Minutes of the 51st ARG Meeting

27-29 June 2014

Paris, France

Attendees: Steve Baird, John Barnes (except Sunday), Randy Brukardt, Alan Burns (Friday, Saturday morning), Jeff Cousins, Brad Moore, Erhard Ploedereder, Jean-Pierre Rosen, Tucker Taft.

Observers: Steve Michell (Friday only).

Meeting Summary

The meeting convened on Friday, 27 June 2014 at 14:30 hours and adjourned at 13:15 hours on Sunday, 29 June 2014. The meeting was held at ECE near the Eiffel Tower in Paris, France on Friday and Saturday, and at premises of AdaLog in the Paris suburb of Issy on Sunday. The meeting covered almost all of the normal AIs and a few amendment AIs on the agenda.

AI Summary

The following AIs were approved:

AI12-0096-1/01 The exception raised by a subtype conversion with a failed predicate check (8-0-0)
AI12-0102-1/01 Stream_IO.File_Type has Preelaborable_Initialization (9-0-0)
AI12-0104-1/01 Overriding an aspect (7-0-0)

The following AIs were approved with editorial changes:

AI12-0042-1/09 Type invariant checking rules (9-0-0)
AI12-0055-1/04 All properties of a usage profile are defined by pragmas (9-0-0)
AI12-0068-1/01 Predicates and the current instance of a subtype (2-0-6)
AI12-0074-1/04 View conversions and out parameters passed by copy (4-0-3)
AI12-0080-1/06 More presentation errors in Ada 2012 (7-0-0)
AI12-0095-1/01 Generic bodies assume-the-worst whether formal type constrained partial views (9-0-0)
AI12-0097-1/01 Tag of the return object of a simple return expression (7-0-0)
AI12-0098-1/00 Problematic examples for ATC (6-0-1)
AI12-0099-1/01 Wording problems with predicates (7-0-0)
AI12-0100-1/01 A qualified expression makes a predicate check (9-0-0)
AI12-0101-1/01 Incompatibility of hidden untagged record equality (7-0-0)
AI12-0105-1/01 Pre is not allowed on any subprogram completion (7-0-0)
AI12-0107-1/01 Is an access to a By_Protected_Procedure interface procedure a protected access (7-0-0)
AI12-0110-1/01 Tampering checks are performed first (7-0-0)
AI12-0114-1/01 Overlapping objects designated by access parameters are not thread-safe (7-0-0)
AI12-0116-1/01 Private types and predicates (6-0-1)
AI12-0120-1/02 “in out” parameters on a default iterator function (7-0-0)

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

AI12-0064-1/02 Nonblocking subprograms (9-0-0)
AI12-0090-1/02 Pre- and Postconditions and requeues (9-0-0)
AI12-0103-1/01 Expression functions that are completions in package specifications (6-0-1)

The following AIs were discussed and assigned to an editor:

AI12-0038-1/04 Shared_Passive package restrictions
AI12-0079-1/01 Global-in and -out annotations
AI12-0106-1/01 Write'Class aspect
AI12-0111-1/01 Tampering considered too expensive
AI12-0113-1/01 Class-wide preconditions and statically bound calls
AI12-0119-1/00 Parallel operations

The following AI was discussed and placed into promising status:

AI12-0117-1/01 No_Tasks_Unassigned_to_CPU (9-0-0)

Detailed Minutes

Apologies

Both Bob Duff and Ed Schonberg sent apologies for being unable to attend.

Previous Meeting Minutes

John has a list of typos marked up on a paper copy of the minutes. Randy will apply those. Approve minutes with changes: 9-0-0.

Date and Venue of the Next Meeting

The next meeting will precede the HILT conference in Portland Oregon, overlapping with the tutorials. The dates are October 18-19, 2014, starting at 9 am, with a full day on Sunday. Note that WG 9 will be Monday, October 20, starting at 7 am.

Thanks

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

Thanks to ECE and Ada-Europe for the meeting facilities on Friday and Saturday.

Thanks to Jean-Pierre Rosen for the arrangements as well as the meeting location on Sunday.

Thanks to the Rapporteur (Jeff Cousins) for running the meeting.

Parallel Processing

This item was discussed in the context of AI12-0064-1, AI12-0079-1, and AI12-0119-1, which now contain the various parts of the most recent proposal.

Corrigendum 2015

WG 9 has requested a Corrigendum no later than their fall 2015 meeting (most likely in October or November).

After some discussion, we agree that we need to finish it technically by our June 2015 meeting. We might even plan to do it earlier (WG 9 would not mind if we did it early); perhaps even by this October. John will help Randy with the introduction.

Require ACATS tests as part of AI work?

Tucker had proposed in e-mail that we require an ACATS test with each AI. Randy had objected as that would be a burden for bug fixes (Binding Interpretations) to the Standard, especially simple typographic changes.

Tucker proposes that we just require the tests for Amendment AIs, encourage for others. The rule would be informal and enforced similarly to the one for wording. (Prior to 1999, we didn't require wording for an AI to be considered complete; the result was that some AIs were nonsense.) Specifically, we would not approve an Amendment AI that did not have an accompanying ACATS test, just like we will not approve an AI that does not have wording.

Require an ACATS test before an Amendment AI can be approved: 8-0-1.

