Minutes of the 54th ARG Meeting

16-18 October 2015

Bennington, Vermont, USA

Attendees: Steve Baird, Randy Brukardt, Jeff Cousins, Gary Dismukes, Bob Duff, Brad Moore, Erhard Ploedereder, Tucker Taft (not Sunday).

Observers: Stephen Michell, Harold the Mouse (Saturday noonish).

Meeting Summary

The meeting convened on Friday, 16 October 2015 at 13:30 hours and adjourned at 12:10 hours on Sunday, 18 October 2015. The meeting was held in the theater room, at Karin's Place (Robert Dewar's former Vermont home, now being converted to an executive retreat). The meeting covered all of the normal AIs and many of the amendment AIs on the agenda.

AI Summary

The following AIs were approved:

AI12-0172-1/01 Raise expressions in limited contexts (8-0-0)
AI12-0176-1/01 6.1.1(26.4/4) only applies to tagged types (7-0-1)

The following AIs were approved with editorial changes:

AI12-0059-1/06 Object_Size attribute (8-0-0)
AI12-0143-1/02 Using an entry index of a family in a precondition (5-0-2)
AI12-0144-1/02 Make Discrete_Random more flexible (7-0-0)
AI12-0156-1/02 Use subtype indication in generalized iterators (6-1-1)
AI12-0163-1/02 Deterministic queue servicing for FIFO_Queuing (5-0-3)
AI12-0173-1/01 Expression of an extended return statement (8-0-0)
AI12-0174-1/01 Aggregates of Unchecked_Unions using named notation (8-0-0)
AI12-0175-1/02 Preelaborable packages with address clauses (8-0-0)
AI12-0178-1/01 Glitches in examples (7-0-1)

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

AI12-0009-1/03 Iterators for Directories and Environment_Variables (6-0-2)
AI12-0064-1/05 Nonblocking subprograms (7-0-1)
AI12-0125-3/01 Add <<>> as a shorthand for the LHS of an assignment (8-0-0)
AI12-0128-1/05 Exact size access to parts of composite atomic objects (6-0-2)

The following AIs were discussed and assigned to an editor:

AI12-0111-1/01 Tampering considered too expensive
AI12-0139-1/01 Thread-safe Ada libraries
AI12-0170-1/02 Abstract subprogram calls in class-wide preconditions
AI12-0171-1/00 Ambiguity in Synchronous_Task_Control semantics

The following AIs were discussed and voted No Action:

AI12-0015-1/02 Ada unit information (4-1-2)
AI12-0024-1/01 Compile-time detection of range and length (7-0-1)
AI12-0025-1/01 Allow 'Unchecked_Access on subprograms (4-0-4)
AI12-0063-1/01 No_Return functions (7-0-1)
AI12-0087-1/02 Reading the default value of a subtype (7-0-1)
AI12-0091-1/04 Add procedure Sin_Cos to Ada.Numerics.Generic_Elementary_Functions (5-0-2)
AI12-0123-1/01 Add 'Subtype attribute (6-0-1)
AI12-0177-1/00 A name resolution oddity (6-0-2)

The following AIs failed to reach consensus and were assigned to an editor to gather additional information for future reconsideration:

AI12-0075-1/04 Static expression functions
AI12-0164-1/03 Max_Entry_Queue_Length aspect for entries

Detailed Minutes

Apologies

Tullio Vardanega, Alan Burns, John Barnes, and Florian Schanda apologize for not being able to attend.

Previous Meeting Minutes

Approve minutes: 7-0-1.

Date and Venue of the Next Meeting

The next meeting will be in Pisa, Italy, associated with the Ada-Europe conference. We decided the dates last time, as June 11-13, 2016. WG 9 would get a slot on Monday afternoon (June 13).

The following meeting most likely will be associated with the 2016 SIGAda conference, for which no dates or location has been settled yet.

Thanks

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

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

Thanks to AdaCore and Jenny Dewar (Karin's Place) for the fine accommodations and food, as well as the interesting weather (at one point on Saturday it was snowing hard when viewed out one window facing south, and sunny and dry when viewed out of an east facing window) The weather was exact opposite of our previous meeting in Madrid – the Sunday night low was 19F (-7C). That's more than a 50C change.

Thanks to Harold the Mouse for entertainment. The mouse ran around the room for quite a while on Saturday morning. Jenny Dewar told us that the mouse was named Harold because she didn't want to kill it. Steve Michell noted that during the April IRTAW meeting, the attendees had been unable to interest the house cat in Harold. We asked, but Harold was not interested in voting on any of the meeting straw polls.

Old Action Items

AI12-0127-1: Steve Baird was unable to get any input from Florian so he didn't update this AI. His other two AIs (AI12-0016-1 and AI12-0020-1) will be worked on “someday”. [As Creedence Clearwater Revival [CCR] sang, “Someday Never Comes” - Editor].

AI12-0002-1: Gary notes that Thomas needs to respond on the importance of these instances and whether banning these is acceptable.

AI12-0119-1: Brad says the AI approach might need some changes, the Gang of Four is reconsidering the issues. He thinks his container AI might fall out of something else (or not).

Erhard has the files for the examples but hasn't cleaned them.

AI12-0058-1: Tucker will do wording for the Fortran annex, he just hasn't gotten to it. [I hear CCR singing again – Editor.]

