Minutes of ARG Meeting 62

Lexington, Massachusetts, USA

5-7 October 2019

Attendees: Steve Baird, Randy Brukardt, Jeff Cousins (electronically except last 30 minutes Saturday and Sunday), Gary Dismukes, Claire Dross (electronically, Monday early only), Brad Moore (electronically, except first 30 minutes Monday), Bob Duff, Ed Schonberg (electronically, except Saturday morning, last 30 minutes Sunday), Justin Squirek, Tucker Taft, Richard Wai (electronically).

Observers: Edward Fish, Erhard Ploedereder (electronically, Saturday and Sunday), Pat Rogers (Saturday before lunch only), Joyce Tokar (electronically, Sunday morning).

Guests: Michael Klemm (electronically, Sunday morning)

Meeting Summary

The meeting convened on Saturday, 5 October 2019 at 11:00 hours EDT and adjourned on Monday, 7 October 2019 at 12:45 hours EDT. The meeting was held in Lexington, Massachusetts, in the conference room in the AdaCore offices. The meeting covered all of the AIs on the agenda that were not previously put on hold (those were listed on the agenda for information only).

AI Summary

The following AIs were approved with editorial changes:

AI12-0280-2/05 Making 'Old more flexible (9-0-1)
AI12-0302-1/03 Default Global aspect for language-defined units (9-0-1)
AI12-0312-1/07 Examples for Ada 202x (10-0-0)
AI12-0340-1/01 Put_Image should use a Text_Buffer (10-0-0)

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

AI12-0334-2/04 Predicates and Global/Nonblocking (8-0-1)
AI12-0343-1/01 Return statement checks (9-0-0)
AI12-0344-1/01 Procedural iterator aspects (6-0-4)
AI12-0345-1/01 Dynamic accessibility of explicitly aliased parameters (9-0-0)

The following AIs were discussed and assigned to an editor:

AI12-0079-2/00 Global in/Global out annotations
AI12-0205-1/02 Defaults for generic formal types
AI12-0215-2/01 Implicit instantiations
AI12-0239-1/03 Ghost Code
AI12-0243-1/02 Subtypes as primitive arguments
AI12-0346-1/00 Ada and OpenMP

The following AIs were discussed and voted No Action:

AI12-0139-1/04 Thread-safe Ada libraries (7-1-1)
AI12-0197-1/01 Generator Functions (9-0-0)
AI12-0197-2/02 Passive Tasks (9-0-0)
AI12-0214-2/03 Boolean conditional case expressions and statements (9-0-1)
AI12-0215-1/01 Implicit Instantiations (8-0-1)
AI12-0229-1/02 Type renaming (6-0-3)
AI12-0240-1/04 Access value ownership and parameter aliasing (9-0-0)
AI12-0240-2/02 Access ownership for Abstract Data Types (9-0-0)
AI12-0240-3/01 Access value ownership and parameter aliasing (9-0-0)

AI12-0240-4/01 Pointer ownership for Abstract Data Types (9-0-0)
AI12-0268-1/01 Automatic instantiation for generic formal parameters (9-0-0)
AI12-0334-1/04 Predicates and Global/Nonblocking (9-0-0)
AI12-0341-1/01 Syntax of conditional case expressions (9-0-1)

The following AIs were discussed and placed on hold:

AI12-0197-4/01 Coroutines and channels (9-0-0)
AI12-0214-1/02 Case statements and expressions (8-0-1)
AI12-0240-5/02 Heap object ownership for Abstract Data Types (9-0-0)

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

AI12-0312-1/06 Examples for Ada 2020 (10-0-0)

Detailed Minutes

Welcome

Steve welcomes everyone to the meeting. We then pause to determine precisely who's attending the meeting remotely (see the attendance list at the start of these minutes).

Apologies

Jean-Pierre Rosen and Tullio Vardanega sent apologies. Erhard Ploedereder sent apologies and a resignation from the ARG as he has retired and no longer has funds to travel to meetings (he still attended the entire meeting electronically). Brad Moore, Richard Wai, and Jeff Cousins sent apologies for being unable to attend the meeting in person; all three attended the majority of the meeting electronically.

Previous Meeting Minutes

No one had any changes to the minutes of the Warsaw meeting #61. Approve minutes: 9-0-0.

Date and Venue of the Next Meeting

Next face-to-face meeting will be co-located with Ada-Europe and WG 9 in Santander, Spain, June 12-14, 2020.

June is a long ways off. We probably should have an electronic meeting, perhaps in the latter part of January. Tucker will be traveling Jan 18-Feb 7, but he can join a meeting during that time. We decide to have the meeting before his travel, and choose Wednesday, January 15th, 11 AM-2 PM EST.

Mailing List Usage

The AdaCore LD Circle has requested that we use Ada-Comment for high-level technical discussion. Ada-Comment is available to any interested member of the Ada community, so it can keep a broader slice of the community informed about our deliberations.

Randy notes that historically, we wanted technical discussion on Ada-Comment. The ARG list originally was not recorded in !appendix – it was for administrative uses only. He started recording technical discussions because moving them wasn't successful (Robert Dewar in particular would not use Ada-Comment for discussion).

Tucker notes that he doesn't remember this history and in any case is more concerned about our direction going forward. We would prefer to keep wording discussions and Steve-type questions (that is, corner-case questions) on ARG, so as to avoid deluging the public with trivia.

We agree that at some point in the future that we will want to migrate Ada-Comment to some more modern infrastructure, but it best to avoid doing that while we are in the home stretch of a revision. Migrating is likely to have a variety of glitches, potentially including losing some users and messages, and it best for now to stay with a system for which we know the issues.

Tucker suggests that AIs that haven't had intent approved should be discussed on Ada-Comment. Steve suggests that "high-level" language issues should be on Ada-Comment while discussion of Standard wording should remain on the ARG list.

Approve policy to use Ada-Comment for AIs without approved intent and "high-level" language issues. 8-0-1. Jeff Cousins abstains.

