WG 9 Comments on ISO/IEC WD 24772-2 Ada language vulnerabilities

*Contributions are labelled by contributor to track the flow of comments.*

[Jeff]

Section 3

I think "overloading" should be added, maybe "unit" too.

[Niklas]

3.2: The term "abnormal representation" is introduced here, but its

relationship to the two related concepts in the Ada RM, "abnormal

object" and "invalid representation", is not made clear. Perhaps

"invalid representation" is meant?

[Niklas]

3.9: Description of "atomic" speaks of "local or register copy" which is

irrelevant to "atomic" (but perhaps applicable to "volatile", and of

curse every atomic object is also volatile). Only the indivisibility of

the memory access is important for "atomic".

[Niklas]

3.10: Description of "attribute" speaks only "characteristics of a declaration", which does not seem to apply to attributes that have dynamic values (T'Callable, T'Terminated, X'Valid, P'Old).

[Niklas]

3.11: Description of "bit ordering" is wrong. It is not about "the way that bits are represented in memory within a single memory unit", but about how bits are numbered in record-representation clauses in program source code.

[Niklas]

3.20: Should the description of "default expression" be extended to cover default initial values of record components? The present description seems to cover only formal parameters.

[jeff]

3.37

language features that have been declared to be obsolescent or deprecated{,} and {which are} documented in Annex J of ISO/IEC 8652

[Jeff]

3.40

The RM says "POSSIBLY in a separate address space", though I would expect it to be the norm.

[Niklas]

Why does the description of a "partition" require separate address

spaces? There are machines -- even multi-core machines -- that use a

single address space for all processes, but still protect memory

accesses by suitable access-rights management. Should there be some

mention of the lack of compilation "dependency" of one partition on

another? And the independent execution of partitions (RM E(3)) and the

ability to stop/start partitions independently (RM E.1(13) and (14))?

[Niklas]

3.46: Include "case expression" in addition to "case statement". See

3.14.

[Jeff]

3.46

expression{ } that is part of a case statement and that determines which choice is taken in executing the case statement or evaluating the case expression; it is of {a }discrete type

[Niklas]

4: second sentence seems to be missing something; the "designed with"

part is otherwise incomplete. Compare to the first sentence of 5.1.1.

Perhaps change to "designed with the aims of".

[Niklas]

5.1.1, last sentence: Move the word "for" to the start of the sentence.

[Niklas]

5.1.6: Subtype declarations can now include static predicates, so all subtype-checks are not dynamic. However, a detail.

[Niklas]

5.1.6 bullet (a): It would be clearer to explicitly add "type" words to show that all "conversions" here are "type conversions", especially as the next-to-preceding paragraph speak of implicit (subtype) conversions.

[Niklas]

5.1.9: Might be good to note that many of these pragmas can now be

replaced by aspects in declarations.

[Niklas]

5.1.9.3: Perhaps the "Intrinsic" convention can be considered "another

language". If not, might be added as one use of this pragma.

[Jeff]

5.1.9.5

Is Discard\_Names particularly relevant to Vulnerabilities?

[Niklas]

5.1.9.6, 5.1.9.7: I suggest to replace "foreign" with "another", in uniformity with 5.1.9.3.

[Niklas]

5.1.9.13: Rather than repeat the description of "volatile", better to refer to its definition in clause 3. Same comment applies to other similar cases.

[Niklas]

5.1.12: Add the word "potentially" before the word "unsafe" in the two

cases where "unsafe" occurs in this clause without the "potentially"

qualifier. Also replace the term "risky operations" by "operations with

machine-specific effects". This section should include the unsafe

programming that can result from using

System.Address\_To\_Access\_Conversions together with System.Storage\_Units

to construct pointers to any desired addresses, even if these standard

Ada packages and functions do not have "Unchecked" in their names. Also

the use of address clauses or Address aspects to declare objects at any

desired address. Some references to these potentially unsafe practices

are already noted in clause 6.11.1 ("unchecked address taking

capabilities").

[Jeff]

5.1.10

I find this definition unclear.

[Niklas]

5.2: Table 5.1, row 5: Marking shared data as "Atomic" is not by itself

a sufficient alternative to using a protected object. Great caution

should be used when shared data is implemented with (just) Atomic, or

with Atomic and Volatile combined. A sufficient sharing protocol must be

identified and care must be taken to verify that the tasks follow this

protocol in all their accesses and in all execution scenarios.

