Coverage for ISO/IEC 8652:2012 and subsequent corrections in ACATS 3.x and 4.x
Clauses 3.9.3 – 3.10.1

A Key to Kinds and subkinds is found on the sheet named Key. Tests new to ACATS 3.0 are shown in **bold**; ACATS 3.1 in ***bold italic***; ACATS 4.0 in **blue bold**; ACATS 4.1 in ***blue bold italic***. ACATS 4.2 in ***green bold italic***.

								Objective's			Submitted tests
Clause	Para.	Lines	Kind	Subkind	Notes	Tests	New	Priority	Objective Text	Objective notes	(will need work).
3.9.3	(1/2)		Redundant								
	(1.1/3)		Syntax		Moved from 6.1 by the Amendment. Aspect_clauses added for Ada 2012.						
	(1.2/2)	1	StaticSem	Subpart	Will be checked as part of other tests (especially legality rules).						
		2	StaticSem	Subpart	Will be checked as part of other tests (especially legality rules).						
		3	StaticSem					7	Check that a class-wide type of an abstract type is not itself abstract.	C-Test. Try declaring objects of such types via object declarations and allocators; try declaring functions returning such types. [Some existing tests have objectives that appear to cover this, but they don't]	
	(2/2)		Legality	Subpart	Other abstract type tests will check this.						
				Negative		B393002 (record type).		5	Check that an untagged type cannot be declared abstract.	B-Tests. Try deriving abstract integer, record, private, etc. types. Also, create a test like B393002 for private types and private extensions (this is a syntax test - giving "abstract" requires "tagged") - probably belongs in 7.3, though. For some reason, ACATS 2.6 has similar tests for records and generic formals, but not regular private types.	
	(3/2)	1	Definitions		"abstract subprogram"						
		2	Legality	Subpart	Abstract type C-Tests will test this.						
				Negative		B393001 (only two examples), B393005 (Example of AARM 3.9.3(3.b)).		5	Check that an explicitly declared abstract subprogram cannot be primitive for a specific non-abstract tagged type.	B-Test. Try tagged records and type extensions as well as private extensions. Try routines that are primitive because of parameters other than the first, primitive because of return types, primitive because of access parameters, or have multiple controlling parameters.	
	(4/2)		Legality	Portion	This is the lead-in for the following two rules:						
				Negative	This is a change from Ada 95.	***C393013***	All		Check that a non-abstract function with a controlling result of type T is inherited as non-abstract and does not require overriding for a null extension of T.		
					From AI05-0068-1.	***B393011***	All		Check that an abstract routine of an abstract partial view overridden by a non-abstract routine in the private part requires overriding when it is inherited if the private part is not visible where it is inherited.		
	(5/2)		Legality					7	Check that an inherited abstract subprogram remains abstract for a tagged abstract derived type.	B-Test: Check both abstract record extensions and interfaces. All we can do here is check that there is no error. Combine with first objective for 3.9.3(6/2).	

					B393005 (as part of third objective).			Check that an inherited non-abstract function with a controlling result is abstract for a tagged abstract derived type with a non-null extension.	Note: We're checking that there is no error here; this is really not testable on its own.
					B393008	All		Check that an inherited non-abstract function with a controlling access result is abstract for a tagged abstract derived type.	Note: We're checking that there is no error here; this is really not testable on its own.
							4	Check that an inherited abstract subprogram remains abstract for an untagged derived type.	B-Test: Check that it cannot be called.
(6/2)	1-2	Legality					7	Check that an inherited abstract subprogram requires overriding for a tagged non-abstract derived type.	B-Test. We have instance cases, but not ordinary ones.
					B393005 (third objective).			Check that an inherited non-abstract function with a controlling result requires overriding for a tagged non-abstract derived type with a non-null extension.	
					B393008	All		Check that an inherited non-abstract function with a controlling access result requires overriding for a tagged non-abstract derived type.	
					C393013	All		For a private extension of type T, check that an inherited non-abstract function with a controlling result does not require overriding if the full type is a null extension of T.	
					B393006 (revised to cover new cases)	All		Check that, if a non-abstract type is derived from an abstract formal private type with the generic declaration, an instantiation is rejected if primitive subprograms that require overriding are inherited by the derived type from the actual (parent) type and they are not overridden.	
	3	Legality							
(7)	1	Legality	Subpart	Any C-Test using abstract types will check this.					
							4	Check that it is illegal to call a non-dispatching abstract subprogram.	B-Test. Try calling abstract primitive subprograms of untagged types; and non-primitive subprograms of tagged types. The latter case is not very important. Note that untagged abstract subprograms cannot be resolved, and other overloadings will get priority. Also see 6.4(8/2).
							7	Check that it is illegal to make a call to an abstract subprogram that is not dispatching.	B-Test. Only calls that are dynamically-tagged are legal; try statically tagged, tag-indeterminate where the tag is statically determined elsewhere, and tag-indeterminate where the tag defaults to the current one. Try ordinary tagged abstract types and interface types.
	2			Added by AI12-0413-1, an Ada 2022 Binding Interpretation.			7	Check that it is illegal for an instance specification to contain a call on equality for a formal type if the actual is an untagged record type that has an abstract equality.	B-Test. This can only be interesting for untagged records (scalar and array types use predefined equality). Don't forget to check in the private part. The body case is handled by 4.5.2(24.1/5), and is tested there.
(8/3)	1	Legality			**B393009** (interfaces), ***BC55001*** (formal interfaces), ***B393012*** (normal types)	Part	3	Check that the type of an aggregate cannot be abstract.	B-Test. Try generic formal abstract private and derived types.
					B393001, **B393009**, BC51012, BC51013, ***BC55001***	All		Check that the type of an allocator cannot be abstract.	