AI12-0140-1: Tucker isn't sure whether he understands the problem. Tucker thought that just a fiat (private view statically matches the full view) is enough. Randy notes that might break transitivity of matching – if A matches B and B matches C, but A doesn't match C. Tucker thinks that can be fixed. Which is why this AI was assigned to him.

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
- AI12-0075-1
- AI12-0127-1 (with Florian Schanda)
- AI12-0128-1 (review by Tucker Taft)

Randy Brukardt:

- AI12-0009-1 (check wording in /03 version)
- Create an alternative to AI12-0064-1, using 'Nonblocking.
- AI12-0112-1 (with Ed Schonberg)
- AI12-0125-3

Editorial changes only:

- AI12-0059-1
- AI12-0143-1
- AI12-0144-1
- AI12-0156-1
- AI12-0163-1
- AI12-0173-1
- AI12-0174-1
- AI12-0175-1
- AI12-0178-1

Jeff Cousins:

- AI12-0009-1 (check wording in /03 version)
- AI12-0171-1 (help Bob Duff)

Gary Dismukes:

- AI12-0002-1

Bob Duff:

- AI12-0171-1 (lead; with Tucker Taft and Jeff Cousins)
- AI to define suppress check name Tampering_Check (see discussion of AI12-0111-1)
- AI to propose a simple capability to turn a loop body into a subprogram (mapping to the call-back iterators) – see discussion of AI12-0009-1.

Stephen Michell:

- AI12-0164-1

Brad Moore:

- AI12-0009-1
- AI12-0119-1
- Concurrent access to separate elements of a container (see discussion of AI12-0139-1 in Madrid)

Erhard Ploedereder:

- AI12-0139-1 (protected containers)
- If possible, acquire some examples where Ada containers are too slow compared to other containers (for AI12-0111-1).

Florian Schanda:

- AI12-0017-1
- AI12-0127-1 (with Steve Baird)

Ed Schonberg:

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

Van Snyder:

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

Tucker Taft:

- AI12-0058-1 (lead to produce wording, with Ed Schonberg and Van Snyder)

- AI12-0064-1
- AI12-0079-1
- AI12-0111-1
- Review AI12-0128-1 on request from Steve Baird
- AI12-0140-1
- AI12-0170-1
- AI12-0171-1 (help Bob Duff)
- Parallel reads for existing containers (see discussion of AI12-0139-1 in Madrid, possibly related to AI12-0111-1)
- Create new AI for new Pre'Class problem – see discussion of AI12-0170-1

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 7 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

AI12-0009-1/03 Iterators for Directories and Environment_Variables

Bob wonders if we could come up with a simpler, more general way to do this. Bob would prefer to implement a lambda. That would seem to be a massive change. Tucker would like to see how that would work.

Bob notes that many other competitive languages are adding that. Someone suggests that is because of the lack of nested subprograms in those languages.

Tucker suggests that perhaps we just support a way to turn a loop body into an implicit subprogram. That could be used to implicitly call the Iterate routine that takes an access-to-subprogram.

Bob will take an AI to investigate to possible ways to do this.

We talk a bit about generators (which Ed had brought up); Tucker says they have a "yield". Bob notes that it is very similar to call back subprograms (like the Search here).

Turning back to Brad's AI. Tucker notes that Brad is using the Start_Search open iteration to implement these.

We quickly return to the extension discussion.

The problem with the existing iterators is that if you want state in the iterator, you have to jump through hoops. The model assumes that you can encode all of the state in the cursor. (This is mainly caused by the iterator parameters being mode **in** in the interface; if they had been **in out** state would have been easy to maintain – Editor.)

Returning again to Brad's proposal, the user view is nice.

Given what we have, this is about as best as we could do.

Brad did prototype the directory entry on GNAT.

We should have someone else review this wording.

Steve Baird wonders why Entry_Presence is a "**new** Boolean", which is weird. It should be a private type.

Jeff has been volunteered to check the wording. Randy will also check the wording. Brad will prototype the environment variables examples.

Approve intent: 6-0-2.

AI12-0015-1/02 Ada unit information

Steve Baird wonders if how these work with instances need to be defined precisely. Compiler version number, for instance is implementation-defined. So the details aren't important – it's already implementation-defined.

But the usage cases that Jeff uncovered seem to depend on the GNAT behavior. If implementation-defined behavior is good enough, one can use an implementation-defined package. It hardly needs to be in the Standard. And if it isn't, we need to define a lot more of the behavior – which is a lot of work.

No Action: 4-1-2.

Jeff votes against as he would like to attempt to keep working on this idea.

AI12-0024-1/01 Compile-time detection of range and length

During the discussion of the previous AI (AI12-0075-1) we considered going in the other direction. We have plenty of experience that making unexecuted code illegal has a downside. So this is unlikely to be a good idea.

No action: 7-0-1.

AI12-0025-1/01 Allow 'Unchecked_Access on subprograms

The idea is that this aspect makes the access type big enough to include a context (like parameter of an anonymous access type). Most compilers already have different representations of access types, so this shouldn't pose any insurmountable difficulties.

Steve Baird says that he finds this dangerous. No one speaks strongly in favor.

No action: 4-0-4.

AI12-0059-1/05 Object_Size attribute

Bob wonders if this matches what GNAT does. Tucker says that it does as far as he can tell.

