Minutes of the 30th ARG Meeting

9-11 June 2006

Porto, Portugal

Attendees: Steve Baird, John Barnes, Randy Brukardt, Alan Burns (except Saturday morning), Pascal Leroy, Steve Michell, Erhard Ploedereder (except Saturday morning), Jean-Pierre Rosen, Ed Schonberg (except Sunday afternoon), Tucker Taft, Bill Thomas (except Sunday afternoon), Joyce Tokar (except Saturday), Tullio Varadega (except Sunday).

Observers: Greg Gicca (except Sunday afternoon), Bibb Latting, Javier Miranda (except Sunday), Brad Moore.

Meeting Summary

The meeting convened on 9 June 2006 at 14:15 hours and adjourned at 14:45 hours on 11 June 2006. The meeting was held in a conference room at the Hotel Le Meridien Park Atlantic Porto in Porto, Portugal. The meeting covered nearly the entire agenda (excepting a few Ada 2005 amendment AIs).

AI Summary

The following AIs were approved with editorial changes:

AI95-00447-1/01 Null_Exclusions allowed in Ada 95 (12-0-1)
AI05-0014-1/01 Accessibility of designated objects (7-0-0)

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

AI05-0008-1/02 General access values that might designate constrained objects (9-0-1)
AI05-0013-1/01 Coextensions considered harmful (8-0-2)
AI05-0015-1/01 Constant return objects (8-0-2)

The following AIs were discussed and assigned to an editor:

AI05-0003-1/01 Qualified expressions and names
AI05-0006-1/01 Nominal subtypes for all names
AI05-0007-1/01 Stream 'Read and private scalar types
AI05-0009-1/01 Confirming rep. clauses and independence
AI05-0012-1/01 Independence and representation clauses for atomic objects

The following SIs were approved with editorial changes:

SI99-0001-1/01 Add new predefined pragmas to Pragma_Kinds (8-0-5)
SI99-0006-1/01 Add new An_Interface_Type_Definition to Type_Kinds (8-0-1)
SI99-0010-1/01 Add Extended_Return to the list of Statement_Kinds (10-0-1)
SI99-0013-1/01 Add support for exceptions with string expressions (11-0-0)
SI99-0014-1/01 Add comments regarding A_Formal_Package_Declaration to ASIS (9-0-2)
SI99-0015-1/01 Correct Corresponding_Type_Operators (8-0-2)

The intention of the following SIs was approved but they require a rewrite:

SI99-0002-1/01 Add A_With_Clause to the list of expected kinds of ASIS.Elements.Trait_Kinds (12-0-0)
SI99-0003-1/01 Support overriding indicators (10-0-0)
SI99-0007-1/01 Add support for new object oriented prefix notation (10-0-0)
SI99-0009-1/01 Handle new aggregate features (10-0-1)
SI99-0011-1/01 Add A_Tagged_Incomplete_Type_Declaration to Declaration_Kinds (10-0-2)
SI99-0012-1/01 Add support for null procedure declarations (10-0-1)

The intention to merge the following SIs into one integrated solution was approved (10-0-1):

SI99-0004-1/01 Add new access definitions kinds to package ASIS
SI99-0005-1/01 Generalize object declaration view
SI99-0008-1/01 Add null exclusion construct for an access type to Trait_Kinds

Detailed Minutes

Meeting Minutes

The minutes of the previous meeting were approved by an e-mail ballot in February.

Schedule and format of ASIS revision

We don't have to consider the format; WG 9 has recommended one.

What should the schedule be? Bill wonders how much work it will be to convert to the new format. Obviously some, and that should be an early step.

Pascal wonders how many SIs will be needed to cover the full standard? Bill says that the 15 we have cover everything. That seems low; there might be other issues that were missed.

Should we use the RM tool to create the new Standard? It probably would be easier for making multiple output formats. A long discussion ensues on the technical details of using the tool.

Greg suggests converting the RTF to the new format, he thinks he can automate this and finish by the end of June. Randy should convert to his tool; that wouldn't be a major problem, but funding is an issue (at least until October).

SI changes would probably be added by hand to the base draft; using the tool they would be marked as changes (as we need for the RM).

Are there other issues to be covered? Jean-Pierre comments that there are some missing features. He is planning to write some soon.

Tucker asks for an example of a missing feature. Jean-Pierre says that it isn't possible to get the profile of a dispatching call; there is no way to get to the declaration. He also wonders if there are any semantic queries that might need to be updated. [Editor's note: Much of this discussion became moot during later examination of individual ASIS issues. We found many problems with the semantic queries in ASIS.]

Pascal reminds us that the document should leave WG 9 in the fall of 2008 (it is a three year schedule). So that means a WG 9 vote in summer of 2008. That's 4 ARG meetings at the twice a year schedule. To meet that, we should have a decent draft of the standard by this time next year. So we have essentially three meetings to get to that point.

New SI cut-off date should be May 1st, 2007. All SIs should be in hand by then.

Future activities of the ARG

WG 9 adopted a priority list. It is roughly ASIS first, language maintenance second, improved libraries third (such as more containers), and other improvements last.

WG 9 also tasked us with creating a Type II Technical Report for improvements to the library, specifically including bounded forms for the containers.

How should we proceed to create that Technical Report? It is thought that the technique used for the Amendment's containers library worked well. That is, have a small drafting subcommittee that creates a draft for the entire ARG to review.

Who should be on the committee? We would like Matt to lead it if possible, so we will invite him to set up a group to draft a Technical Report. Tucker says that Bob Duff wanted to be included. Tucker and Randy volunteer to be

secondary members (neither can commit to work on drafting at this time). Randy will create a mailing list for the containers group.

Date and Venue of the Next Meeting

We'll next meet in Albuquerque after the WG 9 meeting. The dates are November 17-19, 2006.

Do we need Sunday (that is, do we need a three-day meeting)? Perhaps not, but it is too early to say for sure. It's better to schedule the full meeting and later shorten it than to try to lengthen it in the future.

Thanks

By acclamation, the ARG thanks our host, Ada Europe, for the accommodations for the meeting.

Old Action Items

Randy finished the draft of the Amendment, and it has been submitted to (and approved by) WG 9. It is now being processed by SC 22.

ASIS ideas were collected and 15 SIs created.

Pascal completed his action items as well.