					B393001, B393003, **B393009**, BC51012, BC51013, ***BC55001***	All		Check that the type of an object declaration cannot be abstract.	
					BC51012, BC51013, ***BC55001, B393012, B393013***	All		Check that the type of a generic formal object of mode in cannot be abstract	
	2	Legality			B393003, B393009	Part	3	Check that the type of the target of an assignment statement cannot be abstract.	B-Test. Try generic formal abstract private and derived types, and generic formal interface types.
	3	Legality			B393001, **B393009**, BC51012, BC51013, ***BC55001***	All		Check that the type of a component cannot be abstract.	
	4	Legality			B393001, B393005 (second objective, although this appears to be a mistake), **B393009,** BC51012, BC51013, ***BC55001***	All		Check that the result type of a non-abstract function cannot be abstract.	
	5	Legality		Added by AI05-0073-1.	***BC55001, B393012, B393013***	Part	3	Check that the designated type of an access result type of a non-abstract function cannot be abstract.	B-Test. Try generic formal abstract private and derived types.
	6	Legality		Added by AI05-0203-1.	***B393012, B393013***	Part	3	Check that the type denoted by a return_subtype_indication cannot be abstract.	B-Test. Try generic formal abstract private and derived types, and generic formal interface types.
	7	Legality		Added by AI05-0073-1.	***BC55001, B393012, B393013***	Part	3	Check that the result type of a generic function cannot be abstract.	B-Test. Try generic formal abstract private and derived types.
					BC55001, B393012, B393013	Part	3	Check that the designated type of an access result type of a generic function cannot be abstract.	B-Test. Try generic formal abstract private and derived types.
(9)	1	Legality			B393001			Check that the full type of an non-abstract partial view cannot be abstract.	
	2	Legality					5	If a generic formal type is abstract, check that for each primitive subprogram of the formal that is not abstract, the corresponding primitive subprogram of the actual type shall not be abstract.	B-Test. Seems to be missing a test in ACATS 2.x, another case where one line but not the other of a rule was tested.
(10)	1	Legality			B393007, **B393010**	All		Check that abstract primitive subprograms for an abstract type declared in a visible part are not allowed in the private part (unless they are overriding an inherited subprogram).	
	2	Legality			B393007			Check that primitive functions with controlling results for a tagged type declared in a visible part are not allowed in the private part (unless they are overriding an inherited subprogram).	No need for interfaces: all primitive functions have to be abstract and are tested by the objective for line 1.
				Added by AI05-0073-1.			7	Check that primitive functions with controlling access results for a tagged type declared in a visible part are not allowed in the private part (unless they are overriding an inherited subprogram).	B-Test. Test "regular" abstract types. (No need for interfaces: all primitive functions need to be abstract and thus are covered by the objective for line 1).
(11/2)	1	Legality			B393004			Check that a generic actual subprogram cannot be abstract unless the formal is a formal abstract subprogram.	We don't need to test the formal abstract subprogram case here; most C-Tests for that feature will check it.
	2	Legality					5	Check that the prefix of the Access, Unchecked_Access, or Address attributes cannot be an abstract subprogram.	B-Test. Seems to be missing a test in ACATS 2.x, another case where one line but not the other of a rule was tested.
(11.1/2)		Dynamic	Not Testable	Can't test "no effect", because we'd have guess some incorrect effect to look for, which is impractical.					
(12)		NonNormative		A note.					

	(13)		NonNormative		Another note.						
	(14)		NonNormative		Start of an example...						
	(15)		NonNormative								
	(16)		NonNormative		...end of the example.						
3.9.4	(1/2)		Redundant								
	(2/2)		Syntax								
	(3/2)		Syntax								
	(4/2)		StaticSem	Subpart	Any test that uses an interface type in a tagged type context, tests that it is abstract, etc.						
	(5/2)		Definitions		"Limited interface", etc.						
	(6/2)	1	Redundant								
		2	Definitions		"Synchonized tagged type"						
	(7/2)		StaticSem			**C394001**	All		Check that an object of a task interface type can be the prefix of the Terminated and Callable attributes.		
						C394001	All		Check that an object of a task interface type can be passed to an abort statement.		
						CXC7005	All		Check that an object of a task interface type can be the prefix of the Identity attribute.		
				Not Testable	This can't be tested for protected types, because there are no operations of protected types/objects that can be used outside of the protected type (P'Priority can only be used within the type).						
	(8/2)		Redundant								
	(9/2)	1	Definitions		"Progenitor type" and subtype						
		2	StaticSem			**C394002** (limited, simple); **C394003** (nonlim, simple)	Part	8	Check that an interface inherits primitive subprograms from each progenitor.	C-Test. Verify by trying to call such inherited routines. Try each kind of interface. Try routines that are primitive because of parameters other than the first, primitive because of return types, primitive because of controlling access parameters, or have multiple controlling parameters.	CY30017 (task, simple), CY30018 (protected, simple), CY30019 (sync, simple), CY30031 (task, simple), CY30011 (nonlim, access)
	(10/2)		Legality	Subpart	Any interface C-Test will test.						
				Negative		**B394001**	All		Check that a primitive subprogram of an interface type cannot be a subprogram that is neither an abstract subprogram nor a null procedure.		
	(11/2)		Legality	Subpart	Any interface C-Test will test.						
				Negative		**B394A01**	All		Check that the subtype named in an interface list must denote an interface.		
	(12/2)		Legality	Subpart	Any non-limited interface C-Test will test.						
				Negative		**B394A02**	All		Check that a descendant of a non-limited interface cannot be limited.		
	(13/2)		Legality			**B394A03**	All		Check that a type derived from a task interface must be either a task interface, task type, or a private extension.		
	(14/2)		Legality			**B394A04**	All		Check that a type derived from a protected interface must be either a protected interface, protected type, or a private extension.		

