard agrees, saying that he merely wanted to point out an obvious pitfall, arguably „permitted“ by the standard. It would appear that 'Address should be an "aliased" reference to X, namely, another "access" path to X. This could make the use of 'Address subject to the same rules of access path 6.2(12) or to the rules of 'Access. It is still more painful for the user to do 'Address rather than ‘Access because of the needed unchecked conversion of the address to an access value, hence the hope that ‘Access will often replace ‘Address in user code.

Does type compatibility matter? (I.e., seeing an assignment to any dereferenced pointer, whose value is unknown, must one assume that all objects, whose ‘Address was taken, could be affected by this assignment, or can these effects be limited to objects of a type compatible with the target type of the pointer. The latter holds for the taking of ‘Access or ‘Unchecked_Access. Should ‘Address be more devastating on optimization than that ?)
There is considerable and entrenched disagreement on this issue.
The discussion also turned to pool-specific access types and general access types and related issues with ‘Address (as opposed to ‘Access). Tuck argues that use of dereferenced general access types relates to all type-compatible pools and use of ‘Address affects all pools. This model might provide the basis for a set of a rules.

There is reasonable agreement on the following points: the storage for objects whose address was taken needs to stay reserved for that object, as long as the object might be accessible via the address; the declared constantness of an object whose address is taken can be relied upon by the compiler; the effects of aliasing obtained in this fashion must correctly occur in the local context in which the aliases are created and the dereferencing occurs; a more global guarantee of aliasing behavior need not be provided (or else no global variable could be held in a register even temporarily).

Tuck draws a Venn diagram, reproduced here as a tree:

Address-taken objects



Suppose each object is placed in the innermost applicable Venn-diagram circle, or the applicable tree node farthest from the root. Then assignment to a variable in a given category should force the compiler to regard all objects placed in the same category, in enclosing categories, and in enclosed categories to be regarded as potentially modified.

How about the integrity of components of objects whose 'Address is taken? I.e., can one use address arithmetic to obtain a pointer to a different component and then rely on aliasing for that component? It is agreed that no such guarantees need to be provided. The same applies for physically adjacent objects on the stack or the heap.

There is a concern that rules discussed here and proposed in this AI would break existing compiler optimizers, especially in Ada83 compilers. Tuck contends that these compilers most likely assume the worst already, while Erhard munches on his confidentiality agreements.

Can this AI be saved/salvaged? Norm Cohen will rewrite based on the discussion.

AI-176

A "See summary." should be added to the Wording section. Approved 11-0-0. Gary Dismukes will make the change.

AI-184

There are two issues addressed in this AI, which are described in the two paragraphs of the summary and illustrated by the two examples.

Discussion quickly focused on the second issue, described by the second paragraph. Tuck and Bob considered the solution too strong and looked for a less sweeping solutions. To solve the view conversion problem for in out parameters, the simpler decision was to modify 6.4.1(16) to include nominal type when they are indefinite. To solve the explicit view conversion, the simpler solution is to limit explicit view conversion (to ‘in out’ parameters and) to situations where both the operand and target types are tagged.
(An alternative to make view conversions constrained by definition did not offer a solution. The counter-example is

R: C renames C(X).F;
   X := (0,...)
in P of example 2.)

It was noted that an assignment with aliasing inside P of example 2 would be a problem, but that 3.7.2(4) makes it erroneous already. It was suggested that 3.7.2(4) could even be construed to apply to the case 1 as well.

On the first issue, the real problem appears to be specific to untagged generic formal types. It appears that the suggested wording of the AI for 8.5.1(5) addresses this already. It was noted that the !wording and !summary could be made specific to untagged formal types.

Tabled until Pascal Leroy does the rewrite.

AI-187

Offer's comment was aimed at making wording between task attribute operations and operations on atomic objects consistent or else add a binding interpretation to say so. It is clear that they are consistent. Stephen Michell will write it up.