New Action Items

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

Steve Baird:

- AI05-0008-1

Randy Brukardt:

- AI05-0013-1
- AI05-0015-1
- Put the existing ASIS standard into the format of our standard-creating tools (after receiving it from Greg)

Editorial changes only:

- AI95-00447-1
- AI05-0014-1
- SI99-0001-1
- SI99-0006-1
- SI99-0010-1
- SI99-0013-1
- SI99-0014-1
- SI99-0015-1

Greg Gicca:

- SI99-0002-1 (with Bill Thomas)
- SI99-0003-1 (with Bill Thomas)
- SI99-0007-1
- SI99-0011-1 (with Bill Thomas)
- SI99-0012-1 (with Bill Thomas)
- Create an SI on new attributes (see discussion of SI99-0001-1), with Bill Thomas

- Create an SI to fix the text in 3.9.5 about access_definition (see discussion of SI99-0002-1), with Bill Thomas
- Check all of the paragraph references in the current ASIS standard to ensure they are still accurate
- Create a version of the current ASIS standard in the new format as requested by WG 9

Bibb Latting:

- AI05-0009-1
- AI05-0012-1

Pascal Leroy:

- Create an SI to either remove Name and Name_List or use them as a parameter/return type where that makes sense (see discussion of SI99-0004-1)

Jean-Pierre Rosen:

- Create an SI to define an Is_Implicit_Dereference query (see discussion of SI99-0007)
- Provide the ARG with the specification of his complicated ASIS queries package

Tucker Taft:

- AI05-0003-1 (lower priority)
- AI05-0006-1 (lower priority)
- AI05-0007-1
- SI99-0004-1 (Merge SI99-0005-1 and SI99-0008-1 into this SI, creating an integrated solution)
- Create a proposal for better organized semantic queries for ASIS (see discussion of "ASIS Semantic Interface")

Bill Thomas:

- SI99-0002-1 (with Greg Gicca)
- SI99-0003-1 (with Greg Gicca)
- SI99-0011-1 (with Greg Gicca)
- SI99-0012-1 (with Greg Gicca)
- Create an SI on new attributes (see discussion of SI99-0001-1), with Greg Gicca
- Create an SI to fix the text in 3.9.5 about access_definition (see discussion of SI99-0002-1), with Greg Gicca
- Create an SI to add syntactic queries of the form Has_Blah for all ASIS traits (see discussion of SI99-0008-1)

Detailed Review

The minutes for the detailed review of AIs and SIs are divided into Ada 95 AIs, ASIS Issues (SIs), Ada 2005 non-amendment AIs, and Ada 2005 amendment AIs. The AIs and SIs 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 six votes for, one vote against, and two abstentions.

If a paragraph number is identified as coming from the Amendment (with a /2 version), the number refers to the text in the Draft 16 Consolidated AARM. Paragraph numbers in other drafts may vary.

Detailed Review of Ada 95 AIs

AI95-00447-1/01 Null_Exclusions allowed in Ada 95

Randy explains the problem: transition is difficult because it isn't possible to write code that has the exact same semantics in both Ada 95 and Ada 2005. Supporting **not null** in Ada 95 compilers will reduce the problem, as Ada 95 code with **not null** would have the same semantics in Ada 2005.

The syntax is backwards in the AI; it should be:

access_definition ::= [**not null**] **access** subtype_mark

Approve AI with change: 12-0-1.

Detailed Review of ASIS Issues

SI99-0001-1/01 Add new predefined pragmas to Pragma_Kinds

Should these new items to be added last? The last item is An_Unknown_Pragma; that should remain last. And so should An_Implementation_Defined_Pragma. Insert a new block of declarations after the existing block — that keeps the Ada 2005 ones separate; that would allow subtypes like All_Ada_2005_Pragmas.

Tucker wonders why we don't insert these in alphabetical order. Ed says that we don't want to introduce unnecessary incompatibilities. Leaving these in a group can help if you're just processing Ada 95 code.

Approve SI with changes: 12-0-1.

The vote doesn't seem to bring the discussion to a close. Perhaps we should add the subtypes? It sounds like a good idea, and avoids the question of why the particular grouping.

Tucker suggests Ada_95_Pragma_Kinds, Ada_2005_Pragma_Kinds. Jean-Pierre says that the subtypes should be named similarly to the declarations. Thus An_Ada_95_Pragma and An_Ada_2005_Pragma.

There is an argument that all of the pragmas are Ada 2005 pragmas. Someone suggests Not_Ada_95_Pragma as a joke. Someone else points out that this actually would work in the future when it is revised for Ada 201Z.

Tucker wonders if it is worth having these subtypes, given the difficulty of agreeing on names. They're only for transition. Forget the subtypes; we're back to what we previously voted for.

Someone points out that these should be organized similarly to the attributes. Humm, we don't seem to have a SI on new attributes. There needs to be such an SI; this will be an action item for Bill and Greg.

The attributes probably would be in alphabetical order. The pragmas should be, too. So, intersperse the new terms alphabetically.

The recommendation is to use **when others** during transition; it's the only way to have both Ada 95 and Ada 2005 code with ASIS. We vote on this new organization.

Approve SI with changes: 8-0-5.

SI99-0002-1/01 Add A_With_Clause to the list of expected kinds of ASIS.Elements.Trait_Kinds

The wording should say to add a new grouping "Expected Clause_Kinds" (which is declared in 3.9.22) after the existing "Expected Formal_Type_Kinds". (This function allows testing of the presence or absence limited or private.)

The summary should read: "An expected Clause_Kinds paragraph, with A_With_Clause as the only alternative should be added to account for … (rest the same)".

Tucker notes that the wording of 3.9.5 needs to be updated to take this into account. The SI needs to update 3.9.5 to talk about with clauses.

Pascal notes that there is something weird about 3.9.5 for handling access_definition. Don't all access_definitions contain **access**? Jean-Pierre explains that this finds anonymous access types. Right, but the wording is wrong as it doesn't say that. Bill and Greg are given an action item to update the incorrect text in 3.9.5 as a distinct SI (or as part of SI99-0004?).

Tucker wonders if this isn't a place to consider an alternative interface to replace this one? (It's hard to decide if something is limited.). We leave this for future discussion.

Approve intent of SI: 12-0-0.

SI99-0003-1/01 Support overriding indicators

Erhard complains that the name of the function is wrong. Is_Overriding_Declaration is not what we're doing here (that would be a nice thing to have). The name should be Has_Overriding_Indicator.

Erhard wonders why this returns False for unexpected elements. Jean-Pierre says that there anything that starts Is_... does not raise an exception.