	(15/2)		Legality			**B394A05**	All	Check that a type derived from a synchronized interface must be one of a task, protected, or synchronized interface, protected type, task type, or a private extension.
	(16/2)		Legality			**B394A03**	All	Check that a private extension cannot be derived from both a task interface and a protected interface.
	(17/2)		Legality	Subpart	The recheck in an instance boilerplate. The tests for the previous rules should cover this one.			
	(18/3)		Dynamic	Not Testable	Can't test "no effect", because we'd have guess some incorrect effect to look for, which is impractical. Wording changed by AI05-0070-1, but not the semantics.			
	(19/2)		NonNormative		A note.			
	(20/2)		NonNormative		Start of examples...			
	(21/2)		NonNormative					
	(22/2)		NonNormative					
	(23/2)		NonNormative					
	(24/2)		NonNormative					
	(25/2)		NonNormative					
	(26/2)		NonNormative					
	(27/2)		NonNormative					
	(28/2)		NonNormative					
	(29/2)		NonNormative					
	(30/2)		NonNormative					
	(31/2)		NonNormative					
	(32/5)		NonNormative		Modified by AI12-0312-1.			
	(33/2)		NonNormative					
	(34/2)		NonNormative					
	(35/2)		NonNormative					
	(36/2)		NonNormative		...end of examples.			
3.10	(1)	1	Definitions		"designates", "access value"			
		2	General					
	(2/2)		Syntax					
	(3)		Syntax					
	(4)		Syntax					
	(5)		Syntax					
	(5.1/2)		Syntax					
	(6/2)		Syntax					
	(7/1)	1	Definitions		"access-to-object", "access-to-subprogram"			
		2	Definitions		"storage pool"			
		3	StaticSem			C3A0015		Check that a derived access type has the same storage pool as its parent.
		4	StaticSem	Not Testable	A general description of storage pools.			
	(8)		Definitions		"pool-specific" and "general" access types			

(9/3)	1	StaticSem		"aliased view". We use Obj'Access in the test objectives to check the aliased definition (and no more).	B3A0001		Check that a reference Obj'Access is legal if Obj is declared by an aliased object_declaration or aliased component_declaration.	
					B3A0001		Check that a reference Obj'Access is legal if Obj denotes a renaming of an aliased view.	
			Negative		B3A0001		Check that a reference Obj'Access is illegal if Obj is declared by an object_declaration or component_declaration that is not aliased.	
					B3A0001		Check that a reference Obj'Access is illegal if Obj denotes a renaming of an object that is not an aliased view.	
				Added by AI05-0142-4.		5	Check that a reference Obj'Access is legal if Obj denotes an explicitly aliased parameter.	Either B or C-Test.
			Negative	Added by AI05-0142-4.		5	Check that a reference Obj'Access is illegal if Obj denotes a non-aliased parameter of an untagged type.	B-Test. Possibly covered in existing tests.
				Added by AI05-0277-1.		5	Check that a reference Obj'Access is legal if Obj denotes an aliased (extended) return object.	Either B or C-Test.
			Negative	Added by AI05-0277-1.		5	Check that a reference Obj'Access is illegal if Obj denotes a non-aliased (extended) return object.	B-Test.
	2	StaticSem			B3A0001		Check that a reference Obj'Access is legal if Obj is a dereference of an access-to-object value.	
					B3A0001		Check that a reference Obj'Access is legal if Obj is a view conversion of an aliased view.	
			Negative		B3A0001		Check that a reference Obj'Access is illegal if Obj is a value conversion, even if it is of an aliased view.	
	3	StaticSem		Text changed to use "immutably limited" by approved AI05-0053-1.	B3A0001, C3A0013		Check that a reference Obj'Access is legal if Obj designates the current instance of a limited tagged type, or a type with the reserved word limited in its full definition.	
						4	Check that a reference Obj'Access is legal if Obj designates the current instance of a task type or a protected type.	Either B or C-Test.
						6	Check that a reference Obj'Access is legal if Obj designates the current instance of a type that is immutably limited because it has an immutably limited component.	Either B or C-Test. New from AI05-0053-1.
						5	Check that a reference Obj'Access is legal if Obj designates the current instance of a type that is immutably limited for some other reason.	Either B or C-Test. Try generic cases.
						7	Check that a reference Obj'Access is legal if Obj designates a return object whose type is immutably limited.	Either B or C-Test. New from AI05-0053-1. (This is inside of an extended_return_statement.)
			Negative		B3A0001	3	Check that a reference Obj'Access is illegal if Obj designates the current instance of a non-immutably limited type.	B-Test: Nonlimited types and types that are implicitly limited are covered. Other cases??
						7	Check that a reference Obj'Access is illegal if Obj designates a return object whose type is not immutably limited.	B-Test. (This is inside of an extended_return_statement.)
	4	Definitions			B3A0001, C3A0013	4	Check that a reference Obj'Access is legal if Obj designates a formal parameter or generic formal object of a tagged type.	Either B or C-Test. Test generic formal objects.
					B3A0001	4	Check that a reference Obj'Access is illegal if Obj designates a formal parameter or generic formal object of an untagged type.	B-Test: should try generic formal objects.
			Negative		B3A0001		Check that a reference Obj'Access is illegal if Obj is a slice.	
	5	Redundant						

