GNATstack User's Guide

Release 27.0w

AdaCore

Apr 28, 2026

CONTENTS

CHAPTER

ONE

ABOUT THIS GUIDE

This guide contains a general description of GNATstack, a framework for computing and analyzing stack usage for the different subprograms in the application and extracting worst case values prior to execution time.

CHAPTER

TWO

OPERATIONAL ENVIRONMENT

GNATstack operates on any system on which the GNAT compiler is supported, including both native and cross compilers. It requires GNAT Pro 5.04a or later.

Support for Ada, C, and C++ is included. Note that analysis of dynamic dispatching calls in C++ is limited.

CHAPTER

THREE

THE GNATSTACK TOOL

The GNATstack tool statically computes the maximum stack space required by every stack entry point (including tasks) in an application. The computed bounds can be used to ensure that sufficient space is reserved, thus guaranteeing safe execution with respect to stack usage. The tool uses a conservative analysis to deal with complexities such as subprogram recursion, while avoiding unnecessarily pessimistic estimates.

This static stack analysis tool exploits data generated by the compiler to compute worst-case stack requirements. It performs per-subprogram stack consumption computation combined with control flow analysis.

This is a static tool in the sense that its computation is based on information known at compile time. It implies that when the tool indicates that the result is accurate then the computed bound can never overflow. The tool does not rely on empirical run-time information.

The main output of the tool is the worst-case stack requirements for every entry point, together with the paths that lead to these stack needs. The list of entry points can be automatically computed (all the tasks, including the environment task) or can be specified by the user (a list of entry points or all the subprograms matching a certain regular expression).

Note that if the linker performs a partial link (this is the case of VxWorks targets, excluding RTP mode in VxWorks 6) then the main entry point will need to be specified using the *-e* option (see *Options for GNATstack*).

GNATstack also generates a *Visualization of Compiler Graphs* (VCG) file containing the complete call tree for the application, decorated with both local and accumulated stack usage information. The VCG graph description language is used in GCC for other graph outputs.

Apart from the computation of worst-case stack requirements, GNATstack can also perform four additional types of analysis:

- Indirect (including dispatching) calls. The tool indicates the number of indirect calls made from any subprogram.
- External calls. The tool displays all subprograms that are reachable from any entry point for which we do not have any stack or call graph information.
- Unbounded frames. The tool displays all subprograms that are reachable from any entry point with unbounded stack requirements. The required stack size depends on the arguments passed to the subprogram.
- Cycles. The tool can detect all cycles in the call graph. These cycles represent potential recursion and hence potentially unbounded stack consumption.

GNATstack aims at finding an upper-bound to stack usage by static analysis techniques that give accurate results if some conditions are met, such as no indirect calls, no variable-sized local variable, no recursion and full access to the complete environment (including run-time and system calls). These conditions, and in particular the last one, are not easy to meet completely using the full Ada run-time libraries. When all these conditions are not met, there is still interesting information that can be obtained about stack usage, but going all the way to find a reliable stack usage upper-bound may require a fair amount of manual work.

GNATstack provides the infrastructure to allow users to specify in a text file the missing information, such as the potential targets for indirect calls, stack requirements for externals calls, and user-defined bounds for unbounded frames.

CHAPTER

FOUR

GETTING STARTED WITH GNATSTACK

Two steps are needed to analyze stack usage: 1) generation of the basic stack consumption and call-graph information, followed by 2) analysis and report generation.

The first step is to compile the application with the required options to generate the information about per-subprogram stack consumption and call-graph information. The *-fcallgraph-info=su* compiler switch outputs this information on a per-file basis in the common VCG format:

```
$ gprbuild main_unit.adb -cargs -fcallgraph-info=su
```

The use of this compiler switch generates a `.ci` file for every compilation unit.

The compiler can also generate information about dynamically allocated objects adding the *da* marker to the *-fcallgraph-info* compiler switch (*-fcallgraph-info=su,da*). This option helps locating the objects that introduce potentially unbounded stack requirements.

In the second step, GNATstack can then parse these files to generate the complete call graph decorated with stack usage information, so it can look for the longest paths in terms of stack usage:

```
$ gnatstack *.ci
[...]
Accumulated stack usage information for entry points

main : total 376 bytes
 +-> main
 +-> main_unit
 +-> process
```

4.1 Options for GNATstack

GNATstack is configurable with respect to the amount of information that we want to obtain. The set of options available is the following:

-a
: Consider all subprograms as entry points.