[Niklas]

5.2: Table 5.1, row 6: The parenthesis "(and post-conditions)" seems to

be out of place grammatically. I would expect something like "Exploit

the type and subtype system of Ada, and its system of pre-conditions and

post-conditions, to express constraints on the values of parameters."

[Niklas]

5.2: Table 5.1, row 8: This advice seems applicable to other unexpected

termination cases such as exceptions in subprograms, even in sequential

programs.

[Jeff]

6.2.1 says:

"Failure to handle the exceptions raised by failed checks of dynamic subtype properties causes the execution of the whole system, a thread, or an inner nested scope to halt abruptly."

whereas 6.50.1 says:

"If no action is taken to handle the exception, the task or program where the exception occurred will terminate."

They should be more similar. I would have thought that an unhandled exception would typically halt the enclosing partition rather than the whole system/program in a multi-partition system/program. On the other hand, I think you'd be lucky for an unhandled exception just to halt a single task.

Given that task is defined in 3, I think 6.2.1 should say task not thread.

[Niklas]

6.2.2: Would it be out of place to suggest using the compile-time

features for checking physical-unit consistency and physical-unit

conversions that are available in certain Ada compilers (such as GNAT),

although these are not (yet) standardized in Ada? For example, clause

[Niklas]

6.3.2: The last paragraph is a duplicate of the last paragraph of 6.3.1.

Is this intentional?

[Niklas]

6.4.1: I am surprised by the claim that Ada specifies adherence to the

IEEE floating-point standards. No such standard is referenced in RM 1.2

(Normative References). The Ada standard has its own requirements on the

floating-point representations and precision of floating-point

operations, which may well be compatible with the IEEE standards, but I

don't know (and doubt) if that compatibility is complete.

[Niklas]

6.6.2: Same comment as for 6.2.2.

[Niklas]

6.8: If clause 6.9.1 mentions suppressing runtime checks as leading to

the "unchecked array indexing" vulnerability, clause 6.8 should also

mention such suppression as leading to the "buffer boundary violation"

vulnerability.

[Niklas]

6.9.2: Add suggestion to use static analysis and verification tools (eg.

SPARK) to ensure that indexing exceptions cannot occur.

[Niklas]

6.11.1, last paragraph: Change "leaf type" to "pointer to a specific

type in that class". The target type is not necessarily a "leaf"

("final") type.

[Niklas]

6.11.2: Question to language lawyers: Does No\_Use\_Of\_Attribute(Address)

forbid the of Address aspects in declarations? Probably it should, but

RM 13.2.1(6.2) does not say it, I think. Best to include here a

suggestion to use also the restriction

No\_Specification\_of\_Aspect(Address).

[Niklas]

6.12. Contrary to what is stated, Ada certainly does allow pointer

arithmetic by using System.Address\_To\_Access\_Converrsions together with

System.Storage\_Units, so that should be considered "unsafe programming".

See my comment on 5.1.12.

[Niklas]

6.14.1: Should this clause also mention accessibility checks, even if

these apply mainly to stack-allocated (rather than heap-allocated)

objects?

[Niklas]

6.17.1, last paragraph: The "incorrect executable" can also happen when

the two confused names have different types, but occur in a context

where the type does not matter, for example X'Address or X'Size, or in a

context where the type matters but only leads to the selection of a

different overloaded entity, for example Foo(X) can be legal for both

Integer X and Boolean X, if Foo is overloaded for both types.

[Niklas]

6.18: It is out of place to discuss inter-task data-sharing mechanisms

here ("volatile") -- it is not relevant to "dead stores", except as

stated in the first sentence. Better to reference some relevant clause.

[Niklas]

6.18.1 says that "Ada compilers exist which" detect certain errors which

the standard does not require compilers to detect. Checking physical

units is a similar compiler-specific extension.

[Niklas]

6.18.1: It seems to me that a very likely way to introduce dead stores,

by confusing similar variable names, is to intend to assign to a global

variable but by mistake assign to a local variable of the same type and

a similar (or even the same) name. For such cases the comfort offered

by the last paragraph does not apply. Perhaps 6.18.2 should also suggest

forbidding local declarations from hiding more global ones, and suggest

using compilers or static-analysis tools to check for such hiding (as in

clause 6.20.2).

