Adding contracts to the GCC GNAT Ada standard libraries

Joffrey Huguet
• Ada and SPARK
• Context
• Adding contracts to the GCC GNAT Ada Standard libraries
  – Model global effects
  – Protect from run-time errors
  – Add complete contracts
• Related works
Ada and SPARK
Ada and SPARK - The Ada Language

- General purpose language, first released in 1983
Ada and SPARK - The Ada Language

• General purpose language, first released in 1983

• Pascal-like syntax

```
declare
    Y : Integer;
begin
    Y := 1;
end;
```
Ada and SPARK - The Ada Language

- General purpose language, first released in 1983
- Pascal-like syntax
- Strongly typed, with type constraints

```ada
declare
  Y : Integer;
begin
  Y := 1;
end;

type Small_Int is range -100 .. 100;
subtype Small_Nat is Small_Int range 0 .. 100;
type Small_Int_Arr is array (1 .. 10) of Small_Int;
```
Ada and SPARK - The Ada Language

• General purpose language, first released in 1983

• Pascal-like syntax

• Strongly typed, with type constraints

```ada
type Small_Int is range -100 .. 100;
subtype Small_Nat is Small_Int range 0 .. 100;
type Small_Int_Arr is array (1 .. 10) of Small_Int;
```

• Checks introduced at runtime

```ada
declare
  Y : Integer;
begin
  Y := 1;
end;
```

```ada
X : Small_Int := ...;
A : Small_Int_Arr := ...;
Y : Small_Nat := X;
X := A (Y);
-- range check
-- index check
```
Ada and SPARK - Contract-based programming in Ada

- Pre and postconditions for subprograms

```ada
procedure Increment (X : in out Integer) with
Pre  => X < Integer’Last,
Post => X > X’Old;
```
Ada and SPARK - Contract-based programming in Ada

• Pre and postconditions for subprograms

```ada
procedure Increment (X : in out Integer) with
  Pre  => X < Integer'Last,
  Post => X > X'Old;
```

• Strong and weak type invariants

```ada
subtype Sorted_Arr is Small_Int_Arr with
  Dynamic_Predicate =>
  (for all I in 1 .. 9 => Sorted_Arr (I) < Sorted_Arr (I + 1));
```
Ada and SPARK - Contract-based programming in Ada

• Pre and postconditions for subprograms

```ada
procedure Increment (X : in out Integer) with
  Pre  => X < Integer’Last,
  Post => X > X’Old;
```

• Strong and weak type invariants

```ada
subtype Sorted_Arr is Small_Int_Arr with
  Dynamic_Predicate =>
    (for all I in 1..9 => Sorted_Arr (I) < Sorted_Arr (I + 1));
```

• Contracts checked at runtime when assertions are enabled
SPARK:

- Verifies formally absence of run-time errors and contracts

  A : Sorted_Arr := (0, 1, 2, 3, 4, 5, 6, 7, 8, 9);
  --  predicate check proved

  X : Integer := 15;
  Increment (X);
  --  precondition proved
SPARK:

• Verifies formally absence of run-time errors and contracts
  
  A : Sorted_Arr := (0, 1, 2, 3, 4, 5, 6, 7, 8, 9);
  --  predicate check proved

  X : Integer := 15;
  Increment (X);
  --  precondition proved

• Uses deductive verification
Context
Context

- Analysis is modular; each subprogram is analyzed separately, trusting the contracts of called subprograms.
- When using unannotated subprograms, the analysis is weakened.
Context

- Analysis is modular; each subprogram is analyzed separately, trusting the contracts of called subprograms

- When using unannotated subprograms, the analysis is weakened

Example:

Using *Ada.Strings.Unbounded* and *Ada.Text_IO* in proof

```
p.adb:9:15: warning : assuming "Append" has no effect on global items
p.adb:10:21: warning : assuming "Put_Line" has no effect on global items
```

Have subprograms from these libraries really no effect on global items? Can we be more precise about their effects?
Context

- Analysis is modular; each subprogram is analyzed separately, trusting the contracts of called subprograms
- When using unannotated subprograms, the analysis is weakened

Example:

**Using Ada.Strings.Unbounded and Ada.Text_IO in proof**

```
p.adb:9:15: warning : assuming "Append" has no effect on global items
p.adb:10:21: warning : assuming "Put_Line" has no effect on global items
```

Have subprograms from these libraries really no effect on global items? Can we be more precise about their effects?