Old Action Items

Steve Baird has AI12-0016-1 (outline of paper) and AI12-0020-1 unfinished. He says that this is mainly a problem of getting time to work on these. He is now getting some internal pressure to finish this paper so he probably will work on it in the near future.

Bob Duff (AI12-0092-1, AI12-0094-1) and Ed Schonberg (AI12-0002-1, AI12-0058-1) aren't here to explain their sloth.

Tucker Taft has AI12-0058-1 (along with Ed and Van). Randy notes that the next step is to produce some wording changes and verify with Van that they have the needed effect; waiting for Van is not needed. Tucker will try to take a cut at the wording changes by the next meeting.

Current Action Items

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

Steve Baird:

- AI12-0016-1
- AI12-0020-1

- AI12-0090-1
- Problems with iterators and generic bodies (see discussion of AI12-0120-1)
- Problem with multiple controlling parameters in Pre'Class (see discussion of AI12-0113-1) – this might be addressed as part of AI12-0113-1 (coordinate with Randy Brukardt and Tucker Taft)

John Barnes:

- Help Randy with the introduction to the upcoming Corrigendum.

Randy Brukardt:

- AI12-0103-1
- AI12-0112-1 (with Ed Schonberg)
- AI12-0113-1 (with Tucker Taft)
- Changes to 13.13.2(38/3) split from AI12-0106-1

Editorial changes only:

- AI12-0042-1
- AI12-0055-1
- AI12-0068-1
- AI12-0074-1
- AI12-0080-1
- AI12-0095-1
- AI12-0097-1
- AI12-0098-1
- AI12-0099-1
- AI12-0100-1
- AI12-0105-1
- AI12-0107-1
- AI12-0110-1
- AI12-0114-1
- AI12-0116-1
- AI12-0117-1
- AI12-0120-1

Bob Duff:

- AI12-0092-1
- AI12-0094-1

Brad Moore:

- AI12-0119-1

Jean-Pierre Rosen:

- AI for protected type problem raised in the discussion of AI12-0112-1.

Ed Schonberg:

- AI12-0002-1
- AI12-0058-1 (with Van Snyder and Tucker Taft)
- AI12-0111-1
- AI12-0112-1 (with Randy Brukardt)

Van Snyder:

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

Tucker Taft:

- Send a note to Ed Schonberg explaining what he was assigned and why.
- AI12-0038-1
- AI12-0058-1 (with Ed Schonberg and Van Snyder)
- AI12-0064-1
- AI12-0079-1
- AI12-0106-1
- AI12-0113-1 (with Randy Brukardt)
- Create an AI to describe what happens for class-wide preconditions and postconditions if the parameter names are changed. (See AI12-0090-1/01 in the November 2013 minutes). [Editor's note: Perhaps combine with AI12-0113-1?]

Detailed Review

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

If a paragraph number is identified as coming from the working Ada 202x AARM, the number refers to the text in draft 3 of the Ada 202x AARM. Paragraph numbers in other drafts may vary. Other paragraph numbers come from the final Ada 2012 AARM; again the paragraph numbers in the many drafts may vary.

Detailed Review of Ada 2012 AIs

AI12-0038-1/04 Shared_Passive package restrictions

Tucker explains that he simplified this. Erhard doesn't believe it works.

Tucker and Erhard look at an example on the whiteboard. It's unclear anyone else understood what they were talking about (your editor certainly didn't, but perhaps that was because this was discussed after the break on Sunday when many of us would rather think about going home, or lunch, or sightseeing.)

Erhard say that he thinks that it becomes undecidable whether the accessibility works. Tucker notes this is the existing rule, but he's not being very successful at explaining it.

We can't tell if the existing (Ada 95) rule is right, so determining if the new rules are right is impossible.

Send this back to the author for additional examples and understanding, starting at the beginning.

AI12-0042-1/09 Type invariant checking rules

Steve would like an example of the overriding required case in the AI. There may be one in the !appendix.

Inherited invariants do apply to record extensions, this wording now makes those apply.

Tucker wonders why 7.3.2(6/3) only applies to private extensions, since we have rules for record extensions as well. This should say “type extension”. (private => type).

Steve wonders how this works with interfaces. Randy notes that we don't allow interfaces currently. That's in promising AI12-0041-1.

After much discussion, we decide that adding a type invariant from an interface is not harmful; it's just an assertion and there is no semantic problem (it's just a methodological issue). (The problem with preconditions is a semantic problem of precondition counterfeiting, but that's because of the “or”, and that doesn't happen here.)

We have to drop the Language Design Principle, because it really only applies to Preconditions.

Tucker turns to the other new issue: converting to a class-wide type.

It would be possible for a value of type T to leak out via class-wide objects. He added an additional check to prevent this; it's the rule added after 7.3.2(20/3).

We discuss a lot of other holes, but there is no serious issues and making this bulletproof is not possible. (Some of these holes are discussed in AARM 7.3.2(23.a/3)).

The class-wide hole is important because Root'Class is a common thing (used for instance with abstract root types), and Tucker notes that factory methods returning Root'Class are common.

Approve AI with changes: 4-0-5.

AI12-0055-1/04 All properties of a usage profile are defined by pragmas

We start by looking at the wording for the new restriction. Several minor wording changes are requested:

> No task has the CPU aspect specified {to be}[with] a non-static expression.
>
> Name the restriction: No_Dynamic_CPU_Assignment
>
> Each task (including the environment task) that has the CPU aspect specified as Not_A_Specific_CPU will be assigned to a {particular}[single] implementation-defined CPU.