	6	Deleted		The restriction was deleted by Amendment 1; we test it to ensure that compilers have made the needed changes.	C3A0014			Check that if the view defined by an object declaration is aliased, has discriminants, and its nominal subtype is unconstrained, then the object is unconstrained.	
(10)	1	Definitions	Widely Used	"designated subtype"	Any access-to-object test tests this.				
	2	StaticSem	Widely Used	Any general access-to-object type uses this definition.					
	3	StaticSem		"access-to-constant type"	B3A0001 (assignment), B3A0003 (assignment in generics), B641001 (in out params)			Check that a dereference of an access-to-constant type is a constant.	
	4	Definitions	Widely Used	"access-to-variable type"					
					B3A0001			Check that an access-to-variable type cannot designate a constant.	Note: Type conversions will be tested in 4.6, renames in 8.5.1.
					C3A0016	All		Check that a constant value of an access-to-variable type can be used to modify the designated object.	
	5	StaticSem	Widely Used	Any pool-specific access-to-object type uses this definition.	Any Ada 83 access type test!				
(11)	1	Definitions	Widely Used	"designated profile"	Any access-to-subprogram test tests this.				
	2	Definitions		"calling convention"					
	3	StaticSem					5	Check that the calling convention of an ordinary access-to-subprogram type is Ada by default.	B-Test: try to give a subprogram with the wrong convention. This could be tested with 'Access, and will require some sort of substitution to provide an appropriate convention.
							5	Check that the calling convention of a protected access-to-subprogram type is "protected" by default.	B-Test: try to give a subprogram with the wrong convention. This could be tested with 'Access (of an ordinary subprogram). We also could test a similar case to the previous.
				These don't really go here, but since they combine a number of general clauses (3.10, 4.1, 6.4) they make the most sense here.	C3A0001 (functions), C3A0002 (procedures), C3A0003 (functions in generics), C3A0004, C3A0005, C3A0006, & C3A0007 (in data structures), C3A0008 & C3A0009 (passed as parameters), C3A0010 (procedures in generics), C3A0011 (procedures in child), C3A0012 (procedure subunit).		6	Check that a dereference of a named access-to-subprogram can be called and has the appropriate profile. Check that an object of an access-to-subprogram type can designate multiple subprograms.	C-Test: check cases like these for named access-to-protected subprograms. (Surely like C3A0001 and C3A0002.)
(12/2)	1	Definitions	Widely Used	"anonymous access"	Any anonymous access test tests this.				
	2	Definitions	Widely Used	"designated subtype"					
			Widely Used	"anonymous access-to-variable type"					

				"access-to-constant type" - we test this carefully because it is new. The rules checked are really defined elsewhere, but testing it here means that other uses don't need to test all combinations.	***B3A0005***	Part	4	Check that a dereference of an anonymous access-to-constant type is a constant.	C-Test: We should try that a dereference of an access-to-constant can be read at runtime. Low priority because it's unlikely to get wrong.
					B3A0006	All		Check that an anonymous access-to-variable type cannot designate a constant.	
	3	Definitions		"designated profile"	***C3A0017, C3A0018***	Part	4	Check that an anonymous access type can be an access-to-subprogram type, and that it can be called with an appropriate profile.	C-Test: access-to-protected subprogram; try in generics, especially with formal objects. Also try access-to-procedure calls (C3A0017 does not have any procedures, C3A0018 does have one). Also, try functions returning access-to-function returning access-to-function, and similar messy compositions.
(13/2)	1	Definitions	Widely Used	"**null**". Can't get much more widely used than this.					Tests C3A0025, C3A0026, C3A0027 try null for anonymous access types (which is new in Ada 2005).
	2	Redundant							
	3	StaticSem	Widely Used	Sources of access-to-object values. Seems like it should be redundant.					
	4	StaticSem	Widely Used	Sources of access-to-subprogram values. Seems like it should be redundant.					
(13.1/2)	1	Definitions	Widely Used	"excludes null", tests for the legality and checking of null-excluding types will check this.					
	2	Definitions	Subpart	Test as part of testing paragraph 15/2.					
	3	Definitions	Subpart	Test as part of testing paragraph 15/2.					
			Negative		***C3A0030***			Check that an access discriminant only is null excluding when a null exclusion is given.	
			Negative		**C460013, *C3A0030***			Check that a non-controlling access parameter is only null excluding when a null exclusion is explicitly given.	The existing tests each try one such case (a normal subprogram call), but it's hard to imagine a truly different case.
(14/1)	1	Redundant							
	2	StaticSem			B38003A (ordinary types and subtypes); B38003B (formal types); B38008B (doubly constrained subtypes)		3	Check that a constrained access subtype designating an array cannot have an index constraint.	B-Test. Check named general access types.
					B38003A (ordinary types and subtypes); B38003B (formal types); B38008B (doubly constrained subtypes)		5	Check that a constrained access subtype designating a discriminanted type cannot have a discriminant constraint.	B-Test. Check named general access types. Check discriminanted protected and task types, and private types.

