Minutes of the 17th ARG Meeting

11-13 October 2002

Bedford, Massachusetts USA

Attendees: Steve Baird, John Barnes, Ben Brosgol (Friday only), Randy Brukardt, Alan Burns, Gary Dismukes, Kiyoshi Ishihata, Pascal Leroy, Erhard Ploedereder, Tucker Taft.

Observers: Mike Yoder.

Meeting Summary

The meeting convened on 11 October 2002 at 9:10 hours and adjourned at 15:28 hours on 13 October 2002. The meeting was held in a conference room at the nearly deserted offices of Avercom in Bedford Massachusetts. The meeting was hosted by Tucker Taft and his new company, Sofcheck, which has not yet moved into quarters of its own. The meeting covered the virtually entire agenda for amendment AIs, and about half of the agenda of normal AIs. Friday and Sunday were primarily devoted to amendment AIs, while Saturday was primarily devoted to normal AIs.

AI Summary

The following AIs were approved without changes:

AI-196/02: Assignment and tag-indeterminate calls with controlling results (9-0-1)
AI-243/01: Is a subunit of a subunit of L also a subunit of L? (10-0-0)

The following AIs were approved with editorial changes:

AI-147/10: Optimization of controlled types (9-0-1)
AI-195/12: Streams (5-0-5)
AI-212/01: Restrictions on configuration pragmas (9-0-1)
AI-220/01: Subprograms withing private compilation units (10-0-0)
AI-249/07: Ravenscar Profile for High-Integrity Systems (9-0-0)
AI-255/01: Object renaming of subcomponents of generic in out objects (10-0-0)
AI-263/01: Scalar formal derived types are never static (9-0-1)
AI-305/02: New pragma and additional restriction identifiers for Real-Time Systems (10-0-1)

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

AI-178/01: Which I/O operations are potentially blocking? (8-1-1)
AI-219/01: Conversions between related types (9-0-1)
AI-224/06: pragma Unsuppress (11-0-0)
AI-228/02: Premature use of “shall be overridden” (9-0-1)
AI-259/01: Can accesses to atomic objects be combined? (8-1-1)
AI-265/03: Partition Elaboration Policy (10-0-1)
AI-269/03: Generic formal objects can be static in the instance (8-0-1)
AI-297/02: Timing Events (7-0-4)
AI-298/02: Non-preemptive dispatching (11-0-0)
AI-307/01: Execution-time clocks (3-0-7)
AI-317/01: Partial parameter lists for formal packages (8-0-2)

The following AIs were discussed and require rewriting for further discussion or vote:

AI-51/10: Size and Alignment Clauses for Objects
AI-109/07: Size and Alignment Clauses for Subtypes
AI-204/02: Interfaces.Fortran must be absent, right?

AI-218-02/01: Accidental overloading when overriding
AI-251/06: Abstract Interfaces to provide Multiple Inheritance
AI-275/01: Aliased components and generic formal arrays
AI-285/01: Latin-9, Ada.Characters.Handling, 32-bit characters
AI-287/01: Limited aggregates allowed
AI-295/02: Another violation of constrained access subtypes
AI-296/00: Vector and matrix operations
AI-303/01: Library-level requirement for interrupt handler objects
AI-304/00: Reemergence of "=" in generics
AI-310/01: Make 3.9.3(7) an overloading rule?
AI-315/00: Full support for IEEE 754

The following AIs were voted No Action:

AI-299/02: Defaults for generic formal parameters (8-1-1)
AI-313/01: Self reference in subprogram access (8-0-2)
AI-314/00: Standardize Discard_Names (8-2-0)

Detailed Minutes

Meeting Minutes

There were no comments on the minutes of the 16th ARG meeting. The minutes were approved by acclamation.

Review of the "Instructions to the ARG" from WG9:

John reads the document out loud, as we don't have a written one to refer to.

Pascal brings up AI-254. He notes that this document does not give us any guidance on "grey area" proposals. We wonder if it is necessary to show such AIs (finished but may not be included in the Amendment) to WG9. After discussion, we decide to send such AIs to WG9 for "information" as opposed to including in amendment.

The schedule document includes presenting Amendment progress to the Ada community. SIGAda 2002 has reserved a 1.5 hour slot on Tuesday and 2.5 hour slot on Thursday for this purpose. The preliminary program shows a briefing on Tuesday and workshops on Thursday. The conference is December 10-12, with a WG9 meeting on Friday the 13th. Tucker and Ben will be at the show. Tucker has been elected to do the Tuesday presentation. Workshops on Thursday: John and Randy *might* be there. Probably should have parallel Real-time & High Integrity, and everything else tracks. Decide to table this until later in the meeting, hopefully people will have firmed up plans.

Erhard notes that someone needs to tell the Ada Europe program committee to include sessions on the Amendment. Pascal takes this an action items.

Ada User Journal would like a paper on the Amendment for publication. Pascal volunteers to write this article, and thus is given an action item to do so.

On Sunday, the SIGAda workshops are taken up again. Tucker will be there. 70% chance for Erhard. Mike Y. might. John might. Tucker will do workshop; Alan and John will try to find a real-time person.

Next Meeting

At the last meeting, we set a date of February 7-9, 2003. Joyce cannot host. Someone suggests San Diego. Everyone looks at Gary. Gary isn't sure that he can host. Tucker suggests Spain; do we have anyone to organize it? Someone suggests asking Tullio Vardanega (Padua, Italy).

Erhard asks if the dates could be moved back a week if in San Diego (interferes with end of term). That would be February 14-16, 2003. These dates hit the start of Tucker's vacation; but he could make a San Diego (not European) meeting.

Pascal is given an action item to find a host (preferably for February 7-9) and send an announcement to the mailing list as soon as possible.

Social Events

Tucker Taft invited the ARG to a cookout at his house at 7 pm Friday. Ben Brosgol invited the ARG to refreshments at his house at 7 pm Saturday. Despite the rainy weather, a good time was had by all at both events.

Motions

The ARG thanks Tucker Taft and Sofcheck for providing the facilities and social event for the meeting. Approved by acclamation.

The ARG thanks the Rapporteur (Pascal Leroy) for his fine work managing the meeting. Approved by acclamation.

Old Action Items

The old action items were reviewed. Following is a list of items completed (other items remain open):

Steve Baird:

- AI-251
- AI-295

Randy Brukardt:

- AI-224
- AI-299
- AI-301
- AI-303 (AI to fix the contract model violation of C.3.1(8))
- AI-319 (Object_Size attribute)

Editorial changes only:

- AI-216
- AI-217-04
- AI-262
- AI-276
- AI-284

Alan Burns:

- AI-249
- AI-265
- AI-297
- AI-298
- AI-307 (CPU Time, was assigned to Kamrad and Tokar)

Gary Dismukes:

- AI-306

Bob Duff:

- AI-287

Pascal Leroy:

- AI-195
- AI-285

Tucker Taft:

- AI-167
- AI-270
- AI-286
- AI-317 (Make a proposal for partial parameter lists in generic formal packages.)

Mike Yoder:

- AI-314 (Standardize Discard_Name & 'Image)

New Action Items

The combined unfinished old action items and new action items from the meeting are shown below. Note that some of the new action items deal with issues that were uncovered during the editorial review that followed the meeting, and not during the meeting itself. They are included here for easy reference.