Jean-Pierre doesn't like this solution; perhaps a dedicated enumeration would work better:

```
type Overriding_Trait_Kind is
   (No_Overriding_Trait, An_Overriding_Trait, A_Not_Overriding_Trait);
```

Jean-Pierre says that the current scheme favors the ability to determine all traits at once; sometimes you want that. Sometimes you want just one. It's would be hard to handle all at once with a functional interface.

Tucker says that the query approach is more general than the trait approach; probably the main help for the trait approach is in improved performance.

Tucker suggests making traits as an obsolescent feature; we won't add any new traits. (But we still will add new features to existing traits as appropriate; that is SI99-0002). Rather, we'll add new functional interfaces like: Is_Limited and Is_Private.

The discussion morphs into a discussion of semantic vs. syntactic issues. A task does not have the limited trait; Is_Limited would be false for a task. Yuck. A better name would be Has_Limited_Keyword or Is_Limited_Present to make it clear that it is a syntactic function.

Jean-Pierre comments that items usually refer to their declarations, but anonymous types don't have declarations. So, you can't always use the semantic queries that ASIS has. Pascal says that seems like a bug. We look at a specific example. For Corresponding_Expression_Type, class-wide types return Nil_Element. Yuck.

ASIS tends to ignore implicit declarations. It just returns Nil_Element. Jean-Pierre says that it is difficult to program a lot of these things.

Jean-Pierre says that ASIS allows building of fake declarations for implicit/anonymous things. Requiring compilers to do that would thus be compatible.

Tucker suggests that only semantic queries should be required to work on implicit declarations; implicit declarations aren't structural. But semantic queries should always work.

Jean-Pierre says that he has a package for these "complicated" queries. Tucker is amazed that Is_Limited is considered a complicated query. Jean-Pierre says that it takes 97 lines of code to figure out Is_Limited accurately for the current ASIS. But every compiler has a limited bit in its symbol table; that's a lot of code to replace a simple query. We would like to see the sort of things that Jean-Pierre has in his package, others should do so too.

Pascal suggests that (for a start) anything that has a defined term in the standard should probably be considered for one or more queries. Such things are well-defined, and compilers most likely have a way to calculate the property in order to enforce the rules.

Semantic queries currently in ASIS are dispersed in the various packages.

Most of the "Is" functions are semantics. We want a clear way to tell which operations are semantic versus which are structural (syntax). In many cases, these are similar but can give different answers (Is_Limited is an obvious case).

Tucker wonders what the conclusion of this discussion is. Add more low-level semantic queries. All expressions should return a type; structural queries on implicit declarations are allowed to fail. That is, semantics is complete, the kind of declaration doesn't matter, while structural queries depend on what is in the actual source code.

Ed notes that the ASIS design reflects the entire history of compiler development. Language semantics didn't have a very formal basis back in the early 1980s; so they didn't include much semantic information.

A suggestion is to define a new kind of thing called "entity", and then have queries on that.

We definitely want to ask users for routines that they have used.

Tucker would like to define a model of how things fit together. For instance, an expression has a subtype. A subtype has a type. And so on. He takes an action item to create that. (See later discussion below.)

This SI (SI99-0003) should cover just the structural (syntactic) interface. This should be a separate function:

```
type Overriding_Indicator_Kind is (No_Overriding_Indicator,
   An_Overriding_Indicator, Not_Overriding_Indicator);

function Overriding_Indicator_State (Declaration : Asis.Declaration)
   return Overriding_Indicator_Kind;
```

Erhard wonders about the form of the semantic query. Pascal wants to be able to find out what was overridden. That might require a list. Lists already exist in ASIS. Erhard would like a query If_Overrides. Not sure if that is a good idea, best to defer to later.

Approve intent: 10-0-0.

SI99-0004-1/01 Add new access definitions kinds to package ASIS

Much discussion ensues about where this goes. There is no An_Enumeration_Literal_Specification here. The section reference should be 3.10(6) — this is where it is defined. So it goes after An_Others_Choice.

The editor (Greg) should check all of the paragraph references in the ASIS standard to see if they are still correct.

Why is there a new type? 3.9.12 seems to have this same list, why duplicate it? So drop the type.

Randy says that then Not_An_Access_Type_Definition is the wrong thing. Tucker suggests a rename to Not_An_Access_Definition. Pascal says that at this point, we might as well declare the silly new type. So leave that alone.

The section numbers are missing from the wording section; they're given below when known.

16.19: Access_To_Subprogram_Parameter_Profile. This seems inconsistent: this takes a type kind; but the thing that we're passing is a definition kind. Pascal says that the mistake has been made, a formal type definition is a definition kind, and an Access_Type_Definition. So this is OK.

16.20: Asis.Definitions.Access_To_Function_Result_Profile. (No discussion on this one.)

Why is there a new function? It's needed because the syntax is different, and ASIS follows the syntax closely.

The new function should return Name. Randy objects: Name is not used anywhere in the current ASIS. Sure, this doesn't have any semantics, but it is inconsistent. We should have another SI, where either we remove this subtype, or we use it consistently where it makes sense. Pascal takes the action item to create this SI.

Approve SI with changes: 7-0-2.

Later, we decided to merge SI99-0005 and SI99-0008 into this one; that intent was approved: 10-0-1.

SI99-0005-1/01 Generalize object declaration view

An "object declaration view" is the stuff following the colon.

The wording section isn't wording, and is very cryptic. What does this mean?

After extensive discussion and guessing, we decided on the following: Object_Declaration_View needs to be changed to return the access definition. It also needs to take all of the things that Declaration_Subtype_Mark takes. Declaration_Subtype_Mark is obsolete.

Compatibility is a problem; the changes to Object_Declaration_View would make it return different results. Pascal suggests that we need to add a new query here, and the old one is left alone (and raises an exception if Ada 2005 code is used).

SI99-0008 also has to do with anonymous access types. These should all be handled the same way. We should combine SI99-0004, SI99-0005, and SI99-0008 and come up with an integrated solution. Tucker will take this combined SI; creating new interfaces and obsolescing the old ones.

Approve intent of combining SI99-0004, SI99-0005, and SI99-0008: 10-0-1.

SI99-0006-1/01 Add new An_Interface_Type_Definition to Type_Kinds

Add An_Interface_Type_Definition to Type_Kind second last; this is in section number order.

Add A_Formal_Interface_Type_Definition to Formal_Type_Kind at the end (this is also in section number order). Please fix the section numbers for the last three (they reflect the Ada 83 grammar organization, not Ada 95); specifically formal arrays are in 12.5.3, not 3.6.

Definition_Interface_List should be Progenitor_List.

In two places, "subtype mark names" should just be "subtype mark" (a subtype_mark is a name).