OpenMP session

Tucker has arranged a session on OpenMP with the CEO of the OpenMP Consortium, Michael Klemm, for Sunday morning. Tucker sent a link to some slides to the ARG list. Brad wonders if AdaCore has a plan for using OpenMP in their implementation. Tucker replies not yet. Brad suggests inviting Miguel; he is asked to send an invitation to the entire WG 9 list (which he did).

Thanks

We thank AdaCore for the use of their fine facilities, and Tucker Taft for local arrangements (which mainly means choosing good restaurants and making reservations for dinner).

We thank our Rapporteur, Steve Baird, for his hard work keeping us on track.

Thank our editor, Randy for taking good notes and maintaining the website.

We thank Edward Fish for attending the meeting (and taking vacation time to do so); we will recommend him to become a regular ARG member at the next WG 9 meeting in June.

Thank everyone for coming, and to the remote attendees for staying up late or getting up early.

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
- AI12-0243-1 (help Bob Duff on request)
- AI12-0345-1
- AI12-0346-1 (help Tucker Taft with the OpenMP annotations)

Randy Brukardt:

- AI12-0205-1
- AI12-0343-1
- AI12-0346-1 (help Tucker Taft with the user experience portion)

Editorial changes only:

- AI12-0280-2
- AI12-0302-1
- AI12-0312-1
- AI12-0340-1

Bob Duff:

- AI12-0243-1 (with help from Steve Baird)

Ed Fish:

- AI12-0215-2 (with help from Raphael Amiard)

Brad Moore:

- AI12-0346-1 (help Tucker Taft with the GPU portion)

Justin Squirek:

- AI12-0239-1
- AI12-0346-1 (help Tucker Taft with the GPU portion)

Tucker Taft:

- AI12-0079-2
- AI12-0334-2
- AI12-0344-1
- AI12-0346-1 (with input/help from other ARG members as noted)
- Split stream attributes and dispatching Global attributes into a separate AI from AI12-0334-2.

Richard Wai:

- AI12-0346-1 (help Tucker Taft with the user experience portion)

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 22 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

AI12-0079-2/00 Global in/Global out annotations

[Editor's note: This AI was created based on this discussion.]

We start by discussing Tucker’s overview of these features (Global, Nonblocking, compound objects, internal acccess objects). Is it what people had in mind?

Tucker goes over the entire overview.

Edward Fish asks about passing in a function. Tucker notes that Global for an access-to-function parameter should be handled similarly to access-to-object parameters as described in the overview. We need to assume that the function is called and thus Global should be handled by the caller. Tucker should add that to one of the open AIs for Global (AI12-0334-2 or the split from it).

Erhard wonders how you teach this. Tucker notes that you have to teach the small part that students need. This is similar to other complex Ada concepts (like generic units). He notes that the idea of a compound object is pretty fundamental to ADTs, it might new to the RM as a formal term but not new to Ada programmers as a concept.

Erhard suggests that we talk more about this (compound objects) in the RM, specifically in Clause 7. It could use more explanation. Tucker should propose some wording for that.

The second topic we need to discuss was raised by the AdaCore LD Circle. They would like to see some simplification of the Global aspect. The suggestion is to only support **all**, **synchronized**, and **null**. Tucker thinks that we might split the definition into a core part and a detailed part in Annex H.

Steve raises the idea of not using the name Global. The original idea was that the SPARK mechanism would eventually evolve to use the Ada version, but that doesn't seem to be the way things are going to go. If not, we could change the name to avoid any current or future conflict.

Ed Schonberg likes the idea of the separation, the compiler needs to know whether a subprogram has some side-effects, and the details are less important.

Tucker will work out a proposal for such a split. He notes that the access type stuff isn't very useful; if it is used, a conflict is very likely. Randy notes that for a parallel loop, that's true for almost all detailed annotations. This would be treated as an alternative to the original AI12-0079-1.

Edward Fish talks about a possible improvement to access types. Tucker notes that he doesn't think that we want to try to change the way people program. Randy notes that's part of the reason that some sort of conflict checking is so important.

Keep Alive: 10-0-0.

On Monday, Tucker asks about the conflict checks: What we do want to do with the split? After some discussion, we deccide that we want to keep the checks in the core, just based on the simple Global setting that we have (that is, "All_Parallel_Checks"). And the turn it off option also has to exist. The other options can be moved to the annex.

Returning to Saturday, we turn to the third topic.

Randy notes that he and Richard are uncomfortable with the idea that dereferencing internal access objects can be erroneous and that there is no way to detect the problem. Tucker suggests that we could include this sort of thing in the Annex, it would be similar to the way Nonblocking was handled in Ada 2005. (Erroneous in the core, checked in the annex.)

Randy has (privately) proposed a Primary aspect; when it is used the compiler would inserts reachability checks before a dereference. That would ensure that dereferencing an access object that is not reachable is a bounded error -- it either raises an exception or the check goes into an infinite loop (no bad code could execute). Such checks could of course be suppressed (which would return the original situation, but more expectedly).

Richard (privately) proposed a library. Tucker finds that a bit heavyweight. He also suggests using Unchecked_something. That seems reasonable if using regular access types. It's less safe in that case.

We don't draw any firm conclusions on this third topic, nor give anyone an action item.

AI12-0139-1/04 Thread-safe Ada libraries

Randy notes that the author seemed to have lost interest when it became obvious how much work it will be, so it stalled.

This would add a LOT of packages. Tucker notes that we've done a number of things to make this area better (Global => **null**, Nonblocking => True, the extra Brad rules). Tucker also notes that in a multi-core environment one does not want to use locks. But using lock-free algorithms would require redoing everything.

Richard notes that this would be nice to have, but it is a lot of work. Justin notes that his game engine had to wrap everything in protected objects.

Tucker moves that we vote this No Action.

No Action: 7-1-1. Justin opposed, Steve abstains.

AI12-0197-1/01 Generator Functions

Randy notes that alternative 1 was replaced by alternative 3. Alternative 3 was previously put on hold, so this alternative is vestigal.