-ca
: Extract all possible cycles in the call graph.

-c n
: Use *n* bytes frame for subprograms at the beginning of cycles. This can be used to increase visibility of cycles in reports.

-g
Generate internal debug information.

-aO dir
Load `.ci` files from *dir*.

-e entry_point[,...]
Name of symbols to be used as entry points for the analysis. It accepts both symbol and demangled names, and it performs a case-insensitive search.

-files=file
Take as arguments the files listed in text file *file*. Text file *file* may contain empty lines that are ignored. Each non empty line should contain the name of an existing file. Several such switches may be specified simultaneously.

-h or –help
Output a message explaining the usage of *gnatstack*.

-l n
Print the *n* subprograms requiring the biggest local stack usage. By default none will be displayed.

-o=x
Specifies the order for displaying the different call graphs. The options allowed for this qualifier are:

- ***-o=s***

  Select stack usage order (the default mode).

- ***-o=a***

  Select alphabetical order.

-p
Print all the subprograms that make up the worst-case path for every entry point (the default mode).

-pi
Print progress information (for GNAT Studio).

-q
Be quiet: do not display progress information.

-Q
Very quiet: do not display any console message except those indicating an error and the progress information.

-np
Do not print worst-case path for entry points.

-f file
Write the generated graph (VCG format) into *file*.

-r regexp
Any symbol matching the regular expression *regexp* will be considered as an entry point for the analysis. It accepts both symbol and demangled names, and it performs a case-insensitive pattern matching.

-d n
Set default stack size for unbounded (dynamic) frames to *n* bytes.

-u n
Set default stack size for unknown (external) calls to *n* bytes.

-v
Specify the amount of information to be displayed about the different subprograms. In verbose mode the full location of the subprogram will be part of the output, as well as detailed information about inaccurate data.

–version
Print version information.

-tx
Print potential targets for indirect and dispatching calls. The options allowed for this qualifier are:

- ***-ta***
  Print potential targets for both indirect and dispatching calls.
- ***-ti***
  Print potential targets for indirect calls.
- ***-td***
  Print potential targets for dispatching calls.

-Wx
Warning mode. The options allowed for this qualifier are:

- ***-Wa***
  Turn on all optional warnings.
- ***-Wc***
  Turn on warnings for cycles (recursion).
- ***-Wu***
  Turn on warnings for unbounded frames.
- ***-We***
  Turn on warnings for external calls.
- ***-Wi***
  Turn on warnings for indirect (including dispatching) calls.

-x
Generate file *stack_usage.xml* with the information in XML format.

-oc=name
Use the *objcopy* executable *name* to extract the `.ci` files from the object files. If may contain the prefix (such as *powerpc-elf-objcopy*) or the suffix (such as *objcopyppc*).

-s=section
Extract the `.ci` files from section named *section* in the object files.

-k
Keep the temporary files created by GNATstack (such as the `.ci` files extracted from the object files.

–target=tgt
Use *tgt* as program prefix used to build the application being analyzed. This value is only used to help retrieving the stack usage information for the run-time library.

–RTS=rts
Use *rts* as the run-time library used to build the application being analyzed. This value is only used to retrieve the stack usage information for the run-time library.

-Xname=val
Set project variable *name* to *val*. Several *-X* switches can be used simultaneously. If several *-X* switches specify the same *name*, only the last one is used. An external variable specified with a *-X* switch takes precedence over the value of the same name in the environment.

-Pprj
Get the GNATstack options from the specified project *prj*.

4.2 Use of Project Files

gnatstack supports the use of project files. There is an optional package in the project file for GNATstack (which is the package *Stack*) that has an unique attribute *Switches*. This attribute is a simple variable that contains a string list value representing the switches to be passed to *gnatstack*:

```
project Prj is
   package Stack is
      for Switches use ("-p");
   end Stack;
end Prj;
```

The following command will parse the project file `prj.gpr` and call *gnatstack* with the specified options:

```
$ gnatstack -P prj.gpr
```

When *gnatstack* is used with a project file, if gets the set of `.ci` files that correspond to the units that belong to the project and all projects in the project tree.

