5.2, With the callsite coercions of stdlib removed (below), we need to require that Lookup(gamma, delta, f) = fround in the float case of the return statement since it would otherwise validate to have uncoerced calls.

5.4, the numeric literal arg of a float coercion can be any (-)NumericLiteral (no constraints on range whatsoever). The reasoning here is that the fround() coercion will definitely produce a float32 from whatever (-)NumericLiteral you give it

5.5, the 'foreign import' part is missing a case that allows float32 imports

5.5, global variables should be allowed to be comma-separated

6, the asm.js module can also be the FunctionBody of a function constructor call. Also, I'm not sure if these are implied to validate or not, but Odin doesn't accept asm.js modules as the body of fat-arrow functions.

6.8.2, -0 must also be given type double

6.8.6, I think it needs to say "with tau as a subtype of one of its legal store types" since the store types will be things like intish and we want to accept int. Also it'd be good to mention in this sentence that the store types are returned by ValidateHeapAccess.

6.8.7, given the expression -0xffffffff (that's -UINT32_MAX). For symmetry with everything else, this should fail to validate: it's an int literal smaller than -2^31 and it does in Odin. However, based on the wording, it's not clear that, if you fail to validate via the first -NumericLiteral form, you get a second "chance" to validate via the generic -UnaryExpression rule (which *does* validate: 0xffffffff is an unsigned, - operates on ints. Maybe this is a convention in spec definition (there is an implicit "and only if" in "Validating a UnaryExpression node of the form ... succeeds with type signed if ...".

6.8.7, ~~ should accept double? not double

6.8.9, need to change "succeeds with type double" to "succeeds with type tau" and change first bullet to accept (sigma1, sigma2)->tau

6.10, Odin issues a validation error in the constant heap index case where n overflows. That is, where n*typedarraysize > INT32_MAX. The spec accepts any index <= UINT32_MAX, not taking into account the overflow and int32 requirement (which comes form hardware, actually).

7, There is no mention of the heap length requirements imposed when constant heap indices are used. For maximum clarity, I'd mention this requirement in both places:

In 6.10, in the constant-heap-access part, saying "Additionally, at link time, the given ArrayBuffer's byteLength must be greater than n * ElementSize.

In 7, say "The ArrayBuffer's byteLength must be greater than the largest constant heap access validated in 6.10".

7, I can't find a requirement that the value read for FFI global variables should be function objects. For other FFI global variable imports, the values must initially be primitives (otherwise we could execute code when coercing and this could be observable when we re-execute in the general interp).

7, even if this might (I don't know) be implied by requiring that field lookups resolve to data properties, it'd be good to also add "on non-proxy objects"j

8.2, + should accept (double?,double?)

9, imul takes two intish (it semantically performs ToInt32() on inputs)

9, min/max can take double? instead of double (they perform ToNumber() on inputs) and (soon, not yet in release) 'float?'. Also, they must take ‘signed’, not ‘int’.

9, Math.sqrt/ceil/floor should return 'floatish' instead of 'float'

Add Math.clz32 to stdlib (returns fixnum)

Raise lower bound of byteLength range from [12, 24) to [16, 24): http://discourse.specifiction.org/t/increase-minimum-heap-length-to-64kb

Remove callsite coercion requirement for stdlib functions: http://discourse.specifiction.org/t/drop-requirement-for-callsite-coercion-for-stdlib-callees

const global variables: http://discourse.specifiction.org/t/allow-const-global-variables

2.1.12, I'd add that 'extern' is only used to check subtyping for calls to external JS functions and never manipulated within asm.js.

5.2, Even thought Section 4 says "empty statements are ignored", you might want to explicitly say "last (non-empty) statement" in the last sentence.

5.3, I think the 'var' line can be removed as it doesn't participate in the derivation of the function type and gets checked separately by ValidateFunction

5.5 has a nice way of stating "I take delta as input, produce delta' as output". It might be good to replicate this exact form for 5.3 and 5.6

The intro para of Section 6 seems to overlap with Section 4 "Syntax"; perhaps you could just slurp all of Section 4 into the intro of Section 6 (as it is all quite relevant from the POV of validation) and remove Section 4?

6.8.7, It'd be nice to hyperlink to the Unary types table when you say "the type of op is", like you do in the binary case.

6.9, I spent a while trying to understand the difference between the two arg-checking rules. It was only the color of the ellipsis that gave me the hint that the second one was for variadic args. I think it'd be good to come up with a better annotation for variadic types (maybe hat?), otherwise it's overloading based on color and the grapheme used to draw the two ellipses :)

6.10, although you already have a hyperlink to the heap view types in the first para, I'd add hyperlinks for the text "load type" and "store type" in the bulleted lists

6.8.8, need a space between "ValidExpression" and "expr" in last sentence.

8.2, it'd be nice to add a note to the bottom of the table, like you do with the unary table, mentioning the extra cases for add/sub/mul.