No Action: 9-0-0.

AI12-0197-2/02 Passive Tasks

Jean-Pierre proposed this alternative, and then later disavowed its existence. So this is a candidate for No Action.

No Action: 9-0-0.

AI12-0197-4/01 Coroutines and channels

This is a real alternative to alternative 3.

Hold: 9-0-0.

AI12-0205-1/02 Defaults for generic formal types

As part of the discussion of AI12-0215-2, Randy notes that in Warsaw we implied we wanted to revisit whether to allow defaults for generic formal types.

Randy opines that there is little new with type defaults that can't already happen. Ada compilers have code to insert defaults into instances (for formal subprograms), and code to deal with one formal depending on another (for formal array and access types, and for formal subprograms and packages). This adds nothing new to those cases. Steve is not so sure that is the only issue.

We discuss the example:

```
generic
   type Elem is private or use Float;
   type Arr is array (Positive range <>) of Elem
      or use Float_Array;
package ...
```

Steve does not like the possibility that one could have an illegal instantiation if some but not all of the parameters default. We note that this looks quite useful.

Tucker notes that this same sort of dependency can happen with the subprogram <>.

Gary notes that people would hate not being able to do this.

Tucker notes that making these independent is the easiest rule.

Ed Schonberg suggests that the **or use** has to the same class of thing. The rule should be any sort of thing that **could** match the generic.

Tucker suggests that we simply use the actual matching rules and say that we don’t require matching of formal types.

Someone asks for a practical usage of this feature. This would be useful for a storage_pool type, as usually you don’t want to mess with it, but it is helpful to be able to use a specific pool in special cases (for bounded, debugging, etc.). It is noted that C++ provides something like this in their containers.

Tucker suggests that we could adopt the easy rule that a formal type cannot depend on some other formal parameter. That would prevent some sorts of defaults, but they aren’t that likely to be common and the rules could be relaxed in the future.

```
generic
   type Elem is digits <> or else Float;
   type Index_Type is (<>) or use Positive;
   type Arr is array (Index_Type range <>) of Elem
      or use Float_Array; -- (1)
package ...
```

Tucker's suggested rule makes (1) illegal.

Ed Schonberg suggests a very permissive rule. A compile warning is always possible.

A suggested rule is: If a default depends on a formal, then the prior formal must be defaulted. The legality would be that the all of the substitution were made, then the usual rules applied.

We're just talking about the declaration of the generic itself. Instantiation rules don't change.

Randy worries that the use of anonymous instances in defaults could bring up some new issues. For example:

```
generic
   type Elem is digits <>;
   type Arr is array (Positive range <>) of Elem
      or use new Arr_of(Elem).Arr;
   with function F return Elem is new Inst(Elem);
package ...
```

We think after discussion that the issues are similar to formal packages and thus are not new.

Straw poll: Do we want generic type defaults:

Favor: Ed Fish, Tucker, Randy, Justin, Brad, Richard.

Opposed: Erhard, Bob, Steve.

Abstain: Jeff, Ed Schonberg, Gary.

(Reminder: Observers can vote in a straw poll.) This looks like a (weak) consensus to move forward with this.

The rule for a generic declaration is to substitute defaults for all formal parameters, then check the full rules for the generic declaration again.

```
generic
   type Elem is digits <>;
   type Arr is array (Positive range <>) of Elem or use Float_Array;
              -- or use Float_Array is Illegal.
```

This requires Elem to have a default. It would be weird to write this, as it wouldn't be useful (it would be adding an implied contract that Elem = Float if Arr is to be defaulted). Bob notes that we want an AARM note to mention this.

When Elem is defaulted, then this is a reasonable thing to do.

Randy will try to write up this rule.

Remove AI12-0205-1 from Hold status and Keep Alive: 6-1-3. Bob opposes to be consistent with his straw vote. Ed Schonberg, Steve, and Gary abstain.

AI12-0214-1/02 Case statements and expressions

Matching brings in identifiers and visibility, whereas here we just have wildcards. Bob notes that his son thought that the matching identifiers is less important than the wildcard form.

Ed Schonberg thinks it is worth it if it is a full pattern matching, otherwise he doesn't think it worthwhile. Randy finds most of the value in the wildcards, pattern matching brings visibility issues with it.

We decide to split the baby and leave this open for the future when we can discuss it again.

Hold: 8-0-1. Richard abstains.

AI12-0214-2/03 Boolean conditional case expressions and statements

See the discussion and vote on AI12-0341-1.

No Action: 9-0-1. Randy abstains.

AI12-0215-1/01 Implicit Instantiations

We'll use the AI12-0215-2 version if we do this.

No Action: 8-0-1. Brad abstains.

AI12-0215-2/01 Implicit instantiations

The title should be "Anonymous instantiations"; there's nothing implicit here. Randy copied the subject from the other alternative and forgot to change the subject.

We don't want unnamed_instance_name as a production.

Randy bring up the question of type defaults. We implied in Warsaw that we wanted to revisit that, but the AI did not. Randy opines that there is little new with type defaults that can't already happen with formal access and array types. Steve is not so sure. The rest of this discussion is filed under AI12-0205-1.

After a long diversion, we return to the discussion of this AI.

Or use is better for the default for a package, not :=. So make that change in 12.7.

Tucker asks why Steve thought it necessary to have have both anonymous_instance and anonymous_instance_name. Steve explains that when used in an instance or object declaration, the instance is immediately elaborated, freezing happens, and so on. When used as a default, those things happen when the generic is instantiated. Tucker thinks that it is not that important to point out the difference from using this as a default as opposed using it normal. Ergo, it doesn't seem necessary to have have both of these syntax items.

Erhard suggests that in the !problem "generic type" should be "type declared in a generic package".

Erhard notes that in the object case, you can't get anything from the package other than the type. Prefix notation, indexing, iterators, and aggregates are available for a container (or similar user-defined type). So, for at least some uses, you don't need anything explicitly from the package. This is similar to anonymous arrays in that not all operations are available for them.

