Minutes of the 58th ARG Meeting

13-15 October 2017

Lexington, Massachusetts, USA

Attendees: Raphaël Amiard, Steve Baird, Randy Brukardt, Jeff Cousins, Gary Dismukes, Bob Duff, Brad Moore, Tucker Taft.

Observers: Pat Rogers (Friday only), Vasiliy Fofanov (Saturday only).

Meeting Summary

The meeting convened on Friday, 13 October 2017 at 12:15 hours and adjourned at 12:30 hours on Sunday. 15 October 2017. The meeting was held in the conference room at AdaCore's Lexington Massachusetts offices. The meeting covered all of the normal AIs and many of the amendment AIs on the agenda.

AI Summary

The following AIs were approved:

AI12-0064-2/14 Nonblocking subprograms (6-0-2)
AI12-0211-1/03 Interface types and inherited nonoverridable aspects (6-0-2)

The following AIs were approved with editorial changes:

AI12-0127-1/09 Partial aggregate notation (8-0-0)
AI12-0187-1/05 Stable properties of abstract data types (5-0-3)
AI12-0232-1/02 Rules for pure generic bodies (7-0-1)
AI12-0233-1/03 Pre'Class for hidden operations of private types (7-0-1)

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

AI12-0079-1/05 Global-in and global-out annotations (8-0-0)
AI12-0111-1/06 Stable Containers to reduce tampering checks (8-0-0)
AI12-0119-1/04 Parallel operations [loop and block portion] (7-0-1)
AI12-0119-1/04 Parallel operations [reduction expression portion] (8-0-0)
AI12-0189-1/04 loop-body as anonymous procedure (6-1-1)
AI12-0234-1/01 Compare-and-swap for atomic objects (7-1-0)
AI12-0235-1/01 Root_Storage_Pool should be pure (6-0-2)
AI12-0236-1/01 Declare expressions (7-0-0)

The following AIs were discussed and assigned to an editor:

AI12-0208-1/00 Predefined bignum support
AI12-0214-1/01 Case pattern matching
AI12-0218-1/01 Endianness-neutral record representation clauses
AI12-0226-1/01 Generalize expressions that are objects
AI12-0240-1/01 Access value ownership and parameter aliases

The following AI was discussed and voted No Action:

AI12-0238-1/01 Delta package specifications (8-0-0)

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

AI12-0064-2/12 Nonblocking subprograms (7-0-1)
AI12-0127-1/07 Partial aggregate notation (6-0-1)
AI12-0187-1/04 Stable properties of abstract data types (6-0-2)

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

AI12-0064-2/13 Nonblocking subprograms
AI12-0127-1/08 Partial aggregate notation

Detailed Minutes

Apologies

We've received apologies from Erhard Ploedereder, Jean-Pierre Rosen, John Barnes, Florian Schanda, and Alan Burns for being unable to attend this meeting.

Previous Meeting Minutes

No one has any changes to the minutes. Approve minutes: unanimous.

Homework Deadline

Jeff wonders if we could move up the homework deadline somewhat, so he has time to digest the agenda and AIs. We agree to set the deadline two days earlier, and Tucker promises to get his homework in on time.

Date and Venue of the Next Meeting

The next in-person meeting will be in Lisbon, Portugal, June 22-24, following Ada-Europe. The fall meeting will be in Boston (near HILT, possibly at the AdaCore offices again), in mid-October (October 12-14 are the most likely dates).

Randy notes that we need to make progress over the winter if we're going to have much finished by mid-2019. He suggests a series of phone meetings. The group agrees. We will try to set up phone calls 11 am-2 pm EST in January, February, and March. Tucker and Jeff suggest January 29th (a Monday) for the first call. Randy will reconfirm this date in early January. We'll probably try to use Google Hangouts to host the meeting (rather than the AdaCore conferencing system as in the past).

Community Input Announcement

The Community Input Announcement should be sent as widely as possible. Is there anywhere that it should have been posted that it wasn't? Randy posted it on comp.lang.ada. And it was in the Ada User Journal.

Someone suggests that Tucker post it on LinkedIn. Randy will take an action item to send Tucker the text of the announcement for posting there.

ARG Charter

WG 9 asked us to review the ARG charter. Jean-Pierre suggested adding "revisions" to the list of things we might do (before "amendments"), and we probably also should add "technical specifications" to the list as well.

Randy wonders if the ASIS standard should be mentioned here. No, we'd rather that be covered by "other standards as assigned by WG 9". But only the third bullet says "and other standards as assigned by WG9". That should be more generally said (it's true of all the bullets).