Steve Baird:

- AI-51
- AI-109
- Revise AI-216 to account for problems found during the editorial review.
- AI-251
- AI-291

John Barnes:

- AI-204
- AI-218-02
- Write wording for AI-254
- AI-296

Randy Brukardt:

- AI-178
- AI-218-01
- AI-224
- AI-259
- AI-292

Editorial changes only:

- AI-147
- AI-195
- AI-212
- AI-220
- AI-249
- AI-255
- AI-263
- AI-305

Alan Burns:

- AI-265
- AI-266-02
- AI-297
- AI-298

Gary Dismukes:

- AI-158

Bob Duff:

- AI-219
- AI-239
- AI-269
- AI-287
- AI-303
- AI-318
- Be the test creator of last resort

Mike Kamrad:

- Various items to be standardized [jointly with Mike Yoder]
  - External_Tag
  - Storage_IO of tagged types
  - Array indexed by holey enumeration
  - Static elaboration
  - GNAT attributes and pragmas

Pascal Leroy:

- AI-228
- AI-264
- AI-285
- AI-310
- Find a host for the February meeting and sent details to the ARG mailing list.
- Contact the Ada Europe organizers to include a presentation on the Amendment in the conference agenda.
- Write an article on the Amendment for the Ada User's Journal.

Steve Michell:

- AI-148
- AI-250

Erhard Ploedereder:

- AI-237

Tucker Taft:

- AI-133
- AI-162

- AI-188
- AI-214
- AI-230
- AI-231
- AI-252
- AI-275
- AI-282
- AI-288 (split into two AIs: pre/postconditions and invariants).
- AI-290
- AI-293
- AI-295
- AI-304
- AI-317
- Physical units (length/dimensional analysis), whereby square meters are generated by result of multiplying meters; subtypes with special attributes would be used to provide the specifications of these dimensions, reducing the introductions of lots of extra operators.
- Give Amendment briefing and non-real-time workshop at SIGAda 2002.

Mike Yoder:

- AI-315
- Various items to be standardized [jointly with Mike Kamrad]
    - External_Tag
    - Storage_IO of tagged types
    - Array indexed by holey enumeration
    - Static elaboration
    - GNAT attributes and pragmas

Detailed Review

The minutes for the detailed review of AIs are divided into existing amendment AIs and non-amendment AIs. The AIs are presented in numeric order, which is not necessarily the order in which they were discussed. Votes are recorded as “for”-“against”-“abstentions”. For instance, a vote of 6-1-2 would have six votes for, one vote against, and two abstentions.

Detailed Review of Amendment AIs

AI-218-02/01: Accidental overloading when overriding

John has created this alternative using syntax rather than pragmas. **Maybe** seems weird. Eventually it is decided to be OK. Alternative suggestions for syntax immediately surface.

Tucker suggests possibly that the keyword should go after the **is** “**is overriding abstract**” or something like that.

Erhard suggests adding a new syntax for “new” subprograms; then “**maybe**” is the default. “**new procedure** id ...”.

Tucker asks about renames; renames definitely can override. John says that he didn’t consider that.

With so many syntax suggestions, we write them all on a dirty whiteboard:

```
procedure P (...) maybe overrides;
procedure P (...) overrides is abstract;
procedure P (...) is new Q maybe overrides;
overriding procedure P (...)
possibly overriding procedure P (...)
maybe overriding procedure P (...) renames ...
```

The latter possibilities could be formatted like a generic, i.e.:

```
overriding
procedure P (...)

maybe overriding
procedure P (...)
```

Tucker suggests:

```
overriding?
procedure P (...)...

overriding!
procedure P (...)...

overriding~
procedure P (...)...
```

The AI should be sent back to John. He (reasonably) asks for instructions, which starts the whole discussion over from the beginning.

Ben shows an example:

```
package P is
    type T is private;
    function "+" (L, R : T) return T;
private
    type T is range 0 .. 100;
    --(or type T is new <some tagged type>);
end P;
```

Do we want details about overriding in the public part?

Tucker suggests having "**overriding**" to mean "might override", and that's it. That eliminates this problem, but it doesn't solve the original problem.

Another suggestion is to avoid enforcing the pragmas in the private part. But this also eliminates some of the error detection that this feature is intended to provide.

A suggestion is to use the pragma with a full profile in the private part. A full profile is needed in order to specify a particular routine out of a set of overloaded ones.

Bob shows a similar example:

```
package P is
    type T is new T1 ...
    procedure Foo;
private
    type T is new T2 ...
end P;
```

What happens if T1 does not have a Foo, and T2 does? Does this need, or not need the "overriding"?

Tucker suggests allowing the keywords on the body. Then he opines that full profiles with a pragma may be the right way to go.

Mike Yoder says that he prefers the original syntax.

The group starts leaning toward the pragmas. Then it no longer appears in the visible part in these examples.

However, we still need to give John instructions. If it is syntax, where does it go? We take a straw poll.

Front (**overriding procedure** P ...): 2; Middle (**procedure** P (...) **overrides**;): 6; After (**procedure** P (...) **is overriding**;): 1 This vote confirms John's original syntax proposal.

Tucker suggests a solution to the privateness problem and still use syntax is to allow redeclaration. This would look something like:

```
package P is
    type T is new T1 ...
    procedure Foo;
private
    type T is new T2 ...
    procedure Foo overrides;
end P;
```

The privateness problem needs to be addressed in the original version of the AI as well. Randy will handle the original version, John will update this alternative.

AI-224/06: pragma Unsuppress

Change:

"If a checking pragma applies to a generic instantiation or a call to a subprogram that has a pragma Inline applied to it, then the checking pragma also applies to the instance or {inlined} subprogram body."

Also, split this rule into two sentences. (one for inline and one for instantiation).

John notes the proposal section has an extra word: "...with [the] similar scoping rules." He also notes that the example in the problem section is incorrect. It should say "non-negative" saturation, since the code doesn't handle negative numbers properly.

Steve Baird wonders if there is there a problem with requiring suppression to not happen.

```
pragma Suppress (Foo_Check);
procedure P;
pragma Inline (P);
procedure P is ... end P;
...


pragma Unsuppress(All_Checks);
...
Q.P;
```

This example shows a case where we don't want Unsuppress to have an effect. If the call is compiled out of line, then clearly the Suppress(Foo) will apply to the body of P. We want it to be legal for the same to be true in the inlined body.

This is covered by the wording, because Unsuppress only revokes a permission given by a Suppress if the permission applies at the point where the pragma Unsuppress appears. Tucker thinks it's useful to clarify the wording:

"...to which it applies {at the point of the Unsuppress pragma}". (Fix the "it".) Make this sentence more precise.

Tucker suggests changing the bracketed text to say if a Suppress pragma applies to the instance body or inlined copy, the unrevoked permissions also apply to the instance body or inlined copy. Or perhaps make it "unspecified" (we don't want to force documentation of this, so it can't be implementation-defined).

Erhard: The note after 11.5(29) does not apply to nested scopes; it must say "immediately within a declaratative_part".

Approve intent of the AI: 11-0-0.

AI-249/06: Ravenscar Profile for High-Integrity Systems

Alan suggests changing the name to Ravenscar_Profile and adding an implementation advice that profile names end with "_Profile". The group concludes that that is not necessary, as these always appear after "Profile" in pragma profile.

None of these are Dynamic Semantics and Post Compilation Rules; all of that should be Static Semantics. Remove the second sentence ("It is equivalent to the set of pragmas") from this paragraph.

Erhard suggests that the clause title should be changed to be similar to the others in this Annex. Change to "Runtime profiles and the Ravenscar Profile".

The wording should be phrased: "It is equivalent to a list of the following pragmas".