It is suggested to use a derived type:

```
type Der is new new Inst (...);
```

or a subtype:

```
subtype Sub is new Inst (...);
```

In both of these cases, one could use a **use all** clause to get more of the operations. "**new new**" in the derived type gives indigestion.

Tucker wonders why you wouldn't just instantiate the package in a case like this; it's hardly any longer than the subtype and you still have to come up with a name. Whatever advantage there is to an anonymous instance is lost in this case.

Steve would like guidance of how to unify the the wording for the default and non-default case. Tucker will provide.

We definitely need to allow an anonymous instantiation as a formal subprogram default, since generic subprograms are possible and commonly used.

Ed Schonberg wonders if there are any legality rules for the default package. There is a requirement that the actual matches formal. Tucker worries that if there are formals involved, does this actually work?

The only kind of package default proposed is new instance; one cannot use an existing package. An existing package is not syntactically allowed; it would never match if anything depends on the formals.

Richard wonders if we should be allowing

```
      or use <>;
```

And we could also allow a name of a existing instance. If the formal package doesn't depend on the formals of the enclosing generic, then a fixed package instance works. (That is, if the formal package parameters are specified as <>.)

Steve wonders why we are doing this? Ed Fish notes that you can have a signature package with all of the interesting types. In that case, an anonymous instance would eliminate the need to write extra instances just because a package has a signature package.

This handles the case where some users have an existing signature package, and some do not have one and get one on the fly.

Ed Schonberg notes that we now have a picture of the semantic complications. It is not clear if this worth the extra work.

Ed Fish notes that his proposal (in AI12-0268-1) gives a name to the automatic instance; he's not too excited about anonymous stuff.

We would like to pass this off to Raphael Amiard, since the request originated with him. In particular, we would like to know whether this proposal would solve some of his problems or whether it is pointless to him. Ed Fish says that he is interested as well. We give Ed Fish primary responsibility for updating the AI and he will coordinate with Raphael Amiard to answer questions about the value and details of this proposal.

Keep Alive: 10-0-0.

AI12-0229-1/02 Type renaming

Tucker thinks this is a dubious idea. Randy notes that he was the one that proposed it (with slightly different syntax). That doesn't change his current opinion.

Randy notes that this is a problem that we've tried unsuccessfully to solve before. There is little enthusiasm for trying again.

No Action: 6-0-3. Randy, Bob, Gary abstain.

AI12-0239-1/03 Ghost Code

Justin explains the changes.

We think Ghost Policy should be renamed Ghost Kind, Ghost Set, or Ghost Category. It's not a policy, it's an identifier identifying a set of ghost entities.

We go with Ghost Kind (Ghost_Kind aspect). The pragma is Ghost_Policy (not Ghost_Assertion_Policy).

When one specifies Ghost_Kind => <some id>, then Ghost is True for that entity.

Richard asks whether you can turn off all kinds at once. Setting Ghost_Policy without a kind turns off all kinds:

```
pragma Ghost_Policy (Ignore); -- Affects all ghost kinds;
pragma Ghost_Policy (Red, Ignore); -- Affects just the Red kind of ghosts.
```

Tucker notes that by default, the ghost policy should be the same as the (global) assertion policy.

Disabled ghost code can't be referenced from anything enabled. Ghost code can't modify non-ghost state.

Justin suggests:

```
pragma Assertion_Policy (Ghost => Ignore); -- Turns off all ghost policy.

pragma Ghost_Policy (blah => Ignore/Check); -- Blah kinds
pragma Ghost_Policy (Ignore/Check); -- All kinds
```

Gary suggests "Enable" or "Disable" instead of "Ignore" or "Check" since this is code, not necessarily some assertion.

```
pragma Ghost_Policy (blah => Enable/Disable); -- Blah kinds
pragma Ghost_Policy (Enable/Disable); -- All kinds
```

Steve notes that the relationship matters for assertions, we can't have an assertion depending on a disabled ghost. We want to make that easy for users.

So we could have "assertion-kind" ghost kinds.

```
pragma Assertion_Policy (Ignore/Check);
    -- implies Ghost_Policy (Assertion => Disable/Enable);
pragma Ghost_Policy (Assertion => Disable/Enable,
                     G_Kind2 => Disable/Enable);
procedure P (...) with Ghost[, Ghost_Kind => Assertion];
```

Only one kind per entity.

Why do we need the Ghost aspect at all? It is True/False, which can be computed (that is, depend on a static value/expression defined somewhere else). But it is implied when a kind is specified (the above is redundant).

```
procedure P (...) with Ghost_Kind => Performance;
```

Steve asks what happens if someone misspelled the Performance kind. There has to be some sort of check, or there would often be an unintentional proliferation of kinds, which probably wouldn't have the intended effect (but would cause mysterious errors, at least until the spelling error was discovered). It is suggested that you have to be in the scope of a Ghost_Policy for a user-defined policy before you can use the user-defined policy in Ghost_Kind. This gets general agreement.

We forget to take a vote on this one; it gets treated as if it was voted "Keep Alive". [Addendum: Steve Baird claims in his review of the minutes that we couldn't vote "Keep Alive" because of the difficulty of keeping ghosts alive.]

AI12-0240-1/04 Access value ownership and parameter aliasing

We would like something like this, as Parasail has a feature like this. (And Rust does too.) Alternatives 1 through 3 of this AI were replaced by alternatives 4 and 5 (5 is a simplification of 4). So let's just leave alternative 5 open for the future.

No Action: 9-0-0.

AI12-0240-2/02 Access ownership for Abstract Data Types

See discussion of AI12-0240-1.

No Action: 9-0-0.

AI12-0240-3/01 Access value ownership and parameter aliasing

See discussion of AI12-0240-1.

No Action: 9-0-0.

AI12-0240-4/01 Pointer ownership for Abstract Data Types

See discussion of AI12-0240-1.