Tucker wants a comma after "libraries".

Jeff will draft a new version with the changes and circulate it.

Thanks

Thanks Jeff for running the meeting.

Thanks to AdaCore for hosting the meeting.

Thanks to Randy for taking the minutes and editing the Standard.

Unfinished Action Items

Raphaël Amiard failed to do his report. He is requested to do the report early (December) so Florian can use the information in the AI update.

Steve Baird says that A12-0016-1 progress is not likely soon as AdaCore is uninterested in implementing this.

He also says that AI12-0020-1 is not likely to progress. Randy wonders if we should discuss this briefly. Steve was looking at a reflection mechanism. Tucker thinks real-time people don't want that.

Raphaël wonders how Steve got from Image to reflection. Steve replies that one needs a way to walk the components repeatedly and having a way to do that simplifies the rest. Tucker suggests that Image should work much like streaming; perhaps copy streaming.

Bob says we're not trying to read them in. Randy notes that's a significant question: do we want 'Value?

Raphaël notes that one is often trying to iterate over components, so a general mechanism to do that would be useful. That's what Steve was trying to define. Tucker says that should be a separate AI. He would like to see an AI for Image, without necessarily supporting Value, and without a general facility.

Steve will do just 'Image as described here (AI12-0020-1). We'll have a separate AI to extend that to support Value (Tucker will take that AI). Raphaël will create an AI on general component iteration (reflection).

Steve has a question about AI12-0075-1. When does an expression function become static? Tucker proposes that it is decided when an expression function is declared; not on use or specific to a particular view.

Steve comments that Randy had noted that a real static expression can have a different value than the same real runtime expression. That could be dangerous, especially if it shifts unexpectedly during maintenance. Tucker agrees this is an issue; having an aspect Static to determine when an expression function can be static is safer. We agree to define an aspect, without completions for now. Steve will redo the AI this way for discussion next time.

Steve says AI12-0210-1 makes his head hurt. When he tried to think about this, he thinks it is a hard problem. Tucker says this one doesn't deserve more work at this point. The SPARK group needs to consider this further and recommend any changes to the language. Change the priority to Very_Low.

Randy Brukardt says that AI12-0112-1 is waiting for Nonblocking, Stable_Properties, and Global to finish. He probably will start updating it immediately after this meeting.

Alan and Florian are not here and thus can't give their excuses for not doing their homework.

Tucker says AI12-0191-1 is about "part", and it is very painful because it requires looking through the entire Standard. Jeff wonders if we should somehow fix this text. There is value to a taxonomy of the uses of part. Tucker will try to work on this part (pun intended).

Steve notes that a "parent part" is not a "part". Another question to answer in the taxonomy.

Current Action Items

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

Raphaël Amiard:

- AI12-0212-1 (report on C++ facilities)
- AI12-0214-1 (split into two AIs; help from Tucker Taft)
- AI12-0218-1
- AI12-0240-1 (assist Tucker Taft)
- AI to make type marks optional in object renames (see discussion of AI12-0236-1)
- Propose AI on general component iteration (reflection?); see the discussion of AI12-0020-1 in Unfinished Action Items.

Steve Baird:

- AI12-0016-1
- AI12-0020-1 (just Image, see Unfinished Action Items)
- AI12-0075-1 (see discussion in Unfinished Action Items)
- AI12-0208-1 (with help from Bob Duff)
- AI12-0210-1 (get guidance from the SPARK group, then update the AI)
- AI12-0212-1 (assist Florian Schanda)

Randy Brukardt:

- AI12-0017-1 (assist Florian Schanda)
- AI12-0112-1
- Send Tucker Taft the Community Input announcement to post on LinkedIn.

Editorial changes only:

- AI12-0127-1
- AI12-0187-1
- AI12-0232-1
- AI12-0233-1

Alan Burns:

- AI12-0230-1 (with assistance from Tucker Taft)

Jeff Cousins:

- Update the ARG Charter and send the result to WG 9

Bob Duff:

- AI12-0208-1 (assist Steve Baird)
- AI12-0236-1

Brad Moore:

- AI12-0119-1 (split reduction expressions into a separate AI)
- AI12-0234-1

Florian Schanda:

- AI12-0017-1 (with help from Randy Brukardt)
- AI12-0188-1 (see discussion of AI12-0189-1 in Vienna)
- AI12-0197-3 (lower priority than others)
- AI12-0212-1 (with help from Steve Baird)
- New AI for Iterator filtering (see discussion of AI12-0212-1)

Tucker Taft:

- AI12-0079-1
- AI12-0111-1
- AI12-0189-1
- AI12-0191-1
- AI12-0214-1 (assist Raphaël Amiard)
- AI12-0226-1
- AI12-0230-1 (assist Alan Burns)

- AI12-0235-1
- AI12-0240-1 (with help from Raphaël Amiard)
- AI for 'Value for composite types, similar to AI12-0020-1 (see discussion of AI12-0020-1 in Unfinished Action Items)
- AI defining a user-defined literals mechanism (see discussion of AI12-0208-1)
- Potential AI to declare System.Storage_Pools.Subpools Pure (see discussion of AI12-0235-1)
- Potential AI to define () as an empty array (see the discussion of AI12-0119-1)

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 11 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

AI12-0064-2/12 Nonblocking subprograms

Randy notes that he discussed the open issue with John, and his concern is with empty packages. Aspects are fine in non-empty packages. So we will use aspects consistently.

Steve notes that the previous 9.5.1(16) talks about calls, while the previous 9.5.1(18) talks about subprograms. Change the 9.5.1(18) replacement:

> If a language-defined subprogram allows blocking, then {a call on the subprogram}[it] is a potentially blocking operation.

8.3(12.3/3) talks about “they fully conform”, but full conformance applies to profiles, as opposed to subprograms. That wording should be fixed (but not in this AI – it doesn't have anything to do with this topic – do it in the presentation AI).

We worry about whether if this rule needs to apply to Nonblocking. It seems it should.

So we need a rule that if multiple homographs are inherited, then the result is nonblocking if any is nonblocking. Tucker and Randy will take this off-line.

Line 492, “,” should be a semicolon [an AARM ramification, 3rd paragraph of Legality Rules] Tucker says that is fine as is, no change.

Line 529, should be a comma after “works”. Better, change to “because”: “(This works [as]{because} a nonblocking subprogram allows a strict subset of the operations allowed in allows blocking...”.

Approve intent of AI: 7-0-1.

AI12-0064-2/13 Nonblocking subprograms

Later on Friday, Tucker e-mailed new rules. He notes that they all should just apply to dispatching operations. We also need something for other expressions. He will try again.

AI12-0064-2/14 Nonblocking subprograms

Tucker sent new wording in e-mail on Saturday night.

Steve worries about the dynamic semantics effect on Detect_Blocking. Randy notes that only language-defined subprograms that allow blocking (not many) are potentially blocking. User-defined subprogram are never potentially blocking, so Detect_Blocking has no effect on such calls.

Approve AI: 6-0-2.

AI12-0079-1/05 Global-in and global-out annotations

Tucker explains the changes after a confusing start.

He suggested dropping synthesized, as it doesn't provide anything portable. He suggests instead allowing the compiler to synthesize the aspect if none is given; if the user doesn't care about the non-portability, the result would be the same. (And if the compiler can't do that, the program probably would be rejected rather than silently being unsafe.) One hopes for the existence of tools that can explicitly synthesize these aspects, but that has to be outside of the Standard.

Randy notes that we are missing Legality Rules for generic matching, assume-the-best, assume-the-worst.

Randy asks how we can represent packages used in the implementation of the body. In Randy’s case, Ada.Text_IO is implemented using System.Basic_IO. We don’t want to name such packages in the specification. (That's especially important for language-defined packages, where the helper packages are going to be implementation-defined.)

SPARK requires naming the all of the specifications, but a limited with of the package can be used to name that.

Tucker says that we could allow implementations to add limited withs and Global aspects for the implementation packages. We also could use ‘Global to help this problem.

We note that the rules in the introduction to Annex A requires operations to be task safe. Tucker thinks that perhaps “synchronized in out all” might be enough. Randy thinks that should be the default of language defined packages (other than Pure).

Approve intent of AI: 8-0-0.

AI12-0111-1/06 Stable Containers to reduce tampering checks

Randy notes that a paragraph like A.18.2(97.1/4) needs to be moved into A.18, defining “prohibiting tampering with elements”. (The AI already defines “tampering with cursors” in A.18).

He also notes that there are a number of operations in A.18.2 that prohibit tampering with elements, these either need to be modified to say “tampering with cursors” (and the changes undone in the indefinite containers), or tampering with elements needs to be defined as the same as tampering with cursors for the definite containers.

Tucker thinks that he needs to define the Assign operation, as this not an existing operation. Some wording will be needed to define that.

Somewhere we should require implementations to use the Stable containers to implement for loops “of Element”. Perhaps even going so far as change the equivalence of those loops to say that. [Editor's note: That probably would require changing the meaning of the iteration aspects, or defining new aspects that use the Stable containers for iteration; it might be possible to do the latter compatibly; the former has no chance of being compatible.]