If one or more user-defined `.ci` files need to be taken into account for the analysis, the appropriate place to specify them is in the *Switches* attribute of the project file:

```
project Prj is
   package Stack is
      for Switches use (..., "user1.ci", "user2.ci");
   end Stack;
end Prj;
```

4.3 GNATstack output

The execution of GNATstack produces the maximum stack space required by every entry point (including tasks) in the application, together with the paths that lead to these stack needs:

```
$ gnatstack -P my_project.gpr

Accumulated stack usage information for entry points

main : total 408 bytes
 +-> main
 +-> main_unit
 +-> pck.process
```

The maximum stack consumption for the *main* entry point is 408 bytes, and this is a safe bound that can be trusted. The indicated call chain is the path with the longest stack consumption from all possible paths starting from *main*.

Within GNAT Studio the stack usage information is available via the *Analyze* → *Stack Analysis* → *Analyze Stack Usage* menu. For each of the subprograms the information is displayed in the form of annotations containing *local* stack requirements, which refers to a single stack frame, and *global* stack requirements, which refers to the worst case (with respect to stack usage) call chain starting from the subprogram. In the previous example, *local* is the 80 bytes of the *main* subprogram, and *global* is 408 bytes (for the call chain where *main* calls *Main_Unit* and *Main_Unit* calls *Process*):

```
--  main
--
--  Stack usage: local: 80 bytes, global: 408 bytes
```

More details about the locations of the subprograms and their local stack frames is displayed using the *-v* switch:

```
$ gnatstack -v -P my_project.gpr
[...]

Accumulated stack usage information for entry points

main : total 408 bytes
 +-> main at main:b__main_unit.adb:118:4 : 80 bytes
 +-> main_unit at Main_Unit:main_unit.adb:3:1 : 16 bytes
 +-> pck.process at Process:pck.adb:2:4,Process:pck.ads:2:14 : 312 bytes
```

In this case, the first line indicates the total (global) stack requirement of 408 bytes for the whole call chain (adding up all the frames in the call chain), and the following lines contain the individual (local) stack frames for each of the subprograms in the call chain.

When the computed stack consumption cannot be safely bound (because of cycles, unbounded frames, external calls, or indirect calls) the total stack requirements contain a "+?" indication, plus a "*" indication in the frames containing missing information. For example, if we add a dynamic variable to the previous example we would have the following GNATstack report:

```
$ gnatstack -P my_project.gpr

Worst case analysis is *not* accurate because of unbounded frames.
Use -Wa for details.

Accumulated stack usage information for entry points

main : total 576+? bytes
 +-> main
 +-> main_unit
 +-> pck.process *
```

If we add the "-Wa" and "-v" options to GNATstack we get the following additional information:

```
$  gnatstack -v -Wa -P my_project.gpr
[...]

Worst case analysis is *not* accurate because of unbounded frames.

List of reachable subprograms with dynamic unbounded frames:

  In pck.process at Process:pck.adb:2:4,Process:pck.ads:2:14

Accumulated stack usage information for entry points

main : total 576+? bytes (unbounded)
 +-> main at main:b__main_unit.adb:118:4 : 80 bytes
 +-> main_unit at Main_Unit:main_unit.adb:3:1 : 16 bytes
 +-> pck.process at Process:pck.adb:2:4,Process:pck.ads:2:14 : 480+? bytes (unbounded)
```

This analysis indicates that the computed worst case stack usage is not accurate. There are 576 bytes needed, plus the amount required by the dynamic object declared by subprogram *Process*.

Within GNAT Studio, this missing information is expressed with annotations indicating the reasons behind the inaccuracy. For the *main* entry point we would have:

```
--  main
--
--  Stack usage: local: 80 bytes, global: 576 bytes (unbounded)
```

and for the *Process* subprogram we would have:

```
--  pck.process
--
--  Stack usage: local: 480 bytes (unbounded), global: 480 bytes (unbounded)
```

4.4 Information in Non-Loadable Section

The `.ci` file generated for every compilation unit can be written into the generated object as a non-loadable section. For example, it can be written into a section named *.GNU.callgraph* doing:

```
$ gcc -c -fcallgraph-info=su main_unit.adb
$ objcopy --add-section .GNU.callgraph=main_unit.ci main_unit.o
$ gnatstack main_unit.o
```

This section is not downloaded into the target so it does not affect the execution. The advantage is that it may facilitate the retrieval of the required `.ci` files (we simply need the list of object files that make up the application), and it may help guaranteeing the coherence between the call-graph and stack usage information and the corresponding object files.

The linker would merge (concatenate) the `.ci` sections corresponding to the object files being linked into the final executable (including those coming from library files).

