Exceptions avec GNAT

1) Principes

Durant le fonctionnement d'un programme en Ada, les anomalies issues du système comme celles détectées par le programme sont transformées en exceptions. Ada nous propose un certain nombre de façons de réagir face à la levée d'une exception comme ne rien faire se qui va provoquer l'arrêt du programme. Parfois voire souvent, il est utile d'obtenir plus d'information sur l'origine de l'anomalie comme identifier la procédure où elle a été déclenchée. GNAT nous vient en aide en enrichissant les informations proposées de base par Ada.

2) Mise en oeuvre sans récupération par Ada

On laisse l'exception levée dans la procédure P1 terminer programme :

```
 1  procedure STB is
 2   procedure P1 is
 3   begin
 4   raise Constraint_Error;
 5   end P1;
 6   procedure P2 is
 7   begin
 8   P1;
 9   end P2;
10    begin
11    P2;
12    end STB;
```

```
$ gnatmake stb
gcc -c stb.adb
gnatbind -x stb.ali
gnatlink stb.ali
$ ./stb
raised CONSTRAINT_ERROR : stb.adb:4 explicit raise
```

Le programme s'arrête sur une erreur provoquée par la levée d'une exception. Juste le nom de l'exception est indiqué avec le numéro de ligne du code source où elle s'est produite. Il n'y a rien concernant la procédure d'origine, ni même le chemin qu'à pris l'exception pour terminer le programme.

GNAT propose de compléter cette information avec les options "-g -bargs -E" permettant d'obtenir l'historique de l'exception.

L'option "-g" est bien connue pour enrichir le programme exécutable avec les information de déverminage.

L'option "-bargs -E" de "gnatbind" permet d'ajouter du code qui va enregistrer les occurrences des exceptions.

```
$ gnatmake -g stb -bargs -E
gcc -c -g stb.adb
gnatbind -aO./ -E -I- -x stb.ali
gnatlink stb.ali -g
$ ./stb
Execution terminated by unhandled exception
Exception name: CONSTRAINT_ERROR
Message: stb.adb:4 explicit raise
Call stack traceback locations:
0x2a00 0x2a24 0x2a54 0x29a0 0x2168 0x2008 0xffc
```

Des informations supplémentaires apparaissent mais elles ne sont pas exploitables directement. Nous faisons appel à l'utilitaire "addr2line" :

```
$ addr2line --exe=stb --functions --demangle=gnat 0x2a00 0x2a24 0x2a54 0x29a0
0x2168 0x2008 0xffc
0x2a00 in stb.p1.0 at stb.adb:4
0x2a24 in stb.p2.1 at stb.adb:8
0x2a54 in ada_stb at stb.adb:11
0x29a0 in main at b~stb.adb:111
0x2168 in start at /SourceCache/Csu/Csu-57/crt.c:299
0x2008 in ?? at start.s:0
0xffc in ?? at ??:0
```

Maintenant le chemin pris par l'exception est plus explicite :

- déclaration dans la procédure p1 de stb à la ligne 4 du source stb.adb
- remontée dans la procédure p2 de stb à la ligne 8 du source stb.adb
- remontée dans la procédure stb à la ligne 11 du source stb.adb
- remontée dans le programme d'initialisation Ada, C et assembleur final

NE FONCTIONNE PLUS AVEC MAC OS X 10.6.

Nous utiliserons l'utilitaire atos (XCode 3.2.6 mini) :

```
$ atos -o stb -arch x86_64 0x10000121b 0x1000011fb 0x1000011e2
0x1000011c0
stb__p1.1650 (in stb) (stb.adb:4)
stb__p2.1652 (in stb) (stb.adb:8)
_ada_stb (in stb) (stb.adb:11)
main (in stb) (b~stb.adb:121)
```

Maintenant le chemin pris par l'exception est explicite mais moins précis :

- déclaration dans la procédure p1 de stb à la ligne 4 du source stb.adb
- remontée dans la procédure p2 de stb à la ligne 8 du source stb.adb
- remontée dans la procédure stb à la ligne 11 du source stb.adb
- remontée dans le programme d'initialisation Ada à la ligne 111 du source b~stb.adb

Pour MAC OS X 10.7, nous devons également ajouter "-largs -Wl,-no_pie" pour forcer la génération de l'exécutable sur un adressage absolu en mémoire et non entièrement indépendant de la position mémoire, mode par défaut pour accroitre la sécurité.