Approve intent of AI: 8-0-0.

AI12-0119-1/04 Parallel operations

Concurrent blocks

Tucker is confused about "concurrent" vs. "parallel". If interleaving is allowed, then there isn't much difference between "concurrent" or "parallel". Tucker notes that if interleaving is allowed, then the concurrent block doesn't provide coroutines.

These need to avoid preemption to provide coroutines. Tucker would say that they all run in the same Ada tasklet. They only switch when they do a Yield operation (or if they block, if that is allowed at all).

Further discussion on the non-parallel version will be deferred to AI12-0197-4.

Tucker says that these should be handled_sequence_of_statements. Brad was concerned about confusion as to what the handler applies to. Tucker says experience says that we always end up adding those handlers, so we should just do it right away.

There isn't any value to a declare part here for the parallel version. (Perhaps for a coroutine version, we're not talking about that now.)

There doesn't seem to be any wording defining the rules for transfer of control. (Same for parallel loops.) That clearly needs to be added.

The parallel loop text talks about "nonoverlapping ranges", but a range requires contiguous indexes. We don't need to prevent each tasklet processing every 8th item (for instance). Also, one doesn't execute loop parameters.

We talk about the purpose of the feature. Primary purpose is to increase performance without decreasing safety.

Steve notes that there are three cases: forward, reverse, don't care. Tucker says that parallel means to him unordered. We take a straw poll where everyone (excepting Randy, who abstained) agrees that "parallel" means unordered.

So no semantics on exit, and no **reverse** in parallel loops.

All exits (including an exception) work the same way. The wait at the end will quit as soon as possible, picking the first result, and then all of the other tasklets are told to quit. Then that first result is returned.

There is not any preference to a kind of result. There's no reason to give a preference, because an exception (for instance), does not happen at all if it is not picked. After all, some tasklet may only be a millisecond from raising an exception when it quits. There is no way to tell the difference between that and an exception having occurred a millisecond before the tasklet quits but not being the one that provides the result; thus there is no point in giving priority to a particular kind of result.

Brad wonders if exceptions should force a quicker exit. Some programs might run a long time per iteration. Tucker says that the performance of an exception raised case is not interesting. Exceptions mean that there is some kind of error, usually a bug; we don't care about the performance of a program that fails. Bob says he thinks that an exit with a result is more interesting.

Back to blocks: the **do** syntax is fine. We need termination wording. We should call these "parallel blocks", and remove (for this AI) the optional **parallel** (it is mandatory). (AI12-0197-4 will restore the optionality and possibly introduce a different name if **parallel** isn't given.)

Parallel loops

Remove **reverse** as previously noted. Should talk about one tasklet per iteration in the wording; chunking is surely allowed even with that wording.

We need (for both parallel loops and blocks) a definition of the Legality Rules for the race condition checks. We also need runtime checks that array uses of elements are different in each iteration. The wording is easy, but the check is expensive. Probably a need a suppress name for that check as well.

Steve asks if the intent is to support container iterators in the parallel loops. Yes, that is intended; that also needs to be added.

Reduction expressions

We'll add in "Identity" and "reducer" aspects as needed. Tucker notes that T(0) is used in 4.5.3 as opposed to just 0. No good reason for that.

We have freedom for association but not the order. Steve notes that total number of calls on the combiner and reducer are fixed. Tucker agrees that the combiner is called once per iteration; but the compiler decides about the reducer.

There shouldn't be syntax for reducer_function. The combiner_function_call should be an expression (infix and operations aren't function calls). So use *combiner*_expression for this. And drop the syntax definition for combiner_function_call.

We probably should split the reduction expressions into a separate AI, just to make this easier to review. It is huge now.

We don't need **parallel** in the expression contexts. Rather, we can just give implementation advice to execute these in parallel if there are no data dependencies. That includes reduction expression functions.

Brad thinks that there are cases where one does not want to execute a reduction expression in parallel. Perhaps too much detail? The vast majority of users want the best possible performance. Raphaël and Tucker suggest that parallelization should be controlled with other mechanism. Tuning is not portable.

Tucker would prefer to require the initial_reduction_value be required, because <> is box, so you'd have to put a space between them which would be nuts. Someone objects that you then need to have a way to specify an empty set.

Tucker would like to allow empty parentheses to represent an empty aggregate. `(1 .. 0 => <>)` is an empty array;.Tucker would like to allow this to be written as ().

