Minutes of ARG Meeting 61

Warsaw, Poland

14-16 June 2019

Attendees: Steve Baird, John Barnes, Randy Brukardt, Jeff Cousins, Brad Moore, Erhard Ploedereder, Jean-Pierre Rosen, Justin Squirek, Tucker Taft, Tullio Vardanega, Richard Wai (Saturday afternoon only, via Zoom).

Observers: Pat Rogers (Friday only). Juan Antiono de la Puente (before lunch on Friday).

Meeting Summary

The meeting convened on Friday, 14 June 2019 at 11:40 hours CEST and adjourned on Sunday, 16 June 2019 at 13:20 hours CEST. The meeting was held in Warsaw, Poland, in room 1B at the Institute of Aviation on Friday and Saturday, and in a conference room at the Marriott hotel on Sunday. The meeting covered all of the AIs on the agenda that are considered for Ada 202x (there was a number of "for the record" AIs that were not discussed).

AI Summary

The following AIs were approved with editorial changes:

AI12-0191-1/11 Clarify "part" for type invariants (6-0-4)
AI12-0210-1/04 Type Invariants and Generics (10-0-0)
AI12-0214-2/02 Boolean conditional case expressions and statements (10-0-0)
AI12-0240-6/04 Global aspect and access types used to implement Abstract Data Types (7-0-3)
AI12-0326-2/03 Bounded errors for procedural iterators (9-0-1)
AI12-0333-1/03 Predicate checks on out parameters (9-0-1)
AI12-0335-1/02 Dynamic accessibility check needed for some requeue targets (8-0-2)
AI12-0336-1/02 Meaning of Time_Offset (9-0-1)
AI12-0337-1/01 Simple_Name("/") in Ada.Directories (8-0-2)
AI12-0338-1/02 Type invariants and incomplete types (8-0-2)
AI12-0339-1/01 Empty function for containers (10-0-0)

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

AI12-0280-2/03 Making 'Old more sensible (10-0-0)
AI12-0312-1/04 Examples for Ada 2020 (9-0-1)
AI12-0334-2/02 Predicates in Global (10-0-0)

The following AIs were discussed and assigned to an editor:

AI12-0239-1/02 Ghost code
AI12-0268-1/01 Automatic instantiation for generic formal parameters
AI12-0302-1/02 Default Global aspect for language-defined units
AI12-0334-1/04 Predicates and Global/Nonblocking

The following AIs were discussed and voted No Action:

AI12-0280-1/01 Contract_Cases aspect (10-0-0)
AI12-0326-1/02 Consequence for incorrect Allows_Exit (10-0-0)

The intention of the following AIs were voted, but then were discussed again later in the meeting (the final results are above):

AI12-0214-2/01 Boolean conditional case expressions and statements (10-0-0)
AI12-0240-6/03 Global aspect and access types used to implement Abstract Data Types (9-0-1)
AI12-0326-2/02 Bounded errors associated with procedure iterators (10-0-0)

The following AIs were discussed and assigned to an editor, but then were discussed again later in the meeting (the final results are above):

AI12-0239-1/01 Ghost code
AI12-0280-2/02 Make ‘Old more sensible
AI12-0334-2/01 Predicates and Global/Nonblocking
AI12-0335-1/01 Dynamic accessibility check needed for some requeue targets

Detailed Minutes

Welcome

Steve welcomes everyone.

Apologies

No one sent apologies. However, Alan Burns, Gary Dismukes, Bob Duff, and Ed Schonberg are not attending.

Previous Meeting Minutes

No one had any changes to the minutes of the electronic meeting #60G. Approve minutes: 10-0-0.

Date and Venue of the Next Meeting

We don't have the energy for an electronic meeting in the summer. So our next meeting will be a face-to-face meeting in Lexington, Massachusetts, October 5-7, 2019.

Ada-Europe has announced that the 2020 Ada-Europe conference will be 8-12 June 2020, in Santander, Spain. That means that next summer's ARG meeting will be 12-14 June 2020.