D.13(8.3) should be D.13(8/3) in the !wording.

Expand The Proof following the restriction: ... in D.16{, and this restriction guarantees that the activator always has a CPU assigned}.

It seems that there needs to be a sentence saying that the environment task is always assigned to a particular CPU, otherwise the proof isn't true for the environment task.

Steve wonders if the restriction can contradict an explicit rule in D.16. He thinks that “notwithstanding” should be in those rules. No one else seconds that notion.

We settle on the following wording for the restriction:

> No_Dynamic_CPU_Assignment
> No task has the CPU aspect specified {to be}[with] a non-static expression. Each task (including the environment task) that has the CPU aspect specified as Not_A_Specific_CPU will be assigned to a {particular}[single] implementation-defined CPU. {The same is true for the environment task when the CPU aspect is not specified.} [Redundant:{Any other}[A] task without a CPU aspect will activate and execute on the same processor as its activating task.]

Modify the following AARM Ramification:

> If no CPU aspects are specified, then the program will run on a single CPU, as all of the tasks will be activated {directly or indirectly} by the environment task, the rules require the same CPU to be used as the activating task.

Simplify the Ravenscar notes to avoid talking about priority inversion or another specific problem:

> When the Ravenscar profile is in effect (via the effect of the {No_Dynamic_CPU_Assignment} [Only_Static_Task_Assignment] restriction), all of the tasks in the partition will execute on a single CPU unless the programmer explicitly uses aspect CPU to specify the CPU assignments for tasks. The use of multiple CPUs requires care, as {many guarantees of single CPU scheduling no longer apply}[the guarantees against unbounded priority inversion provided by FIFO_Within_Priorities only apply when all of the tasks are assigned to the same CPU].
> It is not recommended to specify the CPU of a task to be Not_A_Specific_CPU when the Ravenscar profile is in effect. How a partition executes strongly depends on the assignment of tasks to CPUs. [In particular, some task assignments can cause unbounded priority inversion not expected by the single CPU semantics. Allowing the implementation to choose assignments exposes the program to risk of future problems caused by changes to the implementation or even just a choice of different options.]

Approve AI with changes: 9-0-0.

AI12-0064-1/02 Nonblocking subprograms

Change the reference to AI12-0026-1 to AI12-0064-1 in AI12-0119-1; this AI is the closest to the proposal from the parallel subgroup.

The proposal is to call the aspect Potentially_Blocking. The AI has it named aspect Nonblocking. That doesn't work well when one needs to talk about the absence of the aspect. “Not Nonblocking” doesn't read well.

An additional suggestion is that there is a configuration pragma that sets the aspect False for a set of packages.

The aspect would be statically enforced on the contents of a subprogram. That means that none of the things defined as potentially blocking in 9.5.1 would be allowed, along with calls to subprograms where aspect Potentially_Blocking is True.

The default value for the aspect is True, for compatibility reasons.

We would not change the behavior of 9.5.1 "potentially blocking" as that would be incompatible.

There would be value to having a restriction (or configuration pragma?) that would detect 9.5.1 violations statically, as that would be possible with this aspect.

Erhard complains that this means that to add this aspect (set to False) to a subprogram, one has to do that to all of the subprograms that it calls. That could be a cancer of sorts. Tucker notes that the proposed configuration pragma would allow switching the default for a package or set of packages to False, which could greatly reduce the "cancerous" effect.

Randy notes that part of this AI has to be ensuring that predefined and language-defined operations have the correct setting for aspect Potentially_Blocking. We surely wouldn't want integer "+" to be potentially blocking.

Steve notes that his version of the aspect (the one described in the current version of the AI) proposed dynamic checks so he didn't have to worry about generic bodies and other nasty cases. Those will have to be addressed in a fully static version of the aspect.

We need the checks to be static in order to provide the additional safety benefits to parallel operations. Dynamic checks find problems too late (and possibly never, depending on testing).

Tucker will take AI12-0064-1. He will look at the rules that need to be added for language-defined packages and operations, as well as those for the configuration pragma.

Approve intent of AI: 9-0-0.

AI12-0068-1/01 Predicates and the current instance of a subtype

Steve does not like this model. He worries that a tagged value is a new concept in the language.

Tucker notes that there many things that you don't want to be able to ask about a current instance. If the value is "3", what is the address of the 3?

Steve says that this is essentially equivalent to putting parenthesis around the object. (Of course, the semantics of that aren't very well-defined, either. There's a hole in the bucket, a hole. :-)

If the current instance is passed as a by-reference parameter, we need to know what the associated object is. So, we need to add some wording somewhere so that we know what the associated object is.

Steve notes that a type conversion of a current instance of a by-reference type would allow one to get back to the object and allow the use of 'Address, etc. Also, of course, you could pass it as a parameter and surely in the subprogram you could take 'Address. There's no sensible way to prevent that (disallowing passing it as a by-reference parameter is not sensible).

Tucker worries that there are cases were there is no object such as literals and named numbers.

Randy suggests that we adopt different rules for elementary and composite types. He doesn't see any sensible way to make it work for both scalar types and for by-reference types.

Tucker suggests adding at the end "If the type or subtype is by-reference, the associated object with the value is the object associated (see 6.2) with the execution of the usage name."

The idea here is that it is a value for a legality purposes.