Steve Baird wonders if "aliased" applies to both component and object. No, just to stand-alone object. So, in 13.3(58) swap "component" and "stand-alone aliased object" (two places), which makes it make more sense.

We turn to the statically matching rules. Steve Baird wonders why we mention "nonconfirming" in static matching.

4.9.1(2/3): their Object_Sizes (see 13.3) are the same if either {has been specified to have a nonconfirming value},

13.3(58) has the rule saying that the object sizes are the same for subtypes that statically match. Tucker says this is a fact, a compiler that does not meet it is wrong. (The user can't make this mistake.)

Gary worries that this statement seems like it is a derived property.

Steve Baird wonders what happens if Object_Size is inherited. Randy says that it is recalculated.

Steve Baird wonders about a derived first subtype. Tucker reads 13.1(15/3): it's inherited but only for statically matching subtypes. Thus:

```
subtype S is ... with Object_Size => Blah;
type T is new S; -- Object_Size is inherited.
```

Steve then worries about:

subtype S1 is ... **with** Object_Size => ...;
subtype S2 is S1;

He says that `S2` is a sort of rename (8.5(6) says as much - Editor), and `S2` should statically match. In such a case, Object_Size should be inherited.

Tucker discovers 13.1(14); we need to add Object_Size here, and the last sentence of 13.3(58) can be removed (it would be redundant). Randy suggests getting rid of the parenthesized items rather than adding anything. He notes that having a list of items here is just a maintenance hazard. Jeff thinks that adding ("{for example}, Size and Alignment) would be better.

Also fix the following AARM note.

We probably need an inheritance rule for Object_Size.

For a subtype with no constraint, Object_Size will be inherited. Tucker will try to make some wording over the next break.

Brad gives us typos: “the the” in the wording, AARM discussion 13.3(58), “oject” in the second editor's note, in the discussion, last paragraph “the meaning that is specifies”.

AI12-0059-1/06 Object_Size attribute

After lunch, Tucker discusses his changes. The 4.9.1 rule is more complicated, but it is necessary and no better wording is obvious.

Put “add the following” before the AARM notes following 4.9.1, as they are new.

Fix the typos from the earlier discussion (Tucker did not fix them).

Remove the editor's note about the Pittsburgh meeting; the one following it should be moved to the end of the discussion.

13.3(58):

> AARM Ramification: The intent is that a compiler need[s] support only those ...
> AARM Discussion: Any extra bits required this way will be [trailing] padding bits.

13.1(23) AARM Note:

> ... specified at compile[-]{ }time, or are ...

Approve AI with changes: 8-0-0.

AI12-0063-1/01 No_Return functions

6.5.1(5/2): “return_expression” should have been “raise_expression”.

For most such uses, using a raise_expression directly is sufficient. And the AI12-0054-1 proposal never ended up in the language, so the primary reason for this AI never happened.

No action: 7-0-1.

AI12-0064-1/05 Nonblocking subprograms

Tucker explains that this adds fixes for entry calls that are procedure calls. Also, he added rules for formal subprograms and access-to-subprogram.

Tucker allows blocking nested units in a nonblocking unit. Such nested subprograms cannot be called. Such nested packages can be useful. Erhard notes that the elaboration of a blocking package could block, and that would be nested within a nonblocking subprogram. That couldn't be allowed, either.

Therefore, there is no value to allowing anything blocking to be nested within a nonblocking subprogram. So the wording should specifically talk about nesting within packages (which is a default) and units inside can be different.

Bob notes that it would apply in child units if specified for a parent.

Steve Baird wonders about the Legality Rule; why is the list is different than the existing list for “potentially blocking” in 9.5.1? Because the “external call on the same object” case is not statically detectable.

Randy wonders if “a call on a procedure that implemented by an entry” can be detected statically. What if we have a dispatching call on a procedure of an interface? How do we know if it is implemented by an entry?

Tucker suggests that we use the Synchronization aspect to determine this:

- entry_call_statement or a call on a procedure that renames an entry;
- a call on a primitive procedure of a synchronized tagged type that does not have Synchronization aspect of By_Protected_Procedure.

Randy wonders about a renames-as-body that is an entry. Tucker claims that is impossible because the convention is entry, which couldn't match the spec. (Or it might be Intrinsic, as it is a prefixed view of an entry, but reading 6.3.1 is inconclusive).

We turn to the rules for generics. Tucker has an override for blocking for an instance.

Steve Baird thinks this doesn't work. In the generic body, it takes a nonblocking subprogram (calling a hash function) and passes it to a global access-to-subprogram. It's not clear whether this works or not.

Randy suggests a different model using Hash'Nonblocking, that would force assume-the-worst in the generic body (that is, everything other than the explicit routines would have to follow the rules of nonblocking).

Steve Baird notes that we need to specify that predefined operators other than "=" for records are nonblocking. He also notes that predefined stream attributes and implicit calls need to be handled.

Erhard suggests that using something like type inference might be helpful. Tucker is unsure; SPARK does a lot of that already.

Tucker thinks that some examples – use the container library both ways (using this model, and using the Hash'Nonblocking) – would be valuable.

Randy will work on an alternative using Hash'Nonblocking, including examples.

Approve intent of AI: 7-0-1.

AI12-0075-1/03 Static expression functions

Steve Baird thinks this is a useful extension, as this lets one factor out duplication.

Randy notes that there is a compatibility problem with this version. That could be fixed by an aspect.