GNATstack can later extract this section from the object files and use the information for the rest of the analysis. GNATstack uses the *objcopy* tool to extract the `.ci` files.

By default, GNATstack uses the native *objcopy* executable. It can be changed using the *-oc* option (see *Options for GNATstack*):

```
$ powerpc-elf-gcc -c -fcallgraph-info=su main_unit.adb
$ powerpc-elf-objcopy --add-section .GNU.callgraph=main_unit.ci main_unit.o
$ gnatstack -oc=powerpc-elf-objcopy main_unit.o
```

The section where the `.ci` files are stored can also be changed, and GNATstack provides an option (*-s*, see *Options for GNATstack*) to specify this section:

```
$ gcc -c -fcallgraph-info=su main_unit.adb
$ objcopy --add-section my_section=main_unit.ci main_unit.o
$ gnatstack -s=my_section main_unit.o
```

CHAPTER

FIVE

ADDITIONAL CAPABILITIES

As indicated in *The GNATstack Tool*, there is a list of potential problems when computing stack requirements: indirect and external calls, unbounded frames, and recursion. This chapter provides guidance about how to address these issues.

The first possibility to address these issues is to avoid the occurrence of these events by means of using the appropriate restrictions and coding style rules. The second is to supply additional information that will help GNATstack to proceed with the analysis.

- Indirect calls.

  Determining statically the target subprogram when dereferencing an access-to-subprogram variable is not always possible. These occurrences can be forbidden through *pragma Restrictions*. The use of *No_Access_Subprograms* avoids access-to-subprogram types. Dispatching calls can also be restricted by using *No_Dispatch* (which prohibits classwide constructs), or the less restricting *No_Dispatching_Calls* (which ensures that no dispatching calls are present).

  The *-Wi* option can be used to signal all the occurrences of indirect calls in the program under analysis.

- External calls.

  These are calls to subprograms for which no stack usage or call graph information is available, introducing an unknown amount of stack usage in a call chain. It can only be avoided by providing the required information for the objects and libraries that make up the application. The *-We* option can be used to signal all the occurrences of external calls in the program under analysis.

- Unbounded frames.

  The use of dynamically sized local objects (such as those whose size depends on an input argument) makes it impossible to statically bound stack usage. The *-Wu* option can be used to signal all the occurrences of unbounded frames in the program under analysis.

- Recursion.

  Very simple cases can be detected by using *pragma Restrictions (No_Recursion)*. This restriction ensures that as part of the execution of a subprogram the same subprogram is not invoked. More complex situations require call graph analysis. *gnatcheck*, see GNAT User's Guide, can be used to detect potential recursion. GNATstack can also detect all the occurrences of cycles in the call graph using the options *-ca -Wc*.

If the occurrence of these potential problem cannot be avoided, GNATstack provides the means to supply additional stack usage and call graph information.

When GNATstack finds incomplete information, such as external and indirect calls, it generates a file (`undefined.ciu`) containing placeholders for the missing information. The format of this file is similar to the `.ci` files, but with *XXX* where the user information must be inserted.

For example, entries generated by GNATstack for external calls have the following pattern:

```
node: { title: "ext_proc" label: "Ext_Proc\\n/path/file.ads:1:2\\nXXX bytes" }
```

User-defined values can be specified in a `.ci` file by simply replacing the *XXX* by the estimated value.

For indirect calls, the `undefined.ciu` file contains entries like:

```
edge: { sourcename: "source" targetname: "XXX" label: "file.adb:6:15" }
```

In this case, the list of potential targets for the indirect call can be placed in a user-defined `.ci` file, replacing the *XXX* by the symbol name of the potential target of the call:

```
edge: { sourcename: "source" targetname: "target1" label: "file.adb:6:15" }
edge: { sourcename: "source" targetname: "target2" label: "file.adb:6:15" }
edge: { sourcename: "source" targetname: "target3" label: "file.adb:6:15" }
```

The use of *label* with the concrete location indicates that the targets specified are only for the indirect call at that location. If no location is put then the user-defined call will be used for every indirect call from *source*.

If there is an unbounded frame for which a maximal size can be derived by some other kind of analysis, this information can be manually added to a user-defined `.ci` file. In this case, GNATstack will take the user-defined (more concrete) information, ignoring the compiler-generated information. The entry in the `.ci` would have the following pattern:

```
node: { title: "dyn_proc" label: "XXX bytes (static)" }
```

where *XXX* needs to be replaced by the stack size obtained by some other analysis.