Steve still wants to call it a constant view. But that would always be an object, what would it mean for a value like 3? Steve wonders about the current situation; that seems to be in the "question not asked" category.

Approve AI with changes: 2-0-6.

No one wants to see it again, and no one objects to progressing this solution, so we agree to consider it approved even with the low number of approve votes.

AI12-0074-1/04 View conversions and out parameters passed by copy

We briefly discuss the incompatibility. Tucker says that this was only legal in Ada 95 (conversions between unrelated access types), not in Ada 83. And hardly anyone uses out parameter view conversions, as most such things are a mistake. So this shouldn't matter in practice.

The 6.4.1 insertion should be:

> If the mode is **out**, the actual parameter is a view conversion, and the type of the formal parameter is an access type or a scalar type that has the Default_Value aspect specified, then
>
> - there shall exist a type (other than a root numeric type) that is an ancestor of both the target type and the operand type; and
> - in the case of a scalar type, the type of the operand of the conversion shall have the Default_Value aspect specified.
>
> In addition to the places where Legality Rules normally apply (see 12.3), these rules also apply in the private part of an instance of a generic unit.

AARM Note: "these rules [is]{are} needed..."

Approve AI with changes: 4-0-3.

AI12-0079-1/01 Global-in and -out annotations

Tucker discusses the basic idea. SPARK uses

```
      Global => (input => (A, B), In_Out => (C, D), Output => E);
```

There are special items **all** and **null**, as well as a way to describe the (hidden) state of a package.

For full Ada, we'd have to add mechanisms for access types (especially for pool-specific access types).

These are fully compile-time checks.

The default for Global is (In_Out => **all**); unless it is a pure package; then it is is (**null**).

Steve mentions that he would like this to help solve the framing problem – which is to reduce the universe of things that might be affected by a call.

Tucker suggests an additional aspect Updated => P1.X, P2.Y to specify all of the parts that are changed (everything else is NOT changed). This describes the changes to the parameters.

Tucker will take this one. Steve thinks it is very important (in general, not just for parallel operations).

Keep alive: 8-0-0.

AI12-0080-1/06 More presentation errors in Ada 2012

We look at each of these in turn.

Erhard is briefly concerned whether (1) fixes the problem, but he is convinced by the group that it does.

Jeff would like a glossary entry for type invariant: "Type invariant. See invariant."

Approve AI with changes: 7-0-0.

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

Steve explains that he sent a note about additional changes to this AI (on Tuesday, after the agenda was sent out).

Someone asks to explain the problem again. Checks (postconditions) that happen at the end of an entry would not happen after a requeue to a different entry, so the caller is not getting the guarantees that they would expect from the called entry. To fix that, we either want the original entry to have no postcondition; or that the postcondition of the target entry is compatible with the original entry.

For type invariants, Steve is suggesting that requeue is not allowed if either have invariants. Tucker thinks that's too strong. He says that type invariants are for the implementation, not the caller. That suggests that we don't care about whether the invariant check is lost. It might open a hole, but it's not that significant (type invariants have other holes, as mentioned when we discussed AI12-0042-1).

Tucker suggests that we don't want to prevent people from using type invariants if they happen to be using requeues. That sort of restriction is always annoying.

Steve Michell thinks that it only matters if you are requeuing to an external entry.

We don't need to claim that the type invariants are bullet-proof in the presence of requeues; it doesn't have to be exactly right.

We need a section on the Dynamic Semantics of requeue in the AI.

Steve then asks about a postcondition that contains Current_Task (for a requeue to a different task). Randy notes that another similar case is a postcondition that contains a task attribute. Tucker thinks that is OK, because one typically is interested in your own properties. Randy worries that the caller could get a false promise. Tucker thinks that this is not important.

Tucker suggests that formal packages handle this case. Steve and Randy both note that we're using conformance rules for the basis of our rules, so we need an extra rule. What formal packages do or don't do doesn't help us.

We turn to the proposed rule. There is a bunch of possessives that need to be removed. “If the declaration of the named entry...”. Tucker wonders how you would do that. Discriminants are suggested as a case where there is a problem.

It should be OK to requeue on the same task for this case, the same current instance is involved. So there is only a problem for external requeue (if it is an external requeue on the same task, then something is already wrong). So this rule only needs to apply for external requeue.

Our intent is that preconditions are evaluated before a requeue; we don't lie for postconditions. We make no guarantee against lying for type invariants.

Approve intent of AI: 9-0-0.

On Saturday, Jeff notes that the !ACATS test section of this AI is talking about overlong encodings. That needs to be fixed.

AI12-0095-1/01 Generic bodies assume-the-worst whether formal type constrained partial views

Summary:

Within a generic body, we assume{ }[-]the{ }[-]worst {as to} whether {or not a} formal {sub}type [have constrained partial views whether a subtype] has a constrained partial view.

In wording, “Modify {AARM} 6.4.1(6.d/3)...”

AARM Discussion: We assume-the-worst in a generic body whether {or not} a {formal} subtype has a ...

Approve AI with changes: 9-0-0.

AI12-0096-1/01 The exception raised by a subtype conversion with a failed predicate check

Approve AI: 8-0-0.

AI12-0097-1/01 Tag of the return object of a simple return expression

“teh” in the AARM To Be Honest.

Summary:

The tag of a return object is that of the return type if that is specific. Otherwise, it is that {of} the return expression of a simple return statement, or the initializing expression of an extended return statement, unless that {return} statement has a subtype_indication of a specific type, {in which case}[then] the tag of that type is used.