Schedule Change

WG 9 has decided on a resolution to provide a bit less than one year for comments on the proposed Ada 202x Standard. The deadline will be June 1, 2020. The ARG will then have no more than a year to finish up the Standard. WG 9 is concerned about the requirements for revision at least every 10 years, so no more time will be available.

The WG 9 resolution will be handled by an e-mail ballot [Update: The ballot was declared unanimously passed on July 1st, 14 votes in favor having been recorded.]

Ada 202x Scope

On Saturday afternoon, we decided not to start/restart work on any additional AIs on our agenda. While we did not take an official action to this effect, this means that we are not currently planning to address in Ada 202x any of AI12-0229-1, AI12-0214-1, AI12-0243-1, AI12-0139-1, or any alternative of AI12-0197.

We instead suggested that everyone with homework work on that for possible discussion during our Sunday session. We indeed spent the entire Sunday session working on (and in some cases, approving) revisions created on Saturday afternoon.

Thanks

We thank Tullio Vardanega for the meeting arrangements. We also thank the Ada-Europe organizers for their hard work. We thank Randy for his hard work. Thanks to Steve for running the group and the meeting.

Unfinished Action Items

Steve Baird has the only unfinished action item, the “dynamic accessibility is a pain to implement” AI (AI12-0016-1). We did not spend any time talking about it this time.

Current Action Items

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

Steve Baird:

- AI12-0016-1
- AI12-0268-1 (really, make a new proposal)
- AI12-0280-2

John Barnes:

- Assist Tucker Taft in creating a paper to explain the model of Global and specifically of compound objects. See discussion of AI12-0240-6.

Randy Brukardt:

Editorial changes only:

- AI12-0191-1
- AI12-0210-1
- AI12-0214-2
- AI12-0240-6
- AI12-0333-1
- AI12-0335-1
- AI12-0336-1
- AI12-0338-1
- AI12-0339-1

Brad Moore:

- AI12-0312-1

Justin Squirek:

- AI12-0239-1

Tucker Taft:

- AI12-0302-1
- AI12-0334-2
- Create a paper to explain the model of Global and specifically of compound objects (with John Barnes). See discussion of AI12-0240-6.

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 21 of the Ada 202x AARM. Paragraph numbers in other drafts may vary. Other paragraph numbers come from the final consolidated Ada 2012 AARM; again the paragraph numbers in the many drafts may vary.

Detailed Review of Ada 2012 AIs

AI12-0191-1/11 Clarify "part" for type invariants

Randy explains the changes. Essentially, parts need to ignore privacy, so the rules need to be Dynamic Semantics.

The answer to the author's question is that the current wording is fine, delete the question.

Steve asks whether limited with incomplete views have invariant checks. It would seem that the "nominal type" of an incomplete type where no completion is available has no parts (and probably represents a possible leak).

Defining such a rule seems ugly, but it seems necessary.

```
with P;
package LP is
   type Feedback is record   -- "Feedback" was on the flip chart page we started with.
     F : P.T;
   end record;
end LP;

limited with LP;
package P is
    type T is private
      with Type_Invariant => ...

    procedure Q (X : access LP.Feedback); -- In body, munges X.F.
```

This is hard, we need a new AI to deal with this issue. (It's been in the type invariant rules forever). The solution probably uses "nominal type".

Approve AI with change: 6-0-4 (Erhard, Justin, J-P, Jeff abstain).

Steve will handle the limited with problem (this became AI12-0338-1 before the end of the meeting).

AI12-0210-1/04 Type Invariants and Generics

Randy explains the "leak" note. Tucker suggests that the wording say "list of known mechanisms whereby this kind of leak can occur". He doesn't think the wording should be positive. Steve suggests : "list of known mechanisms which allow this kind of leak". Several people prefer Tuck's version.

Delete the Editor's Musings from the !discussion.

Brad notes a typo: acheieved

Approve AI with changes: 10-0-0.

AI12-0214-2/01 Boolean conditional case expressions and statements

Steve asks if these can be static. Probably should allow that.