Nous pouvons également utiliser GDB 7.2 qui sera plus précis :

```
$ gdb stb
GNU gdb (GDB) 7.2
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-apple-darwin10.5.0".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /Users/blady/Documents/Programmation/Essais/stb...done.
(gdb) catch exception
Catchpoint 1: all Ada exceptions
(gdb) run
Starting program: /Users/blady/Documents/Programmation/Essais/stb

Catchpoint 1, CONSTRAINT_ERROR at 0x00000001000013fe in stb.p1 ()
    at /Users/blady/Documents/Programmation/Essais/stb.adb:4
4           raise Constraint_Error;
(gdb) bt
#0  <__gnat_debug_raise_exception> (e=0x100013e80) at s-except.adb:43
#1  0x000000010000328d in <__gnat_raise_nodefer_with_msg>
(e=0x100013e80)
    at a-except.adb:833
#2  0x000000010000373c in ada.exceptions.raise_with_location_and_msg (
    e=0x100013e80, f=<value optimized out>, l=<value optimized out>,
    c=<value optimized out>, m=<value optimized out>) at a-except.adb:1019
#3  0x000000010000327c in <__gnat_raise_constraint_error_msg> (
   file=(system.address) 0xf, line=15, column=24,
    msg=(system.address) 0x100013be8) at a-except.adb:823
#4  0x000000010000391b in <__gnat_rcheck_04> (
   file=(system.address) 0x100013e80, line=15) at a-except.adb:1081
```

```
#5  0x00000001000013fe in stb.p1 ()
   at /Users/blady/Documents/Programmation/Essais/stb.adb:4
#6  0x0000000100001423 in stb.p2 ()
   at /Users/blady/Documents/Programmation/Essais/stb.adb:8
#7  0x000000010000140a in stb ()
   at /Users/blady/Documents/Programmation/Essais/stb.adb:11
```

3) Mise en oeuvre avec récupération par Ada

On récupère l'exception et on affiche les informations par la procédure "Exception_Information" de la bibliothèque Ada "Ada.Exceptions". Les informations affichées par cette procédure ne sont pas standardisées. Voyons ce qu'affiche GNAT :

```
 1 with Ada.Text_IO;           use Ada.Text_IO;
 2 with Ada.Exceptions;        use Ada.Exceptions;
 3 procedure STBE is
 4    procedure P1 is
 5    begin
 6       raise Constraint_Error;
 7    end P1;
 8    procedure P2 is
 9    begin
10       P1;
11    end P2;
12 begin
13    P2;
14 exception
15    when E : others =>
16       Put_Line ("----------------------------");
17       Put_Line (Exception_Information (E));
18       Put_Line ("----------------------------");
19 end STBE;
```

```
$ gnatmake stbe
gcc -c stbe.adb
gnatbind -x stbe.ali
gnatlink stbe.ali
$ ./stbe
----------------------------
Exception name: CONSTRAINT_ERROR
Message: stbe.adb:6 explicit raise

----------------------------
```

Le programme s'est arrêtée normalement vu que l'exception a été récupérée. Les informations affichée sont pauvres : le nom de l'exception ainsi que la ligne du source de sa déclaration et de la raison de cette déclaration.

Avec les options -g et -bargs -E :

```
$ gnatmake -f -g stbe -bargs -E
gcc -c -g stbe.adb
gnatbind -aO./ -E -I- -x stbe.ali
gnatlink stbe.ali -g
$ ./stbe
----------------------------
Exception name: CONSTRAINT_ERROR
Message: stbe.adb:6 explicit raise
Call stack traceback locations:
0x2648 0x266c 0x26e0 0x25e8 0x1c88 0x1b28 0xfffffffc

----------------------------
```

Là aussi, des informations supplémentaires apparaissent mais elles ne sont pas exploitables directement. Nous faisons donc appel à l'utilitaire "addr2line".