Approve AI with changes: 7-0-0.

AI12-0098-1/00 Problematic examples for ATC

Question: “need {not}[done] complete”.

Add text at the end of the example: “Note that these examples presume that there are abort completion points within the execution of the abortable_part.”

Brad: “the intended use of the feature” instead of the “the intent of the use of the feature”.

The editor will clean up the discussion to reflect the above.

Approve AI with changes: 6-0-1.

AI12-0099-1/01 Wording problems with predicates

Jeff notes that the !corrigendum is exactly the same in both parts for 3.2.4(12/3). Fix that.

Since a derived type does not have a parent type, we need an “if any” in 3.2.4(4/4) and 3.2.4(12/4). Before “apply” in 4/4 and at the end in 12/4.

Steve would prefer for 4/4:

> For a (first) subtype defined by a {type declaration}[type_declaration], {any}[the] predicates of {any} [the] parent subtype and {any}[the] progenitor subtypes apply.

Tucker suggests:

> For a (first) subtype defined by a {type declaration}[type_declaration], {any}[the] predicates of [the] parent {or}[subtype and the] progenitor subtypes apply.

Looking at 12/4:

> If a subtype is defined by a [derived] type declaration that does not include a predicate specification, then predicate checks are enabled for the subtype if and only if {any} predicate checks are enabled for [at least one of the] parent {or}[subtype and the] progenitor subtypes;

We decide that no change is needed for 3.2.4(29.5/4).

Approve AI with changes: 7-0-0.

AI12-0100-1/01 A qualified expression makes a predicate check

In the question: “onsists” should be “consists”

In the discussion: “For example{:}[, we certainly]”

Also, “stricter than” should be “at least as strict as”.

In the wording, replace:

> {A further check that the value satisfies the predicate of the subtype_mark is made, and if the check fails, the effect is as defined in subclause 3.2.4, "Subtype Predicates".}

by:

> {Furthermore, if predicate checks are enabled for the subtype denoted by the subtype_mark, a check is performed as defined in subclause 3.2.4, "Subtype Predicates" that the value satisfies the predicates of the subtype.}

Approve AI with changes: 9-0-0.

AI12-0101-1/01 Incompatibility of hidden untagged record equality

Randy explains where this AI came from. He had written a ACATS test which did this. Ed noted that this case was supposed to be illegal (which GNAT did not catch). But when he implemented that check in GNAT, many things failed in the AdaCore internal test suite. Randy and Ed agreed that the inconsistency was better than the incompatibility (it's less much likely to occur in practice, and might even be fixing a bug), thus Randy submitted this AI.

Erhard wonders how this affects arrays. There was no change for arrays, and thus there is none here.

Steve worries about being able to call a routine whose specification has not been elaborated. But that can't happen, because an object of the type freezes the type, and record equality can't be declared after freezing (that's a new Legality Rule in Ada 2012).

!question ends in mid-air. Drop that part, add “Do we want this incompatibility? (No.)”

“{The Ada 2012 rule} 4.5.2(9.8/3)...”

The summary is missing. Set it to:

> Delete the legality rule about hidden untagged equality.

Add the missing !wording based on the !corrigendum section.

Approve AI with changes: 7-0-0.

AI12-0102-1/01 Stream_IO.File_Type has Preelaborable_Initialization

Approve AI: 9-0-0.

AI12-0103-1/01 Expression functions that are completions in package specifications

Tucker notes that there is a typo in the question: "is does not" (first paragraph after question)

Update the comment in the example in the question, so it's clearer what is being talked about.

> -- {Is_Empty is} not used in this package specification, so {it is} not frozen until the end of the package specification.

Steve worries that the expression function freezing rules don't apply to a completion. The completion hides the specification from all visibility. So that's not a problem.

Steve persists. He takes the example 13.14(10.3/3) and causes a problem.

```
package Pack is
   type Flub is range 0 .. 100;
   function Foo (A : in Natural) return Natural;
   type Bar is access function Foo (A : in Natural) return Natural;
   P : Bar := Foo'Access; -- (A)
   function Foo (A : in Natural) return Natural is
      (A + Flub'Size); -- The expression is not frozen here.
   Val : Natural := P.all(5); -- (B)
end Pack;
```

If the body of Foo doesn't freeze, then we have reintroduced the problem that 13.14(10.3/3) is intended to solve.

Ugh.

Tucker thinks about changing the elaboration rules (so that Program_Error would be raised somewhere), but Steve points out that elaboration checks are simple, it's freezing that is complex. The group agrees that complicating the elaboration rules isn't a good idea.

This is getting messy and it probably isn't worth the effort. So we revert to choice (1), which is that these completions do freeze in package specifications.

Thus we need to correct 13.14(3/3) – declaration_part should be "declaration list". Then it is clear that it does apply to package specification.

"Declaration list" doesn't work if the completion is in the private part, we'd need to freeze items in the visible part.

> ...before it within the same declarative_part {or a declaration of a library package or generic library package, except for} [that is not an] incomplete type{s}; it only causes freezing of an incomplete type if the body is within the immediate scope of the incomplete type.

Steve worries about nested specifications. Assume the inner package with an expression function as a completion. That might cause freezing of everything in enclosing scopes. Yikes! That doesn't happen for a nested body because that itself freezes.