Approve intent of AI: 11-0-0.

Alan will update the AI.

AI-249/07: Ravenscar Profile for High-Integrity Systems

On Sunday, Alan presents an updated AI. A number of minor wording changes are suggested:

- Add "configuration" pragmas to the first sentence of static semantics.
- Alphabetize the list of restrictions pragmas.
- "Runtime" → "Run-time".
- "a user to define a runtime profile" → "defining runtime profiles"
- "Profile_identifier" → "The profile_identifier"

Approve AI with changes: 9-0-0.

AI-251/06: Abstract Interfaces to provide Multiple Inheritance

Several people complain that the discussion and examples in the AI hasn't been updated. That is very confusing.

Pascal is confused, so he writes some examples:

```
type T is tagged ...;

type I0 is interface;

type I1 is interface;
type I2 is interface with I0 and I1;

type T2 is new T with I1 and I0 and
   record
      ...
   end record;

type T3 is new I1 with I0 and I2 and
   record
      ...
   end record;
```

Objects can be declared for T2 and T3 and for I1'Class.

Pascal doesn't like the ability to derive from interfaces. Tuck points out that you have to allow derivation from them because they match formal abstract types, and you have to be able to derive from them or you have a contract model problem.

Someone asks if T3 should match a formal derived for I0:

```
type D is new I0 with private;
```

It seems like it should.

Steve asks if we can talk about class of all interface types. Yes. That sort of wording happens elsewhere in the standard.

Possibly say a type is "derived from any type it implements". Alternatively, try to generalize the "derive from" terminology rather than inventing a new term "implements".

Randy would prefer the option

```
type T3 is tagged with I1 and I2 and I0 and record ...;
```

This emphasizes that this is starting a new derivation tree; deriving from I0 (say) does not do that. There is some support from the group for that.

What is wrong with this syntax:

```
type I1 is interface;

type I2 is new interface I1 and I2 and I0;

type T3 is new I1 and I2 and I3 with ...;
```

"**interface**" is an unreserved keyword, so it can be a type name. So that makes a type derivation and an interface derivation impossible to distinguish. Therefore, the declaration of I2 above would lead to an ambiguous grammar.

Tucker suggests:

```
type I2 is interface new I1 and I2 and I0;
```

Pascal suggests:

```
interface type I2 is new I1 and I2 and I0;
```

The following suggestions are written on a whiteboard:

```
A)             type I2 is new I1 and I0 interface;

B)             type I2 is new interface with I1 and I0;
```

```
C) (original)     type I2 is interface with I1 and I0;
```

We take a straw vote on these possibilities. For A: 4-6-1; For B: 5-5-1; For C: 8-1-2. C carries.

We discuss the syntax to for derived types. Change the syntax for derived types to:

```
      type T3 is new I1 and I2 and I3 with ...;
```

There are two reasons for this change: Putting the interfaces after the "**with**" seems unnatural. This also gets rid of the appearance that the first interface given is special.

Alan asks if just dropping the "**interface**" would change the meaning. (The Ada syntax tries to avoid cases where a single change has a different, legal meaning.) No, because there is also the record part (which is required for tagged types). Interfaces have no record part.

Mike wonders if this is really a tree. The view downwards is a tree. He continues by asking whether dispatching operations are still constant time. No, some may require a lookup.

If we have the following types:

```
      type T is tagged ...;

      type T1 is new T with ...;

      type T2 is new T and I with ...;
```

then:

```
      X: T'Class;

      if X in I'class then ... -- (1)

      Y : I'Class;

      if Y in T'Class then ... -- (2)
```

Can we do the membership at (1)? If so, we may have to do a lookup of some sort. (2) is similar.

Tucker suggests that if either entity is an interface, then you allow the membership test and the conversion (with appropriate checks) as long as there is an overlap in the subgraph, i.e., as long as the membership test could return True (or the conversion could succeed). This is only needed for classwide types, it isn't necessary for specific types (including specific interfaces).

Steve says that classwide types aren't interface types. Tucker wants to be able to be able to call them interface classwide types. You then would need to say "specific interface type" in type declarations.

Steve asks about the rule that we "pick one and use it" in the case of conflicts in inherited operations.

If two inherited operations are conforming but not fully conforming the type declaration is illegal.

This rule is confusing. Bob writes an example:

```
      type I1 is interface;
      procedure P (X : I1) is null;

      type I2 is interface;
      procedure P (X : out I2) is null;

      type I3 is interface with I1 and I2;
```

Is it just illegal to call P for I3 (as the two routines are homographs, and thus any call would be ambiguous)? Or is the type illegal?

Steve suggests a rule that they must be overridden if not fully conformant but are mode conformant. (Otherwise, parameter names and default expressions could be different.)

There are names which are commonly used (Put, Load, Save), so conflicts are going to happen. Bob would like a way to handle conflicts that come up. Forcing an override for routines that have subtype conformance partially works. But we still have the problem for different modes or subtypes.

Tucker says that a way to name both routines would be too complicated. He wants most of the cake, rather than none of it. Java does not solve this problem. Eiffel's solution is way too complicated.

Summary: Rewrite the *entire* AI. Steve will update the AI.

AI-265/03: Partition Elaboration Policy

It is commented that the "Post compilation rule" is really "Static Semantics". [Editor's Note: This is wrong. "Configuration Pragma" is defined as a post compilation rule, and throughout Section 10 and Annex D, it is written this way. It would be inconsistent (even if technically correct) to change this to "Static Semantics".]

The legality rule should either should be a Post Compilation rule (written without an ordering), or be rewritten to say "already apply" to the partition. The original rule is: "If the Policy_Identifier is Sequential then Pragma Restrictions (No_Task_Hierarchy) must have already been specified for the partition." The suggestion is to change it to: "If the Policy_Identifier is Sequential then a Pragma Restrictions (No_Task_Hierarchy) must already apply to the partition."

Erhard suggests using something like D.2.2(5). The wording should be consistent. We eventually agree with this.

In the wording section, get rid of all the paragraph numbers. There are not any paragraph numbers in the normative text of the standard. The reference to 10.2(6) is incorrect anyway: it should be 10.2(10).

Some members are confused about the "**begin**" of the environment task. They are directed to 10.2(10-12). They remain concerned that users may confuse this with the "**begin**" of the main subprogram. They suggest changing this to "immediately after the elaboration of the library items". We decide this is better.

Change: "In this mode of operation" to "When this policy is in effect", because this is not a non-standard mode. (How could it be? It's in the standard!)

The third paragraph of dynamic semantics has way too much discussion. Simplify it a lot to something like: "Tasking_Error is raised before the call of the main subprogram." Don't re-explain how exception handling works. If this is important, mention it in an AARM note.

Move "Partition_Elaboration_Policy => Concurrent" so it isn't in the Implementation Advice. Again, get rid of "mode of operation" and say "as defined by the standard (see 10.2)".

Alan is queried as to why it is necessary to prohibit potentially blocking operations during startup of the environment task? This rule makes it impossible to reuse any code that has a lock. Yet such code is likely to work fine, as there can't be any other tasks running. Moreover, you can't prohibit deadlock. Of course, you are allowed to raise Program_Error in a deadlock situation (such as a task actually trying to block).

Thus, the group concludes that the implementation advice and the dynamic semantics that prohibit potentially blocking operations should be removed. Replace that by an implementation advice that if it is blocked during the declarative part of the environment task, then Program_Error is raised.