The subprogram names specified in these user-defined `.ci` files are the symbolic names used by the compiler rather than simple Ada identifiers. These symbolic names are not identical to the Ada identifiers.

The most typical scenario is that of subprograms declared in packages. In this case, the names are specified with the fully qualified name, but in lower case and with two consecutive underscores replacing the dots in the full name. For example, given the following package:

```
package P is
   procedure R;
   procedure Q (This : Integer);
   procedure Q (This : Float);
end P;
```

The fully qualified names in Ada would be *P.R* and *P.Q* but the names specified in a user-defined `.ci` file would be *p__r* and *p__q*.

Additional characters make up the symbolic names for overloaded units. Specifically, unique numbers are appended to the symbolic names by the compiler, again using double underscore characters as separators. The first overloaded unit does not get a numeric suffix, but all others in that scope do, starting at two. Hence the corresponding symbolic names specified would be *p__q* and *p__q__2*, respectively.

For subprograms declared in scopes other than packages, such as subprograms nested within other subprograms, a similar format is used as for overloaded units. An additional numeric suffix is added by the compiler, but in this case the separator is a leading dot instead of two underscore characters. Moreover, these numeric suffixes are not deducible simply by counting the overloaded units in the scope. The most convenient approach for determining the full symbolic name is to look in the `.ci` files generated by the compiler.

Consider the following:

```
procedure Demo is

   type Index is new Integer;

   procedure Silly (Upper : Index) is
```

(continues on next page)

(continued from previous page)

```
      Data_List : array (1 .. Upper) of Integer;
   begin
      for Value of Data_List loop
         Value := 0;
      end loop;
   end Silly;

begin
   Silly (Upper => 1024);
end Demo;
```

We need to specify the stack requirements for procedure *Silly*. The full Ada name is *Demo.Silly*, so the symbolic name starts with *demo__silly* but there will be a numeric suffix as well.

If we look in the `demo.ci` file we will see the following:

```
graph: {title: "demo.adb"
node: {title: "_ada_demo" label: "Demo\\ndemo.adb:1:1\\n16 bytes (static)\\n0
dynamic objects" }
edge: {sourcename: "_ada_demo" targetname: "demo.adb:demo__silly.1435"
label: "demo.adb:14:4" }
node: {title: "demo.adb:demo__silly.1435" label:"Silly\\ndemo.adb:5:4\\n80 bytes
(dynamic)\\n1 dynamic objects\\n Data_List demo.adb:6:7" }
}
```

We can see in the last node that the symbolic name for *Demo.Silly* is *demo__silly.1435* and that is what would be specified in the user-defined `.ci` file:

```
node: { title: "demo__silly.1435" label: "4128 bytes (static)" }
```

Note that we have used an arbitrary number (4128) of bytes in this user-defined value.

Alternatively, the symbolic names chosen by the compiler appear in the object code so we can use objdump or nm to show the symbols within:

```
$powerpc-elf-objdump --syms demo.o

demo.o:     file format elf32-powerpc

SYMBOL TABLE:
00000000 l    df *ABS*  00000000 demo.adb
00000000 l    d  .text  00000000 .text
00000000 l    d  .data  00000000 .data
00000000 l    d  .bss   00000000 .bss
00000000 l     F .text  00000120 demo__silly.1435
00000000 l    d  .debug_info    00000000 .debug_info
00000000 l    d  .debug_abbrev  00000000 .debug_abbrev
...
```

As before, we can see that the symbolic name for *Demo.Silly* is *demo__silly.1435*.

When there is some missing information the result of the static stack analysis is not accurate. This situation is indicated by GNATstack when displaying the results with the following warning message:

```
Worst case analysis is *not* accurate because of ...; use -Wa for details
```

In addition GNATstack provides accuracy information on a per-subprogram basis. The mark * is attached to the subprograms when there is any missing information.