		B38008A (ordinary types and subtypes, range and accuracy constraints), B38009A (ordinary access-to-access types, index and discriminant constraints), B38009D (formal access-to-access types, index and discriminant constraints)		4	Check that an access subtype designating an elementary type cannot have any constraint.	B-Test. Check named general access types. Check formal access types with range constraints.
				5	Check that an access-to-subprogram subtype cannot have any constraint.	B-Test. Check access-to-protected-subprogram as well.
				6	Check that an access_definition cannot include an index constraint or discriminant constraint.	B-Test. Try this on unconstrained arrays and discriminated records that would otherwise be legal. Note that this is disallowed by the syntax, thus this is only a medium-priority test; but we still test it because it is an obvious mistake to make (using subtype_indication instead of subtype_mark in the grammar).
		C38002A (ordinary access), C38002B (formal access)		3	Check that a unconstrained access-to-array subtype can be given an index constraint.	C-Test. Check named general access types.
		C38002A (ordinary access), C38002B (formal access)		5	Check that an unconstrained access-to-discriminanted subtype can be given a discriminant constraint.	C-Test. Check record types (both tagged and untagged), private types, private extensions, protected types, and task types. Also check named general and pool-specific access types.
(14.1/2)	Legality	***C3A0019*** (general access-to-object), ***C3A0022*** (pool-specific access-to-object), ***C3A0028*** (access-to-subprogram), ***C3A0029*** (access-to-protected-subprogram)	All		Check that a null_exclusion can be given in a subtype_indication if the subtype_mark is an access subtype that does not exclude null.	C-Test: Try access-to-subprogram types, access-to-protected subprogram types. Possibly combine with 3.10(15/2) objectives.
		C3A0019 (general access-to-object), ***C3A0022*** (pool-specific access-to-object), ***C3A0028*** (access-to-subprogram), ***C3A0029*** (access-to-protected-subprogram)	All		Check that a null_exclusion can be given in a discriminant_specification if the subtype_mark is an access subtype that does not exclude null.	
		C3A0019 (general access-to-object), ***C3A0022*** (pool-specific access-to-object), ***C3A0028*** (access-to-subprogram), ***C3A0029*** (access-to-protected-subprogram)	All		Check that a null_exclusion can be given in a parameter_specification if the subtype_mark is an access subtype that does not exclude null.	

					C3A0019 (general access-to-object), ***C3A0022*** (pool-specific access-to-object), ***C3A0028*** (access-to-subprogram), ***C3A0029*** (access-to-protected-subprogram)	All		Check that a null_exclusion can be given in a parameter_and_result_profile if the subtype_mark is an access subtype that does not exclude null.	
							6	Check that a null_exclusion can be given in a object_renaming_declaration if the subtype_mark is an access subtype that does not exclude null.	C-Test. Possibly combine with 8.5.1 objectives. Test pool-specific, named general, anonymous access-to-object, named access-to-subprogram, named access-to-protected-subprogram, anonymous access-to-subprogram, anonymous access-to-protected-subprogram
					C3A0019 (general access-to-object), ***C3A0022*** (pool-specific access-to-object), ***C3A0028*** (access-to-subprogram), ***C3A0029*** (access-to-protected-subprogram)	All		Check that a null_exclusion can be given in a formal_object_declaration if the subtype_mark is an access subtype that does not exclude null.	
			Negative		***B3A0007*** (normal source)***, B3A0008*** (generic formal source)	All		Check that a null_exclusion cannot be given in a subtype_indication if the subtype mark is not an access type or if it excludes null.	
			Negative		***B3A0007*** (normal source)***, B3A0008*** (generic formal source)	All		Check that a null_exclusion cannot be given in a discriminant_specification if the subtype mark is not an access type or if it excludes null.	
			Negative		***B3A0007*** (normal source)***, B3A0008*** (generic formal source)	All		Check that a null_exclusion cannot be given in a parameter_specification if the subtype mark is not an access type or if it excludes null.	
			Negative		***B3A0007*** (normal source)***, B3A0008*** (generic formal source)	All		Check that a null_exclusion cannot be given in a parameter_and_result_profile if the subtype mark is not an access type or if it excludes null.	
			Negative		***B3A0007*** (normal source)***, B3A0008*** (generic formal source)	All		Check that a null_exclusion cannot be given in a object_renaming_declaration if the subtype mark is not an access type or if it excludes null.	
			Negative		***B3A0007*** (normal source)***, B3A0008*** (generic formal source)	All		Check that a null_exclusion cannot be given in a formal_object_declaration if the subtype mark is not an access type or if it excludes null.	
(15/2)	1	Dynamic		The check is defined by 3.2.2(11-12).			7	Check that Constraint_Error is raised if an index constraint is not compatible with an unconstrained access-to-array subtype.	C-Test. Be sure to check named pool-specific access and general access types. (Anonymous types cannot be named, so they can't occur in a subtype_indication.)
							7	Check that Constraint_Error is raised if a discriminant constraint is not compatible with an unconstrained access-to-discriminanted subtype.	C-Test. Be sure to check named pool-specific access and general access types. Also check record types (both tagged and untagged), private types, private extensions, protected types, and task types as the designated subtype. (This last part could be a foundation.)

	2		Not Testable	This is checked by legality rules, so it shouldn't be possible for this to fail at runtime.					
	3			The "satisfies" relationship is used by memberships and type conversions.			7	Check that Constraint_Error is raised when an access object does not satisfy the index constraint of the target type of a conversion.	C-Test. Be sure to check anonymous access, pool-specific access, and general access types. Use an implicit conversion to check anonymous cases.
				In theory, this should be tested at type conversions; but it makes more sense to do it here rather than to test a hundred rules in one place.			7	Check that Constraint_Error is raised when an access object does not satisfy the discriminant constraint of the target type of a conversion.	C-Test. Be sure to check anonymous access, pool-specific access, and general access types. Use an implicit conversion to check anonymous cases.
	4				***C3A0019*** (named general access-to-object, null exclusion given at point of use), ***C3A0020*** (null excluding subtype of a named general access-to-object), ***C3A0021*** (null excluding named general access-to-object type), ***C3A20022*** (pool-specific access-to-object, null exclusion given at point of use), ***C3A0023*** (null excluding subtype of pool-specific access-to-object), ***C3A0024*** (null-excluding pool-specific access-to-object type)	Part	4	Check that Constraint_Error is raised when a null access value is converted to a null excluding subtype of a named access-to-object type.	C-Test. Check derived pool-specific access and derived named general access. Try objects, components (array, record), discriminants, parameters, return subtypes, and formal objects. Base on existing tests.
					C3A0025	All		Check that Constraint_Error is raised when a null access value is converted to a null excluding anonymous access-to-object type.	
					C3A0028 (named access-to-subprogram, null exclusion given at point of use), ***C3A0029*** (named access-to-protected-subprogram, null exclusion given at point of use)	Part	5	Check that Constraint_Error is raised when a null access value is converted to a null excluding subtype of a named access-to-subprogram type.	C-Test. Check subtype of named access-to-subprogram, null excluding named access-to-subprogram type, subtype of named access-to-protected-subprogram, named null excluding access-to-protected subprogram type, and derived named access-to-protected-subprogram. Try objects, components (array, record), discriminants, parameters, return subtypes, and formal objects.
					C3A0026 (normal subprogram), ***C3A0027*** (protected subprogram)	All		Check that Constraint_Error is raised when a null access value is converted to a null excluding anonymous access-to-subprogram type.	
(16)		Dynamic					5	Check that the subtype_indication of an access-to-object type definition is elaborated.	C-Test. Make sure that any compatibility checks are made, and any dynamic parts are evaluated.
(17)		Dynamic	Not Testable	The creation of a subtype does not have a dynamic effect, the elaboration of the subtype_mark has no dynamic effect, and we can't test for no effect.					