[Jeff]

6.18.1

Was "Atomic" intended rather than "Volatile"? (Would anyone consider "Volatile" as sufficient?)

[Niklas]

"Atomic" can be applied only to small objects, so "Volatile" must be used for larger shared objects. As I understand the RM, for sharing a large object between tasks we must use "Volatile" for the large object, to ensure that load and store instructions are executed, and "Atomic" for a smaller "flag" object, to ensure that a memory barrier or similar memory synchronization is inserted.

I suggest that 6.18.1 should say "Atomic or Volatile".

[Niklas]

6.22.1, third paragraph: It is wrong to say that "Use of an

out-of-bounds value in relevant contexts causes an exception". Using a

variable, say of an integer subtype, in an expression of that type is

not preceded by a range check of the value. The range check is applied

to the result of the computation and depends what is done with that

result. Moreover, the compiler is allowed to assume that the value of

the variable is valid (within the bounds of the subtype) and use that

assumption to possibly optimize away the check of the result. For

example:

 type Pig\_Count is range 0 .. 20;

 P : Pig\_Count;

 N : Natural;

 ...

 N := Natural (P);

 P := P / 2;

z

A run-time constraint check is not required in the assignment to N,

because the compiler can statically detect that every valid value of

Pig\_Count is a valid Natural. However, since P is not initialized, the

resulting value of N is unpredictable, and might even represent a

negative integer.

A run-time constraint check is not required in the assignment to P,

because the compiler can statically detect that dividing a valid

Pig\_Count by 2 cannot give rise to an invalid Pig\_Count. Moreover, even

if the compiler generates a run-time range check, if the initial value

of P is invalid but happens to be such (for example, 36) that P/2 is in

the range 0..20, no exception is raised.

There are some contexts in which a range check is required on the value

of a variable before that variable is used, for example in the

"selecting expressions" in case statements and case expressions. Or at

least it is required that even an invalid value of the selecting

expression must lead to an exception or to one of the real case

branches, and not off into unknown code; see RM 5.4(10.d) and RM

13.9.1(11a). Array indexing is another; while there is no comment in RM

4.1.1 similar to the comment RM 5.4(10.d), there is one in RM

13.9.1(11a).

[Jeff]

6.24.1 1st para

But Ada does allow multiple objects to be initialised in a single declaration, as in:

 X, Y : Integer := 1;

[Niklas]

True, but the RM specifies the order of initializations by making that two-object declaration equivalent to

  X : Integer := 1;
  Y : Integer := 1;

In this case, the comma separator does imply the execution order.

[Niklas]

6.25.1, final example: "Ptr" should be "P". Or "P" should be "Ptr".

[Niklas]

6.27.1, second paragraph: It is not true, I believe, that a case

statement where the selecting expression is outside the range of the

subtype always raises Constraint\_Error, as claimed here. If the

selecting expression is an uninitialized variable of this subtype, with

an invalid value, or an expression that the compiler can statically

deduce must be in range, if all its variables are assumed to be 'Valid,

but this assumption is false, then Constraint\_Error may not be raised,

but execution should still flow into one of the real cases, at least if

there is an "others" alternative. See RM 5.4(10.d).

[Jeff]

6.27.1 middle para

I think an exception in these circumstances is not suppressible, if so it may be worth mentioning this.

[Niklas]

My understanding is that the RM does not require an exception to be raised for an out-of-range uninitialized variable used as a selecting\_expression in a case statement, but does require that one of the case branches should be entered, instead of a jump to some unpredictable code address. So if the case statement contains an "others" alternative, the compiler could generate code to direct all such unexpectedly out-of-range values to "others".

I'm not quite sure if this applies also when there is no "others" - the final sentence in RM 5.4(10d) is not quite clear to me

[Niklas]

Yes, I was too hasty in my comment, sorry. I now think both our suggestion are insufficient, because the problem is that the current sentence says "Place all data [...] in an abort-deferred region", which is already nonsense, whatever is written in the following parenthesis. What should be placed in the abort-deferred region is the access to and update of the vulnerable data.

I suggest to write it as: "Ensure that all accesses and updates to data that would be vulnerable to premature task termination are executed in abort-deferred regions (e.g., protected operations)".