No Action: 9-0-0.

AI12-0240-5/02 Heap object ownership for Abstract Data Types

See discussion of AI12-0240-1.

Hold: 9-0-0.

AI12-0243-1/02 Subtypes as primitive arguments

The problem is that a subtype with a predicate cannot be the controlling parameter of a primitive operation. That prevents something like:

```
type File_Type is tagged ...
subtype Open_File_Type is File_Type
   with Dynamic_Predicate => Is_Open (Open_File_Type);

procedure Put (F : in Open_File_Type; S : in String); -- Illegal,
   -- controlling operand is not of the first subtype.
```

You can work around this by putting the predicate into the precondition of Put, but that can cause proliferation of text and increases the chances that it is missed or incorrect for one subprogram out of many.

The problem is that then overriding is messy. In order for dispatching to work, every overriding needs the same subtype with the same predicate. Since the predicate is an arbitrary expression, conformance is difficult.

Steve created this version with this particular solution, but we've never discussed this version with the full ARG (he did discuss it with Randy).

Tucker and Bob both think the problem should be fixed, but don't like this solution. Bob will take the AI and discuss the corner cases with Steve.

Keep Alive: 8-0-1. Justin abstains.

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

Anonymous instantiations in this case are are not really anonymous since they have a name (the name of the formal for which they are a default).

Justin notes that he feels this is a better idea than AI12-0215-2.

We would like Ed Fish to incorporate this into AI12-0215-2.

No Action: 9-0-0.

AI12-0280-2/05 Making 'Old more flexible

Tucker explains the AI.

Bob and Steve worry that this is a new thing in the backend. Specifically, declaring an object that is initialized only conditionally. Gary notes that it is like initializing a variant record, where one variant doesn't have the component and the other does, with the discriminant determined at runtime. The backend must have some way of supporting that.

Jeff notes that paragraph 6.1.1(22.14) in the draft RM has "a[n] container element association". This occurs in the AI as well.

Gary: AARM Reason "...possible side[-]{ }effects." Steve notes that there are 4 of these. [Editor's note: I only found three, and one was in the discussion of the examples, the other two were in AARM notes.]

"non-aliased" 3 times, once with "nonaliased", and "unaliased" in RM. No consistency with these in the RM. Still, get rid of the hyphen in these cases, as well as in the AI.

Approve AI with changes: 9-0-1. Gary abstains.

AI12-0302-1/03 Default Global aspect for language-defined units

Tucker changed to use overriding for parameter modes. He also added the generic information.

Randy notes that we still need a note to mention that the Global contracts for containers are found in AI12-0112-1.

Approve AI with changes: 9-0-1. Justin abstains.

AI12-0312-1/06 Examples for Ada 2020

In the example after 7.3.2(24/3), the body of Change_Priority is in the private part. Add after **end record:**

```
end Work_Orders;
package body Work_Orders is
```

Brad and Ed Schonberg were discussing the Prime number example.

Tucker notes that this is a bad example because just writing a container aggregate would give you the same result. Brad notes that wouldn't happen in parallel.

Edward Fish suggests a Huffman-encoding dictionary. He is asked so show an example.

Randy suggests splitting out the Reduction examples into a separate AI, so we can finish this big pile of examples and keep working on the reductions. This meets general agreement.

There are two nearly identical Value functions in this AI, can we get rid of one?

Roman_Digit'Value (S(I..I)) would allow us to get rid of the first one. Or use the second Value with an appropriate conversion.

Work_Orders has a parameter "when", that's obviously a reserved word.

Approve intent: 10-0-0.

AI12-0312-1/07 Examples for Ada 202x

Brad sent a updated set of new examples.

Randy notes that ‘Value needs a character literal. Tucker notes that we don’t need the slice in that case. So replace S (D..D) with ‘’’ & S(D) & ‘’’

Put the representation clause in 13.4. Add the “see”s as suggested.

Card_Color example should reference 3.5.1 rather than 3.5.2.

Drop the “with See_3_5_1; use See_3_5_1;” from the 7.3.2 example.

Subsytem should be Subsystem.

The example line in 13.4 needs a period on the end.

Approve AI with changes: 10-0-0.

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

Replaced by AI12-0334-2.

No Action: 9-0-0.

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

Once predicates are involved, it’s necessary that Global/Nonblocking be handled on subtypes. A predicate is an arbitrary expression, which itself can involve global objects, blocking, and pestilence, and it isn't sensible for a type to have to reflect any possible predicate.

Tucker notes that “or the Global aspects come from the same declaration” is missing from the 4.9.1(2/5) paragraph.

The definition of the stream attributes is still missing from this AI. Tucker ran out of time to address that before the meeting. That problem doesn't have anything to do with predicates, and it confuses the issues.

We decide to split dispatching globals/stream attribute definition into a separate AI. This AI would then be all about predicates and subtype constraints, which should make it more understandable.

Tucker will keep both parts.

Approve intent: 8-0-1. Bob abstains.

AI12-0340-1/01 Put_Image should use a Text_Buffer

The bounded buffer should have Implementation Advice not to use dynamic allocation.

Randy asks whether limiting the length of the implemented-defined string for Image when a bad character occurs is a problem. We couldn’t find any.

Jeff says that 4.10(31/5) should say “Image”, not “Wide_Image”.

Bob notes that line 187 in the AI says “[the] any character”. (This is in the last paragraph of the new subclause A.4.12.)

Brad asks about line 156 (a comment in the bounded buffer definition): “Non-abstract overridings of each inherited operation [inherited from] are declared here.” Gary wants the hyphen dropped from “nonabstract”.

The question is asked if these overridings need to be visible. They do for Pre for the Bounded version. Randy made both versions consistent.

Gary notes a typo in the paragraph defining the new line operations: “... that represent[s] a new line ...”.

Approve AI with changes: 10-0-0.