Tucker suggests following the lead of an instance: we would have the expression cause freezing. That is more special than the other possibilities, but it's obviously possible.

So there is a reason for this to be special; surely freezing of entities from other scopes is something we want to avoid. The expression has to freeze so we can set the elaboration bit and not have problems that come from that.

Tucker suggests that we should do that for exception functions as completions anywhere.

He suggests that the model is things that can appear in a package specification freeze as little as possible (like renames-as-body, null procedures, etc.).

So we have to send this back to Randy for wording.

Approve intent of AI: 6-0-1.

AI12-0104-1/01 Overriding an aspect

The redundant wording is replaced by a note to avoid confusion, plus an AARM note to explain the reasoning.

Approve AI: 7-0-0.

AI12-0105-1/01 Pre is not allowed on any subprogram completion

Erhard would prefer to say that this rule applies to "completions of a subprogram declaration".

> A language-defined aspect shall not be specified in an aspect_specification given on a completion of a subprogram or generic subprogram.

Approve AI with changes: 7-0-0.

AI12-0106-1/01 Write'Class aspect

Randy explains the AI.

We start discussing whether this should be inherited. Randy notes that the regular ones aren't inherited and we don't want to change that for this different way to specify it.

In wording (13.13.2(38/3), change to:

> The subprogram name given in such [a clause]{an attribute_definition_clause or aspect_specification} shall statically denote a subprogram that is not an abstract subprogram. Furthermore, if a {specific} stream-oriented attribute is specified for [an]{a specific} interface type [by an attribute_definition_clause], the subprogram name given in the {attribute_definition_clause or aspect_specification}[clause] shall statically denote a null procedure.

Update the !discussion to explain the addition of "specific", use the editor's note for the explanation.

Modify 13.1.1(29/3): if the associated entity is a tagged type, the specification *applies* to all descendants of the type {unless specified otherwise};

Add after 13.1.1(30/3): {Such an aspect is called a class-wide aspect. Other aspects of types or subprograms are called specific aspects.}

Add after 13.1.1(31/3): "The terms class-wide and specific also apply to the corresponding attributes."

This is weird. Try:

"The terms class-wide and specific are used in the same way for the corresponding attributes."

This is still weird.

Tucker wonders whether Write'Class should be defined to be an aspect of the class-wide type. We'd need to make sure that inheritance rule follow properly. But only Type_Invariant is related; so we could handle that specially.

Tucker would like a new AI to address the semantic issues. So pass the interface issue to the new AI, and leave that alone in this AI.

Jean-Pierre argues that it is strange to say T'Class'Write in one case, and Write'Class in the other case.

Randy says that these cases aren't really comparable: the first is an attribute of type T'Class, and the second is an aspect of a specific type T.

He also is concerned that there would be confusion since the contract aspects definitely have a following 'Class. It's easy to imagine a type having something like:

```
with Class'Write => ...,
     Type_Invariant'Class => ...;
```

That is awful.

Steve says that this is a wart, but there is going to be a wart somewhere, and this (Write'Class) is probably the best wart.

Tucker thinks that the AI needs to be rationalized. He volunteers to do this.

Randy thinks that the 13.13.2(38/3) changes need to be separated, because they need to be done regardless of anything changed for Write'Class. (The current wording doesn't work very well for aspects.)

The ACATS test needs to be repaired/removed, because we were unable to decide a course of action.

Tucker will take this AI, Randy will take the 13.13.2(38/3) changes.

AI12-0107-1/01 Is an access to a By_Protected_Procedure interface procedure a protected access

Make two corrections to the Question:

"A[n] proposed..."

```
{package Nested is}
     type Intf is synchronized interface;
     procedure PPN1 (Param : in out Intf) is abstract
        with Synchronization => By_Protected_Procedure;
```

Replace the Summary with:

> A prefixed view of a subprogram with aspect Synchronization being By_Protected_Procedure has convention protected.

Replace the Wording of the AI with:

Modify 6.3.1(10.1/2):

- any prefixed view of a subprogram (see 4.1.3){without synchronization kind (see 9.5) By_Entry or By_Protected_Procedure.}

Modify 6.3.1(12):

- The default calling convention is *protected* for a protected subprogram,{ a prefixed view of a subprogram with a synchronization kind of By_Protected_Procedure,} and for an access-to-subprogram type with the reserved word **protected** in its definition.

Modify 6.3.1(13):

- The default calling convention is *entry* for an entry{ and a prefixed view of a subprogram with a synchronization kind of By_Entry}.

Randy's note: Tucker thinks that the interface subprogram is hidden. As such, this is not a prefixed view. In particular, the 5th sentence of 4.1.3(9.2/2) makes it so that the prefixed view case does not apply here. So there is no problem, delete the note.

Approve AI with changes: 7-0-0.

AI12-0110-1/01 Tampering checks are performed first

Modify the wording change:

> These checks are made before any other defined behavior of the {body of the} language-defined subprogram.

Erhard worries that this wording includes the constraint check that is associated with the subtype conversion part of parameter passing. That's a defined behavior of the call, not of the subprogram body.

Approve AI with changes: 7-0-0.

AI12-0111-1/01 Tampering considered too expensive

Randy explained what he understands about the problem. The primary cost is in the finalization needed to implement the clearing of the tampering state after a call to Reference or Constant_Reference. The cost is **not** in the check itself.