[Randy]

My guess is that Jeff was thinking of the no "others" case. And that is
defined to raise Constraint\_Error -- see 5.4(13). Note that this is \*not\* a
check; the exception is not suppressible - one always gets an exception
here.

[Jeff] Yes!

AARM 5.4(13.a/5) enumerates how this can happen:

  In this case, the value fails to satisfy its (static) predicate (possible
  when the predicate is disabled), is outside the base range of its type,
  or is an invalid representation.

Essentially, if raising an exception is a problem, then one must explicitly
pre-test the expression before the case and/or have an others limb in the
case statement (or case expression, the rules are the same).

If the selecting expression is the name of an object, then one can use
'Valid following by a membership to make the check, but if it is an
expression, the only choice is to have an others ('Valid only works on
objects).

Perhaps some part of this ought to be mentioned, depending on the goal of
the topic (I didn't look at the original document).

[Niklas]

6.30.1, last point (sentinel values): Ada programmers can use sentinel

values for loops, and such programs/programmers are subject to the

vulnerability of failing to provide storage for the sentinel value.

Sentinel values are used to speed up search loops by eliminating the

check for "last index" in favour of the single check for "value found",

and that reason can certainly apply to Ada programs. The claim that this

vulnerability does not apply to Ada is wrong.

[Niklas]

6.30.2: Perhaps add guidance for sentinel loops and ensuring storage for

sentinel values.

[Niklas]

6.32.1, last sentence: Ada does not require that the value returned from

a function is assigned "to the same type variable". Ada does require

that the value is used in some way, so the value cannot be implicitly

discarded. This should be reworded, perhaps to "... the return value

shall be used by the caller in some legal way".

[Jeff]

6.34.1

Though it can call C variadic functions.

[Niklas]

6.34.1, 4th paragraph: It would be preferable to say "unintended

signature" instead of "incorrect signature".

[Niklas]

6.34.2: Why discuss calling other languages here, if 6.34.1 already

points to 6.46 for that case?

[Niklas]

6.34.2: Additional guidelines: Use automatic tools to generate Ada

interfaces for libraries written in other languages, for example tools

to translate C header files to Ada package declarations with Import

pragmas. Use the standard Interfaces packages to declare subprograms and

objects imported from other languages, or exported to other languages.

[Niklas]

6.35.2: I don't see how the No\_Reentrancy restriction applies to

recursion. I suggest to omit this part of the advice (or, if not, to add

parentheses to make it syntactically correct, at least).

[Niklas]

6.35.2: No\_Recursion is also doubtful here, because it does not imply a

requirement for the compiler to detect and report recursion; it just

makes execution erroneous if recursion in fact does happen, RM H.4(27),

which is definitely not desirable. Instead of No\_Recursion, the

guidelines should advise using static analysis tools to ensure absence

of recursion (if that is what is wanted).

[Niklas]

6.35.2: I also find it doubtful to use asynchronous transfer of control

as a time-out for run-away recursion. It is much simpler and more secure

and portable to include a recursion-depth counter in the subprogram

parameters and use it to put a limit on recursion depth (for example

raising an exception if the check fails). In many cases a sane depth

limit can be computed from the data on hand when starting the recursion,

but it is much harder (and quite unportable) to compute a corresponding

time-out value.

[Niklas]

6.36.1, 1st paragraph: The term "enclosing scope" should be clarified to

explain that exceptions can be propagated up the dynamic context (call

chain), not just up the static lexical scope nesting. Perhaps change to

"a statically or dynamically enclosing scope".

[Niklas]

6.36.2: In addition to (or in place of) the Fallback\_Handler, advise the

use of last-chance exception handlers in all tasks and in the main

subprogram.

[Niklas]

6.37.2, third bullet: In addition to advising against address overlays,

there should be advice against using the Address\_To\_Access conversion

functions to overlay objects of different types. Also replace "address

clauses" with "address clauses or the address aspect".

[Niklas]

6.37.2, last bullet: what is the meaning of the last, empty pragma

Restrictions? I suspect it is an editing error -- a placeholder left in

by mistake.

[Niklas]

6.39.2: More guidance (unless already covered in the general part): Use

the predefined container libraries when possible, instead of creating

program-specific dynamic data structures.

[Niklas]

6.44.1, 1st sentence: Change "applies" to "apply" (if there are indeed

several vulnerabilities).