The return type Expression_List is annoying, but that seems to be the way it is done. Why isn't it Name_List? Jean-Pierre says that every name in ASIS is considered an expression (it is hard for a compiler to tell them apart syntactically). After this, Tucker finds Name_List in the definitions for types.

This should be Name_List. Someone complains that is a problem as it returns an expression kind. But that's still OK, there isn't a Name_Kind. Calling it Name_List is better documentation, so do that.

Approve SI with changes: 8-0-1.

SI99-0007-1/01 Add support for new object oriented prefix notation

Jean-Pierre finds the name of this function (Is_Prefix_Notation) confusing.

The SI is focusing on calls, but that's wrong; it is the name that matters (these can be renamed, passed to a generic formal, etc.).

This should apply to the selected component; and you need queries on the call. But there is no new syntax here, so this is not a syntactic query, it's really a semantic query. There appears to be an interface for analyzing entry names, so we could try to use that one.

The semantic interface needs information for Prefixed_View (that should be added to Callable_View: Is_Prefixed_View should be changed to Boolean). Jean-Pierre notes that it would be useful to have Is_Implicit_Dereference.

This results in a long discussion of how prefixes work in ASIS. Jean-Pierre will take an action item to create an SI on the Is_Implicit_Dereference query. (It would be nice if the prefix could be returned with an implicit dereference node.)

There doesn't seem to be any need for an additional structural interface (there is no new syntax for prefixed views). Jean-Pierre says that recognizing it can be done with existing calls. Pascal suggests voting No Action for this SI. Steve Michell prefers that we verify that the existing ASIS can do this before we kill this SI. Greg takes an action item to check the existing wording of ASIS doesn't prevent prefixed views from being analyzed. Tucker will ensure that the new semantic interface will cover this. No function will be added.

Approve intent of SI: 10-0-0.

SI99-0008-1/01 Add null exclusion construct for an access type to Trait_Kinds

Do we want a Trait, or would a better to have a Boolean function? Randy wonders if the Boolean function would handle other ideas (like **accept null**). Pascal says that we're doing the language as it is and not worrying about future compatibility.

Jean-Pierre says that traits don't scale up. But Boolean functions require multiple calls to get the needed information. Jean-Pierre prefers a Boolean array result. Tucker suggests that if we want a routine that returns all of the results, it should return a record. The different results have different names in that case. Jean-Pierre says that he'd like to iterate over the result. Others are skeptical that that would be useful, and Jean-Pierre doesn't have an example.

This smacks of premature optimization; we shouldn't worry about the performance difference between one and two calls — it will depend very much on the implementation. Pascal says it is would be very complex to define all-in-one queries.

Jean-Pierre suggests a Has_Trait query that passes in the trait. Pascal comments that this isn't as readable as a simple query.

Ed suggests that we stay as close to the existing style of ASIS as possible. So maybe we should stick with the traits here.

Tucker suggests adding syntactic queries in addition to the traits. For instance, Is_Abstract_Present, Is_Null_Exclusion_Present.

So try to update the traits if we can, and add syntactic queries for everything.

A separate SI should be created to add these syntactic queries.

Approve intent of merging SI99-0004, SI99-0005, and SI99-0008: 10-0-1. Tucker took this SI.

We suggest to use Has_Blah for the syntactic queries. Bill will take this SI.

SI99-0009-1/01 Handle new aggregate features

Shorten the title to get rid of noise text [Editor's note: This was done above, else it won't fit here.]

Tucker says that returning Nil_Element is uncomfortable; users would have compare against Nil_Element to write a box. There should something more obvious. An objection is raised that the interface already returns Nil_Element in some cases. Two wrongs don't make a right.

3.9.8 uses A_Box_Default. So we can use the word "Box" here.

The SI is suggesting changes in 17.17-17.20. That seems weird; the box replaces the expression. There should be an expression kind for box.

Jean-Pierre wonders what the subtype would be for this expression (which represents box)? Much of the time, it has one; but there are cases where there isn't a well-defined subtype. That occurs (where there are different subtypes of the same type) in Ada 95. Really? `(others => 10)` if the components have different subtypes. For Ada 2005, you could really have different types: `(others => <>)` could be different types. In either of these cases, the expression would not have a defined type.

The normalized version would be expanded, if there is an initializing expression. But if there is no expression, you would still get box.

So, create a new kind of expression called a Box expression; it will always have a type for a normalized aggregate, for an unnormalized aggregate, it will have one if possible.

This would be added in 3.9.17, called A_Box_Expression -- 4.3.1 after A_Named_Array_Aggregate.

It is pointed out the current standard doesn't specify which Corresponding_Expression_Type you get in `(others => 3);` if there are two possible subtypes. Jean-Pierre says that it should just return the first named subtype for an expression, since it is usually not a subtype.

Tucker will look at Corresponding_Expression_Type, as this is part of the semantic interface; it's quite possible that it will be considered an obsolete query.

Replace in the wording Nil_Element with an expression element with the expression kind A_Box_Expression.

Approve intent of SI: 10-0-1.

SI99-0010-1/01 Add Extended_Return to the list of Statement_Kinds

The summary is wrong, as is the first sentence of question. This is about extended return statements.

Pascal is annoyed by A_Return_Statement; this should be A_Simple_Return_Statement. Of course, we changed the name; it was correct for Ada 95. Pascal would like a renames (we can make the old one obsolete). Tucker suggests a subtype, but then withdraws that idea because it would be incompatible.

Tucker suggests putting the compatibility features in a child package. Randy objects, because it would be incompatible. Tucker says that an extra with clause isn't too bad. Randy still objects, people who don't use use clauses would have to do a lot more than just adding extra with clause.

What exactly do we mean by compatible here, anyway? Our definition of compatibility is that the only change required is adding **when others** to the existing code (assuming it was in case statement style); the resulting code would be forward and backward compatible.

So change the literal to A_Simple_Return_Statement, add a renaming for "compatibility with a prior version of this standard" as A_Return_Statement.

Is the name A_Return_Object_Declaration correct? Yes, this is a return object. But Tucker notes this is part of another construct, those are usually called specification rather than declaration (these are things without semicolons in them). So A_Return_Object_Specification.

That doesn't cause issues with 15.9 and 15.10; they already have specifications (15.9 from another SI).

The function Return_Object_Declaration should also be Return_Object_Specification.

There is no empty sequence of statements, so Nil_Element can only be returned if the statements are not present. That means that there is no ambiguity, and a further query is not needed.

Approve SI with changes: 10-0-1.