Randy notes that the overhead isn't needed for many calls to those routines (if the finalization of the object happens before any calls, no tampering is possible). The compiler could remove that overhead on those cases (much like other checking optimizations).

He explains that a language-level option is to remove the tampering checks from the Reference/Constant_Reference routines for the bounded containers, since the bad problem can't happen in those containers.

We also talk about a global suppress for tampering checks mechanism. The check itself is fairly cheap; the problem is that the body of Reference needs to avoid the return of a controlled object. So it has to be a full replacement of the container package, a local suppress would not be enough.

Tucker suggests having a configuration pragma (or perhaps an aspect on the instantiation).

J-P suggests having Unchecked_Containers. Tucker does not want to have Unchecked_Containers in the language; it would complicate the definition of the language and give an appearance of a lack of safety. We've traditionally handled this sort of performance issue with pragmas (and now aspects).

Tucker suggests tasking Ed with suggesting a solution (configuration pragma, aspect, or whatever). Within AdaCore, this is thought to be an important performance problem, and Ed would be the person who would need to address it. We certainly would like to hear his thoughts before continuing. So assign this AI to Ed.

Tucker will communicate this to Ed.

AI12-0112-1/01 Preconditions for container operations

Randy shows his example of preconditions and simple postconditions.

Steve thinks that the majority of the postconditions should not be there. Tucker thinks that we probably want postconditions only when the entire semantics could be described as a postcondition. Set_Length for instance (Post'Class => Length(Container) = Length). Otherwise, we don't want a postcondition.

Tucker suggests that perhaps AdaCore would have an intern do this work.

Steve wonders about what happens when Assertion_Policy is Ignore. Randy says that we would need a rule to state that Ignore and a failed precondition is unspecified – we surely would not want implementers to spend any effort preventing problems caused by a precondition being ignored.

Tucker suggests that this gets included in the mail to Ed. Tucker will write this mail to Ed about AIs 111 and 112.

Keep alive: 7-0-0.

Jean-Pierre asks about wrapping containers with protected objects. He notes that protected functions can be executed in parallel; that does not provide exclusion. But one might want to use a function to access a container (especially if the element type is indefinite).

He spends a long time convincing the rest of the group that this is a real problem. Eventually he succeeds, which only makes more work for himself. He (Jean-Pierre) will look at this problem, and propose an AI (with a solution if possible). One possibility is an aspect Full_Exclusion to use on a protected object (to mean no concurrent readers).

AI12-0113-1/01 Class-wide preconditions and statically bound calls

Randy tries to explain the problem.

Randy notes that the body of P11.Proc would be surprised if some other Is_Valid is called. Tucker notes that Is_Valid can be weakened and still follow LSP – so overriding it with a function that always returns True would be OK via LSP. But then P11.Proc could be called when it's own Is_Valid is False. Essentially, with the current definition of Pre'Class, it tells you nothing when you are inside of a body – the Pre'Class known to the body cannot be trusted.

That can't possibly have been the intent.

Tucker suggests that when one inherits a class-wide precondition, the parent types are systematically replaced in the expression with the new types.

Randy worries that wouldn't work for dispatching calls.

Steve worries about a case of an abstract type with a concrete Proc and an abstract Is_Valid. He says that a statically bound call would call an abstract Is_Valid.

Erhard wonders if LSP is being violated in these cases (practically). Randy notes that the rules have to make sense for all uses, whether LSP is followed or not.

Tucker explains that this works for LSP. Tucker notes that Liskov did not consider statically bound calls.

Tucker notes that the parent call case is especially weird here. Your caller would evaluate your Is_Valid. And then your call to your parent's implementation would also call your Is_Valid; your parent's Is_Valid would never be checked. That's very strange. Tucker also notes that we're requiring rechecking your own Is_Valid in the middle of the operation when it might not be true (the setup to make it true might have to follow the call to the parent operation).

Tucker now convinced there is a bug. Steve is worried about the inconsistency (we'd be silently changing the behavior). Tucker argues that we need to fix this bug, most users will bump into it sooner or later.

Steve wonders about a two parameter case:

```
Is_Valid_2 (A, B : T);
G : T'Class;
Pre'Class => Is_Valid_2 (X, G);
```

For type T1, we would have

```
Is_Valid_2 (A, B : T1)
Is_Valid_2 (X, G) -- makes no sense (G does not match T1).
```

So this is an existing problem. But this isn't really related to this problem. It should be handled separately. Steve argues that the 'Class substitution is broken and we need to fix it.

Tucker doesn't want to complicate the basic fix with this complication. Steve will get an action item to investigate and fix his problem in the context of the fix for AI12-0113-1. If it turns out that it naturally gets fixed as part of AI12-0113-1, then we should address it there.

Randy and Tucker will coordinate a fix for this problem on the lines of substitution.

Steve is still concerned about the inconsistency of a fix here.

J-P notes that the idea is that the Pre'Class for the body of the Proc (the procedure in the original example) of T1 is that of the Proc of T1. He notes that sounds like common sense.

Tucker reiterates that the problem is with the description of inheritance.

Tucker notes that a pre-test for the precondition doesn't work with the current rule:

```
X : T;

if Is_Valid (X) then -- Statically bound
   Proc (X); -- Implicit call to Is_Valid is dynamically bound.
end if;
```

If X actually is a T1, the explicit Is_Valid call is not going to work, because the actual implicit call would effectively call Is_Valid for T1. It would have to be written as:

```
 if Is_Valid (T'Class(X)) then
```

That seems like another data point in favor; no one is going to write the above until they've been burned.