Affichage en clair des info avec addr2line :

```
$ addr2line --exe=stbe --functions --demangle=gnat 0x2648 0x266c 0x26e0
0x25e8 0x1c88 0x1b28 0xfffffffc
0x2648 in stbe.p1.0 at stbe.adb:6
0x266c in stbe.p2.1 at stbe.adb:10
0x26e0 in ada_stbe at stbe.adb:13
0x25e8 in main at b~stbe.adb:148
0x1c88 in start at /SourceCache/Csu/Csu-57/crt.c:299
0x1b28 in ?? at start.s:0
0xfffffffc in ?? at ??:0
```

Maintenant le chemin pris par l'exception est plus explicite :

- déclaration dans la procédure p1 de stb à la ligne 6 du source stbe.adb
- remontée dans la procédure p2 de stb à la ligne 10 du source stbe.adb
- remontée dans la procédure stb à la ligne 13 du source stbe.adb
- remontée dans le programme d'initialisation Ada, C et assembleur final

Même en récupérant l'exception nous avons trouvé le moyen d'obtenir les même informations. Cependant le moyen d'y parvenir n'est pas direct.

NE FONCTIONNE PLUS AVEC MAC OS X 10.6.

Nous utiliserons l'utilitaire atos, voir paragraphe précédent.

4) Mise en oeuvre avec récupération en clair

GNAT nous aide encore avec la procédure "Symbolic_Traceback" de la bibliothèque "GNAT.Traceback.Symbolic".

```
 1 with Ada.Text_IO;            use Ada.Text_IO;
 2 with GNAT.Traceback.Symbolic; use GNAT.Traceback.Symbolic;
 3 procedure STBG is
 4    procedure P1 is
 5    begin
 6       raise Constraint_Error;
 7    end P1;
 8    procedure P2 is
 9    begin
10       P1;
11    end P2;
12 begin
13    P2;
14 exception
15    when E : others =>
16       Put_Line ("----------------------------");
17       Put_Line (Symbolic_Traceback (E));
18       Put_Line ("----------------------------");
19 end STBG;
```

```
$ gnatmake stbg
gcc -c stbg.adb
gnatbind -x stbg.ali
gnatlink stbg.ali
$ ./stbg
----------------------------

----------------------------
```

Attention rien ne s'affiche si nous n'utilisons pas les options -g et -bargs -E adéquates :

```
$ gnatmake -f -g stbg -bargs -E
gcc -c -g stbg.adb
gnatbind -aO./ -E -I- -x stbg.ali
gnatlink stbg.ali -g
```

```
$ ./stbg
---------------------------
0x24a8 in stbg.p1.0 at stbg.adb:6
0x24cc in stbg.p2.1 at stbg.adb:10
0x2540 in ada_stbg at stbg.adb:13
0x2234 in main at b~stbg.adb:148
0x18d4 in start at crt.c:299
0x1774 in ?? at start.s:0
0xfffffffc in ?? at ??:0

---------------------------
```

Nous obtenons directement le chemin pris par l'exception :

- déclaration dans la procédure p1 de stbg à la ligne 6 du source stbg.adb
- remontée dans la procédure p2 de stbg à la ligne 10 du source stbg.adb
- remontée dans la procédure stbg à la ligne 13 du source stbg.adb
- remontée dans le programme d'initialisation Ada, C et assembleur final

NE FONCTIONNE PLUS AVEC MAC OS X 10.6 :

```
$ ./stbg
---------------------------
Symbolic_Traceback not implemented on this target
---------------------------
```

Un contournement est possible avec l'appel de l'utilitaire atos dans le code Ada de l'unité Excep_Sym_Trace_Workaround (voir le code en annexe).

```
 1. with Ada.Text_IO; use Ada.Text_IO;
 2. --  with GNAT.Traceback.Symbolic; use GNAT.Traceback.Symbolic;
 3. with Excep_Sym_Trace_Workaround; use Excep_Sym_Trace_Workaround;
 4. procedure STBGA is
 5.    procedure P1 is
 6.    begin
 7.       raise Constraint_Error;
 8.    end P1;
 9.    procedure P2 is
10.    begin
11.       P1;
12.    end P2;
13. begin
14.    P2;
15. exception
16.    when E : others =>
17.       Put_Line ("---------------------------");
18.       Put_Line (Symbolic_Traceback (E));
19.       Put_Line ("---------------------------");
20. end STBGA;
```

```
2$ gnatmake -f -g stbga -bargs -E
gcc -c -g stbga.adb
gcc -c -g excep_sym_trace_workaround.adb
gnatbind -E -x stbga.ali
gnatlink stbga.ali -g
bash-3.2$ ./stbga
----------------------------
stbga__p1.2247 (in stbga) (stbga.adb:7)
stbga__p2.2249 (in stbga) (stbga.adb:11)
_ada_stbga (in stbga) (stbga.adb:14)
main (in stbga) (b~stbga.adb:266)

----------------------------
```