Typo in wording after 4.5.7(21/3): "is is evaluated".

Steve wonders why the wording says "converted to the type of the case expression". That is needed for class-wide types.

4th line, "...containing {this} choice condition [that is True] is [is] evaluated."

Jeff notes a wording glitch:

> Otherwise (no choice_expression is True, or multiple choice_expressions are True), [then] Program_Error is raised.

Jeff also notes that curly brackets are missing in 4.5.7(14/3):

> A condition is expected to be of any boolean type. {A choice_condition is expected to be of type Boolean.}

!proposal:

A Boolean conditional case expression consists of a number of alternatives guarded by Boolean conditions. The conditions are such that only one is true ...

... mutually {exclusive}[exclusion] conditions ...

Approve intent: 10-0-0.

AI12-0214-2/02 Boolean conditional case expressions and statements

Randy provided a update overnight, so we take this up again on Sunday.

Randy notes that he did not really give this kind of case expression/statement a name in the formal wording. We should get rid of it in the summary and proposal.

> !summary
> A new form of case expression/statement {without a selecting_expression}[, called the Boolean conditional case,] is introduced.

The AARM note for 4.5.7(21): The middle sentence should be "It cannot be suppressed.".

Modify 4.9(12.1/3):

- a conditional_expression all of whose conditions, *selecting*_expressions, {choice_expressions, }and *dependent*_expressions are static expressions;

Modify 4.9(32.5/3): – This is a new item, so it should say Add after 4.9(32.5/3). And this is a bullet, also fix equals.

- a *dependent*_expression of a case_expression whose associated choice_condition is static and whose value equals False; or

Steve notes that a case_expression where all of the choices are False (or more than one is True) should not be a static expression. Since this is not a check, we need an explicit rule for that.

Add to 4.9(12.1):

> In addition, for a case_expression without a *selecting*_expression, exactly one of the choice_conditions is True.

!proposal

Jeff: The conditions [are]{should be} such that only one is true

All of this wording should be consistent using choice_condition rather than choice_expression.

Same fixes for the case statement.

Approve AI with changes: 10-0-0.

AI12-0239-1/01 Ghost code

Randy (two years ago) suggested that it would be nice if this mechanism could be used for anything that shouldn't be in production code. Possibly partitioning of this code would be valuable.

He also suggested ghost pragmas to be usable for setting configurations.

Jean-Pierre suggests that a good compiler would remove dead code. Tucker claims that this is good documentation.

All assertion expressions are classified as "ghost". No ghost parameters or components, but that isn't necessary for Ada; SPARK has these limitations.

Ghost code cannot write any non-ghost objects. Tucker would like “Ghost in” - it is something that is read, that is not used in the non-assertion code.

Erhard wonders about the local state of an Assertion_Policy. Ghost is designed such that it it is illegal to reference something that it is not enabled.

Erhard notes it is much like a conditional compilation. Randy thinks that is the value of a ghost code mechanism, in that it checks that dependencies don’t break when you turn such checking on or off. That’s a very useful thing not available in Ada now (or many other languages, either).

Lunch (including enormous hamburgers) is delivered to the meeting room and the discussion is suspended.

We slowly restart the discussion as people finish eating.

Tucker asks what we would want to use this for beyond assertions. Randy suggests tracing/logging/debugging code. Tucker thinks that that would need about the same entities as we need for assertions.

We would need ghost files in that case. That seems OK.

So it makes sense to investigate the use of ghost code for tracing/logging. Some sort of partitioning (multiple sets of ghost things that can be controlled individually) would seem valuable for that, that also should be investigated.

We look at Steve’s questions. If we want to support tracing, we need external interactions, so most of these rules should be maximally permissive.

Steve will keep the AI.

Keep alive: 9-0-1 (Jeff abstains).

Later, Steve decides he's taken on too much for now and gives this AI to Justin.

AI12-0239-1/02 Ghost code

Justin provided a new version overnight; we take it up on Sunday.

Tucker suggests that one would like to be able to turn off all ghost code at once. And then also be able to turn off individual policies.