He would define a function "()" return T, which gives an empty object. It would be predefined for arrays. Randy wonders if it is defined for every array or just one-dimension arrays.

This should be a separate AI, it is only loosely related to the reduction expression.

The identity is really only needed if a reduction expression is used in parallel; so we don't need to require reducer/identity. Raphaël worries that if a user wants parallelism but didn't use an appropriate function, we have no way to tell the user whether parallelism actually happens. (The **parallel** keyword would have provided such a mechanism.)

Bob says that this seems more like a compiler issue. We imagine that a compiler could have a mode where it warns about reduction expressions that can't be parallelize. [Why should this be implementation-defined? - Editor.]

Approve intent of reduction expressions: 8-0-0.

Approve intent of parallel blocks and loops: 7-0-1.

AI12-0127-1/07 Partial aggregate notation

Steve asks about the case where two components are declared in different variants.

```
type R (Flag : Boolean) record
   case Flag is
      when True  => F1 : ...
      when False => F2 : ...
   end case;
end record;
```

```
    (... with delta F1 => ..., F2 => ...);
```

Tucker says this rule is simple: “A single record delta aggregate shall not refer to two components declared within different variants of the same variant part.” Bob says it’s not quite right. He doesn’t expand on that.

Straw poll: 7-reject at compile time; 0-runtime; 1-abstain.

Tucker wonders why we don’t allow this on private extensions. Randy notes that we don’t allow any aggregates on private extensions. Tucker says this is new, so we can define it however we want. So long as there are some visible components, we should allow a delta aggregate. And that falls out from visibility rules (if there aren't any visible components, then one can't name any such components).

Tucker then wonders why we don’t allow class-wide types. After much discussion, Randy notes that the wording says that we do in fact allow them for delta aggregates (that is the change to the wording). Tucker leaves at this point (Friday evening).

Steve will try to fix the two rules and bring it back tomorrow.

Approve intent of AI: 6-0-1.

AI12-0127-1/08 Partial aggregate notation

Tucker notes that we need to say “descendant of a record type or record extension”. This allows the type to be any type with some visible record components.

The general resolution rule 4.3(3/3) doesn’t allow private types. How to fix that??

After much discussion, we decide to distribute the general resolution rule to each aggregate type (to get rid of this problem for good) and then delta aggregates can be different. Otherwise, we would be changing resolution of existing aggregates and that would be a compatibility problem.

AI12-0127-1/09 Partial aggregate notation

Steve sent revised resolution rules during the break. Essentially, the existing aggregate rules are changed to Legality Rules. Steve claims that no change to existing rules is intended.

Bob claims that the existing rules don’t actually implement the intent; Steve disagrees but it doesn’t really matter. Randy notes that compilers probably follow the ACATS rather than the exact wording.

There is a cut-and-paste error in the delta aggregate case; it needs to talk about delta aggregates rather than extension aggregates.

The change is to 4.3.1(8/2), not (3).

Tucker says that he doesn’t like this change. Some of us argue that no one has ever understood the original rule.

Bob leans toward doing less. So just replace 4.3(3):

> The expected type for a delta_aggregate shall be a single array type, or a single descendant of a record type or of a record extension. The expected type for any other aggregate shall be a single array type, record type, or record extension.

For 4.3.4, use the following for the resolution rules:

> The expected type for a record_delta_aggregate shall be a single record type or record extension.
>
> The expected type for an array_delta_aggregate shall be a single array type.

Approve AI with changes: 8-0-0.

AI12-0187-1/04 Stable properties of abstract data types

We discuss the use of this feature. The example shows how all of these features are needed, even in a simple case like Text_IO.

Steve asks about the definition of stable properties. The second sentence is weird. So combine into a single sentence:

> A *property function* of a type *T* is a function with a single parameter and a nonlimited return type, that is a primitive operation of *T* or a function whose parameter is class-wide and covers *T*.

Gary notes that we ought to say that the mode is **in**. We need to say that the parameter is *T* (as the function could be primitive by returning *T*).

Tucker would rather get rid of the term “property function” altogether, and use a set of Legality Rules. They would put restrictions on “stable property function”.

Those rules would go in the Legality Rules section so that they are not duplicated.

Why is this valuable? This works well with the containers. It decreases clutter in the postconditions of an ADT. Several people note that they’ve tried to do this explicitly and it gets very long and obscures the interesting part of the postcondition.

Approve intent: 6-0-2.

Randy will try an update (before the meeting ends).

AI12-0187-1/05 Stable properties of abstract data types

Randy sent a redraft during the break based on our earlier discussion.