→ We need to annotate the subprograms to have correct assumptions
Model global effects of subprograms
Subprograms from Ada.Strings.Unbounded actually have no effect on global items
Subprograms from Ada.Strings.Unbounded actually have no effect on global items

```ada
procedure Append
    (Source   : in out Unbounded_String;
     New_Item : Unbounded_String)
with Global => null;
```

Adding the Global annotations removes the previous warnings
However, subprograms from Ada.Text_IO have an effect on the memory and file system, but no global variable represents the file system.
However, subprograms from Ada.Text_IO have an effect on the memory and file system, but no global variable represents the file system.

One solution: create a virtual object to represent the file system.

```ada
package Ada.Text_IO with
  Abstract_State => File_System
is
  ...
  procedure Get (File : File_Type; Item : out String) with
    Global => (In_Out => File_System);
  ...
```
However, subprograms from Ada.Text_IO have an effect on the memory and file system, but no global variable represents the file system.

One solution: create a virtual object to represent the file system.

```ada
package Ada.Text_IO with
  Abstract_State => File_System
is
...
  procedure Get (File : File_Type; Item : out String) with
    Global => (In_Out => File_System);
...
```

This way, we are able to model the effects of subprograms on the file system; the warnings are removed and the assumptions are correct.
Protect from run-time errors
The Ada Reference Manual states:

```ada
function Insert (Source  : in String;
                Before   : in Positive;
                New_Item : in String)
                      return String;
```

78/3 Propagates Index_Error if Before is not in Source'First .. Source'Last+1; otherwise, returns Source(Source'First..Before-1) & New_Item & Source(Before..Source'Last), but with lower bound 1.
The Ada Reference Manual states:

```ada
77 function Insert (Source   : in String;
                  Before    : in Positive;
                  New_Item  : in String)
           return String;

78/3 [Propagates Index_Error if Before is not in Source'First .. Source'Last+1; otherwise, returns Source(Source'First..Before-1) & New_Item & Source(Before..Source'Last), but with lower bound 1.]```

The following code fails at runtime:

```ada
1 procedure Main with SPARK_Mode is
2   Str_1 : String := "abc"; -- Source’Last = 3
3   Str_2 : String (1 .. 4);
4 begin
5   Str_2 := Insert (Str_1, 5, “d”); -- 5 is not in 1 .. 4
6 end Main;
```

But SPARK doesn’t say anything about it!
Add a precondition:

```ada
function Insert
  (Source  : String;
   Before  : Positive;
   New_Item : String) return String
with
  Pre => Before - 1 in Source'First - 1 .. Source'Last
    and then Source'Length <= Natural'Last - New_Item'Length;
```
Add a precondition:

```ada
function Insert
  (Source       : String;
   Before       : Positive;
   New_Item     : String) return String
with
  Pre => Before - 1 in Source'First - 1 .. Source'Last
   and then Source'Length <= Natural'Last - New_Item'Length;
```

Re-run the proof:

```
main.adb:5:16: medium: precondition might fail
    5 | Str_2 := Insert (Str_1, 5, "d");
       ^~~~~~~~~~~~~~~~~~~~~~~~~~
```

Now SPARK detects that the parameters don’t satisfy the precondition
Protect from run-time errors - Second example

Another extract from the Reference Manual:

6 procedure Open(File : in out File_Type;
    Mode : in File_Mode;
    Name : in String;
    Form : in String := "");
...

8 The exception Status_Error is propagated if the given file is already open. The exception Name_Error is propagated if the string given as Name does not allow the identification of an external file; in particular, this exception is propagated if no external file with the given name exists. The exception Use_Error is propagated if, for the specified mode, the external environment does not support opening for an external file with the given name (in the absence of Name_Error) and form.

12 procedure Delete(File : in out File_Type);
...

14 The exception Status_Error is propagated if the given file is not open. The exception Use_Error is propagated if deletion of the external file is not supported by the external environment.

27 function Is_Open(File : in File_Type) return Boolean;
28/3 Returns True if the file is open (that is, if it is associated with an external file); otherwise, returns False.
Let’s add preconditions...