Do we say that Red ghost code can’t access a Green ghost code? Probably not, they could be hierarchical usage. There are already rules to make it an error for a enabled thing to depend on a disabled thing,

Item 7 should be removed for Ada; the extra policies handle the effect if needed, we don’t need to be restrictive.

Jeff wonders if the name Policy is appropriate, it usually is very top-level (like Assertion_Policy).

Policy is used for Ignore or Check. It seems weird to call this a policy.

Perhaps we shouldn’t use Assertion_Policy for this, since that takes aspect names. Tucker suggests **pragma** Ghost_Policy (Ghost_Id => Assertion_Policy);

The intent is that Ghost code can read but not write non-ghost stuff. A call to a non-ghost subprogram from ghost code is OK if the parameters and Global only write ghost objects.

Tucker thinks that the only time that ghost comes in implicitly is a call to a ghost subprogram. Calls to ghost subprograms has to be in ghost code.

Erhard worries that implicit ghost could get abused. A ghost call can only write parameters that are ghost objects.

Keep alive: 10-0-0.

The AI will stay with Justin.

AI12-0240-6/03 Global aspect and access types used to implement Abstract Data Types

Tucker discusses the "modifies" problem. Erhard and Randy note that modifies doesn't prevent conflicts with other global objects, so it effectively doesn't work. Tucker says that his latest approach is "**private in out** File". Randy notes that all that is required is to note that the mode of a compound object is effectively **in out** rather than what it was declared as (usually **in**). In that case, the subservient object(s) are already treated as part of the compound object for the purposes of the Global aspect, it is merely necessary to mention that they can be written (as well as read).

Tucker also thinks that it isn't necessary to mention access parameters in Global. It is obvious that they are going to be dereferenced. So these should be implicit.

This means that the .**all** notation isn't needed at all. Tucker says that Randy and Steve will be happy.

What to name this new thing? Many suggestions are made, of varying seriousness. In this list of suggestions, "param" must be limited or controlled:

```
Global => private in out param
Global => indirect in out param
Global => real_mode in out param
Global => truth in out param
Global => not_fake_news in param
Global => in out param access -- Erhard
Global => own in out param
Global => in out param -- Steve, and only for an in param.
```

Tucker worries that there might be a mistake, if param is a global *and* a parameter.

```
Global => overriding in out param -- only in param.
```

This could be described as "Overriding the default associated with the parameter mode". This last suggestion seems best.

Approve intent: 9-0-1 (Erhard abstains)

AI12-0240-6/04 Global aspect and access types used to implement Abstract Data Types

Tucker sent a new version overnight.

The .**all** stuff was removed. There is new wording for access parameters. And **overriding** was added to the syntax.

In 6.1.2(37/5):

> treated as a read or update {of X} (respectively);

Drop the "(respectively)" two places, it's more confusing than helpful.

In 6.1.2(32/5): "identifies variable state" should be "identifies the variable parts"

Richard sent a typo in e-mail: "augment the Glabal" Steve says it should be "ougment the Glabal", which gets general laughter.

In the summary: "The parmeter mode"

Gary notes that the second paragraph of the Legality Rules has "nothwithstanding".

Richard complained about "physical". Tucker and Steve already battled over that, and Tucker had sent a raft of uses showing that this is appropriate. It would be nice to send Richard that same raft of uses. Tucker will find that e-mail and sent it to him (or to the list).

After further discussion, we decide to accept that particular wording change that Richard proposed, since it is arguably is better. (Steve, who previously complained about this, does like this rewording better.)

Change “trust” to “rely on” in the discussion paragraph Richard mentioned. As to the rest of his comment, the ARG always reserves the right to make erroneous code illegal in a future Standard. There’s no good reason to say that here, and we try to avoid talking about future Standards (that always begs the question of why we didn’t do it now, if it’s such a good idea).

Approve with changes: Erhard, Justin, Jeff oppose.

Erhard thinks that the Global aspect should not be extended into the heap. He is concerned about there being many false positives from the static checking. Tucker points out that the current wording for Global gives no way to avoid false positives – this AI is **removing** many of the false positives.