	(18)		NonNormative		A note.					
	(19)		NonNormative		Another note.					
	(20)		NonNormative		A third note.					
	(21)		NonNormative		Start of examples...					
	(22/2)		NonNormative							
	(23)		NonNormative							
	(24)		NonNormative							
	(25)		NonNormative							
	(26)		NonNormative		...end of examples.					
3.10.1	(1)		General							
	(2/2)		Syntax							
	(2.1/2)	1	Definitions	Subpart	"incomplete view". Tested as part of the following legality rules. (The "unconstrained" part is tested as part of paragraph 6.)					
		2	Definitions	Subpart	"tagged incomplete view". Tested as part of the following legality rules.					
		3	Redundant		The normative rule is now given in 7.5(6.1/3), changed by AI05-0178-1. Test is there.					
	(2.2/2)		StaticSem	Lead-in						
	(2.3/2)		StaticSem	Subpart	Tested as part of the following legality rules.					
	(2.4/3)		StaticSem	Subpart	Tested as part of the following legality rules. (This occurs from limited withs).					
	(2.5/3)		StaticSem	Subpart	Tested as part of the following legality rules. Modified by AI05-0208-1. This is thought to just make the wording compatible with existing practice.					
	(2.6/3)		StaticSem	Subpart	Tested as part of the following legality rules. Modified by AI05-0162-1.					
	(2.7/3)		StaticSem	Subpart	Tested as part of the following legality rules. Modified by AI05-0162-1 and AI05-0208-1.					
	(3/3)	1	Legality	Widely Used	Any incomplete type would test.					
					AI05-0162-1 allows incomplete types to be completed by private types.	**B3A1003**	Part	4	Check that an incomplete type can be completed by a private type declaration or a private extension.	C-Test. We only test this because it is a change; the B-Test checks that it is allowed, so we have a low priority to test that it actually works.
				Negative		***B3A1001, B3A1002***	All		Check that an incomplete type is illegal if there is no full type that completes it.	
						B3A1003	All		Check that an incomplete type cannot be completed by another incomplete type declaration.	
						B3A1003	All		Check that an incomplete type cannot be completed by a subtype declaration.	
		2	Legality		This is listed as "redundant" in the AARM, but the note 3.10.2(3.b) makes it clear that there is nothing redundant about the package rule.			7	Check that an incomplete type given in the visible part of a package can be completed in the same visible part.	C-Test: Try packages and generic packages; try tagged incomplete and regular incomplete types. Try to combine this test with other objectives.
				Negative		***B3A1001, B3A1002***	All		Check that an incomplete type given in the visible part of a package cannot be completed in the private part or body of the package.	

			Negative	This is a normal completion rule (and is really redundant), but we'll test it here since it makes sense to be complete here.	***B3A1001, B3A1002***	All		Check that an incomplete type given in a declarative part or package cannot be completed in a more nested declarative part or package.	
	3	Legality		"Taft amendment types". This is also redundant, but since the semantics are special, we'll test it explicitly here.	C38108A, C38108B, C38108C (all normal incomplete, normal package, comp. in body), C38108D (normal incomplete, normal package, comp in body subunit)		6	Check that an incomplete type given in the private part of a package can be completed in that private part or in the package body.	C-Test: Try packages and generic packages; try tagged incomplete and regular incomplete types. Try to combine this test with other objectives.
(4/3)	1	Legality	Subpart	Tested as part of any tagged incomplete type test.					
			Negative		***B3A1004***	All		Check that a tagged incomplete type cannot be completed by an untagged type.	
					C3A1002 (with discriminants)		3	Check that a normal incomplete type can be completed by a tagged type.	C-Test. Try tagged types that do not have discriminants. Try to combine this objective with another.
				Allowed by AI05-0162-1.			5	Check that an incomplete type can be (directly) completed by a private type.	C-Test. Try both tagged and untagged types.
	2	Legality			C38104A (normal incomplete)		4	Check that an incomplete type can have a known discriminant part.	C-Test. Try tagged incomplete types. Try to combine this objective with objectives (it's not worth testing by itself).
			Negative		B38103A, B38103B, B38103C, B38103D, B38103E		4	Check that an incomplete type with a known discriminant part is illegal if the full type does not have a fully conforming discriminant part.	B-Test. Try tagged incomplete types. Copying B38103A (perhaps with a few cases from B38103C) would be enough. Try private types.
	3			Technically redundant, but probably wouldn't be tested elsewhere.	C3A1001 (normal incomplete, untagged records and PTs), C3A1002 (normal incomplete, tagged records and tasks)		5	Check that an incomplete type with unknown discriminants can be completed by any type, including a type that has discriminants.	C-Test. Try tagged incomplete and regular incomplete types. Try completing with unconstrained array types, and various definite types, as well as private types.
					C3A1001 (normal incomplete, untagged records and PTs), C3A1002 (normal incomplete, tagged records and tasks)		5	Check that an incomplete type without discriminants can be completed by a type that has discriminants.	C-Test. Try tagged incomplete types.
				Mentioned only by omission, but this has to be tested somewhere.	C38102A (int, enum, con arrays, uncon arrays, untagged records), C38102B (float), C38102C (fixed), C38102D (task), C38102E (formal discrete, int, float, fixed, array, private)		4	Check that an untagged incomplete type without discriminants can be completed by any type that does not have discriminants. (Discriminant cases are covered by another objective.)	C-Test. Try modular, decimal, protected, interface, formal access, formal modular, formal decimal, formal derived, formal interface types, private types. Best if that can be done as part of other tests.
(5/2)		Legality	Portion	This is the lead-in for other rules.					
(6/3)		Legality		AI05-0098-1 makes an insignificant change to this paragraph.			8	Check that the name of an incomplete view can be used as the subtype mark in an access-to-object definition.	C-Test. Try regular and tagged incomplete types, and incomplete types from limited views. This is the primary use of incomplete types, it should be tested throughly. But it's nearly “widely-used”, many cases likely exist in existing tests.