AI12-0341-1/01 Syntax of conditional case expressions

Tucker proposes No Action on the entire thing (that is reopen AI12-0214-2 and kill it). This feature was proposed for SPARK and they don't have any interest. There's value, but not enough to spend time arguing for (or about) it.

No Action: 9-0-1 (Randy abstains because it will be a lot of work to remove from the draft.)

AI12-0343-1/01 Return statement checks

Randy tries to explain the problem and his suggestion. Basically, the wording in 6.5 does not specify whether the dynamic checks occur before or after the execution of the sequence of statements of an extended return. This matters for some of the checks, since the statements can change the value returned, and returning an unchecked incorrect value would defeat the purpose of some of the checks.

Additionally, Steve noted that predicates definitely are enforced when the object is created. We explicitly added a check to ensure that predicates hold for **in out** and **out** parameters, and it seems necessary to have a similar guarantee for the return object. Note that there is no problem for a simple return, but the use of an extended return could cause the predicate to not hold on the object ultimately returned.

Tucker suggests that the paragraph about executing the statements gets put between the "early" and "late" checks. Then we can say that the order of the text defines the checks.

Edward Fish asks if **constant** changes the place where the checks are made. That could work, but we don't want the behavior of the checks to change over **constant**. That would be too confusing to users.

We agree with Randy's proposal. Randy will take the wording.

Approve intent of AI: 9-0-0.

AI12-0344-1/01 Procedural iterator aspects

Brad had suggested that procedural iterators in general should allow only one such parameter. Then the <> isn't necessary, and it can be removed. Tucker thinks this is a good idea. Randy notes that we still would need the rule in the aspect definitions, since they apply to subprograms outside of the context of a procedural iterator.

Steve notes that "at {exactly}[least]" should be "{exactly}[at least]".

Steve asks what happens if one of the access-to-procedure parameters is hidden and becomes visible later. We all go "arrrghhh!!!". Eventually, Randy realizes it isn't possible, since the parameter has to be anonymous, and that can't be hidden. We all relax.

Bob wonders if Parallel_Calls should be allowed for any subprogram, as opposed to just procedures. So this should be "subprogram" rather than "procedure". We laugh about conservation of "subprograms" and "procedures" in this wording.

Steve does not like the "assuming" wording. Randy copied most of that from 9.10.1, although it uses "presuming" as opposed to "assuming". So replace "assume" with "presume".

"Parallel_Calls" really belongs with 9.10.1.

Tucker will take this one.

Approve intent of AI: 6-0-4. Gary, Justin, Richard, Ed Schonberg abstain.

AI12-0345-1/01 Dynamic accessibility of explicitly aliased parameters

Randy explains that the model was that it never would be necessary to pass in an accessibility level. The intent was that explicitly aliased parameters would act like normal parameters in regard to accessibility except when they are compared against the master of the call (the accessibility of the function result), which is defined to pass.

It's pretty clear that the static accessibility rules implement this model, but the wording for dynamic accessibility doesn't try to do so, and that can be determined by using access parameters or stand-alone objects of an anonymous access type.

We note that this model means that passing 'Access of an explicitly aliased parameter to some other access parameter gets local accessibility, which is an annoying result (the special case gets lost in that case). But the other possibility of depending on the details of the return is more annoying (as the accessibility of a function could change because of some maintenance on the type of a deeply nested subcomponent). So we agree to go with Randy's model.

Edward Fish asks if the level of the type of the explicitly aliased parameter would work. No, that would be too strong (a lot of types are library-level, but the calls are local – consider a container indexing).

Tucker thinks that the wording isn't right to implement the intended model. We need to simply talk about comparing against the "master of the call".

Steve will try to come up with better wording.

Approve intent of AI: 9-0-0.

AI12-0346-1/00 Ada and OpenMP

[Editor's note: This AI number was assigned after the meeting.]

On Sunday morning, we had a special session on OpenMP. We had a (remote) guest, Michael Klemm, who is the OpenMP Consortium CEO.

Tucker shows his set of slides giving an overview of proposed parallel support for Ada 202x. [Editor's note: These slides can be found at: https://drive.google.com/file/d/156vq44aK2FF60cbd_I8hIOXmqEGjl1H7/view]

He starts with the difference between Concurrent/Parallel programming, then Ada 202x goals.

He notes that a parallel block could be implemented as a parallel loop containing a case statement with each iteration being a single branch of the parallel block.

He discusses relevant features of Ada 202x: Iterators, filters, parallel constructs, and reduce.

Michael Klemm takes over with a set of slides of his own. [Editor's note: Unfortunately, Michael was not able to give us the entire slide deck that he showed us as it contained some proprietary information on slides that he didn't use.]

Michael starts with the evolution of OpenMP. OpenMP 1.0 was a portable interface to threads.

Later versions support unstructured parallelism. The various versions use pragmas to configure. Initialization pragmas starts a "team of threads", and then the runtime will decide how to map those.

Note that an OpenMP "thread" is essentially what Ada calls a task. And an OpenMP "task" is what Ada 202x calls a "logical thread of control". We'll try to prefix "task" and "thread" with "Ada" or "OpenMP" as needed when it could be confusing.

Returning to Michael's presentation. Multiple patterns are possible (single creator, multiple creator, nested tasks).

C example using the single creator pattern:

```
    #pragma omp parallel
```

```
#pragma omp master
while (elem != NULL) {
    #pragma omp task
         {compute(elem);}
    elem = elem->next;
}
```

Michael also showed various Fortran examples, they are rather similar so none of those are recorded here.

All threads in a thread team are candidates to execute tasks.

Tasks are created with a pragma in C:

```
#pragma omp task [clause] {structured block}
```

Note that the block is logically part of the pragma.

Various options are in `clause`. Can specify objects to be "private" (which are task-specific) or "shared" (which use one object for all tasks). Priorities can be set (but these are "hints", can be ignored).

OpenMP was built on top of existing languages, so they used pragmas and the like, rather than changing the definition of the languages.

Basic OpenMP is "sequentially equivalent"; turning off OpenMP gives a program which still will work sequentially. For instance, in the example above, ignoring the pragmas gives one a sequential while loop.