The accumulated worst-case stack usage for the different entry points will contain the mark *+?* to indicate that extra stack utilization may happen:

```
$ gnatstack *.ci
[...]
Worst case analysis is *not* accurate because of cycles, external calls.
Use -Wa for details.

Accumulated stack usage information for entry points

main : total 504+? bytes
 +-> main
 +-> proc_a *
 +-> proc_b *
```

CHAPTER

SIX

EXAMPLES

This section contains a number of examples of the use of GNATstack showing how to use the tool and how to interpret the results. Let us start analyzing a simple program like the following:

```
procedure Main_Unit is
   type Data_Type is array (1 .. 5) of Integer;

   function Inverse (Input : Data_Type) return Data_Type is
      Result : Data_Type;
   begin
      for Index in Data_Type'Range loop
         Result (Index) := Input (Data_Type'Last -
                                  (Index - Data_Type'First));
      end loop;

      return Result;
   end Inverse;

   Data   : Data_Type := (1, 2, 3, 4, 5);
   Result : Data_Type;
begin
   Result := Inverse (Data);
end Main_Unit;
```

A typical project file would contain the flags to compile the application with the required options to generate stack consumption and call-graph information. In addition, it would contain the desired option for the stack analysis phase:

```
project Prj is
   for Main use ("main_unit.adb");

   package Compiler is
      for Default_Switches ("Ada") use ("-fcallgraph-info=su");
   end Compiler;

   package Stack is
      for Switches use ("-p");
   end Stack;
end Prj;
```

We can then easily compile and analyze the application using this project file:

```
$ gprbuild -P prj.gpr
[...]
$ gnatstack -P prj.gpr
Accumulated stack usage information for entry points

main : total 376 bytes
 +-> main
 +-> main_unit
 +-> main_unit.inverse
```

Note that we do not need to specify the `.ci` files for the application. *gnatstack* computes automatically this list from the project file. In addition, the main entry point has also been automatically detected by GNATstack.

The execution of the tool tells that the longest path (in terms of stack consumption) is that made up by *main* (the program entry point) -> *main_unit* (the main Ada program) -> *main_unit.inverse*. The maximum stack consumption following this path is 376 bytes, and this bound can be trusted.

Focusing on the capability of detecting problems related to the computation of stack requirements we can use the following example:

```
package Ext is
   procedure Set_Alignment (Minimum : Integer);
   function Get_Size (Value : Integer) return Integer;
end Ext;

package body Ext is
   Threshold : Integer;

   procedure Set_Alignment (Minimum : Integer) is
   begin
      Threshold := Minimum;
   end Set_Alignment;

   function Get_Size (Value : Integer) return Integer is
   begin
      return (((Value - 1) / Threshold) + 1) * Threshold;
   end Get_Size;
end Ext;

with Ext;
--  Ext is compiled without stack usage information

procedure P is
   Set_Alignment : access procedure (Minimum : Integer) :=
     Ext.Set_Alignment'Access;
   --  Indirect call

   procedure R;

   procedure Q is
      Data : array (1 .. Ext.Get_Size (10)) of Character;
      --  Dynamically sized local objects
   begin
      R;
```

(continues on next page)

(continued from previous page)

```
   end Q;

   procedure R is
   begin
      Q;
   end R;
begin
   --  Set minimum alignment (via indirect call)
   Set_Alignment.all (Integer'Alignment);

   --  Recursion: Q -> R -> Q
   Q;
end P;
```

To compile and analyze this program we do:

```
$ gcc -c ext.adb
$ gprbuild p.adb -cargs -fcallgraph-info=su,da
[...]
$ gnatstack *.ci
[...]
Worst case analysis is *not* accurate because of cycles, unbounded frames,
external calls, indirect calls. Use -Wa for details.

Accumulated stack usage information for entry points

main : total 472+? bytes
 +-> main
 +-> p
 +-> p.q *
 +-> p.r *
```

The result of the analysis is not accurate. As indicated in the warning message, we can check which is the origin of this inaccuracy using the *-W* option. Additional information can be extracted increasing the verbosity level. Adding the *-v* option will also print detailed information about the location of the different subprograms and per-subprogram stack requirements:

```
$ gnatstack -Wa -v -p *.ci
[...]

List of reachable cycles:

<c1> p.q
 +-> p.q at Q:p.adb:11:14 : 144+? bytes (unbounded,cycle) (<c1>)
 +-> p.r at R:p.adb:9:14 : 104+? bytes (cycle) (<c1>)
 +-> p.q at Q:p.adb:11:14 : 144+? bytes (unbounded,cycle) (<c1>)

List of reachable subprograms with dynamic unbounded frames:

  In p.q at Q:p.adb:11:14
    Data at p.adb:12
```

(continues on next page)