Erhard worries about interrupts that arrive before handlers are attached. There is always such a time period. Perhaps we need to say that the result is implementation-defined. Better, we should reuse the C.3 wording about blocked interrupts.

Approve intent of the AI: 10-0-1.

Alan will update the AI.

AI-285/01: Latin-9, Ada.Characters.Handling, 32-bit characters

Pascal explains the AI to the group. It addresses two areas: support for Latin-9, and support for 32-bit characters. Pascal believes that these are different problems, and should be handled in separate AIs.

Tucker would rather say “corresponding packages” for Wide_Wide_ rather than spelling out exactly which ones. However, that wouldn’t be as clear as the current description.

Bob is concerned that “Image” is defined in terms of “Wide_Image”, and “Wide_Image” is defined in terms of “Wide_Wide_Image”. He would prefer a single leap. Randy points out that Wide_Wide_Image is optional, so care is needed.

Mike complains about the name Wide_Wide_Character. He is reminded that new names in Standard are dangerous - they can silently change the meaning of a program. This name is relatively safe in the sense that it is unlikely that people are using it already.

Tucker wonders why you can’t write character literals of Wide_Wide_Characters in Ada source. After much discussion, we agreed that this should be allowed, but it should be a separate option.

Why do we need 32-bit (Wide_Wide) characters? Because they are putting mathematical characters outside of BMP (as well as dead languages and archaic languages). Some users will need these characters.

What about full source representation of the language in Wide_Character? Kiyoshi reports that there is a push in SC22 to allow full wide characters in identifiers.

But that brings up the questions that no one really wants to answer: how do you define which characters are letters? how do you define case equivalence? Mike says we should just reference the existing character standard, and say that anything defined to be a “letter” there is a letter in Ada. Randy thinks that this is likely to be very complex in the compiler and in the run-time. Tucker suggests use anything out of row 00 as a letter. Kiyoshi says that would not be acceptable to Japan, which is preparing a standard for which characters are allowed in identifiers.

Mike will research this, and try to determine just how complex the code for determining a letter would be. These would become classification tables to use in Wide_Handling, and compilers would use them or something similar for handling identifiers. He also will include information about UTF-8.

The discussion turns to Latin-9.

Pascal notes we don’t need Latin-9 support if we have full BMP support for identifiers, because Latin-9 is just switching some BMP characters around (in terms of coding).

Pascal’s AI defines Ada.Characters.Latin_9.Handling and so on. Kiyoshi says that this is weird, because Standard.Character represents Latin_1. That means it is weird to treat a Latin-1 character as if it is Latin-9. Others think of Character as an 8-bit coding representing any character set.

Pascal understands Kiyoshi’s concern, and tries to explain it with an example. Consider the enumeration identifier “ÿ” (latin small letter y diaeresis). E’Image(ÿ) = “ÿ” in Latin-1 (there is no upper case version), but “Ÿ” in Latin-9 (there is an upper case version). So we would need the identifier semantics to be changed depending on the character set. Pascal claims that this is important to reading French.

So Pascal is leaning toward thinking about full BMP support, which does not have this problem. Others disagree; it seems unlikely that the 8-bit Latin-9 representation will disappear. We could require the implementation of a Latin-9 mode.

Someone asks if there is any real need for Latin-1. But we can’t change to Latin-9 without breaking existing programs, as it would be a silently incompatible change.

Tucker comments that part of supporting localization is being able to support the local character set. That implies that there should be some way to do this.

Tucker wonders if the rename of the Ada.Characters.Handling could specify the character set. That would work, but it would not be obvious in a program what character set it depended on. It would be better if there was some explicit indication. He suggests legislating a Latin-9 mode as an alternative to the usual Latin-1 mode, and using a configuration pragma to select the mode. This should be hooked into 3.5.2(5) somehow. Pascal is unenthusiastic.

Pascal will try to rewrite the AI.

AI-287/01: Limited aggregates allowed

We consider the author's questions. The first question is "should subtype_marks be allowed for array components?" Everyone says that we want the consistency. Bob says that there is a lot of extra wording to support that.

The second question is whether we should require that the subtype_mark statically denotes the component subtype versus a run-time check. Bob says that he wanted it like extension aggregates. However, these use a run-time check. Bob then asks about the last time this was discussed – in Cupertino, we voted for a run-time check in this case.

Tucker suggests using wording like 4.8(9-10). However, this too broad (it is for a subtype indication, not a subtype mark). Another possible wording to use is 4.3.3(8), which describes the check for extension aggregates. But this is too narrow, as the ancestor must be constrained and tagged record. We need this to work for all types. In addition, we don't have the "dueling discriminants" problem of the ancestor part.

Bob would like the subtype to be "compatible". But there is no such case in the language. The closest is the allocator case. That uses the discriminants from the subtype mark. And allocators don't slide.

Bob wonders what the human would expect to see. The response is that the human would like to see what is actually happening. That argues for being close to static matching (at least for elementary types).

Tucker is afraid that some types cannot have an appropriate subtype to specify if you require static matching. For instance, discriminant dependent components don't have any subtypes which would statically match.

Several people suggest that there would be value in allowing a constrained subtype to set the discriminants of a component.

Bob comments that we would like the following to work:

```
type R (Len : Natural) is
   record
      S : String (1 .. Len);
   end record;
X : R := (Len => Y, S => String);
```

where the component S automatically takes its constraint from the discriminant.

Bob will rewrite the AI based on these ideas.

AI-296/00: Vector and matrix operations

John reports on the progress on this AI. Brian Wichmann has done a prototype implementation which uncovered three issues with the draft: sliding of parameters, division by zero, and accuracy of scalar product.

AI-297/02: Timing Events

Alan explains the changes from last time. The problem from last meeting is to be addressed by AI-303 (the Timing_Event_Handler protected object needs to be library level, as the callback needs to have the object exist).

The first paragraph about Timing_Event_Handler is irrelevant (the accessibility check will insure this).

Do we need to have a library-level requirement for Timing_Event objects? Tucker asks if we can't just say Cancel_Handler is called when the object is finalized.

Another suggestion is to make it an access type:

```
    type Timing_Event_Access is access all Timing_Event;
    for Timing_Event_Access'Storage_Size use 0;
```

Then the accessibility check does the trick.

Randy objects to the ugly syntax of calls with this idea. This objection seems pretty convincing.

Is this problem solely because the object is returned in a handler? No, it happens because the object is likely to be linked in a global data structure.

Randy would like to see the solution for AI-303 before deciding this issue. After some argument, we move on to the other parts of the proposal. Cancel_Handler needs to return the previous handler; otherwise you have a race condition. Consider having an overload that doesn't return the handler. Alan disagrees. The main race condition is whether the timer went off or not (these are a one-shot). If other tasks are involved, all bets are off anyway. So a Boolean out parameter is enough.

Fix "boolean" in all lower case in Is_Handler_Set (it should be "Boolean", of course).

We discuss AI-303 in order to have a better understanding of the solution to the library-level problem.

Afterwards, Alan says the we'll use "pragma No_Deeper_Objects(Timing_Event);", defined in AI-303.

Approve intent of the AI: 7-0-4.

Alan will update the AI.

AI-298/02: Non-preemptive dispatching

Alan explains the changes. The locking policy was folded in. A discussion was added to the AI to explain why the "token" model is necessary.

The third bullet in dynamic semantics needs to take into account interrupts (as an interrupt could change the priority). Alan says the last sentence of that bullet should be removed. Fix typo in the previous sentence "...retain{s} the execution token."