SI99-0011-1/01 Add A_Tagged_Incomplete_Type_Declaration to Declaration_Kinds

Should this be a trait instead? There is no tagged trait, and we don't want to start down that road.

The new literal should go after An_Incomplete_Type_Declaration, both so the subtype ranges aren't perturbed and to keep the incomplete types together.

Pascal wonders if we should rename this to An_Untagged_Incomplete_Type_Declaration. Randy comments that the full type might be tagged, so that's misleading.

We realize that tagged private types are not handled here. They are handled in the definition part. Tagged record is handled differently again. Erhard objects that there is no definition in an incomplete type. That's true, but it is also true for the private type, and the current ASIS still treats it as a definition. Private type is the closest analog to incomplete types, let's follow that.

So add An_Incomplete_Type_Definition and A_Tagged_Incomplete_Type_Definition to Definition_Type in 3.9.9, before A_Private_Type_Definition.

Add An_Incomplete_Type_Declaration to the allowed kinds in 15.8, and allow the return of the new items.

Erhard wonders about the ER and CR comments. The ER and CR markers (and others) are similar to the subheaders in the RM, and should be formatted that way in the new ASIS Standard. We should move these (they appear before the subheaders to which they apply), and eventually we could put them into an appendix. The new definition literals should go here as ER notes "no child elements".

Approve intent of SI: 10-0-2.

SI99-0012-1/01 Add support for null procedure declarations

This should be a trait; it should be same as abstract subprograms.

So add A_Null_Trait to Trait_Kinds at the end (3.9.5). Change the comment on An_Ordinary_Trait to include **null** (but be careful with null exclusions in access_definition).

Add A_Null_Default to 3.9.8 in front of A_Nil_Default. (We don't really like these names, but nothing else seems evident.)

What is the Corresponding_Body (15.27) for a null procedure? Abstract isn't mentioned specifically either. It should be; it should return Nil_Element if the subprogram has the abstract or null trait.

There is a typo in the subject: "declara{t}ions".

Jean-Pierre wonders what the actual for a normalized generic instantiation for a formal with a null default would be. You would have to create a null procedure in this case, with the correct name. But there is no name for the actual. We don't seem to have a conclusion here, but this is mainly a semantic question.

A side discussion erupts about macro expansion of generics. This shouldn't be required of implementations that support code sharing of bodies, but the definition of normalized instantiations seems to require it. We find an implementation permission function that queries whether it is supported (Generic_Macro_Expansion_Supported), but there is no definition of what it means if it returns False. That ought to be fixed somehow. Arguably, the blanket permission to return Nil_Element covers it, but that permission makes portable use of ASIS very difficult — it probably should be reduced. (And the semantic interface will have no such permission.)

Returning to the case of the default null procedure, we look at 15.45. There are permissions that seem to cover the case of implicit stuff. Tucker suggests that we need no extra words here. In any case, the normalized version is in the semantic domain, whereas the unnormalized version is in the structural domain. Randy thinks that we may need to separate the normalized version from the unnormalized version, because we'll want to return types appropriate to the semantic domain. That seems fairly heavy, but in any case it is premature.

Approve intent: 10-0-1.

SI99-0013-1/01 Add support for exceptions with string expressions

Remove extra comma from the first sentence of the wording.

The name of this function doesn't say it has anything to do with raise statement. Jean-Pierre suggests Raise_Statement_Message.

Approve SI with changes: 11-0-0.

SI99-0014-1/01 Add comments regarding A_Formal_Package_Declaration to ASIS

The first part of the question is nonsense; A_Formal_Package_Declaration_With_Box is an ASIS thing, not an Ada 2005. It should say that Ada 2005 generalizes the actual part of a generic formal package.

The proposal is confusing and surely isn't complete. "We should consider"?!?

Tucker points out that there are two declaration kinds for formal packages, that's wrong, but changes would be incompatible.

There is a suggestion that the version with box is special; all of the others would fall under the A_Formal_Package_Declaration. That's because box by itself is a special case that doesn't fall under the normal syntax.

Why not obsolete A_Formal_Package_Declaration_With_Box? That would be incompatible: Ada 95 applications would not be prepared to handle the new box returns for A_Formal_Package_Declaration. Besides, the syntax is different; while `(others => <>)` is semantically the same as `(<>)`, it surely isn't the same syntactically.

The first bullet for 17.21 in the SI is fact OK, and should be used.

The second bullet should be changed from Nil_Element to Expression_Element of kind A_Box_Expression (both places).

The third bullet doesn't say anything useful. It should be dropped. The changes for 15.45 are also dropped, there is no change for A_Formal_Package_Declaration_With_Box.

The fourth bullet essentially says we should add something like A_Box_Expression, which we did long ago. (Well, as part of another SI.) It should also be dropped.

Approve SI with changes: 9-0-2.

SI99-0015-1/01 Correct Corresponding_Type_Operators

The subject of this SI is insanely long; shorten it as above.

Tucker says that A_Record_Definition and A_Null_Record_Definition are lower level; these are used to get the components. Erhard says that the problem is that they are in the garbage list of definitions; these are all of things there that might have operators.

Randy asks what is returned if you ask for the Definition_Kind of a Type_Definition of a record type. A_Type_Definition is the answer after a lot of discussion.

Randy wonders about this function: what operators you ought to get depends on visibility. It seems like a junk function; it should be replaced by a semantic function that can be context dependent. Tucker says that this function seems to require returning every operator that ever could exist.

Tucker would like a wording change to say that all of the operators are returned, except for private types. For private types his preference is to return only those visible to the clients. Randy objects, he thinks that we should consistent and return those that are declared on the private type, even if it is in the private part (but not those on the full type).

Jean-Pierre suggests that we improve this in the semantic domain, and leave it the way it is here. Any change might be incompatible anyway. So, follow the SI except drop A_Record_Definition and A_Null_Record_Definition.

Tucker says that in the 7th paragraph of the definition, we need to drop “For limited private types,”, since Ada 95 allows redefining "=" on all types.

Approve AI with changes: 8-0-2.

