User report

GNATprove User's Guide observations:

1.- In general I think that explanation order is erroneous, because on the first chapters the narrator assume that reader should have many concepts learned; some concepts even are explained on the forward chapters.

2.- Each coding section should have line numbered, like some other documentation or books, i.e., *ada for software engineers*.
More precisely in page 9, when narrator explain the GNATprove output *p.adb:4:07* ... is not to clear.

3.- Typing errors collection:
Page 12:

function **Is_**Even
function **Is_**Odd

page 23:
with **with** boundary values.

4.- Maybe is useful, to insert direct links to some non-trivial definitions; just like Wikipedia, it make reference to other pages located in wikimedia.
The concepts that personally I didn't understood and I searched on documentation was:

page 5:
the existing SPARK annotated subset of Ada

restrictions similar to the ones found in Ravenscar or RavenSPARK.

Annotations (its formal definition)

page 6:
SPARK, JML, ACSL, Spec#

page 9:
pragma Annotate

page 10:
a sub-expression

page 11:

This other type should have a base type ranging from -10 to 9
That's correct, but I never read before this important limitations, I think that no one explained it, at least no on the text books
pure logical
I think this expression is not clear enough

page 12:

Name => static_string_Expression,
-- what is the length?

aspects
ensures expression
--not found

page 13:

for **some** J in Table'Range

some
-- is a quantifier, I never used before on the for loop

for all J in Table'First

all
-- is a quantifier, I never used before on the for loop

If loop invariants are note supported yet. why do you talk about it?

page 14:

quantified expression

VC
--pedantic ... "Ada Standard", is not really clear

page 16:

worst-case interpretation of the Ada standard

I think should be more explained

page 17:

**I think that you should high-light this very important points (takle the control of the monkey)
if some code is useless to establish the subprogram's postcondition, the contract is either wrong or incomplete.
**The programmer can:
a) correct the contract
b) remove the offending portion of code
c) accept the warning

Fixing GNATprove User's Guide:

Trying to fix the GNATprove user's Guide, and specially to understand it.

I research on the web (mainly in) [1] [2] and my ada books[3] [4] all concepts that, on my own point of view, are not on the general knowledge, at least not for the inexperienced developers.

I fixed the documents :

- alfa.html
- gnattest.html
- usage.html

Each modification is writed separated in:

- GNATprove-userManual-alfa.html.txt
- GNATprove-userManual-gnattest.html.txt
- GNATprove-userManual-usage.html.txt

[1] http://www.ada-auth.org

[2] http://docs.adacore.com

[3] Ada 2005, John Barnes, Addison Wesley.

[4] Ada for Software Engineers, Ben-Ari, Springer Second Edition.

Installation:

No observations, everything is easy and works perfectly on x86 fedora 17.

GPS plugin:

Everything is easy and works well on x86 fedora 17.

Observations:

When you select a file on the left side of the project perspective, the folder tree. And then on the contextual menu you select “prove line”. It takes line:0, what does it means?, any way parameter can be changed manually.

When you select Edit -> preferences, its appear in no prove option, so you can not have access to some advanced switches like --pedantic.

Using GNATprove

It works as it's described on documentation, I think it works really good.

I didn't use GNATcheck nor GNATtest features.

Observations:

I think that you should explain better gnatprove.out cause if you are not really understand what does

functionally proved means, you can get alarmed. Or maybe you change your “coding policy rules” just keeping in mind to elevate your “in alfa percentage”. Principally because static analysis is a good thing and many people see it like an a tautological concept.

But avoiding the usage of exceptions, controlled types or access types. Just to elevate your alfa percentage, I don't see it as a good idea.

For example controlled types had promoted the usage of a type on the entire “user control”. Theoretically this was a good feature, but when you use GNATprove and you read the gnatprove.out you feel like some kind of dangerous programmer,... unbounded_stacks is an example. Comes to mind the expression, “and his (programmer) decision was washed away”

The same thing about the access features because many people speak bad about C / C++ features that permit to programmer make arithmetical operations (adding subtractions ... calculus) with memory addresses. And theoretically a good ada feature is that compiler let you use the potentiality of the address usage, limiting the potential messed up (calculating wrong the memory addresses), preventing dangling pointers with ada statements like aliased.

Same about Exceptions, at the beginning many people were proud about the fault-mechanism, that let you raises up some particular errors. Then Software engineers said that **go to** operation is really dangerous, and then it was deprecated. When deprecation was already accepted, then some people say that exceptions are same think than **go to**... so exceptions are good feature or not?

The point here is that nobody has the same point of view. I think that is better if GNATprove include its “functionally proved definition”on its output message, it will be better.

What is the best thing?

I think that gnatproove encourage a disciplined programming style, after few time coding with pre/postconditions, you know that all the contracts are specified on the ads.
And if something were wrong you have to tune the contract, without touching the implementation, at least.
Improve the reliability, by find problems earlier, after few time coding with this pre/conditions you feel more secure that your coded program is well done. I mean you take much time to write it, by one time that is finish, you are more confident that is well done.

What is easy to do?