(continued from previous page)

```
List of reachable external subprograms:

  ext.get_size at Get_Size:ext.ads:3:13

List of indirect (including dispatching) calls:

  1 indirect calls in: p at P:p.adb:4:11
    at p.adb:24

Accumulated stack usage information for entry points

main : total 472+? bytes (unbounded,cycle)
 +-> main at main:b~p.ads:24:14 : 112 bytes
 +-> p at ada_main_program:b~p.adb:28:17,P:p.adb:4:11 : 112 bytes
 +-> p.q at Q:p.adb:11:14 : 144+? bytes (unbounded,cycle) (<c1>)
 +-> p.r at R:p.adb:9:14 : 104+? bytes (cycle) (<c1>)
```

We can see that the result is not accurate because there is recursion, dynamic frames, and symbols for which there is no stack usage information.

GNATstack has detected one cycle in the call graph (*Q* calling *R* which in turn calls *Q*). The cycle has been tagged as *<c1>* for later references. The following section of the report indicates that the use of a dynamically sized local object (*Data*) derives into a dynamic unbounded frame (subprogram *Q*). We then have the list of subprograms for which there is no stack usage information (package *Ext* has been compiled without the required flags). Finally, there is a list of locations where there is an indirect call.

Looking at the information about entry points we can see that the computed stack requirements for the *main* entry point is 472 bytes. However, this result is not accurate (as indicated by the mark *+?*), and we see that the two problems that have been found is the existence of cycles and unbounded frames.

The call chain information provides more detailed information about the subprograms that are part of a cycle (*Q* and *R* in this case), and the dynamic size of the frame for subprogram *Q*.

CHAPTER

SEVEN

MACHINE SPECIFIC TOPICS

Because stack usage information is only generated for Ada or C code, if you interface with code written in assembly you may need extra details.

When using a cross compiler with restricted run-time libraries, the stack usage information for the run-time libraries is stored together with the libraries. The options *–target* and *–RTS* can be used to retrieve the information for the run-time library. These options are used the same way as for building the application, as shown in the following example:

```
$ gprbuild --target=powerpc-elf --RTS=powerpc-elf/zfp-prep -P prj.gpr
$ gnatstack --target=powerpc-elf --RTS=powerpc-elf/zfp-prep -P prj.gpr
```

7.1 AVR

The stack used by the *call*, *rcall*, *icall* or *eicall* instructions is counted in the callee. That is all subprograms use at least 3 bytes on avr6 architecture (*atmega256x*) and 2 bytes or avr5 architecture.

If you link with libgcc (which is the default unless you use -nostdlib), add 3 bytes (on avr6 architecture) to the maximum stack usage reported by GNATstack as there is some left subprograms written in assembly.

CHAPTER

EIGHT

GNU FREE DOCUMENTATION LICENSE

Version 1.1, March 2000

59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

8.1 PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document 'free' in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of 'copyleft', which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

8.2 APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The 'Document', below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as 'you'.

A 'Modified Version' of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A 'Secondary Section' is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The 'Invariant Sections' are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The 'Cover Texts' are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A 'Transparent' copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not 'Transparent' is called 'Opaque'.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The 'Title Page' means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, 'Title Page' means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

8.3 VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

8.4 COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

8.5 MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- State on the Title page the name of the publisher of the Modified Version, as the publisher.
- Preserve all the copyright notices of the Document.
- Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- Include an unaltered copy of this License.
- Preserve the section entitled 'History', and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled 'History' in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the 'History' section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- In any section entitled 'Acknowledgements' or 'Dedications', preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- Delete any section entitled 'Endorsements'. Such a section may not be included in the Modified Version.
- Do not retitle any existing section as 'Endorsements' or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled 'Endorsements', provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

8.6 COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled 'History' in the various original documents, forming one section entitled 'History'; likewise combine any sections entitled 'Acknowledgements', and any sections entitled 'Dedications'. You must delete all sections entitled 'Endorsements.'

8.7 COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

8.8 AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an 'aggregate', and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8.9 TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

8.10 TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

8.11 FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License 'or any later version' applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

8.12 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (c)  YEAR  YOUR NAME.

Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.1
or any later version published by the Free Software Foundation;
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
A copy of the license is included in the section entitled 'GNU
Free Documentation License'.
```

If you have no Invariant Sections, write 'with no Invariant Sections' instead of saying which ones are invariant. If you have no Front-Cover Texts, write 'no Front-Cover Texts' instead of 'Front-Cover Texts being LIST'; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

INDEX

Symbols

C

D

E

I

N

P

R

U