Justin thinks this is more relevant to SPARK and it makes more sense if we let them go first. Tucker and Steve note that the SPARK ownership stuff does exactly this. Justin will change his vote.

Jeff would like to have more checking; he does not like the fact that this is unchecked. We note that this is intended to be the baseline, with checks defined in the future.

Jean-Pierre says that he is not sure if Global should be in Ada at all. Does it cause anything to be checked? All of the conflict checks are based on top of that. And the aspect itself is statically checked, the usages in a body of a subprogram must be consistent with its Global aspect.

The original vote seems to have been based on several misconceptions. Let's vote again.

Approve AI with changes: 7-0-3 (Erhard, Jeff, Justin abstain).

Tullio suggests that we might need to beef up the explanation of this feature. He goes on to say that we need a ! vision section for this one.

We need a paper in the Journal to explain the model here. Tucker and John will try to create such a paper.

AI12-0268-1/01 Automatic instantiation for generic formal parameters

Jean-Pierre suggests that we just put the instantiation directly in the outer instantiation.

Tucker says that would look a lot like an allocator. This seems doable, and it would be OK so long as it occurs in a place where an instance is allowed.

We talk a bit about allowing an anonymous instance for a stand-alone object.

```
X : Vectors.Vector(Natural, Character);
```

We allow anonymous array types, why not anonymous containers? The usage cases are very similar (especially for vectors).

The only places where an anonymous instance would be allowed is where declarations are already allowed. That should eliminate issues caused by elaboration code occurring in odd places.

We think **new** is better for such an instance:

```
X : new Vectors.Vector(Natural, Character);
```

We think that one can get away without accessing anything else by name in the instance. Prefix notation, object iterators, container aggregates, and user-defined indexing would allow using X without direct reference to the generic unit. Thus, we don't need any mechanism to reach those other entities; if someone needs them, they have to make a named instance.

The syntax for a stand-alone object should have the type name at the end:

```
X : new Vectors(Natural, Character).Vector;
```

So the idea is to allow this (anonymous instance) anywhere that an anonymous array type is allowed, in instances, and in the default for generic formals. That of course means allowing defaults for formal packages and types (else there isn't a place to use this for them).

Steve will take this one. We will incorporate ideas from AIs 268, 215, 297, and 205, and make that a new AI.

AI12-0280-1/01 Contract_Cases aspect

AI12-0280-2 and AI12-0214-2 provide a more general way of solving this problem.

No Action: 10-0-0.

AI12-0280-2/02 Make 'Old more sensible

Replace the summary by:

> X'Old may be used in a potentially unevaluated subexpression so long as the condition controlling the subexpression can be evaluated on entry yielding the same result.

!problem

... ({for example, }[especially] if the object is large ...

Wording. Steve worries about user-defined literals here. These could have side effects or depend on globals such that the answer would change.

Erhard would like to see if this term can be generalized and possibly used in other contexts. Randy notes that the new term "operative constituent" might allow shortening this list of junk.

Try to fold the notwithstanding rules into the existing "potentially unevaluated". Look at generalizing.

Jeff notes that there are several "nothwithstanding"s.

Keep alive: 8-0-1. (Justin abstains; Tucker is out of room.)

On Saturday morning, we bring Tucker up to date on this discussion. Randy asks if there is a better term - "Pre-evaluable" sounds like it is talking about preelaborable. Steve notes that the place that this applies to is rather restricted.

Tucker suggests "known on entry".

Randy will keep this one, and redraft using this term.

AI12-0280-2/03 Making 'Old more sensible

Randy provided a new version on Saturday evening, and we take it up again on Sunday morning.

!proposal

We define the term "known on entry{"} subexpression["] to describe {sub}expressions that

Add after 6.1.1(19/3):

- a literal whose type does not have any Integer_Literal, Real_Literal, or String_Literal aspect specified, or[,] the function specified by such an {aspect}[attribute] has aspect Global specified to be null;

Add before 6.1.1(20/3):

> A subexpression of a postcondition is {*}always evaluated{*} if it is not potentially unevaluated. ...