The "private" option exists mainly because C and Fortran didn't have local declarations when OpenMP was defined (that's now fixed for both). So relatively unnecessary for Ada (we can just use a block).

There are various pre-determined data-sharing attributes. Variables declared inside the construct are private. Tucker asks about "threadprivate", Michael says he would like it to go away. He thinks a program that needs it is broken, and C, C++ have their own way to do it. Tucker notes it is useful for the scheduler, but it doesn't seem useful for user-level code.

A task has to resume on the same thread that it started on, because of threadprivate. "Untied" allows it start on any thread (no use of threadprivate).

Task reductions (using a taskgroup).

```
#pragma omp taskgroup task_reduction (op : list) {structured-block}
```

Applies the reductions to the objects declared by all of the tasks.

The implementation can chose how to do this.

Loops can be declared with:

```
#pragma omp taskloop grainsize(TS)
```

which allows loops to be managed by OpenMP as opposed to specifically.

"Simd" is an option, that allows simd execution of the loop body. Loop can be "simdized", chunks are the size of a simd register.

Simd = 1 is allowed (so the compiler is supposed to use vector instructions when available, but not absolutely required).

Michael explains that this is available as auto-vectorization often fails; this directive essentially says this is OK to do so.

The "simd" option can compose with tasks and threads, so each chunk can be vectorized.

It's possible to declare "simd" functions, compiler will create a vector version of the function.

We turn to discussing the OpenMP model for GPUs. OpenMP currently supports one host; multiple accelerators/coprocessors of the same kind. (That means that you can't use both Nvidia and Intel video processors at the same time).

OpenMP automatically enables the coprocessors.

Michael notes that the big production systems are homogeneous (all the same GPU), so there doesn't seem to be a lot of need for multiple kinds of coprocessors in a single program.

The type of coprocessor is not significant. A coprocessor can be specified with:

```
        #pragma omp target {structured-block}
```

This can specify the device and more.

The execution model for GPUs is that transfer of control is sequential and synchronous, the direction is determined by the data. This happens inside of a single OpenMP thread.

The compiler has to determine the data that has to be transferred (it does not have to determine if the data is changed, so it will transfer in both directions). These are the objects referenced in the target region.

The compiler is not required to detect non-conforming code, so it can blow up when run.

Michael is asked if OpenMP supports debugging on the GPU. He tells us that it does have a debugger interface, but the tools have to come from the compiler vendor.

The OpenMP data environment is lexically scoped; this means that allocated buffers/data is automatically released.

When pointers are used, the compiler may need help to determine the size of the objects to transfer, or even the objects needed.

The compiler has to produce the code for the loop body both the GPU and for the host (in case the OpenMP implementation decides to not use the GPU for some reason). As always, the code is supposed to be executable even if the OpenMP implementation does nothing. The code in the loop has to be restricted to what can be executed on the GPU. [Presumably, this is target-dependent, but this was not discussed during the presentation. - Editor.]

Tucker asks about OpenMP APIs. These are specific to the OpenMP implementation. Michael notes that GCC uses their own implementation of OpenMP. Clang/LLVM uses the Intel implementation of OpenMP.

For each directive, there is a set of API routines that implement the directive. We'll talk more about this later.

Continuing Michael's presentation. OpenMP separates offload and parallelism. Therefore, one needs to explicitly create parallel regions on the target devices. This allows a subset of usual OpenMP to be used on the target device.

This model allows any sort of coprocessor to be used (FPGA, GPU, whatever).

Data that can be copied has to be bitwise-copyable. OpenMP 5.0 allows serialization subprograms.

Target allows "nowait", in which case the host thread does not wait. In OpenMP-speak, the targets are OpenMP tasks, all of the OpenMP task constructs can be used when using corprocessors.

Tucker explains how we propose to map to OpenMP. We are providing a parallelism approach for Ada that could be implemented using OpenMP. And possibly having a standard OpenMP interface.

Michael notes that how one could use a foreign program (say in C) which uses OpenMP with an Ada program that also uses OpenMP is an important issue. Michael says that OpenMP is really bad at interoperablity with the threading model. For instance, a threading region that calls a subprogram that contains a threading region causes trouble.

The only model is that completely interoperable is the OpenMP tasking model. [It is suggested later that we could use the usual distinction between a main in Ada and a main in a foreign language to deal with this — an Ada main would set up OpenMP threads and we'd assume the foreign main did that if needed. I didn't record when this was mentioned, so I don't know if this is the right place for this in the minutes, but nowhere else seems relevant. - Editor.]

Michael suggests ignoring the OpenMP threading model; the runtime has to initialize threads and after that they should be forget about that and use OpenMP tasks (which correspond to Ada "logical threads of control").

He suggests using a "taskloop" to map a parallel loop.

Tucker shows a layering: syntax (parallel for, parallel reduce, and so on); pragmas (possibly standard, possibly implementation-defined); Ada.Parallelism package (to be called by compiler-generated code); System.LWT interface and then specific packages System.LWT.OpenMP, System.LWT.Default, ... - used to implement Ada.Parallelism. The interface is primarily a dispatch table.

Tucker shows an example of using a scheduler object to initial scheduling. Steve notes that his example would be too late if package elaboration uses any task or parallel features (Tucker's example put the scheduler object in the main subprogram). Tucker comments that it could be placed in a package that is elaborated early as well.

Michael notes that too many layers can have an impact on performance. Especially if data gets copied. Tucker notes that Ada implementations typically pass large objects by reference, so that's not usually a problem.

OpenMP translation uses a translation of loop bodies to some sort of subprogram (generally called a "thunk"). The Intel compiler passes in pointers to the shared/firstprivate data. Since Ada uses up-level addressing, we wouldn't have to pass more than the up-level addressing data to reach shared objects.