Erhard thinks that the "token" language is too complex for a simple concept. Various suggestions are made. One is to define the processor as non-preemptive. But it seems better to give that a different name – "token" is suggested again (that is, we're back to where we started).

Alan notes that we might want to allow combinations of tokens in the future. Mike agrees; if there was a "both" policy needed, that would be easier to define with the token approach.

Tucker thinks that this is still too complex. He says that all we need to say is that a new task becoming ready is not a task dispatching point in this model. That means that in this policy, D.2.1(8) is not true.

After thought and discussion, Alan says that that idea seems to work.

Tucker suggests that D.2.1(8) should be fixed to say that these are new task dispatching points. Bob notes that the introduction paragraph D.2.1(1) needs to get rid of preemptive.

Steve Baird and Tucker discuss D.2.1(8) back and forth. They agree that D.2.1(8) should just say that “A task dispatching point is when a task dispatching policy requires a task to go back to the ready queue.” Then, move the remainder of D.2.1(8) to D.2.2.

Alan will try rewriting this AI again, and come back either with a rewrite as suggested by Tucker and Steve, or a reason that it cannot be done.

Approve intent of the AI: 11-0-0.

AI-299/02: Defaults for generic formal parameters

Bob comments that he just wrote some code that could have used these. Mike Yoder also says that he also would use this.

Straw vote on keeping the AI alive: 5-5-1.

We ask the members who would kill this AI why they feel that way. The consensus is that this feature is not important enough at this time.

No action on AI: 8-1-1.

AI-304/00: Reemergence of “=” in generics

In the e-mail discussions about this, Robert Dewar said he was completely opposed to this. Pascal expresses agreement with Robert.

Tucker says that he thinks that they (the Ada 95 designers) made a mistake in this area. “=” should have been defined to compose for all record types, not just for tagged types. Of course, compatibility concerns at the time would probably have made it difficult to sell this idea. For scalar types, (especially numeric types), it would be hard to write a useful algorithm that would work if the operators could change meaning. (Think about Text_IO.Integer_IO.) But that isn’t true for record types.

Pascal repeats that causing the runtime behavior of existing programs to change is *very* dangerous.

Erhard says that he would like to see a fully worked out proposal. Everyone would be for this idea if it wasn’t an incompatibility.

John points out that the incompatibility directive from WG9 would allow us to consider this AI.

Tucker thinks that this should be an operational attribute for a type. Mike Yoder comments that the smelly solution is to have a pragma to apply to a type which says to compose “=”.

Erhard would just change the semantics. Compilers are likely to have a switch to go back to Ada 95 semantics, so any problem code could be compiled with that switch. But, it is likely that users wouldn’t even know they had a problem until deployment. We don’t want to cause an Ariane-style problem.

John thinks that programs that worked around the lack of composition probably would continue to work even if this was changed.

Gary comments that some people use generics to get the original “=” as a trick. Pascal agrees that he has seen (and written) such code as well.

Erhard suggests a configuration pragma Composite_Equality_Composes_Properly. The pragma also would be allowed on a single type. What pragma applies to the type declaration would determine whether or not the “=” would compose.

It might be valuable to have a way to specify that it is done the old way. Then a user could insure that the semantics would not be changed by a maintenance programmer.

We take a straw poll on keeping this AI alive: 7-2-2.

The group agrees that any solution must not be upward incompatible. We might include text to suggest that the default for this may change to composition switch someday.

Tucker will try to find a solution.

AI-305/02: New pragma and additional restriction identifiers for Real-Time Systems

No_Task_Termination should say “It is implementation-defined...”

Simple_Barriers should say “a static Boolean expression or the value of...”.

Tucker asks to make Max_Entry_Queue_Length implementation-defined. After discussion, it was decided that that is wrong; the AI is correct.

The requirement for Detect_Blocking is not a Post Compilation Rule, it is Dynamic Semantics; change the heading.

What does “foreign language domain” mean? What not say in imported subprogram or machine code insertion? Someone says that that does not include a callback to Ada from foreign code. Bob would like to keep it vague.

Steve Baird says that this statement already is true. If you violate any Ada semantic requirements in C (by stomping on memory, for instance), all bets are off. So this should be no more than a note. You may not say “may not”; must say “might not” (this is an ISO requirement for the language of standards).

Approve with editorial changes: 10-0-1.

AI-307/01: Execution-time clocks

Alan explains the proposal. There are two versions. The time type version is similar to an Ada 9x proposal. The package version is simpler.

Pascal notes that this does not seem to be a big change to the compiler either way, just the runtime. Randy notes that there are already multiple time types; the big work is going from one to two.

Bob wonders why this needs a new time type. The answer is that the accuracy of CPU time is not necessarily the same as that of monotonic time.

Tucker wonders about the cost of implementing this. He asks what is the advantage of waiting on the time of another task. Alan doesn’t have an example. Tucker wonders why the time needs to include an indication of the task.

Tucker thinks that we should get rid of Clock_Id and just use Task_Id There doesn’t seem to be any reason for the separate Clock_Id.

Tucker prefers the package approach. He thinks (advanced) end users could create this on real-time operating systems today.

Mike Yoder asks if this is a timely notification when a clock expires. Alan answers that that depends on the underlying system; we include metrics so we can tell how it works.

Straw poll on the proposals: New time type: 1; Package: 6; Abstain: 4

Tucker asks (again) why we have a Clock Id. Alan thinks it is because it came from POSIX. OK, we’ll get rid of Clock Id (replacing it by Task_Id).

Time_Of should return a CPU_Time. It should not have a Task_Id parameter.

Pascal would get rid of the separate Initialize and Finalize procedures, making the Task_Id an access discriminant. Tucker goes further, putting the Task_Id as an argument to Arm. He'd also overload Arm with a CPU_Time.

Tucker would prefer to have all the math operations for CPU_Time. The reason that they are not provided is that comparing or operating on times from different tasks is meaningless. Tucker doesn't care if the values come from different tasks. Mike says that this is basic strong typing.

The basic problem is that the origin of time for each task is unspecified. Tucker suggests that ">" etc be defined as converting to the appropriate time spans. He would prefer to specify that the origin is zero; if the underlying OS doesn't use zero, the implementation needs to correct for that. That seems possible. Mike Yoder says that you might not want a zero point, as you might be getting it from an OS.

Time_Span has to cover an hour. Pascal notes that this makes "-" not very useful. Randy notes this is just like Ada.Calendar and Duration.

Summary:

Get rid of Clock_Id; force each clock to start at zero; is it intended that these aren't comparable; get rid of Initialize and Finalize, Arm takes the Task_Id, and overload with Time_Span, or with CPU_Time. (Arm with a CPU_Time is useful for time budgeting.)

Approve intent of the AI: 3-0-7. Since that isn't conclusive, we check that the group wants to continue to work on this AI. That passes easily: 9-0-1.

AI-310/01: Make 3.9.3(7) an overloading rule?

If this was an overloading rule, then it would be possible to remove operations from an untagged type.

Steve Baird thinks that this seems like a bad use of the abstract mechanism.

Erhard points out that the abstract call would have to remain a candidate for dispatching, else you couldn't dispatch on them. That would be devastating. Thus, we could only do this if the call could not be a dispatching call. That means that would only be a name resolution rule for subprograms that are not primitive operations of a tagged type. If it is a private type, it gets messy.

Ben comments that he is sympathetic to the problem. But he's concerned that we're fiddling with the most fragile part of the compilers.