We look at the compatibility problem. There are several examples in an AI.

Tucker suggests that we could except these sorts of calls from being illegal, they would become non-static expressions instead.

Thus it raises an exception at runtime; but it cannot be used as a static expression.

Bob suggests that we do that for all types (illegal would go to nonstatic with an exception at runtime). Steve Baird worries about unexpected results, Tucker worries about customers depending upon it.

Straw poll:

(1) No action - 3
(2) Original proposal – 0;
(3) Original proposal with aspect – 1
(4) Static unless fail runtime check - 5
(5) Bob's proposal - 2

Straw poll 2: (best choice)

(1) No action - 3
(4) Static unless fail runtime check – 5.

This idea fixes one compatibility problem, but not the other (the first one in the AI, the case statement one). It's suggested that the case statement problem would be extremely unlikely. Others are not so sure.

Gary suggests running the GNAT test suite on this.

Steve Baird gets the AI, and should make some prototype implementation in GNAT, plus running the GNAT test suite on it.

Approve intent: 5-3-0.

Randy voted against as he thinks this solution is too complicated to implement and describe.

Bob voted against as he says the cost benefit doesn't seem worth the trouble.

Jeff voted against as he feels this is a nice-to-have, not worth the effort.

AI12-0075-1/04 Static expression functions

On Sunday morning, we look at new wording that Steve Baird sent overnight.

The additional condition is a double negative. Can't it be written better? Erhard suggesting using "unless the *function*_name or *function*_prefix statically denotes an expression function and the compile-time evaluation of the call fails any language-defined check."

Steve Baird does not like this rewording, he worries about mixing “ands” and “ors”.

The Note should be an AARM Note.

Steve Baird gives in on the unless wording. The wording should read:

- a function_call whose *function*_name or *function*_prefix statically denotes a static function, whose actual parameters, if any (whether given explicitly or by default), are all static expressions, unless the *function*_name or *function*_prefix statically denotes an expression function and the compile-time evaluation of the call fails any language-defined check.

Approve AI with changes: 3-1-3.

Randy still votes against as he thinks this is way too complicated for the potential benefit. He might be swayed by adopting Bob's suggestion (as then all static expressions would work the same).

We think that we need to revisit this as we do not have a consensus as to how to proceed. That leads to a brief meta-discussion of what votes are considered sufficient to be a consensus. We decide to take that up in the future.

We still would like Steve Baird to get some information about incompatibilities (if any) in the GNAT test suite. It would also be useful to determine Bob's suggestion (of making static expressions that fail a language-defined check be non-static) causes any problems. [After all, it would only change the behavior when a static expression occurs in a non-static context; it seems unlikely that most users would know the difference – Editor.]

AI12-0087-1/02 Reading the default value of a subtype

The question is what is the good for. Erhard suggests that one might want two types with the same default. Bob suggests that it is a constant. That is a constant of the type, so that doesn't work (unless it is a named number).

Erhard says Has_Default_Value can't be illegal in any case. We're confused. He gives an example:

```
if S'Has_Default_Value then
   -- use S'Default_Value
else
   -- use some alternative.
end if;
```

If S'Default_Value is illegal when Has_Default_Value is False, then this doesn't work. That would defeat the purpose of having Has_Default_Value.

We'd have allow S'Default_Value in all cases and raise Program_Error if there is no value.

Yuck.

No action: 7-0-1.

AI12-0091-1/04 Add procedure Sin_Cos to Ada.Numerics.Generic_Elementary_Functions

Jeff says that the view of the UK Ada Panel had been that it should be left to the compiler to optimize such things. As such, he no longer wants to champion this proposal. No one else from the ARG wants to champion it, either.

No Action: 5-0-2.

AI12-0111-1/02 Tampering considered too expensive

Tucker explains his proposal. He is suggesting that we could define a “locked reference”; such a reference would not need a (separate) tampering check. The container would be locked (with a tampering check) and then locked references generated and used; finally the container would be unlocked.

Bob says that the GNAT containers have been reworked and are now as much as 40 times faster in some cases (without changing the semantics). They also implemented a Suppress name for the tampering check.

Bob notes that GNAT implemented an atomic check for tampering in containers, which might not work on all targets but works well on most.

Bob says that if Suppress is in effect at the point of the instance, the return type of Reference becomes non-controlled. There is an aspect Disable_Controlled, which is a compile-time Boolean, based on the value of the suppress flag.

Bob says that the code is erroneous if the check would fail (as the RM says for all suppressed checks).

Returning to Tucker's proposals. He suggests that we limit the tampering with elements check to just indefinite containers, and then make the tampering checks for other containers be just tampering with cursors. (We originally made the containers as similar as possible, which meant that we made the checking rules the same for all of the containers. But we don't really need tampering with elements for definite containers, as changing the contents of an element shouldn't cause a reallocation.) Then his “handle” to precheck tampering for a container would be tampering with cursors for definite containers, and tampering with elements for indefinite containers.

Most of the operations would be implemented on the handle, and they could work in parallel. The “of” iteration could automatically use this code.

Bob thinks that is essentially what he did: move some of the expensive stuff out of the loop.

Tucker suggests that it would require supporting concurrency of tampering checks on the underlying container.

Bob wonders if a read-only handle would help anything. Probably we need to prototype to see if it helps. Possibly “in” would give read-only and “in out” would give read-write, and then declare the read-only one as a constant.