Tucker suggests adding the rule for prefix notation (borrowed from use all clause) to allow a class-wide function to be a stable property function. That makes the Legality Rule:

> A stable property function of a type *T* (including a class-wide stable property function) shall have a nonlimited return type and shall be:
> - a primitive function with a single parameter of mode **in** of type *T*; or
> - a function that is declared immediately within the declarative region in which an ancestor type of *T* is declared and has a single parameter of mode **in** of a class-wide type that covers *T*.

Approve AI with changes: 5-0-3.

AI12-0189-1/04 loop-body as anonymous procedure

The !wording is new. (Ignore the !proposal, there is too much there.)

This is only added to loop statements, not other kinds of things that are loop-like (aggregates, quantified expressions).

Tucker goes through his new wording. He notes that there might need to be a rule preventing multiple uses of the name identifier in the list. Bob notes that if these are declared, the homograph rules would prevent multiple uses.

Steve notes that there is not really a loop here; it’s inside of the procedure. Tucker notes that most often such functions are called multiple times, so there is something like a loop. Raphaël notes that this could be used like a lambda, where there is no loop in site.

Bob notes that he has a call-back usage (to open, process, and close a file) that he would be sorely tempted to use via this syntax.

Steve wonders if Exit_Exception could be nonoverridable. That seems cleaner than having a bunch of rules.

Tucker notes that we need Implementation Advice that exit makes use of the exception mechanism.

Several people wonder why we have to specify the exception name here. Randy notes that this is some name pollution for the exit exception; one could also see just promising that exceptions from the <> routine are propagated.

Bob wonders how this works if you have two nested loops for the same procedure. You are then using the same exception to exit both.

Tucker draws the code on the whiteboard:

```
L3: for (a) of Iter(<>) loop
   L2: for (b) of Iter(<>) loop
      ...
      exit L3;
   end loop L2;
end loop L3;
```

which translates to:

```
declare
   Target : Natural := 0;
   procedure Loop_Body (B : ...) is
   begin
      Target := 2;
      raise E; -- Iter's Exit_Exception.
   end Loop_Body;
begin
   Iter (..., Loop_Body'Access, ...);
exception
   when E =>
      if Target = 2 then
         exit L3;
      end if;
end;
```

The alternative is to add an aspect to say that all cleanup is handled by finalize. Then the implementation can do whatever makes sense (perhaps with an anonymous, unhandleable exception). One presumes that language-defined routines would get the aspect.

Randy says that while he don't like this much, it at least addresses his concern about existing routines that handle cleanup with an **others** handler.

Jeff wonders how you put an aspect on an anonymous routines. It gets put on the Iterate, not on the call-back routine.

Raphaël votes against, he thinks it is a leaky abstraction; nothing would prevent anyone to use it as a non-loop. And he doesn't think there is enough value; he would rather create a collection and iterate through that. Tucker notes that this makes iterators with state much easier to create.

Approve intent of AI: 6-1-1.

AI12-0208-1/00 Predefined bignum support

Bob would prefer to add it directly to the language.

Tucker would prefer to make packages more like built-in. Specifically, he would like to see user-defined literals. Tucker will take an action item to propose a user-defined literals mechanism. Randy notes that should work for string literals as well as numeric literals (and possibly even **null**).

Steve will take this AI with help from Bob.

Keep alive: 8-0-0.

AI12-0211-1/03 Interface types and inherited nonoverridable aspects

Tucker cleaned up the wording in 13.1.1(3/5) and added wording 13.1.1(4/4). 13.1.1(3/5) defines confirming for names, 13.1(18.2/3) defines confirming for representation aspects (values).

Randy notes that "confirming" is only defined for aspects that are names and for representation aspects. Luckily, all uses are one or the other. So we're OK so far.

Gary doesn't like "confirming of".

Straw poll: "Confirming of" 5, "confirm" 2, abstain 1.

Approve AI: 6-0-2.

[Editor's Note: This AI had no !summary, so it did have to have a change applied before checking in the approved version.]

AI12-0214-1/01 Case pattern matching

Raphaël explains that this is intended to make case statements more useful.

Tucker notes that membership tests were recently expanded, and he thinks that case statement is related to that. They should be a close together.

Raphaël notes that case requires completeness, which means that the full generality of memberships is unlikely.

Tucker would like the case statement generality be a separate AI from the binding stuff. By mixing them, there are different issues being covered. Moreover, a proposal that is too complex is at much more risk of being rejected completely for not being fully baked. Randy agrees with this sentiment.