...not Global => **null** {since} those cause a ...

Tucker wonders if we should give a name to "functions that have Global => **null**" and he wonders if Nonblocking => also should be specified. Steve says that a function could start two tasks and determine the answer by which one finishes first. We should do this separately.

There are still two uses of "not potentially unevaluated".

Steve asks if we should have quantified expressions. That's possible but hard because the iterator is possibly user defined. Tucker thinks we are just trying to get the low-hanging fruit here, not every possibility.

"then" should be "the" added before the word "entire" in paragraph 21 – Tucker.

The terminology is confusing to Tucker and Steve; they do not believe that this reflects potentially unevaluated. There really is a three state ("always evaluated", "never evaluated", or "maybe evaluated"). After discussion, Steve will try to find new terminology for this one.

Approve intent: 10-0-0.

AI12-0302-1/02 Default Global aspect for language-defined units

Tucker still needs to look at the ‘Global of generic formal parameters. Most of them need to be in the default Global for a generic unit.

Steve and Tucker talk about build-in-place, and the conclusion is that there is no problem.

Randy notes that only non-generic units that are Pure are Global => **null** by default, so Tuck needs to describe what those do. Randy suggests that Tuck look at the Nonblocking for some ideas on how those should be handled.

Keep alive: 10-0-0.

This goes to back to Tucker.

AI12-0312-1/04 Examples for Ada 2020

John complains about the 4.2.1(10/5) example. It is not an integer at all. Jean-Pierre says that you have to put this value in ISO notation. We decide that this example needs to be replaced.

Erhard suggests that a line break is needed in 4.5.7(21/3):

```
(case Digit is{LF} when 'I' => 1,
```

4.5.7(21): Use a different name so the sex is more ambiguous (Suggest to use Pat or Chris.)

In 4.5.10(53/5):

```
[parallel for Val of M when Val > 100.0 => Val]
   'Reduce(Reduce, (Sum => 0, Addend_Count => 0), Combine);
```

This seems complex. Tucker wonders if a container Set could be used for this example. Brad will look at that.

5.5(23/5): Use square brackets: for the prefix of Reduce:

```
True_Count : constant Natural :=
   [for J in Grid'Range(2) =>
      (if Grid (I, J) then 1 else 0)]'Reduce("+",0);
```

7.3.2(24/3): There needs to be an explanation of what's going on here. There needs to be some operations that could violate the invariant. Change_Priority or Reschedule are examples. Brad will take this one back.

Tucker suggests that we could imagine a type invariant on the container s that Length(V) <= Capacity(V).

Jeff complains that Brad has not addressed any of the problems mentioned in his e-mail review. That should be done.

Approve intent: 9-0-1 (Justin abstains).

AI12-0326-1/02 Consequence for incorrect Allows_Exit

We are going to use the other alternative.

No Action: 10-0-0.

AI12-0326-2/02 Bounded errors associated with procedure iterators

We discuss the issues. Erhard wonders why this is a bounded error? Statically detecting this is impossible, and a dynamic detection is expensive.

Steve wonders if the consequences are enough. He wonders if the transfer of control does not go to the right place. He is invited to propose wording that he feels is better. (After Tucker mentions that he'd asked Steve to do that multiple times last month, and he never proposed anything.)

Erhard would like a "positive" statement of what **parallel** is for. Tucker will propose something, perhaps a Dynamic Semantics section. Randy wants to ensure that this is a parallel construct.

There is an extra line in the second paragraph of the !proposal.

The first paragraph of the problem statement is confusing. Replace it with:

> AI12-0189-1 introduced the Allows_Exit aspect to specify that a subprogram is designed to allow use {with} [in] a procedural iterator {where the loop body has an exit or other transfer of control out of the loop}.

5.5.3(18/5): Specifying the Allows_Exit aspect {to be} True for a subprogram

Jeff notes that the AARM note following 5.5.3(18/5) is missing a word: An indication {of} whether ...