Does this rule (name resolution only if not a primitive operation of a tagged type) cause problems with generics? It is claimed that it does not.

Tucker suggests that 6.4(8) should be modified instead of changing the nature of the rule in 3.9.3(7).

Straw vote on keeping the AI alive: 3-4-3.

It is suggested that perhaps the problem is with using the abstract mechanism for two different problems. But using a different mechanism to solve this problem does not draw any interest.

Straw vote on pursuing the AI only for untagged types: 4-4-3.

Pascal will rewrite the AI to try again.

AI-313/01: Self reference in subprogram access

Bob: This would be a difference from the interpretation of the type name inside its declaration; usually a type is a current instance, not the type itself.

Steve Baird notes that the workaround is illegal – you can’t have a parameter of an incomplete type. (You could if it was a tagged incomplete, see AI-217-04.)

Tucker makes a motion to classify this AI as No Action. Gary seconds the motion.

No action on AI: 8-0-2.

AI-314/00: Standardize Discard_Names

Mike Yoder says that he asked people at TopLayer. The people he asked only cared about storage minimization and that the attributes would still work for debugging purposes.

Pascal says let’s leave this implementation-defined. Otherwise, we force every implementation to change.

Mike Yoder says that the advantage would be to testers. That even with Discard_Names, ‘Image would still work in debugging code.

Tucker suggests that when adding Discard_Names to existing code, it would be good that ‘Image continue to work.

The obvious implementation is to substitute some Integer'Image for the enumeration type. But even compilers that do that currently (GNAT and AdaMagic) don’t agree on whether to use the representation or the position value as the source integer.

Straw poll: should we standardize effect of Discard_Names? This is defeated 3-7-1.

Erhard: Discard_Names is not in the index of the RM, the editor should fix that.

No action on AI: 8-2-0.

AI-315/00: Full support for IEEE 754

Mike Yoder explains that he’s read the Figueroa paper, and thinks it doesn’t go far enough in some cases. For instance, you need per task state (otherwise you have a mess when tasks switch). He also thinks we need an intervals package.

Mike Yoder will take this on. Pascal is also interested, and will provide advice.

AI-317/01: Partial parameter lists for formal packages

Tucker explains the proposal. Primarily, omitted parameters are specified by “others => <>”.

Steve Baird wonders if there is any complication with the matching rules. No one can think of any.

The implementation effort is probably relatively small – it is “just” a generalization of (<>).

Erhard asks about the problem that prevents saying “Formal.Formal.xxx”. It seems related to this issue; Tucker is asked to fix that problem as well.

Erhard asks if there is a reason that the formal parameter names disappear when an actual is provided. Tucker says that the formal and actual type may have different properties (such as “=” or “:=”). We would have to define which properties are available where via what name. That’s much too complicated.

Approve intent of the AI: 8-0-2.

Tucker will rewrite the AI to provide wording.

Detailed Review of Regular AIs

AI-51/10: Size and Alignment Clauses for Objects
AI-109/07: Size and Alignment Clauses for Subtypes

Pascal asks Steve to volunteer to handle these AIs, along with AI-291. He agrees.

AI-147/10: Optimization of controlled types

We discuss the changes; all are discussed in the e-mail in the appendix to the AI.

Pascal thinks that bullet (a) of the wording seems imply that any of these optimizations can only be done for types that don't have an implicit Initialize call. The group agrees that it is confusing.

Tucker suggests replacing it with "If removing an implicit Initialize call, it does not invoke a user-defined Initialize, and"

Erhard counters with: "the implicit Initialize call to be omitted if no implicit Initialize call that invokes a user-defined Initialize call is removed, and"

We start writing the wording on the whiteboard:

"no implicit Initialize call that invokes a user-defined Initialize procedure is omitted"

Change the "controlled type if" to "controlled type provided that:"

"all calls to user-defined Initialize procedures are preserved, and"

Erhard says that this would prevent optimization if a call to Initialize occurred in an Adjust or Finalize. That would be impossible to tell.

We try several more alternatives for the wording:

"all calls to Initialize that invoke a user-defined Initialize procedures are preserved, and"

"if a call to Initialize is omitted, it does not invoke a user-defined Initialize procedure"

"such an implicit Initialize call does not invoke a user-defined procedure, and"

"for an implicit Initialize call does not invoke a user-defined procedure, and"

"any omitted Initialize call does not invoke a user-defined procedure, and"

It is intended to add a "to be honest" AARM note that this does not apply to any Initialize calls which happen to be in an Adjust or Finalize that is omitted.

"any omitted Initialize call is not a call on a user-defined Initialize procedure, and"

We finally agree to this wording.

Add a comma after Adjust in the first new sentence, remove hyphen from "non-limited".

Approve AI with changes: 9-0-1.

AI-178/01: Which I/O operations are potentially blocking?

Erhard says that he opposes this, as he says it might have a non-blocking I/O. Randy (and others) reply that this is defining *potentially blocking*, not what *is* blocking.

Erhard says that he would want the definition of what is potentially blocking to be implementation-defined. The group thinks that's horrible, you couldn't write much of anything portably in that case.

Tucker asks why anyone would want to use Set_Line_Length in a protected object (without being able to use Put). No one can think of a reason.

Bob believes that the current wording of the standard is fine. This AI probably should be a ramification.

Erhard is worried that answering this question might cause implementers to raise Program_Error on calls to Text_IO that would otherwise work. Other members retort that that is what *should* happen. He suggests making the AI No action.

Pascal objects, as he thinks that the wording is not that clear and a ramification is useful. Bob brings up Robert Dewar's rule that the RM does not say absurd things. So the questions aren't very interesting (only one answer to each question is not absurd).

Make it a ramification: 8-0-2.

Approve intent of the AI: 8-1-1.

AI-195/12: Streams

Pascal describes the changes to the AI. First, the wording for class-wide availability in 13.13.2(36/1) was clarified. Second, 13.13.2(35) was changed to make the point where checks are made unspecified.

Tucker suggests 13.13.2(27) should say "normal {default} initialization"; also, "passes it to S'Read" (not "initializes").

Steve Baird wonders if we need to say that the order of assignments is unspecified. No, it seems unlikely that it is possible to tell the order because of 11.6 permissions.

Tucker would like 13.13.2(36/1) moved either to 13.1 or 3.9.3, saying "Unless otherwise specified, the subprogram name given in an operational attribute clause shall not denote an abstract subprogram." Pascal objects, as he would prefer not to put rules in place for operational attributes that may limit their applicability. (It is this reason, after all, that caused us to define operational attributes in the first place.) After discussion, Tucker withdraws this suggestion.

Tucker asks if the specific case of 13.13.2(36/1) should be "limited record extension" instead of "limited type extension", because you can't see the components when it is private. Pascal and Randy agree.

Tucker worries that these rules require lookahead; if T'Class'Input is used between the declaration of T and the specification of T'Input, it should not be available. This is missing from the wording.

Tucker suggests putting the 4th bullet of the second bulleted list in 13.13.2(36/1) 2nd, as the other bullets are recursive.

Tucker again asks why bullets 2 & 3 three cannot be changed to:

- when the corresponding attribute of T is available.

Pascal explains (again) that this covered in the discussion of the AI. There are problems with attributes made available in the private part, because descendants aren't required to have an available attribute in that case. And then we could dispatch to an unavailable attribute.

Tucker suggests the following bullet:

- where the corresponding attribute of T is available, provided that if T has a partial view, then corresponding attribute is available at the end of the visible part where T is declared.

This seems to work.