Tucker will try to design this solution (Possible names: “Frozen_Handle”, “Stable_Handle”).

Gary suggests naming this “Stable_Vector”, as it should have the name of the kind of container in it when it is used explicitly. “Stabilize” gives you a stable view. The container would be unlocked when the handle is finalized.

Should we make Bob's suppress check names part of the Standard?

Yes, as a separate AI. Bob will take this AI. With just “Tampering_Check”. (“Container_Check” could be handled by turning off preconditions in Ada 2018, as all of those checks will become preconditions; it doesn't seem necessary to be a separate flag).

Tucker will write up his proposal in more detail.

Keep Alive: 7-0-1.

AI12-0123-1/01 Add 'Subtype attribute

The use case in the AI was provided by Obj'Image (added in the Corrigendum). And it is a can of worms.

No Action: 6-0-1.

AI12-0125-3/01 Add <<>> as a shorthand for the LHS of an assignment

Tucker worries about this getting buried in longer expressions.

He also suggests that being similar with other languages would be an advantage (he's arguing in favor of alternative 2).

Jeff objects, he says new programmers are more flexible than that.

Erhard suggests that using <<>> is too close to existing operators. He would like something much more different.

Different characters would work, $ and [] are suggested.

We turn to discussing alternative 2.

Tucker now suggests :+. There aren't any comparisons, so := is well-defined. (Comparisons don't have the right types for these operations.)

Bob asks why :& is included? It would almost always fail with a Constraint_Error for Strings. Tucker replies that they're for user-defined types (including language-defined types like unbounded string).

Someone complains that

```
X:=-3;
X:-3;
```

are only one character different, but they mean two very different things.

Straw poll:

Prefer Alternative 2: 2
Prefer Alternative 3: 4

Abstain: 2

We return to discussing Alternative 3. What character(s) should we use for this?

The character(s) used need to be visible (it can't get lost in an expression) and can't conflict with other existing uses.

$ is politically incorrect. % is used as a string replacement. # might work (but there might be issues with based numbers). @ would work. ^ is too unobtrusive.

Erhard suggests that $ is close to a 2.

Straw poll (live with):

$	5
#	8 (if it works)
@	8
%	4
<LHS>	3

Straw Poll - Preference: # 2; @ 6.

Looking at the details.

Resolution: Randy notes that the LHS of an assignment statement can be overloaded (with the RHS mutually providing the type), and there might be issues with resolution from plugging the LHS type into the RHS. Bob suggests that if there is any @ in the RHS, then the LHS becomes a complete context. In that case, the resolution is straightforward.

The renames rules are not liked. Use the conversion to a procedure with one **in out** parameter, as in Alternative 2.

Is this view a constant? Randy notes that the Taft anti-aliasing rules would fail in such cases if it is a variable, so there is little use to it being a variable.

Tucker suggests that all of the expansions turn into a qualified expression, which would make the @ uses a constant view. If it was a variable view, one could change the discriminant and that would make a mess.

Should the name “target name” be used? It's not a name, “target” or “assignment target” are suggested. [Well, actually, two minutes later we decided it *is* a name. - Editor.]

Gary wonders if having this as a separate clause seems a bit weird. Bob thinks that it is fine.

@ is a primary. Or is it a name? Bob notes that the LHS is a name, so @ is a name.

Erhard suggests that it is just a variable view of the LHS and the @ is a constant view of the LHS. This seems the same as Tucker's suggestion.

Approve intent: 8-0-0.

AI12-0128-1/04 Exact size access to parts of composite atomic objects

Randy argues that this only makes sense when combined with Volatile. It's only about accessing hardware, an access of hardware should be volatile.

Steve Baird goes further to argue that the item should be Atomic. It doesn't make sense to access a multiword object to exact size.

Tucker suggests that composite atomic objects just do this. Randy notes that C.6(20) and C.6(22/2) would need changes for that. (C.6(20) should also be fixed for volatile components that aren't a multiple of the Storage_Unit size).

Atomic objects are always read or written atomically, even if they are components of a larger atomic object. (4 8 byte components in a 32-bit object, on I86, for instance). OTOH, if they aren't atomic and are part of a larger atomic object, then the atomic object is read-modify-write.

Approve intent: 6-0-2.

AI12-0128-1/05 Exact size access to parts of composite atomic objects

On Sunday morning, we take up Steve Baird's late night update.

Notwithstanding is not liked much.

Randy notes that we have a similar problem with C.6(20), we need to allow bit-mapped volatile components to use a read-modify-write. So we may be able to merge those.

Steve Baird suggests that he move some of his implementation note into C.6(20), rewriting it to read like normative wording. He goes on to suggest appending to C.6(20):

> However, there may be target-dependent cases where reading or writing a volatile object (typically a component) necessarily involves reading and/or writing neighboring storage, and that neighboring storage might overlap a volatile object.

This deletes the second paragraph of the Implementation Note (as it is covered by the above). (Starts with “as another example...”)

Randy notes that 9.10(1/3) says that parts of Atomic objects aren't independently addressable. So we do not need the Notwithstanding text, get rid of it.

Gary wants nonatomic to not have a dash. (All of them in the RM – one – does not have a dash.) “4” should be “four”. “run[-]{ }time” Randy wonders if that is correct; Gary says it is when “run time” is used as a noun]

The very last sentence of the wording has a “which” that should be “that”.