[Niklas]

6.49.2: Possible additional guidance: Use automatic tools to translate

interface specifications from other languages to Ada code and vice

versa.

[Jeff]

6.52.1 There are more than 12 categories of checks, and the number is growing.

[Niklas]

6.53.1: Period missing after first occurrence of "Ada".

[Niklas]

6.54.2, second bullet: Add the phrase ", or features not needed in the

present program" to this bullet (as an alternative to "obscure

features"). However, this should be done only for restriction pragmas

that imply a check at compile time, so one should not, for example, use

No\_Recursion as an attempt to force a programmer to avoid recursion.

[Niklas]

6.56.1, last bullet: The Ada RM has no concept "abnormal

representation". There are "abnormal objects" and "invalid

representations". Perhaps the term is meant as an abbreviation of the

cases RM 13.9.1(6 and 6.1) where the returned representation does not

represent a value of the expected subtype. Moreover, the creation of

such a value does not itself lead to erroneous execution; what happens

depends on how the created value is used. See RM 13.9.1(12 and 12.e).

These are details, and nit-picks, but why not be exact, in the Ada

tradition.

[Niklas]

6.56.2, second bullet: Same comment re Atomic as for Table 5.1, row 5.

[Niklas]

6.59.1, last sentence: Any use of Task'Terminated as a condition for eg.

communicating with a task is open to race condition problems. The only

safe use is to check that a task \_is\_ terminated. It is not clear to me

how this sentence applies in this context.

[Niklas]

6.59.2: I don't see how one can catch activation failures of

library-level tasks; that handler would have to be in the environment

task, which is provided by the implementation and cannot be modified or

extended by the user. The second and third bullets are not compatible

with each other.

[Jeff]

6.62.2 "protected operation" would be better than "protected object".

[Niklas]

 I disagree; one cannot "place ... data" in a protected operation, but can place data in a protected object.

[Niklas]

 I disagree; one cannot "place ... data" in a protected operation, but can place data in a protected object.

[Jeff]

The data is placed in a protected object, but isn’t it the individual operations of a protected object that are abort deferred regions, not the whole protected object?

[Niklas]

Yes, I was too hasty in my comment, sorry. I now think both our suggestion are insufficient, because the problem is that the current sentence says "Place all data [...] in an abort-deferred region", which is already nonsense, whatever is written in the following parenthesis. What should be placed in the abort-deferred region is the access to and update of the vulnerable data.

I suggest to write it as: "Ensure that all accesses and updates to data that would be vulnerable to premature task termination are executed in abort-deferred regions (e.g., protected operations)".

[Jeff]

Thanks Niklas, that would be better.

[Niklas]

 After the quoted comments by Jeff and myself, we discussed the point in WG9 email and agreed that a good wording would be: "Ensure that all accesses and updates to data that would be vulnerable to premature task termination are executed in abort-deferred regions (e.g., protected operations)".

Please add that to the comments.

[Jeff]

8 3rd bullet:

Wouldn't "pragma Conflict\_Check\_Policy" cover this? (Not that it's likely to be implemented any time soon).

[Jeff]

8 6th bullet:

Many users have been after this for years. I think that aspect "Global => null" would cover it. Also, GNAT has "pragma Pure\_Function" for this (though I had heard that it was ignored).

Typos.

I've followed the ARG convention of {} where an insertion is required, and [] where a deletion is required.

5.1.8

types.{ }In

5.2 Number 6

"Exploit the type and subtype system of Ada to express (and post-conditions) on the values of parameters." doesn't make sense; presemably several words missing.

6.5.1 2nd para

Enumeration representation specification{s} are

6.5.1 3rd para

outside {of }the

6.11.1

(see 5.1 Language concepts)[ ]. Other

6.16

to Ada[.] as

6.17.1

Ada.{ }There

6.35.2

pragma Restriction {(}No\_Reentrancy{)};

[Niklas]

6.44.1, 1st sentence: Change "applies" to "apply" (if there are indeed

several vulnerabilities).

[Niklas]

6.49.2: Possible additional guidance: Use automatic tools to translate

interface specifications from other languages to Ada code and vice

versa.

[Niklas]

6.53.1: Period missing after first occurrence of "Ada".

[Niklas]

6.54.2, second bullet: Add the phrase ", or features not needed in the

present program" to this bullet (as an alternative to "obscure

features"). However, this should be done only for restriction pragmas

that imply a check at compile time, so one should not, for example, use

No\_Recursion as an attempt to force a programmer to avoid recursion.