						7	Check that a discriminant constraint can be used when the name of an incomplete view is used as the subtype mark in an access-to-object definition.	C-Test. Try regular and tagged incomplete types, and incomplete types from limited views.
		Negative	This is technically redundant (an incomplete view doesn't have the right class for these other constraints and exclusions), but we'll test it here for completeness.	***B3A1007***	All		Check that constraints other than discriminant constraints cannot be used on the name of an incomplete view when used as the subtype mark in an access-to-object definition.	
			Added as a parenthetical remark by AI05-0098-1.	***B3A1007***	All		Check that a null exclusion cannot be used on the name of an incomplete view when used as the subtype mark in an access-to-object definition.	
				B3A1007	All		Check that constraints (other than appropriate discriminant constraints) cannot be used on an access-to-incomplete type.	
(7/2)	Legality					8	Check that the name of an incomplete view can be used to declare a subtype.	C-Test. Try regular and tagged incomplete types, and incomplete types from limited views. This is a change from Ada 95.
		Negative		***B3A1007***	All		When the name of an incomplete view is used to declare a subtype, check that any constraint or null exclusion is illegal.	
(8/2)	Legality			***C3A1003, C3A1004*** (tagged incomplete views, type declarations)	Part	7	Check that the name of an incomplete view can be used as the subtype_mark in an access_definition.	C-Test. Try regular and tagged incomplete types, and incomplete types from limited views. This is the primary use of incomplete types, it should be tested throughly. Try uses in object declarations, component declarations, and as parameters and function results.
(8.1/3)	Legality		Added by AI05-0151-1.	***C3A1003, C3A1004*** (subprograms, tagged incomplete views)	Part	7	Check that the name of an incomplete view can be used in the profile of subprograms, access-to-subprogram types, and anonymous access-to-subprograms used other than in bodies.	C-Test. Try in procedures, functions, and named and anonymous access-to-subprograms. Try tagged and untagged incomplete types and tagged and untagged incomplete views imported from limited views. The limited view case for normal subprograms (the reason for the rule change) is critically important; the other cases less so.
(8.2/3)	Legality		Added by AI05-0203-1.	***CC51010*** (tagged incomplete views), ***CC51011*** (tagged incomplete types)	Part	4	Check that the name of an incomplete view can be used as the actual parameter in an instance corresponding to a formal incomplete type.	C-Test. Still need to try untagged incomplete types, untagged incomplete views from limited views, and incomplete formal types. It's hard to think of usage cases for these, thus the low priority.
(8.3/2)	Legality	Portion	This is the lead-in for the following rules. Careful, the paragraph number was changed by AI05-0151-1 and AI05-0213-1.					

(8.4/3)	Legality		Careful, the paragraph number and contents were changed by AI05-0151-1.	***B3A1A01*** (incomplete views), ***C3A1003, C3A1004*** (subprograms, incomplete views)	Part	4	Check that the name of a tagged incomplete view can be used as the subtype_mark of a parameter in a subprogram_body, entry_body, or accept_statement.	C-Test. Try in procedures, functions, and named and anonymous access-to-subprograms, as well as entry_bodies and accept_statements. Try tagged incomplete types and tagged incomplete views imported from limited views. Existence testing in the B-Test, subprograms only in the C-Test, only for incomplete views from limited views. The limited view case is sort-of important, incomplete types is meh.
		Negative		***B3A1A01*** (incomplete views), **B3A1006** (incomplete types)	All		Check that the name of an untagged incomplete view cannot be used as the subtype_mark of a parameter in a subprogram_body, entry_body, or accept_statement.	
		Negative		***B3A1A01*** (incomplete views), **B3A1006** (incomplete types)	All		Check that the name of a tagged incomplete view cannot be used as the subtype_mark of the result of a function body.	
		Negative		***B3A1A01*** (incomplete views), **B3A1006** (incomplete types)	All		Check that the name of an untagged incomplete view cannot be used as the subtype_mark of the result of a function body.	
(9/2)	Legality		Note: This is also allowed for untagged incomplete types as an obsolescent feature, so we don't test illegal cases.	***B3A1A04, C3A1003*** (parameters, limited views)	Part	6	Check that the name of a tagged incomplete view can be used as the prefix of 'Class when that is used in a context allowed for a tagged incomplete view.	C-Test. Try tagged incomplete types and tagged incomplete views imported from limited views. Try as the parameter type in a formal_part, as the designated subtype in named and anonymous access types (including those used as parameters), and in a subtype_declaration. Note: Existence testing in the B-Test, parameters of limited views in C-Test (C3A1004 has similar cases).
		Negative		***B3A1A04*** (incomplete views), **B3A1006** (incomplete type, function result)	All		Check that the name of a tagged incomplete view cannot be used as the prefix of the Class attribute used in a context that does not allow the use of a tagged incomplete view.	There is only one test of tagged incomplete types in B3A1006, but since any such case violates freezing rules or indefinite type rules as well as this rule, it's not worth testing further.
(9.1/2)	Deleted		Deleted by AI05-0151-1.					
(9.2/3)	Deleted		Deleted by AI05-0151-1.					
(9.3/2)	Legality	Widely used	Modified by AI05-0151-1; essentially moved 9.2 here. The legal case is the normal case, not worth testing in general.					
		Negative	Note that subprograms using imported incomplete views cannot be primitive for the imported type, so this rule does not apply to them. Thus we only need to test incomplete types.	***B3A1005***	All		Check that if a use of an incomplete type T is part of the declaration of a primitive subprogram of T, and T is given in the private part of package P, T cannot be completed in the body of P.	
(9.4/2)	Legality		Note: We covered parameter subtypes and function result subtypes under 8.2/2, above.	***B3A1A02***	All		Check that the name of an incomplete view cannot be used in the subtype_indication of an object declaration.	
				B3A1A02	All		Check that the name of an incomplete view cannot be used in the subtype_indication of a component declaration.	
				B3A1A02	All		Check that the name of an incomplete view cannot be used in the subtype_indication of the rename of an object.	
				B3A1A05	All		Check that the name of an incomplete view cannot be used in the subtype_indication of a generic formal object	