ASIS Semantic Interface

ASIS does not currently have a consistent semantic model of Ada. Tucker was given an action item to create such a model. As usual, he’s done already; he shows his preliminary chart of entities, constructs, and queries which will make up an Ada semantic model for ASIS.

```
Entity:
  Program Unit
    Package
    Subprogram
    Task Unit
    Protected Unit
    Single Entry
    Entry Family
    Generic Unit
      Generic Package
      Generic Subprogram
  Object
  Value
  Exception
  Type/Subtype/Constraint

Construct:
  Declaration
  Body
  Name
  Expression
  Compilation Unit

Semantic Attributes:

  Entity
    Kind_Of_Construct -> enum
    Has_Declaration -> Boolean
      Entity_Declaration -> Declaration construct
      Entity_Identifier -> Identifier
      Is_Overloadable -> Boolean
      Enclosing_Scope -> Declarative_Region
      Enclosing_Compilation -> Compilation unit construct
      Is_Renaming -> Boolean
        Renamed_Entity -> Entity
      Is_Program_Unit -> Boolean
        Entity_Body -> Body construct
```

```
        Entity_Stub -> Body construct
        Is_Compilation_Unit -> Boolean
        Is_Library_Unit -> Boolean

Declarative_Region ->
  Enclosing_Entity -> Entity
  Formal_Part -> List of Entitities
  Visible_Part -> List of Entities
  Private_Part -> List of Entities
  Body_Part -> List of Entities

Subprogram
  Is_Abstract -> Boolean
  Is_Dispatching -> Boolean
     Associated_Tagged_Type -> Subtype
  Is_Overriding -> Boolean
     Overrides -> Set of Subprograms
  Is_Null -> Boolean
  Is_Primitive -> Boolean
     Primitive_On_Types -> Set of Subtypes
  Profile -> Optional_Return_Subtype, List of Parameter-specs,
  Is_Function -> Boolean
     Return_Subtype -> Subtype
  Is_Instance -> Boolean
     Actual_Part -> List of Associations

Package
  Is_Instance -> Boolean
      Actual_Part -> List of Associations
      Is_Formal_Package -> Boolean
  Is_Pure -> Boolean
  Is_Preelaborated -> Boolean
  Is_Remote_Call_Interface -> Boolean
  Is_Remote_Types_Package -> Boolean

Generic
   Formal_Part -> List of Entities
   Current_Instance -> Entity

Subtype
  Category -> {boolean, character, ordinary_enum, signed int,
               modular int, ordinary fixed, decimal fixed,
               float, access-to-obj, access-to-subp,
               record, record extension, array, task, protected,
               interface, private, private extension}
  Are_Of_Same_Type(Subtype) -> Boolean
  Is_Constrained -> Boolean
     Constraint -> ?
  Is_Definite -> Boolean
  Is_Derived -> Boolean
     Parent_Subtype -> Subtype (corresponding subtype?)
  Progenitors -> Set of Subtypes
  Ultimate_Ancestor(Subtype) -> Subtype
  Is_Descendant(Subtype) -> Boolean
  Is_Ancestor(Subtype) -> Boolean
  Is_Incomplete_View -> Boolean
     Complete_View -> Subtype
  Is_Partial_View -> Boolean
     Full_View -> Subtype
```

```
    Is_Limited -> Boolean
    Is_Abstract -> Boolean
    Is_Tagged -> Boolean
      Is_Classwide -> Boolean
        Root_Subtype -> Subtype
      Is_Synchronized_Tagged -> Boolean
      Is_Interface -> Boolean
    Is_Universal -> Boolean
    Is_Discrete -> Boolean
    Is_Scalar -> Boolean
      Base_Subtype -> Subtype
      Low_Bound,High_Bound -> Expression/Value/?
    Is_Elementary -> Boolean
    Has_Known_Discriminants -> Boolean
      Discriminants -> List of Discriminant_Specs
    Has_Unknown_Discriminants -> Boolean
    Is_Preelaborable -> Boolean
    Needs_Finalization -> Boolean
    Contains_Task -> Boolean
    First_Subtype -> Subtype
    Is_Formal_Type -> Boolean
    Is_Aspect_Specified(Aspect) -> Boolean
    Is_Aspect_Directly_Specified(Aspect) -> Boolean
    Is_Access -> Boolean
      Is_Access_Parameter_Subtype -> Boolean
      Static_Accessibility -> Natural
      Excludes_Null -> Boolean
      Is_Anonymous_Access -> Boolean
      Is_Access_To_Object -> Boolean
        Designated_Subtype -> Subtype
        Is_Access_To_Constant -> Boolean
        Is_Pool_Specific -> Boolean
      Is_Access_To_Subprogram -> Boolean
        Designated_Profile -> Profile, Is_Protected

  Object
    Is_Component -> Boolean
      Enclosing_Object -> Object
    Is_Constant_View -> Boolean
    Is_Aliased_View -> Boolean
    Nominal_Subtype -> Subtype
    Initial/Default_Expression -> Expression

  Expression
    Denotes_Entity -> Boolean
      Denoted_Entity -> Entity
    Expression_Subtype -> Subtype

  Name_Of_Object/Value/View_Of_Obj/Value ->
    Nominal_Subtype -> Subtype
    Is_Constant_View -> Boolean
    Is_Aliased_View -> Boolean
```

Tucker wonders whether “object” should be *any* object, or just stand-alone object. He thinks that we need something that represents any sort of object. Steve Baird asks if that is the static view of an object. Tucker says yes; perhaps this should be called Object_Name.

But we also need to handle names that denote values. So perhaps Object/Value Name, and then Is_Object => Boolean. This is anything that has a type, so maybe Typed_Name. "Name" seems too syntactic. Tucker suggests "View".

So replace the latter part of the above with:

```
Typed_View
  Nominal_Subtype -> Subtype
  Is_Component_View -> Boolean
    Immediately_Enclosing_Object -> Typed_View
  Is_Constant_View -> Boolean
  Is_Aliased_View -> Boolean
  Is_Object_View -> Boolean (else it is a value)

Callable_View
  Profile_of_View -> Profile
     (Convention, List of parameter specs (names, defaults), Return_Type)

Subtype_View
  Is_Partial_View -> Boolean
    Full_View -> Subtype_View
  Is_Incomplete_View -> Boolean
    Complete_View -> Subtype_View
  Category -> enum
    (Character, Boolean, ordinary enum, signed int, mod int, ord. fixed, dec. fixed,
     Float, acc-to-obj, acc-to-subp, record, record ext, private, private ext, array, task,
     Prot, interface)
  Is_Tagged -> Boolean
     Is_Classwide -> Boolean
        Specific_Type -> Subtype_View
     Is_Specific -> Boolean
        Classwide_Type -> Subtype_View
     Is_Interface -> Boolean
     Is_Synchronized_Tagged -> Boolean
  Is_Abstract -> Boolean
```

Ed asks how you get back to the structural domain. (The structural domain means the current syntax-oriented ASIS interface.) Tucker says that the Entity_Declaration gets back to the structural domain. Which may not work if there isn't an explicit declaration (classic ASIS allows but does not require the creation of implicit declarations). We probably need something similar for Typed_View: Expression_Denoting_View -> expression construct. Similarly for Callable_View and Subtype_View.