Nous obtenons directement le chemin pris par l'exception :

- déclaration dans la procédure p1 de stbg à la ligne 7 du source stbg.adb
- remontée dans la procédure p2 de stbg à la ligne 11 du source stbg.adb
- remontée dans la procédure stbga à la ligne 14 du source stbga.adb
- remontée dans le programme d'initialisation Ada

5) Affichage des traces des exceptions

Nous pouvons souhaiter ne pas ajouter de code d'affichage pour chaque traitement d'exception mais vouloir tout de même activer leur affichage à des endroits particuliers. GNAT nous propose un mode Trace à activer sur demande. Prenons l'exemple suivant, seule l'exception non rattrapée apparait:

```
 1. procedure STBH is
 2.    procedure P1 is
 3.    begin
 4.       raise Constraint_Error;
 5.    exception
 6.       when others =>
 7.          null;
 8.    end P1;
 9.    procedure P2 is
10.    begin
11.       P1;
12.       raise Storage_Error;
13.    end P2;
14. begin
15.    P2;
16. end STBH;
```

```
$ gnatmake -f -g stbh -bargs -E
gcc -c -g stbh.adb
gnatbind -E -x stbh.ali
gnatlink stbh.ali -g
$ ./stbh
Execution terminated by unhandled exception
Exception name: STORAGE_ERROR
Message: stbh.adb:12 explicit raise
Call stack traceback locations:
0x100001407 0x1000013da 0x100001336
```

L'exception dans P1 n'a pas été affiché puisque traitée sans affichage. Le programme s'arrête sur l'exception dans P2 puis que non rattrapée.

Ajoutons les appels au mode trace de GNAT avec "Trace_On" :

```
 1. with GNAT.Exception_Traces;
 2. procedure STBH is
 3.    procedure P1 is
 4.    begin
 5.       raise Constraint_Error;
 6.    exception
 7.       when others =>
 8.          null;
 9.    end P1;
10.    procedure P2 is
11.    begin
12.       P1;
13.       raise Storage_Error;
14.    end P2;
15. begin
16.    GNAT.Exception_Traces.Trace_On (GNAT.Exception_Traces.Every_Raise);
17.    P2;
18. end STBH;
```

```
$ gnatmake -f -g stbh -bargs -E
gcc -c -g stbh.adb
gnatbind -E -x stbh.ali
gnatlink stbh.ali -g
$ ./stbh
Exception raised
Exception name: CONSTRAINT_ERROR
Message: stbh.adb:5 explicit raise
Call stack traceback locations:
0x100001147 0x1000011d5 0x1000011bc 0x10000110e

Unhandled Exception raised
Exception name: STORAGE_ERROR
Message: stbh.adb:13 explicit raise
```

```
Call stack traceback locations:
0x1000011e9 0x1000011bc 0x10000110e
```

Les deux exceptions sont bien affichées. "Trace_Off" permet de désactiver l'affichage.

Ajoutons alors l'affichage symbolique avec "Set_Trace_Decorator" :

```
 1. with GNAT.Exception_Traces;
 2. with Excep_Sym_Trace_Workaround;
 3. procedure STBH is
 4.    procedure P1 is
 5.    begin
 6.       raise Constraint_Error;
 7.    exception
 8.       when others =>
 9.          null;
10.    end P1;
11.    procedure P2 is
12.    begin
13.       P1;
14.       raise Storage_Error;
15.    end P2;
16. begin
17.    GNAT.Exception_Traces.Trace_On (GNAT.Exception_Traces.Every_Raise);
18.    GNAT.Exception_Traces.Set_Trace_Decorator
19.      (Excep_Sym_Trace_Workaround.Symbolic_Traceback'Access);
20.    P2;
21. end STBH;
```

```
$ gnatmake -f -g stbh -bargs -E
gcc -c -g stbh.adb
gcc -c -g excep_sym_trace_workaround.adb
gnatbind -E -x stbh.ali
gnatlink stbh.ali -g
$ ./stbh
Exception raised
Exception name: CONSTRAINT_ERROR
Message: stbh.adb:6 explicit raise
stbh__p1.2247 (in stbh) (stbh.adb:6)
stbh__p2.2249 (in stbh) (stbh.adb:13)
_ada_stbh (in stbh) (stbh.adb:20)
main (in stbh) (b~stbh.adb:268)

Unhandled Exception raised
Exception name: STORAGE_ERROR
Message: stbh.adb:14 explicit raise
stbh__p2.2249 (in stbh) (stbh.adb:14)
_ada_stbh (in stbh) (stbh.adb:20)
main (in stbh) (b~stbh.adb:268)
```