Legality Rules: The change should be placed after C.6(13.2).

Steve Baird needs to rewrite the discussion section. [Your editor is tired of doing his homework for him.]

Randy notes C.6(22) [the Implementation Advice] conflicts with the normative wording. That needs to be fixed, probably needs some similar “however” wording. Probably “except in the case of a volatile but nonatomic subcomponent of an atomic object.” gets added to the end.

Randy will send Steve Baird a summary of the wording changes, he will write up a discussion section, Tucker will review the result.

Approve intent: 5-0-2.

AI12-0139-1/01 Thread-safe Ada libraries

Erhard says that people initially want thread-safe code, not caring about efficiency. He also says that handling the packages individually would take a huge amount of work.

Thus, he wants to mandate (for a special version of the libraries) that they are sequentially consistent.

Randy worries that call-backs would potentially cause deadlock. Erhard intends that those would be a Bounded Error. (He said that he sent a second version with that, but no one has seen it.) [Editor's note: It was stuck in a spam filter, and didn't get delivered until October 28.]

Bob worries that this duplicates the entire library. He worries that this causes a maintenance headache (double the work). Erhard argues that one can use tools to cut the duplication. But building and maintaining the tools would also be a headache (Ada does not have real conditional compilation.)

The “protected” runtime is completely separate, with a different set of types. The other “unprotected” runtime is different.

Erhard describes his second version that no one saw because it was stuck in the mail system. He changed “protected” => “safe”. He added a rule that call-backs are a bounded error. He added some additional justification. He argues that changing the interfaces are a problem for users, because they can't switch easily to the parallel version.

Steve Michell suggests that we might want a configuration pragma to globally change to the protected version.

Erhard says that he considered that, but he decided that explicitly changing the library references was important because many other tools are involved (for source analysis and the like).

Randy wonders how attribute 'Identity works (it returns Ada.Task_Identification.Task_Id), if it has to return one of two (or many) types. This would be a problem for several of the language-defined attributes ('Address, 'Identity for exceptions, etc.) It seems to him that some part of the library has to be the same for all versions. We decide to defer the question as too detailed at this point.

Keep alive: 7-0-0.

AI12-0143-1/02 Using an entry index of a family in a precondition

Erhard suggests that the first paragraph of the discussion be moved to the end. The group agrees.

Erhard gripes about the difference in access in the body versus the precondition

Steve Baird suggests that an alternative would be to allow an optional syntax to specify the name in the specification. But that only works for **protected**, and it seems like swatting a fly with a baseball bat.

Approve AI with changes: 5-0-2.

AI12-0144-1/02 Make Discrete_Random more flexible

Bob: It is hard to implement this right, users usually get it wrong. (The e-mail thread proves that multiple times.) So we should provide it.

Add the correct algorithm to random generation for integers to the discussion.

Steve Baird says that the wording in A.5.2(41) require a uniformly distributed result over the result subtype. But that's not what we want when we have the First and Last parameters.

So add to A.5.2 (41): "..., or in the case of the version of Random with First and Last parameters, over the range First .. Last."

Erhard suggests: "...uniformly distributed over the range provided by the parameters First and Last, or the range of the result subtype." He then withdraws that.

Straw poll: Use a postcondition 6-1-0. [Editor's note: I have no idea what this in relation to.]

[Editor's note: Reviewers suggest that A.5.2(42) was also intended to be changed. Since there are two unanswered questions here, the AI has been reopened to settle those questions.]

Approve AI with changes :7-0-0.

AI12-0156-1/02 Use subtype indication in generalized iterators

Steve Baird discusses the changes to define the accessibility of loop parameters.

We take a series of straw polls on the various parts of this AI.

> Subtype_indication on generalized indicator: 7-1-0
> Anonymous access type, only useful for array component iterators: 4-2-2.
> Clarify the AARM note: 7-0-1.

It's time to quit for dinner (our reservations are in 30 minutes), so we'll look at the wording tomorrow.

On Saturday morning, we continue looking at this AI.

loop_parameter_subtype_indication should be loop_parameter_subtype_definition. That's because there are similar names in 3.6 (discrete_subtype_definition, index_subtype_definition).

Approve AI with changes: 6-1-1. Bob votes against as he does not think these changes are worth the trouble.

AI12-0163-1/02 Deterministic queue servicing for FIFO_Queuing

The implementation model is that there is a sequence count in the protected object (for all queues), and the older arrival is used if multiple tasks are ready. (Or you can just use one queue for all entries, and the order of items provides the order of arrival.)

"earliest arrival time" is misleading; it's never possible for two entries to be queued simultaneously. "The call that arrived first is selected." And then we don't need the sentence about the "more than one call with the same arrival". The last paragraph is similar.

The middle paragraph stays as is; it's about delay alternatives so it is really about time.

The last discussion item last sentence should be changed to: Applying arrival first as the selector provides fairness. (The rest is deleted.)

Wording:

> When more than one condition of an entry_barrier of a protected object becomes True, and more than one of the respective queues is nonempty, the call that arrives first is selected.
>
> If the expiration time of two or more open delay_alternatives is the same and no other accept_alternatives are open, the sequence_of_statements of the delay_alternative that is first in textual order in the selective_accept is executed.
>
> When more than one alternative of a selective_accept is open and has queued calls, the alternative whose queue has the call that arrives first is selected.