Tucker notes that the binding isn't that interesting for variants, but it might make sense for class-wide operations (where the child components aren't visible until the item is converted).

Raphaël would like Tucker to write a short description of what he is describing for class-wide.

Raphaël will update this AI as described above, with help from Tucker (especially for the class-wide cases).

Keep Alive: 7-0-1.

AI12-0218-1/01 Endianness-neutral record representation clauses

This is implemented in GNAT. Bob says that he doesn't understand it.

Steve wonders what happens if this is applied to a record with an aliased scalar component. That has to be banned.

Raphaël wonders if it is important that this isn't in the Standard. Tucker says that his sense is that the customers don't care that it is not standard.

Tucker agrees with Randy that this could be very expensive in compilers, as it possibly effects the back end. (Back ends are usually different for each target, rather than being the same for all targets like the majority of an Ada compiler.) He doesn't think it is worth doing; it is not going to attract any new Ada customers or help that many users.

Raphaël will take the AI, but he will try to get Thomas Quinot to write it (Thomas was the original designer of this feature).

Keep alive: 6-1-1. Randy thinks it is too much effort for implementers, and Ada 2020 does not need to be hard to implement for other implementers. Any implementer can add this if they want without it being part of the Standard.

AI12-0226-1/01 Generalize expressions that are objects

Bob would like to get rid of this entire difference. Tucker worries about "associated object", as that might be issues. We look at that and it only applies to by-reference types, so elementary values are not a problem.

Tucker also would like to reverse the list of constants to a list of variables, as there are far fewer kinds of variables and we're much less likely to add a new kind of variable.

Randy notes that this AI originated with him, so he would be the author by default. However, he already proposed the changes that he understands; it doesn't seem possible to him to merge the meaning of "value" and "object" and making everything an "object" seems like a lot of additional mechanism. Will someone else take the AI? Tucker reluctantly offers to take the AI.

Keep alive: 8-0-0.

AI12-0232-1/02 Rules for pure generic bodies

Bob wonders if anyone would care. Tucker says that Hristian (of AdaCore) was going to implement this wrong, because generics were not properly covered.

There are curly brackets missing around all of the new text.

Randy notes that Pure is only defined for compilation units, which means that nested units can't be pure, which blows up the whole thing.

Tucker suggests that we change "compilation unit" to "program unit" in (15.1/3) [two places], then in the new text change "elaborable construct" to "program unit" in the last new bullet.

Steve worries that the new bullet includes the item itself. We discuss for a while and conclude it would be weird to say that the elaboration of an item performs the elaboration of itself, so there is no real issue here. Steve can propose a To Be Honest if he disagrees.

Approve AI with changes: 7-0-1.

AI12-0233-1/03 Pre'Class for hidden operations of private types

Tucker says that he tried to make this work based on views and it didn't work, so he reverted to the obvious model.

The last two paragraphs of the discussion will be split into a new AI and assigned to Tucker to complete. Then he rereads the definition of renames-as-body and decides that it is "obvious". So blow away those paragraphs and forget this topic.

Gary has editorials: !response, 1st paragraph "as apposed". !discussion, 3rd to last "for now[, the]{;} a compiler or"

Approve AI with changes: 7-0-1.

AI12-0234-1/01 Compare-and-swap for atomic objects

Bob says that GNAT already has such a package, so we don't need one. Randy notes that atomic objects are portable, so we need portable access to modify atomic objects. Bob says there aren't any other Ada compilers (but then why are we here?)

Use GNAT or GCC or LLVM as a model.

Brad will take this on.

Approve intent of AI: 7-1-0. Bob doesn't think this particular case is worth standardization. GNAT's facilities are good enough and this sort of code isn't sufficiently portable anyway to go further. We also need to not put too many features into Standard so that other compilers do exist.

AI12-0235-1/01 Root_Storage_Pool should be pure

This AI was discussed last on Friday, after Tucker had left.

Bob notes that to use this in a Shared_Passive package, you couldn't have much in the way of access types, and a Storage_Size = 0 pool doesn't seem useful. And no pool object can be used in such a package, so what is the point?

This could cause problems for existing implementations, if they have anything prohibited by Pure in the implementation of the package.

Straw poll: No Action: 6-0-1.

However, we want to let Tucker explain why he was in favor before a final resolution.

On Saturday morning, we took up this AI again.

Tucker explains that we try to make everything Pure. He doesn't believe that there would be any underlying mechanism for Root_Storage_Pool.

Randy says that he worries about subpools, as there is various semantics (for finalization, for instance) that may require some underlying mechanism.