```ada
procedure Open
  (File : in out File_Type;
   Mode : File_Mode;
   Name : String;
   Form : String := "")
with
  Pre  => not Is_Open (File),
  Global => (In_Out => File_System);

procedure Delete (File : in out File_Type) with
  Pre  => Is_Open (File),
  Global => (In_Out => File_System);
```
And try:

1 procedure Main with SPARK_Mode is
2 File_1, File_2 : File_Type;
3 begin
4   Delete (File_1); -- wrong usage; File_1 is not open
5   Open (File_2, In_File, "hello_world.txt");
6   Delete (File_2);
7 end Main;

main.adb:4:04: medium: precondition might fail, cannot prove Is_Open (File)
And try:

```ada
1 procedure Main with SPARK_Mode is
2   File_1, File_2 : File_Type;
3 begin
4   Delete (File_1); -- wrong usage; File_1 is not open
5   Open (File_2, In_File, "hello_world.txt");
6   Delete (File_2);
7 end Main;
```

Preconditions are not enough to prove the correct usage of the library:

```
main.adb:4:04: medium: precondition might fail, cannot prove Is_Open (File)
main.adb:4:12: high: "File_1" is not initialized
main.adb:5:04: medium: precondition might fail, cannot prove not Is_Open (File)
main.adb:5:10: high: "File_2" is not initialized
main.adb:6:04: medium: precondition might fail, cannot prove Is_Open (File)
```
Protect from run-time errors - Add more contracts

Let’s add more contracts:

```ada
type File_Type is limited private with
    Default_Initial_Condition => (not Is_Open (File_Type));

procedure Open
    (File : in out File_Type;
     Mode : File_Mode;
     Name : String;
     Form : String := "")
with
    Pre   => not Is_Open (File),
    Post  => Is_Open (File),
    Global => (In_Out => File_System);

procedure Delete (File : in out File_Type) with
    Pre   => Is_Open (File),
    Post  => not Is_Open (File),
    Global => (In_Out => File_System);
```
Protect from run-time errors - And try them out

And re-run the proof:

main.adb:2:04: info: initialization of "File_1" proved
main.adb:2:12: info: initialization of "File_2" proved
main.adb:4:04: medium: precondition might fail, cannot prove Is_Open (File_1)
main.adb:5:04: info: precondition proved
main.adb:6:04: info: precondition proved

Now we are able to prove when Status_Error won’t be raised at run-time.
And re-run the proof:

main.adb:2:04: info: initialization of "File_1" proved
main.adb:2:12: info: initialization of "File_2" proved
main.adb:4:04: medium: precondition might fail, cannot prove Is_Open (File)
main.adb:5:04: info: precondition proved
main.adb:6:04: info: precondition proved

Now we are able to prove when Status_Error won't be raised at run-time.

However, this is not the only error:

• Mode_Error is related to modes (In_File, Out_File, ...)
• Name_Error is raised when the file does not exist on the file system
• End_Error is raised when a file terminator is read in a procedure
• Use_Error is related to the external environment
Add complete contracts to subprograms
Add complete contracts - Going further...

Take the example with string handling again:

```ada
1 procedure Main with SPARK_Mode is
2   Str_1 : String := "abc";
3   Str_2 : String (1 .. 4);
4 begin
5   Str_2 := Insert (Str_1, 4, "d");
6   pragma Assert (Str_2 = "abcd");
7 end Main;
```

An assertion has been added after the call to verify that Str_2 is equal to "abcd" after the call.
Take the example with string handling again:

```ada
procedure Main with SPARK_Mode is
  Str_1 : String := "abc";
  Str_2 : String (1 .. 4);
begin
  Str_2 := Insert (Str_1, 4, "d");
  pragma Assert (Str_2 = "abcd");
end Main;
```

An assertion has been added after the call to verify that Str_2 is equal to "abcd" after the call.

But it is not proved:

```
main.adb:3:04: info: initialization of "Str_2" proved
main.adb:5:13: info: precondition proved
main.adb:5:13: medium: length check might fail
main.adb:6:19: medium: assertion might fail, cannot prove Str_2 = "abcd"
```
Indeed, we don’t have any information on Str after the call to Insert:

```ada
function Insert
  (Source   : String;
   Before   : Positive;
   New_Item : String) return String
with
  Pre => Before - 1 in Source'First - 1 .. Source'Last
    and then Source'Length <= Natural'Last - New_Item'Length;
```

Add complete contracts - Going further...
Indeed, we don’t have any information on Str after the call to Insert:

```ada
function Insert
  (Source : String;
   Before : Positive;
   New_Item : String) return String
with
  Pre => Before - 1 in Source'First - 1 .. Source'Last
  and then Source'Length <= Natural'Last - New_Item'Length;
```

The Reference Manual states:

77 function Insert (Source : in String;
  Before : in Positive;
  New_Item : in String)
  return String;

78/3 Propagates Index_Error if Before is not in Source'First .. Source'Last+1; otherwise, returns Source(Source'First..Before-1) & New_Item & Source(Before..Source'Last), but with lower bound 1.
We need to reflect that through a postcondition:

Post \Rightarrow

\text{Insert'\textbf{Result'}} \text{First} = 1

\text{\textbf{and then} Insert'\textbf{Result'}} \text{\textbf{Length}} = \text{Source'} \text{\textbf{Length}} + \text{New_Item'} \text{\textbf{Length}}
Add complete contracts - Add postconditions

We need to reflect that through a postcondition:

Post \Rightarrow

Insert'Result'First = 1

and then Insert'Result'Length = Source'Length + New_Item'Length

and then

Insert'Result (1 .. Before - Source'First)

= Source (Source'First .. Before - 1)
Add complete contracts - Add postconditions

We need to reflect that through a postcondition:

Post  =>

\text{Insert}'\text{Result}'\text{First} = 1
and then \text{Insert}'\text{Result}'\text{Length} = \text{Source}'\text{Length} + \text{New}_{-}\text{Item}'\text{Length}
and then
\text{Insert}'\text{Result} \ (1 \ldots \ \text{Before} - \ \text{Source}'\text{First})
= \text{Source} \ (\text{Source}'\text{First} \ldots \ \text{Before} - \ 1)
and then
\text{Insert}'\text{Result}
\ (\text{Before} - \ \text{Source}'\text{First} + 1
\ldots \ \text{Before} - \ \text{Source}'\text{First} + \text{New}_{-}\text{Item}'\text{Length})
= \text{New}_{-}\text{Item}
Add complete contracts - Add postconditions

We need to reflect that through a postcondition:

\[
\begin{align*}
\text{Post} & \Rightarrow \\
\text{Insert'\text{Result'First} = 1} & \\
\text{and then} & \text{Insert'\text{Result'Length} = Source'\text{Length} + New\_\text{Item'Length}} \\
\text{and then} & \text{Insert'\text{Result} (1 .. Before - Source'First)} \\
& = Source (Source'First .. Before - 1) \\
\text{and then} & \text{Insert'\text{Result}} \\
& (Before - Source'First + 1 \\
& .. Before - Source'First + New\_\text{Item'Length}) \\
& = New\_\text{Item} \\
\text{and then} & \text{(if Before - 1 < Source'Last}} \\
& \text{then} \\
& \text{Insert'\text{Result}} \\
& (Before - Source'First + New\_\text{Item'Length} + 1 \\
& .. Insert'\text{Result'Last}) \\
& = Source (Before .. Source'Last))
\end{align*}
\]
And now the assertion is proved:

main.adb:3:04: info: initialization of "Str_2" proved
main.adb:5:13: info: precondition proved
main.adb:5:13: info: length check proved
main.adb:6:19: info: assertion proved

The library Ada.Strings.Fixed provides different kinds of operations on Strings:

- Search subprograms
- String translations
- String transformations
- String selectors
- String constructors
Related works
Related works - Projects

On standard libraries:

- **C standard libraries:**
  - annotated header files packaged with Frama-C
  - external work on annotating header files done by GrammarTech

- **Java standard libraries:**
  - some libraries are annotated for OpenJML

- **Community participation:** annotationsforall.org
Related works - Projects

On standard libraries:

- C standard libraries:
  - annotated header files packaged with Frama-C
  - external work on annotating header files done by GrammarTech

- Java standard libraries:
  - some libraries are annotated for OpenJML

- Community participation: annotationsforall.org

On third-party libraries:

- SPARK binding of TweetNaCl and Libsodium libraries
  
github.com/isavialard/TweetNaCl_binding
  
github.com/isavialard/Libsodium_binding

- SPARK binding and partial verification of CycloneTCP
  
github.com/AdaCore/Http_Cyclone
Related works - Planned next steps

- Specifying more GCC GNAT Ada standard libraries
- Verifying a given implementation of the library
Conclusion

- There are different levels of detail
- These levels can serve for different purposes
- This is a substantial effort
Online resources

- Blogpost on annotating third-party libraries
  
  blog.adacore.com/secure-use-of-cryptographic-libraries-spark-binding-for/libsodium

- Online Ada and SPARK Courses
  
  learn.adacore.com

- Download page for the SPARK toolset
  
  adacore.com/download

- Source code of the SPARK proof tool
  
  github.com/AdaCore/spark2014