Drop the second paragraph of the !problem.

Approve AI with changes: 5-0-3.

AI12-0164-1/02 Max_Entry_Queue_Length aspect for entries

Bob wonders what the user demand for this feature is?

Steve Michell says that this was proposed for future Ravenscar, where they want to relax some of the restrictions on Ravenscar.

The suggestion in the AI about using the precondition doesn't work because a precondition is outside of a protected action, so the length of the queue could change.

One could use a pragma Assert, but it would be in the body, thus it wouldn't be visible to the caller.

Delete the last paragraph of the discussion (it's based on an incorrect idea).

Tucker notes that the restriction says “Max”, so it does appear to be a global maximum, not a default value. We wouldn't want to be overriding a global maximum.

Reword the first Legality Rule:

> If a restriction Max_Entry_Queue_Length applies to the partition, a nonconfirming value of an aspect Max_Entry_Queue_Length (for a type or individual entry) shall be less than or equal to the value of the restriction.

The same for the other Legality Rule. (The first Legality Rule might in fact be a Post-compilation Rule, most restrictions are. Randy will check that.)

Tucker would rather use “-1” as “(representing an unspecified queue length)”.

> If an entry call or requeue would cause the queue for any entry of a type to become longer than the nonconfirming value for Max_Entry_Queue_Length for the type, then Program_Error is raised at the point of the call or requeue.

Approve intent: 5-1-2.

Bob votes against, he says he doesn't think there is any real user demand for the feature, and there is a cost.

AI12-0164-1/03 Max_Entry_Queue_Length aspect for entries

Randy notes that we failed to answer the editor's note about where this aspect should be defined. Bob notes it has to go in Annex D as the restriction is there. So D.4 is fine.

Restrictions are post-compilation checks, that's the rule. Compilers can detect post-compilation checks at compile-time if they can; they don't *have* to wait for link-time.

Approve AI with change: 3-1-3.

Bob votes against for the same reason as yesterday. The language is getting too big – we don't want to add nice-to-have stuff.

As with AI12-0075-1, we don't have a clear enough consensus.

Randy suggests that if this ends up being used in expanded Ravenscar, then we need it, else we don't.

Steve Michell will take this back to IRTAW to find out how it will be used (in particular, will this end up getting used in a relaxed Ravenscar).

Steve Michell says that Ravenscar can be organized by having a PO per task, as mailbox where other tasks call protected procedures to give it work; the task waits on the entry if there is no work.

Brad says that he thinks this came out of a paper from Pat Rogers, perhaps he should be asked.

AI12-0170-1/02 Abstract subprogram calls in class-wide preconditions

Tucker says this draft is wrong. The nominal NT is nonabstract. So we want to check all types (including T itself) for legality when 6.1.1(18/4) is applied. That is the application of 6.1.1(18.2/4).

In the !discussion (toward the end), the type AT needs to be renamed because **at** is a reserved word. **Tagged** is missing in the type declaration.

So undo the change in 3.9.3. And then undo the change in 6.1.1(18/4), or better add "Redundant[(including T itself)]".

All of the examples need to be rechecked that they get the right answers. And redo the discussion. So it goes back to Tucker.

Tucker also has another problem in this area that probably wants a new AI.

Consider an overriding:

```
package Pkg7 is
   type AT1 is tagged null record;

   function F1 (X : AT1) return Boolean;

   function F2 (Y : AT1) return Boolean
      with Pre'Class => F1 (Y);

end Pkg7;

with Pkg7;
package Pkg8 is
   type NT is new Pkg7.AT1;
   overridding
   function F1 (X : NT) return Boolean;
end Pkg8;
```

Consider a dispatching call:

```
F2 (AT1'Class(Obj_of_NT));
```

The precondition that the caller uses is going to call F1 (for NT).

The body, OTOH, only knows F1 (for AT1).

Thus we want two different preconditions.

The solution (took 2 beers + wine last night) is to treat inheritance as a wrapper (a call to the inherited routine), so the precondition that the body knows about gets evaluated again (with the original F1).

(Making something illegal or "shall be overridden" would be better, but it would be massively incompatible and painful to use.)

Much more discussion ensues. A summary is that it is a bug if the new predicate is true but the old predicate is false. (Consider a type Is_Open and Read that requires Is_Open; derived from this is a type Open_File [which is always open], Is_Open is always True for type Open_File. This is OK so long as the old Is_Open is always going to be true for a file of the new type.)

Tucker will write up various ideas to solve this problem. He will make sure the two AIs are consistent. (We probably need to generic NT model to avoid calling things that aren't primitive, to deal with the abstract cases and changes.)

Tucker ought to try to produce this as soon as possible so we can review it.

Keep alive: 6-0-2.

Difficulty of the new AI is hard. Probably the old one too. The priority of these AIs is high.

AI12-0171-1/00 Ambiguity in Synchronous_Task_Control semantics

Some argue that these are designed to handle one task, and this example is handling two tasks.

Bob suggests that the task that creates the object is the only one that can suspend on it.

That's incompatible.

Straw poll: Must work 5, Does not have to work 3.

AARM note D.11(6.a) suggests the intent; it was not intended that multiple tasks could call Suspend_Until_True. ("The count of callers [on Suspend_Until_True] is at most one.")

Bob is concerned about the possibility harming efficiency. Steve would like to know whether IRTAW has concerns about these (either way).