Is this worthwhile? Much discussion ensues. Existing tools seem to be mostly syntax based. Steve Michell says that might be because the existing ASIS doesn't support semantic analysis very well. This new interface might enable a whole new class of ASIS applications.

Tucker will draft an SI with a more detailed proposal.

Detailed Review of Ada 2005 regular AIs

AI05-0007-1/01 Stream 'Read and private scalar types

Pascal says that the rule enforced depends on the view at the point of the attribute_definition_clause. Randy says that's a good model, but it doesn't seem to follow from the standard.

Randy is worried that with this model, calls to stream attributes will check the range of the "formal" (that is, the specification of the attribute as determined by the view at the point of the call), and call a routine that expects only to

be called with the first subtype. For example, if the call is in the body of the defining package, the attribute would be defined to work on 'Base, but the actual routine would only work on the first subtype. It seems like some checking would get missed.

Steve Baird worries that streaming 'Base might fail in this case. Tucker says that that is already true, if the Stream_Size is defined.

Tucker would like to allow T as well as T'Base. It seems to help users; this is a common mistake (and the error message is usually cryptic). But it might cause problems with Constraint_Error being raised when 'Base is streamed. Randy comments that such problems would mostly be self-inflicted (streaming of 'Base is a bit suspicious anyway — if you want to do that, define a full-range type).

Tucker will write up this AI.

AI05-0008-1/02 General access values that might designate constrained objects

Randy tries to explain the AI. The model for designated objects was they are always constrained; but we changed that without changing the corresponding rules. The big problem is the 4.1 rule: as taken literally, it requires a constrained flag in all designated objects. The 'Access and renames cases are not a real problem, as careful reading of the wording makes clear that “is” is required, not “might”. But it would be clearer to revise the wording.

Pascal would like to define a technical term “known to be constrained” so that this text doesn’t get repeated.

Tucker suggests that we add such a definition after 3.3(23).

The use in 3.10.2(26/2) would look like:

“The view shall not be a subcomponent that depends on discriminants of a variable unless that variable is known to be constrained. In addition to the places where Legality Rules normally apply, these rules apply also in the private part of an instance of a generic unit.”

Does this occur for constant views? Yes, unfortunately, general access to constant can point to an aliased variable object.

Pascal points out that changing from variable to object is incompatible. This can happen currently (in parameters, for instance). That is covered by erroneousness in 3.7.2(4). So we don’t have to change this.

Ed comments that this isn’t very satisfactory; Ada is supposed to be the language of safety. The 'Access case is fairly nasty: if it happens, the bad access created can last forever. And cause trouble forever. Pascal decides that the incompatibility is probably better than erroneous execution.

We decide to go with the incompatibility; replace “variable” by “object”.

Now for the definition of “known to be constrained”:

“A composite object is *known to be constrained* if:

- its nominal subtype is constrained;
- its nominal subtype is indefinite;
- it is part of a stand-alone constant (including a generic formal object of mode **in**);
- it is a stand-alone variable constrained by its initial value; or
- it is a dereference of a pool-specific access type, and there is no partial view of its type that is constrained.

AARM NOTE: We do not include dereferences of general access types because they might denote stand-alone aliased unconstrained variables. That’s true even for access-to-constant types.

For the purposes of determining within a generic body whether an object is known to be constrained, a subtype is not considered indefinite if it is a descendant of an untagged generic formal derived type, nor is an access type considered pool-specific if it is a descendant of a formal access type.”

Is this view-specific? Not really, the problem is when we have a constrained view and there are other views that are unconstrained.

We bend into a discussion of 3.7.2(4). It seems that the discriminants of an **in** parameter passed by reference are not covered by that rule. (They're not usually subcomponents that depend on discriminants.) So compilers ought to assume that they can change. No one thinks that his compiler does this. Tucker says that doing this violates the entire user model (that is, an **in** parameter can be treated as a constant), so we don't want to pay extra for code that assumes that they can change.

Someone wonders if there should be a check. This doesn't seem much better from a user perspective (the user still can't trust her subtypes, now they might blow up without warning, and the check is likely to occur far away from the change that caused the problem) and there would be a distributed overhead.

Steve Baird will research this problem and propose a solution (or see if it is already covered). He'll also update the rest of the AI.

John notes a typo in the AARM note, "objects {of} the type". Tucker notes another: "can designate[d]".

Otherwise, the changes in 4.1 are OK.

The renames case should be changed similarly to the access case; there are a few extra rules that must not be lost.

Approve intent of AI: 9-0-1.

[Editor's Note: Steve also needs to try to include return objects here; see the discussion of AI05-0015.]

AI05-0009-1/01 Confirming rep. clauses and independence

Tucker would prefer that independence when using pragma Convention is implementation-defined. No, we have this new pragma to request independence; otherwise it is not independent. We don't want nor need the non-portability that comes from making this implementation-defined.

Tucker would like to see Independent on individual components for record types.

Bibb will write up this AI, including a recommended level of support.

AI05-0012-1/01 Independence and representation clauses for atomic objects

Bibb comments that he doesn't like the resolution of AI-363; he would prefer rejection of such slices. That's not possible, because of dynamic bounds and uses in generics. In any case, existing approved AIs are out of bounds.

Tucker says that pragma Pack should always work. Randy argues that this does not match the recommended level of support.

Tucker doesn't understand; he says that 13.2(6.1) provides what he wants. Randy says that 13.2(6.1) is just IA; it can't override the Recommended Level of Support (which is a requirement when Annex C is in force). Advice can only be applied after meeting all requirements.

Tucker says his model is that Pack happens last, after other representation clauses are processed. Randy says that there is nothing anywhere in the Standard or AARM that even hints at such a model. The Recommended Level of Support should be modified to take other representation clauses into account.

Bibb says that is a problem. If you add Atomic_Components, you won't find out that the packing changed. Tucker's response is that if you care, you should have used Component_Size to specify the size of the components.

Component_Size isn't quite as absolute as it appears; it says that padding is allowed unless pragma Pack is given! We seem to be getting circular.

Tucker still would like pack to be advisory for records; arrays would be less important; it could be rejected. It's hard to get packing for records, whereas using Component_Size is relatively easy.

There is more discussion about this. There is no firm conclusion.

Bibb will write up this AI.