Nous obtenons les informations complètes pour chacune des deux exceptions.

Pascal Pignard, novembre 2005, juillet 2011, mars 2012.

ANNEXE : code source de l'unité Excep_Sym_Trace_Workaround

```
with Ada.Exceptions;
with Ada.Exceptions.Traceback;
package Excep_Sym_Trace_Workaround is
  function Symbolic_Traceback
    (Traceback : Ada.Exceptions.Traceback.Tracebacks_Array)
     return      String;
  function Symbolic_Traceback
    (E    : Ada.Exceptions.Exception_Occurrence)
     return String;
end Excep_Sym_Trace_Workaround;

with System.Address_Image;
with Ada.Command_Line;
with Ada.Text_IO;
with Ada.Strings.Unbounded;
with GNAT.OS_Lib;
with Ada.Strings.Unbounded.Text_IO;
package body Excep_Sym_Trace_Workaround is
  function Exception_Information_Workaround
    (Address_List : Ada.Strings.Unbounded.Unbounded_String)
     return        String
  is
     use type Ada.Strings.Unbounded.Unbounded_String;
     use type GNAT.OS_Lib.Argument_List;

     File     : Ada.Text_IO.File_Type;
     Exe      : constant String := Ada.Command_Line.Command_Name;
     Filename : constant String := Exe & "-exception.txt";
     Text     : Ada.Strings.Unbounded.Unbounded_String;
     Result   : Integer;
     Ok       : Boolean;

     Args : GNAT.OS_Lib.Argument_List_Access :=
       GNAT.OS_Lib.Argument_String_To_List
        ("-o " &
          Ada.Command_Line.Command_Name &
         " -arch x86_64 " &
          Ada.Strings.Unbounded.To_String (Address_List));
  begin
    if GNAT.OS_Lib.Is_Executable_File (Exe) then
```

```
      Ada.Text_IO.Create (File, Ada.Text_IO.In_File, Filename);
      GNAT.OS_Lib.Spawn ("/usr/bin/atos", Args.all, Filename, Ok, Result);
      GNAT.OS_Lib.Free (Args);
      if Result = 0 then
        Ada.Text_IO.Reset (File);
        while not Ada.Text_IO.End_Of_File (File) loop
          Ada.Strings.Unbounded.Append
           (Text,
            Ada.Strings.Unbounded.Text_IO.Get_Line (File) & ASCII.LF);
        end loop;
        Ada.Text_IO.Close (File);
        return Ada.Strings.Unbounded.To_String (Text);
      else
        return Ada.Strings.Unbounded.To_String (Address_List);
      end if;
    else
      return "Executable '" &
           Exe &
            "' not found, probably called from JAVA context.";
    end if;
  end Exception_Information_Workaround;

  function Symbolic_Traceback
   (Traceback : Ada.Exceptions.Traceback.Tracebacks_Array)
    return     String
  is
    Address_List : Ada.Strings.Unbounded.Unbounded_String;
  begin
    for Ind in Traceback'Range loop
      Ada.Strings.Unbounded.Append
       (Address_List,
        " 0x" & System.Address_Image (Traceback (Ind)));
    end loop;
    return Exception_Information_Workaround (Address_List);
  end Symbolic_Traceback;

  function Symbolic_Traceback
   (E    : Ada.Exceptions.Exception_Occurrence)
    return String
  is
  begin
    return Symbolic_Traceback (Ada.Exceptions.Traceback.Tracebacks (E));
  end Symbolic_Traceback;

end Excep_Sym_Trace_Workaround;
```