AI-188

The issue is how promptly the set priority takes effect. Multiple processors are excluded, and this includes pthreads-based runtime systems (according to Dewar). Tuck will write this up to provide the capability requested in the AI. Erhard will be anxiously awaiting the AI to be convinced by Tuck's argument, since the cautious wording of the RM in this regard was crafted quite intentionally the way it is.

AI-189

Stephen Michell will write it up.

AI-190

Erhard will write up a first draft for discussion.

AI-192

There is a hole in the RM which would inadvertently permit a subprogram body to complete a generic instance. This is clearly not intended. John Barnes will write up the AI.

AI-193

Tuck brings a problem of an exception being raised while Adjust is performed in the middle of a list of controlled component values during the initialization of a controlled object. Must an implementation stop adjusting the rest of controlled components in the list? The goal of the answer should be to prevent the failure of one controlled components from corrupting the other controlled components.

There are two initializations that can be considered: implicit and explicit. Implicit initializations remain unchanged; they are different, since a failed initialization can easily suppress finalizations selectively. The initialization of the object still must maintain track of which controlled components are successfully initialized in order to only finalize the successfully initialized components. Explicit initialization is determined to be conceptually equivalent to the

action of the assignment (for which we know that finalization will eventually happen after the adjust -- we do not want to require any suppression of such finalization). When the Adjustment of the initialized (or assigned) object value begins (after the bits have been copied), then all the controlled components are adjusted. The failure of a controlled component will then cause all the controlled components to be finalized. This will cause (potentially fatal) problems for the Finalize routine of the failed adjusted controlled component.

Wording in 7.6.1(4) will be changed to reflect this.

Tuck will write the AI.

AI-195

Discussion is divided into the questions found in the AI:

- Questions 1 and last: Do default initialization.
- Question 2: On the privacy issues the answer to the question is yes, it is illegal.
- Question 3: On inheritance, the discussion included the inheritance of the "=" operator as specified in AI-108. AI-108 should apply only to non-limited tagged types.) It was felt that 'Read should compose where "=" composes and inherit, where "=" doesn't compose; however limited tagged types are difficult, since 'Read then needs a new implementation, i.e., is de-facto abstract. It better then not inherit. This general logic should be applied to inheritable attributes, including 'Read and 'Write. This logic shouldn't apply to the 'Input and 'Output attributes.
- Question 4: the answer is that italicized T should be redefined to mean the same as for a user-defined Read.
- Question 5: If the parent type only has the predefined 'Read and 'Write, these attributes are not inherited attributes for the child type. The child type gets the predefined attributes that will handle the child type correctly (according to the category rules for record type). For user-defined 'Read and 'Write for the parent type, the child type will inherit the attributes. The user can override these attributes to tailor them for the child type. The resolution is to leave things as they are.
- Question 6: The discussion of exceptions goes to Chap 11.6 for guidance on this issue. 11.6(6) is very clear that language defined exception can produce abnormal values. Otherwise, user-defined exceptions, raised by user-defined 'Read must be safe, meaning that a temp must be created. If the type is controlled, the Initialize and Finalize of the temp must be executed. The application of this is narrowed to only unconstrained types, since the main point is to preserve safety with respect to discriminants.
- Question 7: Yes, Constraint_Error is raised.
- Question 8: Yes, make it abstract.
- Question 9: No, it doesn't specify. (Performance over perversion).

Tuck will write up this AI.

AI-197

Gary Dismukes will write it up.

AI-199

This AI addresses an error in AI-41. AI-41 incorrectly covers program unit pragmas, Inline, Convention, Export and Import, which makes sense for instantiations of generics with these pragmas applied to the generic specifications. AI-41 should only apply to library unit pragmas. Tuck will complete the write-up.

AI-201

Bob recommends that the category be changed to No Action as the issue was dealt with in the messages found in the

appendix. Approved 10-0-2 as No Action and Bob will do the clerical work to make it so.

AI-202

The sort of question that implementors would ask. Gary Dismukes will write it up.

AI-203

John Barnes will write it up.