Steve Baird asks about a derived type:

```
package P is
   type LP is limited ...;
private
   type LP is limited ... ;
   for LP'Read use ...;
end P;

package R is
   type D2 is new LP ...;
   -- Has no 'Read.
end R;

private package P.Q is
   type D is new LP ...;
   ... D'Class'Read ...
end P.Q;
```

No, this call can only dispatch within the tree rooted at D; it can't dispatch to D2.

Approve AI with changes: 5-0-5.

AI-196/02: Assignment and tag-indeterminate calls with controlling results

Corrigendum section: Needs a closing > for @xbullet.

Erhard wonders if qualification changes the function to a non-dispatching call. No, 3.9.2(3) says qualified expressions can still be dispatching.

Erhard finds this surprising. In:

```
X : T'Class := ...
X := T'(F);
X := F;
```

Why doesn't this return a tag of T? Because, qualification is for resolution; it is ignored in dynamic semantics.

Bob points out that this is consistent with parameter passing:

```
P (T'(F), X);
P (F, X);
```

We know that these two do the same thing. We want assignments to work the same way.

Erhard gives in to the analogy argument.

Approve AI as written: 9-0-1.

AI-204/02: Interfaces.Fortran must be absent, right?

Randy notes that there is no statement in Annex B that these packages are optional. Some people argue that it was the intent was that these were optional, they ask Randy to show where in the standard it says otherwise. But this is a normative annex; the packages in Annex A do not explicitly say that they are required. It is just implied. The group agrees that there needs to be a statement that these packages are optional.

Erhard points to B.3(61/1), which says that Convention C_Pass_by_Copy is required if C-eligible types are supported. Why is that a problem? He is concerned about people providing the package without the semantics.

Pascal points out that the Interfaces.Fortran assumes that Complex derives from Ada.Complex, which assumes that the representations are the same in Ada and Fortran. This should be a separate idea, with no requirements on the representation.

We read the minutes of the previous discussion (from the 8th (!) ARG meeting in Burlington). We agree the minutes are OK, but the AI doesn't reflect them at all – optionality of the packages is not included, and it is necessary to revisit the wording of the convention pragmas in the Annex (i.e. B.5(20)).

We send the AI back to John to revise.

AI-212/01: Restrictions on configuration pragmas

We discuss how this AI came about.

Gary asks that "must" be added to the summary. Others disagree. We decided to let it stand.

Remove last comma in last sentence of the summary.

Bob suggests changing the recommendation to (See Summary.) as it is a copy.

Approve AI with changes: 9-0-1

AI-219/01: Conversions between related types

Bob asked this question, but didn't provide an answer. He says (in his question) that he believes that the intent of the designers was that this would be allowed.

Tucker comments that we don't want to lose the ability to convert as we know more about the type.

Approve intent of AI: 9-0-1.

Bob will provide wording.

AI-220/01: Subprograms withing private compilation units

Change summary from "separate" to "distinct" to avoid confusion with subunits.

The !corrigendum section is missing the original paragraph.

Tucker says that we've fixed the wrong part. The body is still a body, and still included in the second part. Perhaps it should say:

"If a with_clause of a given compilation_unit mentions a private child of some library unit, then the given compilation_unit shall be either the declaration of a private descendant of that library unit or the body (but not a subprogram body acting as a subprogram declaration, see 10.1.4) or a subunit of a (public or private) descendant of that library unit."

Perhaps make this into bullets:

"If a with_clause of a given compilation_unit mentions a private child of some library unit, then the given compilation_unit shall be either

- the declaration of a private descendant of that library unit; or
- the body (but not a subprogram body acting as a subprogram declaration, see 10.1.4) or a subunit of a (public or private) descendant of that library unit."

The second line is still hard to understand. Factor it out:

"If a with_clause of a given compilation_unit mentions a private child of some library unit, then the given compilation_unit shall be either

- the declaration of a private descendant of that library unit; or
- the body (but not a subprogram body acting as a subprogram declaration, see 10.1.4) of a (public or private) descendant of that library unit; or
- a subunit of a (public or private) descendant of that library unit."

Bob would prefer:

"If a with_clause of a given compilation_unit mentions a private child of some library unit, then the given compilation_unit shall be either

- the declaration of a private descendant of that library unit; or
- the body of a (public or private) descendant of that library unit (but not a subprogram body acting as a subprogram declaration, see 10.1.4); or
- a subunit of a (public or private) descendant of that library unit."

Tucker suggests:

"If a with_clause of a given compilation_unit mentions a private child of some library unit, then the given compilation_unit shall be either

- the declaration, body, or subunit of a private descendant of that library unit; or
- the body or subunit of a public descendant of that library unit, but not a subprogram body acting as a subprogram declaration, see 10.1.4."

This last suggestion is approved.

Approve AI with changes: 10-0-0.

AI-228/02: Premature use of "shall be overridden"

Gary: In summary, "overriden" should be "overridden".

In response: "contruct" => "construct".

Remove "author's notes".

In response, last paragraph, fix: "squirreling".

There "exist" → "exists". In first paragraph of response, last sentence.

Tucker: How does freezing apply here? In the example which has P'Access, P is frozen, so it can't be overridden. It would have to be a default expression. Pascal wants to know where this rule is. After much argument, 13.14(4/1) is pointed out.

There is some discussion about what compilers currently do with these examples. Through the wonders of modern technology (the laptop), several people try this example on their favorite compiler. Gary reports GNAT says: "prefix must be abstract". Tucker says that AdaMagic complains about freezing; as a default expression, it works; the squirreling rename says "fatal internal memory access error"!

The example in the question is wrong, it should not have "abstract" in package Types. It would be useful to distinguish between the function example (non abstract type) and the procedure example (abstract type).

The examples are screwed up, so the AI is sent back to Pascal to fix them.

We discussed squirreling renames, and some members of the group are uncomfortable with the fact that this is not a squirreling rename, but it looks identical to it. That is different from the normal case. Randy says that he thinks the conclusion of AI-211 is correct (that the rename is illegal), but the logic to get to that conclusion is wrong. We probably need a binding interpretation to fix this.

The group concludes that the best solution is to make these illegal. That should be done by inserting a new rule after 8.5.4(5): "if the *callable_entity*_name of a renaming is a must-be-overridden subprogram, then the renaming is illegal".

Approve intent of the AI: 9-0-1.

AI-243/01: Is a subunit of a subunit of L also a subunit of L?

Someone asks if the wording change causes problems? Randy points out an analysis in the appendix.

Approve AI as written: 10-0-0.

AI-255/01: Object renaming of subcomponents of generic in out objects

Tucker suggests changing the wording to leave the original wording alone, changing the last period to a semicolon, and then adding "for a composite type, its nominal subtype is unconstrained if the first subtype of the type is unconstrained, even if the subtype_mark denotes a constrained subtype." This wording is adopted.

Approve AI with changes: 10-0-0.

AI-259/01: Can accesses to atomic objects be combined?

Erhard comments that this is wrong for Volatile, which ought to apply Atomic, not the other way around. Mike Yoder says synchronization is a use for Atomic without Volatile. Tucker disagrees. Volatile says that some other task might beat on the object. Erhard would like the capability of not requiring writing to memory, but still have the indivisible properties of Atomic.

Tucker says that Atomic is to synchronize with other threads or device registers; while Volatile implies some other task might change the object (it might be a large object).

Tucker moves to approve the AI, Bob seconds.

Kiyoshi asks why this is a ramification, it has wording. Randy explains that Implementation Advice is not binding, and historically, those have been ramifications.