[Niklas]

6.56.1, last bullet: The Ada RM has no concept "abnormal

representation". There are "abnormal objects" and "invalid

representations". Perhaps the term is meant as an abbreviation of the

cases RM 13.9.1(6 and 6.1) where the returned representation does not

represent a value of the expected subtype. Moreover, the creation of

such a value does not itself lead to erroneous execution; what happens

depends on how the created value is used. See RM 13.9.1(12 and 12.e).

These are details, and nit-picks, but why not be exact, in the Ada

tradition.

[Niklas]

6.56.2, second bullet: Same comment re Atomic as for Table 5.1, row 5.

[Niklas]

6.59.1, last sentence: Any use of Task'Terminated as a condition for eg.

communicating with a task is open to race condition problems. The only

safe use is to check that a task \_is\_ terminated. It is not clear to me

how this sentence applies in this context.

[Niklas]

6.59.2: I don't see how one can catch activation failures of

library-level tasks; that handler would have to be in the environment

task, which is provided by the implementation and cannot be modified or

extended by the user. The second and third bullets are not compatible

with each other.

[Jeff]

6.50.1 doesn't make much sense, I suggest:

If the library convention is to report {errors by means of }error codes and not by exceptions, then[ ], if the library components themselves are written in Ada, [then Ada's exception handling mechanisms ]let all called units trap any exceptions that are generated and return error [conditions]{codes} instead.

[Jeff]

6.57.1

1st sentence of 3rd para basically repeats 1st sentence of 2nd para.

[Niklas]

6.63.2, last bullet: Change "the Ada's" to "Ada's" or "the Ada".

[Niklas]

7: Really nothing? How about run-time violation of the ceiling priority

of a protected object? Or call-back from a protected object "externally"

to the same protected object, causing deadlock if not detected?

Index

This needs regenerating. Maybe I was unlucky, but every entry I checked was incorrect.

**EDITORIAL**

Wrong font:

5.2:

Last para.

6.3.2 last bullet:

First letter of "No\_Unchecked\_Conversion".

6.4.2 5th bullet:

Closing bracket of "Generic\_Elementary\_Functions)".

6.11.1:

Last para.

6.25.1 4th para:

First numeral of "100 .. 103".

6.30.1 first indented para:

"while … loop" should be in bold.

6.35.2 last bullet:

Opening bracket of "(No\_Recursion)"

6.44.1

"in" in "(“Obj in Target’Class”)"

6.56.1 3rd bullet

last letter of "abort"

6.56.2 2nd bullet:

"Atomic"

6.56.2 last bullet's first sub-bullet:

"abort"

6.57.2 3rd bullet:

"package System"

6.58.1

"pragma Restrictions (No\_Obsolescent\_Features)"

6.65.2 2nd bullet:

Access

6.65.2 3rd bullet:

Unchecked\_Access

8 penultimate bullet:

'Address

Index

The lists under Attribute, Exception and Pragma use a mixture of fonts.

[Alan Burns] The concurrency vulnerabilities should include discussion of Time.

[Stephen] Part 1 addresses time issues under “application vulnerabilities” clause 7.

Inconsistent spellings:

The American "behavior" 4 times, English "behaviour" 45 times.

"implementation defined" 10 times, "implementation-defined" 20 times (the RM uses both, in 123:322 proportions).

"user defined" once, "user-defined" 10 times (the RM uses both, in 1:107 proportions).

"floating point" 4 times, "floating-point" 16 times (the RM only uses the former).

"null-value" once, "null value" twice (the RM only uses the last).

"run-time" 28 times, "runtime" 3 times, "run time" 3 times (the RM uses all three, in 94:26:48 proportions).

"type-conversion(s)" 11 times, "type conversion" 13 times (the RM only uses the latter).

[Tuck and Jeff] words are hyphenated when used as a adjective and left unhyphenated when used “stand-alone”

[Niklas]

Index: The index is out of date or incorrect. For example, the first

entry, for "Abnormal representation", refers to page 1, but there is no

page 1. The first page, after the Roman-numbered pages, is page number

10. That page does contain a glossary entry for "abnormal

representation", so perhaps the problem is only a constant page-number

offset to account for the nine Roman-numbered pages.