Exposing Uninitialized Variables:

Strengthening and Extending Run-Time Checks in Ada

Uninitialized Variables

- Common cause of bugs that are difficult to find
- Often lead to unpredictable behavior
- May show up under special circumstances not encountered during testing

Example

One release of the Eurcontrol CFMU Air Traffic Flow Management app had the following bug in compatibility code:

```
--  Initial version
if  Reading_Current_Version  then
    Boolean'Read (Stream, A_Flight.New_Field);
end if;
```

```
--  Correct Version
if  Reading_Current_Version  then
    Boolean'Read (Stream, A_Flight.New_Field);
else
    A_Flight.New_Field := False;
end if;
```

Detecting Uninitialized Variables

- **Static detection**
  - Formal validation techniques
  - Compiler Warnings
- **Run-Time detection**
  - Purify-like solutions
  - Ada 95 Normalize_Scalars pragma

Static Detection of Uninitialized Variables

- **Formal validation techniques**
  - Difficult to apply to large-scale applications such as Eurocontrol's (1.5 M SLOC)
  - Even harder if the application exists already

- **Compiler warnings**
  - GNAT produces warnings about dubious code such as

```
procedure P is
      K : Natural;
begin
      K := K + 1;
      …
end P;
```

Compile time warning

More on Compiler Warnings

- GNAT emits such warnings in various cases by tracing possible static flow paths
- Problem is undecidable in general
  - E.g. array element initialization

```
procedure Q (N : Positive) is
      A : array (1 .. 3) of Natural;
begin
      A (2) := 0;
      A (1) := A (N) + 1;
      …
end Q;
```

- Generating too many false alerts is counter-productive

Example of Compile-Time False Alarm

```
procedure  Read_Or_Write (Read_Mode : Boolean; A : in out Natural) is
begin
   if  Read_Mode  then
      A := …;        -- Read from somewhere
   else
      Write (A);   -- Write somewhere
   end if;
end  Read_Or_Write;
```

Run-Time Detection: Purify-Like Solutions

- An all or nothing tool, cannot be applied selectively
- Instrumented object code is 3 to 5 times slower and takes 40% more memory
- Precludes the use of Purified applications in operational context
- Purify did not detect all of the problems that GNAT's new pragma Initialize_Scalars detected

Run-Time Detection: Normalize_Scalars

- **Pragma Normalize_Scalars (Ada 95 Annex H)**
  - Designed to eliminate non-determinacy from safety-critical apps
- **Requires application wide consistency**
  - Precludes its use for testing small portions of a large application
- **Manual coding is required to detect invalid values**

```
if  A_Flight.New_Field'Valid  then
   …  -- The field can be used
else
   …  -- Error handling
end if;
```

GNAT New Solution

- New pragma Initialize_Scalars
- Ability to select the initial value for uninitialized scalars
- Compiler support for additional validity checking levels

Pragma Initialize_Scalars

- **Behaves like Normalize_Scalars**
  - That is it initializes uninitialized scalars
- **You can apply this pragma just to some units**
  - Don't have to apply it to the whole program like Normalize_Scalars
- **Can be conveniently used for large portions of a large application**
  - For instance for newly introduced units

Choice of Initial Values

- **The initial value can be selected at bind time between**
  - All bits 0
  - All bits 1
  - Invalid value if possible (like as in Normalize_Scalars)
  - A specified bit pattern

- **Running the app with different settings can detect more bugs**

- **This is particularly useful when no invalid value exists**
  - Variation in behavior can indicate the existence of uninitialized variables

Selective Validity Checking

-gnatVa/n	Turn ON/OFF	all validity checks (including RM)
-gnatVc/C		checks for copies
-gnatVd/D		RM checks (on by default)
-gnatVf/F		checks for floating points
-gnatVi/I		checks for "in" parameters
-gnatVm/M		checks for "in out" parameters
-gnatVo/O		checks for operators
-gnatVr/R		checks for returns
-gnatVs/S		checks for subscripts
-gnatVt/T		checks for tests

Constraint-Error raised
If invalid value is detected

Eurocontrol uses -gnatVaM

```
procedure  Read_Or_Write (Read_Mode : Boolean; A : in out Natural) is
begin
    if  Read_Mode  then
        A := …;        -- Read from somewhere
    else
        Write (A);   -- Write somewhere
    end if;
end  Read_Or_Write;
```

Application to the Eurocontrol Application

- **GNAT has only reported real errors (uninitialized scalar usage)**

- **GNAT helped detect subtle bugs**
  - Procedure waiting for an X protocol event up to a certain deadline. When deadline was reached before event occurred the variable that said if X event was pending was left uninitialized. This left open the possibility to a call to X to handle an unexisting event
  - Very helpful for instance in numerical algorithm where bugs could only otherwise uncovered by checking the precision of the computation

- **GNAT helped detect efficiency bugs**
  - Not all bugs lead to functional problems, some subtle ones can lead to useless searches in a list

Performance Impact

Mode	Current Use	BUILD Time	Executable Size	Run Time
No optimization RM checking		100	100	100
No Optimization Inizialize_Scalars All validity checks ON	Development	118	107	160
Optimization All validity checks OFF		190	68	69
Optimization RM checking	Operational	197	69	70
Optimization Inizialize_Scalars All validity checks ON		252	72	91

Summary

- **Eurocontrol experience with Initialize_Scalars has been very positive**
  - Recommend the use of -gnatVa
- **GNAT fine-grain control over validity checking makes it practical for use in existing applications**
- **Trend in programming guidelines to "force" initializing everything at declaration can lead to wrong code that is much harder to detect**

```
B : Natural := 0;  -- NOT a good idea :)
…
if  … then
   B := 5;
elsif … then
   B := 8;
end if;
```