					B38105B (normal incomplete type, formal private types [inc. limited])		6	Check that the name of an incomplete view cannot be used as the actual type for a generic formal type other than a formal incomplete type	B-Test. Try normal and tagged incomplete types; and incomplete views imported from limited views. Try formal private types, and types where the full type would match.
					B3A1A02	All		Check that the name of an incomplete view cannot be used in an allocator	
					B3A1A03	All		Check that the name of an incomplete view cannot be used in a use_type_clause.	
(10/3)		Legality			***B3A1009*** (incomplete views), ***B3A1010*** (formal incomplete types)	All		Check that the name of a parameter that has an incomplete view cannot be used as a prefix.	
					B3A1008 (incomplete types), ***B3A1009*** (incomplete views), ***B3A1010*** (formal incomplete types)	All		Check that a dereference of an access-to-incomplete type cannot be used as a prefix.	
			Negative	If any other ways to use a dereference of an access-to-tagged-incomplete type can the thought up, they should also be tested.	***C3A1003*** (limited views)	Part	6	Check that a dereference of an access-to-tagged-incomplete type can be passed directly as a parameter.	C-Test. Still need a case using a tagged incomplete type (Taft—amendment type, probably, used in a child). Note that there needs to be a subprogram with the tagged incomplete parameter declared elsewhere (in another unit) in order to make the call. This is the reason that this feature exists, and it should be tested.
					C3A1004 (limited views)	Part	4	Check that a parameter of a tagged incomplete type can be passed directly as a parameter.	C-Test. Still need a case using a tagged incomplete type (Taft—amendment type, probably, used in a child) – somewhat unlikely, thus low priority. Note that there needs to be a subprogram with the tagged incomplete parameter declared elsewhere (in another unit) in order to make the call.
	2	Legality		Added by AI05-0151-1.	***B3A1A06***	All		Check that the actual parameter to a call cannot be of an untagged incomplete view.	Could try these local to a spec, but any such call has to fail an elaboration check and thus is not useful.
	3	Legality		Added by AI05-0151-1.	***B3A1A06***	All		Check that the result object of a function call cannot be of an incomplete view.	Could try these local to a spec, but any such call has to fail an elaboration check and thus is not useful.
	4	Legality		Added by AI05-0151-1.			8	Check that a prefix cannot denote a subprogram having a formal parameter or result of an incomplete view.	B-Test. Example was 'Access of a subprogram with an incomplete view parameter before the completion.
(10.1/5)		Legality		Added by AI12-0155-1, an Ada 2022 Binding Interpretation. Only possible for regular incomplete types (limited views cannot have primitive subprograms).	***B3A1011***	All		Check that the controlling parameter or controlling result cannot be an incomplete view if the call is dynamically tagged.	3.10.1(9.3/2) means that only local calls that always fail an elaboration check are possible.
(11/2)		Deleted							
(12)		Dynamic	Not Testable	We cannot test "no effect", as it would require guessing an incorrect effect to check for.					
(13)		NonNormative		A note.					
(14)		NonNormative		Start of examples...					
(15)		NonNormative							

(16)	NonNormative	
(17)	NonNormative	
(18)	NonNormative	
(19/2)	NonNormative	
(20/2)	NonNormative	
(21/2)	NonNormative	
(22)	NonNormative	
(23/5)	NonNormative	Modified by AI12-0312-1. End of examples.

	Paragraphs:	Objectives with tests:	Objectives to test:	Total objectives:	Objectives with submitted tests:
4	123	125	73	156	1

Must be tested	Objectives with Priority 10	0
	Objectives with Priority 9	0
Important to test	Objectives with Priority 8	4
	Objectives with Priority 7	16
Valuable to test	Objectives with Priority 6	8
	Objectives with Priority 5	19
Ought to be tested	Objectives with Priority 4	16
	Objectives with Priority 3	10
Worth testing	Objectives with Priority 2	0
Not worth testing	Objectives with Priority 1	0
	Total:	73

Objectives covered by new tests since ACATS 2.6	82
Completely:	63