Erhard says that the same is true in 5.5.3(17/5): The Allows_Exit {aspect} of an inherited primitive subprogram...

Brad: The last sentence of the !problem should have a period.

Randy notes that the last sentence of the AARM Reason after 5.5.3(19/5) is speculating about the future; that has no place in the AARM. Drop that sentence.

No vote for now.

On Saturday morning, we take up this AI again. Tucker sent some new wording overnight:

> *Dynamic Semantics*
> [Redundant: For the execution of a loop_statement with an iteration_scheme that has a procedural_iterator, the procedure denoted by the name or prefix of the iterator_procedure_call (the *iterating procedure*) is invoked, passing an access value designating the loop body procedure as a parameter. The iterating procedure then calls the loop body procedure zero or more times and returns, whereupon the loop_statement is complete. If the **parallel** reserved word is present, the iterating procedure might invoke the loop body procedure from multiple distinct logical threads of control.]

This should have an AARM proof to explain why the text is redundant. Most readers won't have Tucker standing by to explain yet again why this is implied by the rest of the subclause.

> Proof: The stated Dynamic Semantics are implied by the Static Semantics given above and the Bounded Errors given below.

Jean-Pierre does not like the fact that parallel is unchecked. A given iterating procedure either is planning to or is not intending to use parallel execution. He wonders if something could be specified to check that this is parallel, especially since the author will have had this choice. Tucker suggests a Requires_Parallel aspect, parallel would have to be given for such a routine, and not given otherwise.

Approve intent: 10-0-0.

Tucker will try to do this before the end of the meeting.

AI12-0326-2/03 Bounded errors for procedural iterators

Tucker sent a new version of this AI Saturday evening. He added a new aspect Parallel_Iterator.

There is a typo “primitieve”.

The insertion brackets {} are missing around 19/5, the last sentence is new.

Approve AI with changes: 9-0-1 (John abstains).

AI12-0333-1/03 Predicate checks on out parameters

Erhard is concerned that this rule would apply to the reverse conversion. Tucker says it does not. Seems like we need an AARM note here (after 4.6(51)):

> AARM Ramification: The reverse conversion applied to by-copy **out** parameters is *not* a view conversion and thus any enabled predicate checks a*re* performed.

Approve with changes: 9-0-1 (John abstains).

AI12-0334-1/04 Predicates and Global/Nonblocking
AI12-0334-2/01 Predicates and Global/Nonblocking

Tucker explains the problems (common to both of these solutions).

Randy notes the addition of the stream attribute cases to the first alternative; something needs to be done for them in the second alternative.

Jean-Pierre notes that the principle is that a subtype is a subset of the type. So to him, that implies that the subtype is bound by the contract of the type.

Steve argues that in the case where future maintenance adds new subtypes/predicates – you can’t know what the future will bring.

Randy notes that users need their contracts to be trustworthy. If someone changes a predicate that is supposed to be nonblocking to be blocking, that can break some other group’s code. This has to be properly contracted to prevent this – an “implicit global aspect” is an oxymoron.

The access subtype case is described and we decide that should be implicit for that. For predicates, the Global has to be explicit (and can be specified for a subtype-specific cases).

Tucker will try to find a way for predefined Image to not cause a problem (that is, be Global => **null**).

Tucker will try to update alternative 2.

Approve intent: Erhard opposes. He thinks this is an unsolvable problem, will be as hard as accessibility, and it is not worth the effort.

Jeff asks how type invariants are handled. Randy notes that Pre/Post is an issue, but type invariants made his head hurt. They’re formally at the call site. Steve thinks that they need to be included in the subprogram. Randy notes that a formal subprogram needs to know about the predicates. Tucker is convinced.

We abandon the “intent” vote.

Keep alive: 10-0-0.

AI12-0334-2/02 Predicates in Global

Tucker provided a new version of the AI overnight, we look at it on Sunday.

The idea is that a routine can kick the can for dispatching calls, so that the caller can use the appropriate Global for the static call. So that the caller would know more about the actual operations.