We vote on Tucker's motion to Approve the AI as written: 5-3-2. This is not a consensus.

Pascal was asked why he opposed. He says that he doesn't like Atomic and Volatile being tied together. Randy objects to that — that would be a change to the language (which clearly does tie them together). But this is not an Amendment AI; this is not a change to the language.

Erhard asks if this problem is occurring in other languages than C. People say POSIX probably does that.

Someone asks if we need a third pragma to cover this additional case. Even the people who don't like the current semantics don't want that.

Erhard offers a compromise: add Implementation Advice that says that you cannot combine Volatile operations. This then would be implied by Atomic. The wording "A sequence of instructions that read or update a volatile object should not be combined with the reads or updates of another object. The implementation should document those instances in which it is not possible to follow this advice." is suggested.

Tucker suggests that we change the wording from paragraph 16, to "For a volatile object all reads and updates of the object as a whole are performed directly to memory, and shall not be combined with reads or updates of other objects." This is then a binding interpretation, and the title and question should be updated. Make sure that the original question is answered in the discussion.

Approve intent of AI: 8-1-1.

AI-263/01: Scalar formal derived types are never static

Steve Baird explains the changes to the AI. The recent change has to do with arrays. The definition of static string subtype should be broadened by deleting the text "(and whose type is not a descendant of a formal array type)". This was not reflected in the AI as written.

Pascal asks that the string example should be moved into the discussion, and explained. (It is pretty convincing.)

Approve AI with changes: 9-0-1.

AI-269/03: Generic formal objects can be static in the instance

The wording rule that makes the change to make formal objects static seems to be missing. The wording change given is to avoid problems with the missing new rule.

Tucker thinks this was handled in the RM. He points to 12.3(15). Then Bob notes that 12.3(14.c) discusses this case.

We turn back to what we want. Legality rules aren't the issue. We don't want exact evaluation in the body or the private part. For example:

```
generic
   A : in Float;
package G is
   Y : constant Boolean := A/3.0 = 1.0/3.0;
end G;

package P is new G (1.0);
```

If Y is in the visible part, it must be True for instantiation P. If it is in the body, we don't want to insist on that. If Y is in the private part, do we want this to be exactly evaluated, or not? Tucker says we want the private part the same as the body. Others disagree, both because of child packages and because there is no problem implementing exact evaluation in the private part.

Tucker wonders about the legality rules for static expressions. Like all legality rules, these are not checked in the private part, perhaps that would cause is a problem. There are three such rules (4.9(34-6)). There is a rule 4.9(37) that partially eliminates some of these checks for generic formals. So, if we insist on exact evaluation in the private part we need to check these legality rules in the private part of an instance.

AARM 4.9(37.e) is clearly wrong for a formal object that happens to be static.

Tucker summarizes the changes he would make. He would remove 4.9(37). Then, add text to say that neither the legality nor accuracy rules are performed in an instance body, but all of them are performed in the instance specification (including the private part).

Steve Baird wonders if there is a problem with generic subprograms. No, this is essentially the same as a single subprogram in a package.

After some thought, Tucker points out that 4.9(36) would be problem for a generic since you don't know the 'Small. Moreover, this case would not raise an exception at runtime. So it would be hard to write a generic formal decimal value (you would have an implicit contract). So 4.9(37) should still apply to 4.9(36).

We also need a change so that 4.9(24) applies to formal objects. Tucker suggests that we simply should change 12.4(10) to "a formal_object_declaration of mode **in** is a full constant declaration and declares, etc." Bob says this needs to be in the index if we make this wording change.

To summarize the wording changes needed. Recheck 4.9(34-35) in specification of instances; do not recheck or require exact evaluation in an instance body. Never recheck 4.9(36) in an instance. Change *full constant declaration* to include formal **in** parameters.

Approve intent of the AI: 8-0-1.

Bob Duff volunteers to write the new AI.

AI-275/01: Aliased components and generic formal arrays

This is a problem similar to an existing one. The fix is not upward compatible.

The formal type A in example is wrong, as the index subtype in a generic formal must be a subtype_mark. Change 1..10 into Short_Integer or some such.

Gary wonders why type A1 is legal. It is since T's partial view is constrained.

Bob tries to write an example of how this can be a problem. Add the following to the existing example:

```
package P is
   ...
   X : A1;
end P;

package body P is
   type Ptr is access all T;
   subtype Con_Ptr is Ptr(1);
   Y : Ptr := X(1)'Access;
   X : Con_Ptr := Y;
end P;
```

Gary would like to make the array illegal, but that is hard to do. Tucker would rather make the 'Access illegal. "If you somehow end up with something unconstrained, the 'Access is illegal." This doesn't seem easy, either.

Could the constrained access subtype be made illegal in this case? That would be incompatibility, but it wouldn't be that bad. And it would put the illegality on the problem feature.

Generic contract model issues prevent disallowing constraints on just general access types. (Both general access and pool-specific type match generic formal access types.)

The only other solution seems to be look through privacy.

We vote. Approve AI as written: 3-1-6. This is not conclusive enough to adopt.

Tucker will search for a better solution.

Bob suggests a runtime check on access subtypes. Or possibly, make this just erroneous. (That essentially makes access subtypes erroneous.) Tucker says that making the dereference erroneous might work.

AI-295/02: Another violation of constrained access subtypes

Steve Baird describes the alternatives considered. Two of them are fatally flawed, so we're left with the runtime check.

It is suggested that the runtime check might fix AI-275 as well.

Tucker has the beginnings of an idea. He describes a problem with their unbounded string type which has discriminants. When a user allocated one of these on the heap, it suddenly was constrained. So he suggests that the access types (where one is access to a view without discriminants and the other is access to a view with discriminants) are different. That means that we essentially prevent conversions between the view without discriminants and the view with discriminants. Then there is no need to constrain the view without discriminants in the heap.

Steve complains that this would make heap object sizes different, possibly making some objects very large. Tucker is skeptical that that would happen.

Randy wonders if this solves the problem. There is only one access type in the example for AI-275. Tucker explains that an extra legality rule would be needed here.

Tucker will take both AIs and try to find a solution.

Pascal wonders if 3.7.1(7/1) actually fixes the extended example for AI-275. That seems to be true. Is there another way to have a problem with this example? We can't find one, so maybe AI-275 is a red herring in the first place.

AI-303/01: Library-level requirement for interrupt handler objects

Steve Baird says that **new** needs to be added to the instantiation in the example.

Tucker suggests adding a new pragma Library_Level, then saying that Interrupt_Handler requires such a pragma.

If we also added this pragma for generic units, then we could use an assume-the-worst rule. That is, Library_Level types could not be used in a body unless this pragma is applied.

This seems appealing. We discuss the pragma name. Tucker suggests "No_Nested_Instances". Tucker claims that "instance" is used for objects. "Deeper" is preferred: "No_Deeper_Instances". Pascal objects to using "instances" for types. Tucker suggests using "No_Deeper_Objects" for types.

Bob Duff will write this up.

Ben asks if this is inherited by derivation. That seems necessary (otherwise you could convert a nested object of a derived type into a "No_Deeper_Object" type — implicitly!). It must also compose correctly for composite types.

Tucker suggests changing 3.10.2(20-21) to take advantage of this knowledge. If the pragma No_Deeper_Instances or No_Deeper_Objects applies, the static levels in these rules can be defined to be the same.

Partial and full views need to agree on this property, so it must be specified on the partial view. That's unfortunate.