Tucker suggests splitting root storage pool, and changing that to Pure, and then making a separate AI to analyze subpools. Bob says analyzing subpools is a waste of time.

Approve intent of AI: 6-0-2.

AI12-0236-1/01 Declare expressions

The syntax probably doesn't need the optional **declare**.

Steve thinks we should allow renames. The semantics is rather like renames. It would be better if **constant** worked like a constant declaration (that is, makes a copy). So allow object renaming and constant declarations.

Someone suggests use clauses. But that would be ugly, as part of a complete context would have different visibility than the rest. Indeed, we should try to only declare things that aren't overloadable in the middle of expressions lest whether a new name is used be conditional on the types of subexpressions.

We talk about the syntax. Tucker doesn't like **begin**, we should use =>. **Do** is suggested and rejected.

Randy comments that we have kept expressions (if expression, case expression, raise expression) as close as possible to the matching statement. Shouldn't that apply here??

Raphaël suggests that we make **declare** non-optional. He says that makes it obvious for the reader.

Steve suggests that the semicolon should be required in front of **begin**.

The subtype can be omitted if the type is inferrable from the initialization expression.

Raphaël worries that this would give an incentive to use an expression rather than a declaration.

Bob suggests that the expression should be treated as an "any type", like the operand of a type conversion (unless, of course, a type is explicitly given).

Randy worries about the maintenance hazard of the object silently changing types. Tucker thinks that already happens in type conversions and it hasn't been an important problem.

Is the colon needed in a renames? That is:

X : [subtype_mark] **renames** expression;

X : **renames** <expression>;

vs.

X **renames** <expression>;

We should make the type optional in regular object renaming as well. That should be a separate AI.

Straw vote: **declare** - 6; nothing: 3.

Which separator keywork is liked the best: **begin** 1; "; **begin**" 4; => 2; "; **in**" => 2

Jeff leaves (this was the last AI discussed).

Bob will take the AI. Raphaël will create an AI to make the type name in **renames** optional.

Approve intent: 7-0-0.

AI12-0238-1/01 Delta package specifications

Vasiliy notes that they have customers that cannot modify the specifications to add Pre/Post and other contracts. The delta package would allow adding stuff to the specification.

Tucker notes that there is concern what happens if the delta package is not available. He would propose that a delta package can't affect dynamic semantics other than Assertion_Policy.

Steve argues that this should be handled by a preprocessor; it doesn't belong in the language.

Randy notes that preconditions in particular are important to the caller. They are necessary to making correct calls. It seems weird to allow hiding them in a separate place.

Jeff notes that his organization would never change a frozen spec under the covers, as a new contracts would also have to be agreed by the entire group.

Steve notes that this reduces readability, as you would have to know that there is a delta package. Raphaël notes that some contracts are huge and they hurt readability.

Vasiliy says that FACE wanted to ban contracts, and delta packages would have allowed working around that. (Didn't happen though.) Tucker suggests that a restricted subset of Ada might restrict contracts.

Raphaël notes that one could annotate Ada 83 code using a delta package so that proof tools could be used on the code, but still compile with their Ada 83 cross-compiler.

Straw poll: Continue with it in this process: 0; Send it back to AdaCore for a technical specification: 9.

No action: 8-0-0.

AI12-0240-1/01 Access value ownership and parameter aliases

Tucker says the real goal is to allow you to safely use pointers and allocators: no dangling references, no storage leaks, no explicit deallocation. The justification in the AI misses this completely; it should be rewritten to make the real goal clear.

Randy suggests that this only work for a pool-specific type. If that restriction was applied, then most of the type conversion rules follow automatically (since pool-specific types can only be converted to a related type that shares the same pool). And we surely can't manage 'Access-created accesses this way. Nor can we give an aspect to an anonymous access type, so those can't be used anyway. Tucker agrees that this is a good idea.

Randy asks how this works in generics. For instance, how does a generic formal access type work. Tucker says that the aspect would have to match in that case. (Probably a generic derived type would be the same). Tucker thinks that this would normally be used program-wide; perhaps the aliasing restriction would require that (else the checks can't be made).

Steve said that Parasail has optional components with implicit dereference. Wouldn't a mechanism like that be a lot easier to define instead of this one? No answer is recorded.

Randy raises the issue of generic formal private types – they might have "owned" access components; Tucker says those will need to be addressed, along with other generic issues.

Tucker says that this may be a lot of work, but it also may be a huge win for Ada users.

Steve notes that we have to worry about interactions with Global.

Raphaël volunteers to assist Tucker.

Keep alive: 7-0-1.