The change to 9.5(26/5) was removed, so that paragraph does not need to be mentioned.

Paragraph 9.5(24/5) should say “subtype” rather than “type” in two places.

4.9.1(2/5) needs to require Nonblocking and Global to match for static matching of subtypes. Probably need to define “matching” for Global (“come from the same declaration”).

Paragraph 9.5(xx) needs to mention that Nonblocking is inherited from the ancestor for a subtype, unless specified.

[Editor's note: 9.5(49/5) should not say that the Nonblocking is inherited from the predicate. Probably this paragraph shouldn't be changed at all by this AI.]

It needs to be illegal to specify it False if it is inherited True. Steve notes 13.1.1(18.1/4) covers this for type-related aspects. We need a similar rule for subtype-specific aspects, probably in the same place.

Approve intent: 10-0-0.

AI12-0335-1/01 Dynamic accessibility check needed for some requeue targets

What happens when the check fails? There's no wording to describe that. Program_Error is raised, the wording should be looked up.

“4)” should be “4.”.

The structure of these items should be all written the same way. Steve will take this back to make the wording consistent.

3.3.1 (15) and following has a sequence of steps to use as a guide.

Foo should be Some_Type in the example in the !discussion (the type names should be the same). There are other uses of Foo.

No vote for now.

AI12-0335-1/02 Dynamic accessibility check needed for some requeue targets

Steve sent a rewording overnight.

The static rule excepts “part of the formal parameter”, we need the same here.

“If the target object is not a part of a formal parameter of the innermost callable construct...” Eliminate the references to “enclosing body”, since that is covered by the definition of callable construct in 6(2/3).

Approve AI with changes: 8-0-2 (Erhard, John abstain).

AI12-0336-1/02 Meaning of Time_Offset

Redundant[The Time_Offset for UTC is zero.]

Randy may swap UTC_Time_Offset and Local_Time_Offset so that Local_Time_Offset is the important one.

Approve AI with changes: 9-0-1 (Steve abstains).

AI12-0337-1/01 Simple_Name("/") in Ada.Directories

Modify the AARM note after A.16(74/2):

> The "basename" command ignores a trailing '/' and then returns the part of the name in front of {the trailing}[a] '/'.

Tucker suggests:

> If the filename ends with a '/', and is not a root, then "basename" returns the part in front of all of the trailing '/'s.

The change after A.16(47/2) should be:

> A *root directory* is a directory {that has no containing directory}[whose name cannot be decomposed].

Drop the second sentence of A.16(76.a/2).

In the AARM note after A.16.1(17/3):

> ...Is_Root_Directory if it {is} necessary to {distinguish}[tell between] roots and other simple names.

The last bullet in A.16(81/3) should be: [Editor's note: these references should be to paragraph 82, not 81.]

- the string given as Name is a root directory and{,} [either of] Containing_Directory or Extension is nonnull.

The closing } for the entire A.16(81/3) change is missing.

Drop the editor's note after the A.16(81/3) change.

Approve AI with changes: 8-0-2 (Steve, John abstain).

AI12-0338-1/02 Type invariants and incomplete types

This is a new AI (number just assigned) that Steve sent in e-mail Saturday night.

The "leak" note needs to be incorporated into the new leak note from AI12-0210-1. The editor will do that, this is just an AARM note.

Approve AI with changes: 8-0-2 (Jean-Pierre, John abstain).

AI12-0339-1/01 Empty function for containers

This is a new AI (number just assigned) that Brad sent in e-mail Saturday night.

In the summary, the default value should be an "implementation-defined value". [In the functions, the text should be "*implementation-defined*" to match the rest of the RM, as in 13.7 and A.18.1 – Editor.]

The postcondition shouldn't use prefix notation for Length and Capacity.

Should Empty be an expression function? Yes, that is OK, we want to avoid the extra English definition needed. [Editor's note: This only works for Empty when the container has no capacity, as we don't have the right semantics for any existing function.]

Approve AI with changes: 10-0-0.