Jeff will talk to Alan about this. Bob will investigate the effect in GNAT with help from Tucker. Bob announces that there are 15 versions of the implementation for GNAT for different targets.

Keep alive: 6-0-2.

AI12-0172-1/01 Raise expressions in limited contexts

Approve AI: 8-0-0.

AI12-0173-1/01 Expression of an extended return statement

Second line of discussion, "xtended".

Approve AI with change: 8-0-0.

AI12-0174-1/01 Aggregates of Unchecked_Unions using named notation

In the wording, change: "or within the selector_name" to "or as a selector_name".

In discussion, "anonynous".

Blow away the parenthisized part of the penultimate sentence. Actually, just drop the whole sentence "But that's a shaky argument...", and the "And in any case," changes to "But".

Approve AI with changes: 8-0-0.

AI12-0175-1/01 Preelaborable packages with address clauses

Randy explains his proposal; he notes that it required more changes than he expected. Most of those are things, like relational operators, that are inexplicably not allowed on static strings and would need to be allowed here. He's not quite as excited by this idea as he originally was.

If we don't make these expressions static, how else could be solve the original problem?

Tucker suggests introducing the idea of preelaborable function, which includes static functions and some additional things (To_Address, Unchecked_Conversion, the parameter(s) have to be static, – but the result type can be anything.

Someone notes that there is potential runtime check. Tucker wonders why, so an example is given:

```
function UC is new Unchecked_Conversion (Boolean, Integer);
C : constant Positive := UC(False);
```

C would have a runtime check. The corresponding check for static expressions makes the declaration illegal. Tucker suggests that we require static matching on the result. Exceptions are OK here, so we don't need any restrictions on the generic instances (no static matching).

Tucker asks about allocators. Steve Baird says that it would make an implicit call to Allocate, which would violate 10.2.1(7). Then Steve Baird notes that the Pure rules explicitly say allocators aren't allowed. So we conclude that they are allowed for preelaboration. And GNAT allows them.

One could ask for the address of an access type via Address_to_Access_Conversion. If 'Address is allowed, then Address_to_Access_Conversion should be allowed as well.

The list of functions included: To_Integer, To_Address (13.7.1), To_Address, To_Pointer (13.7.2), Unchecked_Conversion (13.9).

The words would go in 10.2.1. Change 10.2.1(7):

> A call to a subprogram other than:
> - a static function;
> - an instance of Unchecked_Conversion;
> - functions To_Pointer and To_Address declared in an instance System.Address_to_Access_Conversions;
> - all functions declared in System.Storage_Elements.

Make the priority of the AI medium.

Approve intent of AI: 6-0-2.

AI12-0175-1/02 Preelaborable packages with address clauses

On Saturday afternoon, we look at a rewrite of the AI posted by Randy.

Typo in the third bullet, need an “of” between instance and System.

Bob complains about mixed plural and singular.

Replace the wording with:

> A call to a subprogram other than:
> - a static function;
> - an instance of Unchecked_Conversion;
> - a function declared in System.Storage_Elements; or
> - the functions To_Pointer and To_Address declared in an instance of System.Address_to_Access_Conversions.

Bob would like an example of this problem. The example in the mail is good.

Gary very small editorial. Should be a blank line between the first two paragraphs of the !discussion. Actually, some lines are too long.

Approve AI with changes: 8-0-0.

AI12-0176-1/01 6.1.1(26.4/4) only applies to tagged types

Approve AI: 7-0-1.

AI12-0177-1/00 A name resolution oddity

Steve Baird explains the problem from his e-mail. Tucker says he is worried about a more general problem:

```
package Pkg3 is
   A : Integer := 1;
end Pkg3;

with Pkg3; use Pkg3;
package Pkg4 is
   function F is (A)
      with Post => F'Result = A;
   A : Natural := 2;
end Pkg4;
```

(Tucker's version uses separate objects with the same name rather than overriding functions.)

In this case, the function is returning `Pkg3.A`, while the postcondition is using `Pkg4.A`. That's going to be surprising.

Is this is a pathology or something that we have to fix?

Bob says that he doesn't know of anything reasonable that we could do.

Randy notes that the Post expression might not resolve at this point (entities within it may not yet have been declared); doing a "partial resolution" is a non-starter.

SPARK would like this to be illegal, but it is just too hard to check.

No action: 6-0-2.

AI12-0178-1/01 Glitches in examples

4.3.3(45/2) – leave the parens, just drop the String'

12.3(24), change 12.1(22) to include (<>).

12.3(24), change "Square" on lines 4 and 5 to "Square1" and "Square2". (The fifth line would fail as suggested in the AI..)

!discussion

Quote "MR_Pool."

4.3.2(13)
Tucker says we need to fix this:

```
(Expression with Left => new Literal'(1.2), Right => new Literal'(3.4))
```

This (and the following fixes) get moved to the !wording section.

Fix 10.1.2(29):

```
type Department is [private]{...};
```

Change 11.4.3(6/2):

Add "..." above Open to show there are additional declarations (specifically File_Exists).

Then in 11.4.3(5/2) add:

```
private
   ...
```

To show there is a private part.

Fix A.18.32(9/3): Remove the use of "Adjacency_Lists" from the body.

Change 11.2(12):

```
{Ada.Exceptions.}Exception_Message
```

Approve AI with changes: 7-0-1.