Tucker asks Michael about the Clang API. He wonders about implementing a create thread operation (for example). Michael shows us the API in the reference documentation. [It can be found at `http://openmp.llvm.org/Reference.pdf` - Editor]

Task groups are created by start group and end group; the implementation keeps track of the groups. The APIs don't require passing a taskgroup handle, it is always the current taskgroup.

Steve asks about canceling execution. Michael says that OpenMP has a best effort task cancelation. Other than cancelation points, it doesn't stop execution. But nonstarted tasks immediately become completed. It is works on groups. He shows us the exact example that Steve was talking about. Cancel taskgroup cancels yourself.

Tucker asks about exceptions. OpenMP does not worry about exceptions. It assumes one path into a construct and one path out. So a parallel construct needs a handler inside and reraise outside.

Brad asks if OpenMP supports non-commutative reductions? Not directly, but it now supports user-defined reductions, and that could be used to support a non-commutative reduction.

We asked Michael to share any slides that he can. We thanks him for his time and efforts.

On Monday, we discuss the OpenMP session from the context of what we need to do for the Ada Standard. Tucker notes that we don't want to depend on OpenMP, but we do want to be able to create a binding to it.

We want to support controlling on a partition-by-partition basis what light-weight thread scheduler is being used.

Tucker shows his layering proposal (see Tucker's slides from Sunday).

Steve asks whether this is like other runtime configuration. If it is, then it should be controlled by switches and the like. Tucker's response was not recorded.

Ed Schonberg notes that OpenMP Ada needs to be as easy to use as the Fortran version, or it will be a non-starter.

Tucker thinks that it would important for people to be able to write their own schedulers, similar to the way to we allow redefining Storage_Pools.

Justin wonders if people that won't allow tagged dispatching would handle this interface, which is heavily defined by dispatching. Steve worries that there is certification things that ban all forms of indirect calls. Tucker notes that

DO179C has an annex on tagged types, so OOP is OK there. If an organization isn't moving to DO179C, they aren't going to be doing multicore stuff either.

Tucker thinks that OpenMP pragmas should not be in Ada 202x, but we will have to define those (hopefully for the OpenMP folks). Tucker notes that our ideal goal would be to get Ada into the OpenMP standards.

OpenMP threads are similar to Ada tasks; OpenMP tasks are simple to Ada logical threads of control. (We noted this yesterday.)

Interoperability seems to be possible with foreign code that uses OpenMP tasks, probably not with OpenMP threads (according to Michael Klemm). The thread issues exist in just C code as well, it's a general problem.

Steve notes that we differentiate between the cases where the main is in C and where the main is in Ada. That seems appropriate here. An Ada main needs to initialize OpenMP threads, otherwise the Ada runtime would not do that (but still could use OpenMP tasks).

Richard wonders if the schedular object could be declared inside the sequential package. He's trying to find a solution to Steve's previous problem of notes that parallelism would not work in elaboration code if the scheduler object hasn't been elaborated yet. Note that OpenMP makes it pretty clear that such code is run sequentially, and the implementation always needs to be ready to do that. So it's not clear that it is a problem.

Who wants to work on making proposals in this area? Tucker notes that there are at least two AIs – one is a plug-in standard, and the other is an OpenMP instantiation with pragmas.

Ed Schonberg notes that GPU support is another part of this. We don't have a way to describe where data lives or when it gets copied. Tucker notes that OpenMP has pragmas for that, he thinks that these are specific items for OpenMP.

Ed Schonberg suggests an annex. Randy notes that we might not even be able to talk about OpenMP in a ISO standard (generally, we can only talk about ISO standards). Perhaps we would have to use a Technical Report or something similar.

Richard wonders if it really necessary to specify data location. The compiler can do the right thing. But that is very inefficient. So some sort of data locality pragmas are needed. [Editor's note: don't aspects make more sense? An object location specification is entity-specific, and it seems to be a representation aspect.]

Ed Fish notes that these pragmas aren't really necessary, it's better for the compiler to figure it out. But that is a research project except in the simplest of cases. We probably should describe the pragmas as "hints", similarly to the way inline is described. (GNAT does plenty of inlining that doesn't depend on the hints.)

There is a bit of discussion about the importance of GPUs. It seems important enough for some sorts of uses that we shouldn't ignore it.

Richard notes that he would like to work on it but he doesn't know enough. Ed Fish concurs. Randy notes that we need a leader for this area. Tucker is the only one to have enough experience and interest to do that. Justin notes that a model similar to the DSA might make more sense for GPUs and he would like to pursue that.

Ed Schonberg asks if the syntax would be enlarged. Tucker thinks that we probably would look at pragmas and aspects, but probably not syntax per-se. [Editor's note: Randy made a syntax extension proposal almost as soon as arriving home, so a small enlargement is very possible. His point is that parameters specific to parallel loops and blocks probably should be part of the syntax in the way that aspects are part of the syntax for declarations. We have a long history of trouble with pragmas getting separated from the associated construct.]

So we will create an AI about mapping parallel constructs into OpenMP and other schedulers. Tucker will lead, and everyone interested will help Tucker. Justin volunteers to work on GPU issues, Brad is also interested. Richard volunteers to look at the user experience. Tucker suggests that he create examples. Randy volunteers to help Richard.

Keep alive: 9-0-0.

Tucker wonders about the OpenMP dependence setting. [Editor's note: The details of this setting were not recorded in the main notes above; they essentially allow specifying that some chunk of code depends on some other, and then let OpenMP schedule the chunks in parallel so long as all of the chunks it depends upon have finished first.]

Do we just say the compiler can figure it out? Or do we prevent such things from being parallel at all? Michael had an example of five loops that were interdependent, and some parallelism is possible. How do we map this to Ada? Do we want to try to solve this problem?

Steve notes that we want to be able to do as well as C.

Tucker notes that representing a graph in source code is hard. Brad notes that it is a static analysis thing. Tucker notes that we don't want to call that "parallel", it seems like it is was sequential code that could have pragmas to allow parallel "as-if" implementation.

Steve will look into this issue of mapping dependence annotations.