AI05-0013-1/01 Coextensions considered harmful

Tucker says that he thinks that the rule 7.6(9.4) can just be deleted. It would have to be illegal for some other reason before the discriminants.

The problem is actually with access types; if the designated type is a Taft type or a limited with type, you cannot know whether it needs finalization (at least not until link-time).

Tucker says that we need to remember that this definition is only used for the restriction No_Nested_Finalization. So we can change that if it will help.

Tucker says that changing the restriction will help. He suggests changing the restriction to checking allocators. At that point, you have to know about the designated type (or the allocator is illegal anyway). The rule would be that the allocator cannot be used if the access type is not at library level.

As a side effect, this fixes the coextension problem.

The replacement wording for D.7(4/2) would be:

"Objects of a type that needs finalization (see 7.6) shall be declared only at library level. There are no allocators for an access type whose designated type needs finalization if the access type does not have library-level accessibility."

We have to talk about library-level accessibility in order to cover coextensions; they might be not be declared at library-level but still have library-level accessibility.

Change the title to "No_Nested_Finalization is difficult to enforce".

Pascal asks about Max_Size_In_Storage_Elements. The wording includes coextensions in the size, and those can be unknown or even recursive. Tucker says that this attribute probably will have to return Max_Int when co-extensions are possible. Randy complains that he pointed this out when this coextension wording was introduced into the standard.

Approve intent of AI: 8-0-2.

AI05-0014-1/01 Accessibility of designated objects

The problem is that the accessibility of a designated object is needed even if there isn't a dereference in the sense of section 4.

Typo: "presen[s]{c}e or absent[s]{c}e" in the discussion.

Typo: "applys" should be "applies" in the recommendation.

Approve AI with changes: 7-0-0.

AI-0015-1/01 Constant return objects

Pascal says that the syntax change is not enough; the presence of keyword **constant** doesn't trigger 3.3(15-22) or any other rules about constant views. So some wording is needed.

Add "the return object declared by an extended_return_statement with the reserved word **constant**;" after 3.3(21).

Tucker would prefer having it before 3.3(21). There are some groans.

Tucker wants to change 3.3(10). That seems out of bounds (we're not in Amendment mode where we fix everything that looks messy); it's not broken. He complains that 3.3(21) is inconsistent; Randy says then we should fix that. Pascal agrees.

Tucker proposes changes to 3.3(21):

- the return object created as the result of evaluating a function_call;
- the result of evaluating an aggregate;

Steve Baird worries that paragraph 10 and the new paragraph 21 have the same wording with different meaning.

Pascal suggests dropping "result" from 3.3(10); the result of the function is irrelevant. That is, "the return object of a function_call (or of the equivalent operator invocation — see 6.6);"

Erhard doesn't believe that 3.3(21) is clear that this is outside the call. (Access through an extended_return_statement is not necessarily constant.) Perhaps an AARM note will clarify.

Tucker suggests 3.3(10) should just be "the return object of a function"; the rest is unnecessary.

After further discussion, Tucker suggests just replacing 3.3(21) with "the object denoted by a function_call or an aggregate;"

Should return objects be stand-alone objects? Stand-alone is defined in 3.3.1(1). Tucker says this is a distributed definition; the index doesn't reflect that. (That should be fixed.) Pascal says that's weird for a function used as a component of a limited aggregate. But that is modeled with assignment that must be optimized away.

What is stand-alone used for? The renames and 'Access cases, for one. The definition of stand-alone should be added to 6.5(5.7).

Pascal says that we don't want the rules for stand-alone object to be triggered; that requires tasks to start at the wrong place, and address clauses to be supported and other nasty stuff. Tucker withdraws his suggestion.

Steve Baird needs to include these return objects in AI05-0008; we want them to be allowed in 'Access and renames even though they're not stand-alone.

Pascal says that a return object constant should be a full constant declaration. That allows them to be static and other things. We need to add that to 6.5(5.7): "If the reserved word constant appears in an extended_return_statement, then the return object is declared by a full constant declaration." Yuck. Tucker suggests: "An extended_return_statement with the reserved word constant is a full constant declaration for the return object."

Steve Baird says that we need a Legality Rule requiring an initializer for constant return objects.

Add to the end of 6.5(5): "An extended_return_statement with the reserved word constant shall include an expression."

That's it.

Do we want it? Erhard speaks against it. He thinks the examples are rather contrived. Tucker thinks many people will make the entire body be the extended return statement. This would be the one place in the language where you can't put in **constant**.

Randy says that the clean-up example seems natural; in Ada 95 it is very hard to do clean-up after creating the return object. It would be impossible in Ada 2005 for limited types without using an extended_return_statement. Once you have the extended_return_statement, it's natural to want to put **constant** on the object if it is not modified in the body code.

Approve intent of AI: 8-0-2.

Detailed Review of Ada 2005 amendment AIs

AI05-0003-1/01 Qualified expressions and names

The problem is that it isn't possible to use a qualified expression to disambiguate an overloaded function call in contexts where a name is required. Tucker had suggested a grand unification of names and expressions. That seems like a huge change for a relatively limited problem.

Jean-Pierre says that just saying that a qualification of a name is a name would be sufficient. Steve Baird wonders if conversion of a name should be a name. But then you could rename the result of a conversion, and we'd have to define what kind of view, etc. That would get messy for real value conversions:

```
G : renames Float(X); -- X is Integer.
```

You would have to thunk this; if X was assigned then G should change, and if G was assigned, X should change. The representation differences would make this ugly to implement.

Tucker suggests that we just change prefix to allow qualification. Steve Baird says that would allow unusual prefixed view calls:

```
T'(aggregate).Func
```

Steve Baird returns to Jean-Pierre's idea. He wonders about the qualification subtype check on an assignment:

```
T'(X) := 10;
```

Anything you do seems strange. Making the check before the assignment seems useless and likely to be annoying (what if X is uninitialized?); making the check after the assignment adds a whole new kind of subtype check. We probably don't want to allow assignment into qualified expressions.

For the prefix idea, qualified expression would be added to the cases allowed in a prefix.

Steve Baird (who is in fine form) suggests that in that `case String'("fjvfvd")(I)` would work. Pascal doesn't like that.

Randy suggests mixing the two ideas: a prefix allows a qualification of a name. Tucker takes that a bit further and suggests maybe we want to define qualified_name ::= subtype_mark'(name), which is only used in a prefix.

Tucker will write up the AI, low priority.

AI05-0006-1/01 Nominal subtypes for all names

Tucker will write up the AI, low priority.