All actions are really easy to do, you need just to execute the command.
For changing the statistics levels in alfa / not in alfa is also easy,

You just need to change the way in which you are coding. But I think that is easier when you are just testing the tool for a simple and single application like I do. But obviously will not being the same for a complex application because you have many dependencies.

What is difficult to do?

I consider that thinking in real-world pre /postconditions that are useful without modify the original requirements is not really simple. It seems easy but in practice is not at all. I think that is like exceptions, is not an obvious concept, and how do you use it, is tricky to understand.

The overflow checks, I'm not pretty sure if I really understand what does it means.

General GNATProve observations:

I think that in general the reader make a lot of assumptions, and is better if you explain well each goal.

Many times we can not see the problems on a complex system until you deploy it. This kind of problems will not be discovered by GNATProve, maybe is clear for everybody, I understood at the end.

Instead GNATprove can help to discover a bug on an early phase of the development, cause contracts pre/postconditions and contract_case can help you to write interfaces that get closer to **system requirements specification**.

Some errors will be discovered on the **coding phase** cause practically with GNATProve, you are adding more checks to your code, checks that compiler will exec.

Other pre /postconditions and contract_case will help to find problems on the single test phase and other contracts will help to find bugs on the integration phase. Compiler insert byte code checks that will be executed at run-time. Quantified expressions is a good example, you need unit tests to coverage the entire code.

The main point here is that if you find a bug on an earlier phase you will save-work programmer hours on the entire Software Development (V diagram). And even more you review the System Requirements that are coded on the interfaces.

GNATProve don't work alone, if you want to ensure reliability system quality, you have to use also other ada tools, GNATcheck, GNATtest and aunit features.

General Ada observations:

In my opinion adopt the ada way to develop software is really hard. Specially when you comes from the classical teaching C based methods.

If it is the case, you have bad customs, like:

> Have not obligation to coding using best practices.
>
> You keep coding also if you don't understood by entire the specific current used feature, you usually search some example on the web and you adapt it for your current particular development characteristics.
>
> You don't need to understand the complex Theory behind programming language to code a program. You just keep moving and at the end development goes on, even if it has bugs.
>
> **When you have some errors** (also problems or doubts), the compiler give you the error causes. If you don't understand the message you copy/paste on a web search engine. And that's all, you have many posts explaining the problem.

For my particular experience learning Ada is tied to expend many more hours per day to coding/learning, and at the end you saw that you wrote just few lines... for me this is some kind of frustration language-feature.

You have to read a lot; if you read more, you “better understand” that you need to read more, because each time you understand a particular concept A, that concept “gives you” the necessity to understand other (ignored) concept B definition... and you keep walking. At least now I didn't met the final ignored concept, and when you get back to an already read it concept you “read it another time” and you understand that the first time you read it, you miss-understand a tiny part that now you know is important.

When you have some errors, the compiler give you the message. But you are alone, no web search engine can save you. If you are lucky, you can ask to your mentor (thank you for always being available), or your University professor you find somebody on IRC channel that explains you. But if the problem is a depth feature, you have to read! but where to begin?

Ada gem is ever the best place to begin, but ada-gem web site also contain scientific journal articles... understanding a journal article is not easy at all.

In this point you asked you, where did I arrived here?, I just had a compiling/running error. The scientific journals are for the scientific researchers, not for developers!, right?

The debugger, GPS has integrated the debugger but is not really easy to use it, is not really clear where are you, at least I get lost really often, you don't know if it show updated information or not.... at the end I prefer the original gnu command lines.

When you talk about ada experts, they look everything really simple... easy to learn/understand/do.

But easy for whom?

> For the compiler implementor
>
> For the guys who writes the ada text books
>
> For the people to have to write the formal logic demonstration for some specific feature
>
> For people that is learning to code, and ada is their first programming language

Unfortunately (or fortunately) ada experts lives on the clean environment (at least it seems), maybe each of the above responses are simple true.

For my point of view, real world is dirty, each developer have pathological characteristics, some worse than others. I think that ada promoters should have that in mind.

Fixing the Ada way

I think that the most important thing is documentation. Seriously is really exhausting to keep reading for hours and then simply lost the focus.

I think that hi-lite, can begin to a new way to make ada documentation. You can insert links to the RM, when you make reference to “a not obvious concepts”. Also you can high-light a short description on the mouse-over link; the title property of the html link tag.

With this simple policy you will save time to the inexperienced developers. And you are hiding obvious concepts, to the experimented developers. Experts won't be obligated to read a too verbose documentation explains.

The other option is to make an script that generates documentation when you download the product. The default script option should be verbose, then experts can change some flag option to generate “documentation jewels pure” / “documentation garbage free”.

Changing of internship subject:

I know that I'm not the best developer, but also I know that I'm not the worst one. Maybe just this internship oversight me. Or maybe the problem is that nobody can pretend to contract an ada developer for few months; just like others programming languages developers (at least not if you aren't an ada expert), because get insight have a high timing cost. Anyway I appreciate the detail.

SoCiS internship:

At the end I'm really happy to share this summer of code with GNAT adacore members, seriously is an honor for me.

Also if my girlfriend is not happy at all; she said that I work too much, she talks on her psychologist perspective.

Anyway I hope to finish at least the cellphone partition, human_migration simulation until 28th.

Thank you again for the book.