Keep alive: 7-0-0.

AI12-0114-1/01 Overlapping objects designated by access parameters are not thread-safe

Fix !summary:

> ...nonoverlapping objects {designated}[designed] by parameters of access type.

In !wording, remove the changes from AI-52, just assume they are changed. (Specifically “any language-defined subprogram” instead of “the same”.)

> “...and all parameters of access type{s} designate...”

Steve worries that parameters that are by-reference could overlap with a designated object. The wording doesn't cover that.

> “...as all objects that are denoted by parameters that could be passed by reference or designated by parameters of an access type are nonoverlapping.”

Approve AI with changes: 7-0-0.

AI12-0116-1/01 Private types and predicates

Steve and Tucker would prefer to delete the second sentence of 13.1(9/3) and 13.1(9.1/3) and then generalize it and then follow those two paragraphs with a general restriction:

> If a representation item, operational item, or aspect_specification is given that directly specifies an aspect of an entity, then it is illegal to give another representation item, operational item, or aspect_specification that directly specifies the same aspect of the entity.

Drop the change to 3.2.4.

Redo the discussion to make sense based on the change to the wording.

Brad notes that the example in the question should have a full type of **null record**, as **private** is never a full type.

Summary: An aspect cannot be specified on two views of the same entity.

Approve AI with changes: 6-0-1.

AI12-0117-1/01 No_Tasks_Unassigned_to_CPU

"No CPU aspect is specified to be statically equal to Not_A_Specific_CPU. The CPU aspect is specified for the environment task." Drop the second sentence of the original version.

Set the status to Promising.

Promising with changes: 9-0-0.

AI12-0119-1/00 Parallel operations

parallel would be a new keyword.

A parallel block would look like:

```
parallel
   x := ...
and
   y := ...
and
   z := ...
end parallel;
```

Tucker says that it would be a suppressible error to do something that isn't safe for parallel operations. (For instance, to call a function for which aspect Potentially_Blocking is True – see discussion of AI12-0064-1. There also would be rules associated with the Global aspect.) This would allow users to bypass the checking if they were sure what they were doing.

A parallel loop would look like:

```
for I in parallel 1 .. 1_000_000 loop
   ...
end loop;

for E of parallel Any_2 loop
   ...
end loop;
```

Usually, these would be broken into chunks (implicitly):

```
parallel
   for I in 1 .. 500_000 loop
      ...
   end loop;
and
   for I in 500_001 ,.. 1_000_000 loop
      ...
   end loop;
```

```
        end parallel;
```

Tucker then shows some method of splitting an object that otherwise would prevent the parallelism by using the whiteboard wonderfully. (Erasing entities and adding lines.) It's not possible to follow this with linear notes. The basic idea is that the map-reduce view of the world would be supported, so that partial sums (or whatever) for each chunk is reduced at the end. This is supported by a new construct:

```
        Total_Count := Count'Reduce (Reducer => "+", Identity => 0.0);
```

The reducer is a function with two parameters and a result, all of the same type (the type of Count). The identity is the initial value of Total_Count.

Someone asks what happens if one of the branches of a parallel block ends because of a transfer of control (that is, a non-local exit or goto, a return, or the propagation of an exception).

```
        parallel
           x := ...
              -- goto/exit/return/exception
        and
           y := ...
        and
           z := ...
        end parallel;
```

When a transfer of control out of the statement occurs, then the other threads are stopped. Not necessarily immediately, but as soon as possible. The transfer of control can't happen until the other threads are stopped (or complete on their own).

Brad will take this AI.

Keep alive: 9-0-0.

AI12-0120-1/02 "in out" parameters on a default iterator function

Reorder the wording for 5.5.2(6.1/4):

> A container element iterator is illegal if the call of the default iterator function that creates the loop iterator (see below) is illegal.

Steve asks about this in a generic unit. For a formal derived type, looping legality can't be known.

Randy notes that this problem already exists (it's not related solely to these new rules). If the root type has a reversible iterator, and a derivation of that type has only a forward iterator, then a generic could use a reversible iterator on the formal (derived) type. But an instance using the derived type ought to fail (but of course there is no recheck in a generic body). This is clearly a problem, Default_Iterator should not be changed for a derived type. But this is a different problem, so Steve is given an action item to create an AI for this.

The exception statements are quite vague. Tuckers asks if these could say something about the calls. Randy notes there are a lot of different calls, and the assignment could also fail (if the cursor is controlled and Adjust is called, any exception could be raised).

Tucker suggests making that a user note.

> Note: A generalized iterator can propagate an exception as a result of a call or assignment...

Erhard wonders what can handle these exceptions. That's important, and Randy was trying to cover that. It is propagated by the entire loop statement.

So add after 5.5.2(13/3):

> Any exception propagated by the execution of a generalized iterator or container element iterator is propagated by the immediately enclosing loop statement.

Delete all of 5.5.2(10/3 and 13/3) in the AI, including AARM notes. We won't add a user note.

Add a sentence in the !discussion about describing where an exception propagated can be handled. In particular, one cannot handle an iterator exception in the body of the loop.

Change subject to "Legality and exceptions of generalized loop iteration".

Approve AI with changes: 7-0-0.