Software Analysis Tools @ AdaCore

Yannick Moy

LSL Seminar, CEA-LIST
December 8th, 2009
Outline

Ada & AdaCore

Dynamic Analysis Tools @ AdaCore

Static Analysis Tools @ AdaCore

Project Hi-Lite
Outline

Ada & AdaCore

Dynamic Analysis Tools @ AdaCore

Static Analysis Tools @ AdaCore

Project Hi-Lite
Ada Timeline

1975
US DoD
"Strawman"

1977
4 proposals selected
Green, Red, Blue, Yellow

1979
Green

1983
Ada83

1995
Ada95

2005
Ada2005

201X
Ada201X
Integers in Ada

```ada
subtype Eggs_Number is Integer range 0 .. 12;
type Eggs_Number is new Integer range 0 .. 12;

if Eggs_Number'First < Num and then
  Num < Eggs_Number'Last
then ...

for Num in Eggs_Number'Range loop ...

Val : Integer;
Num := Eggs_Number'(Val);
Num := Eggs_Number(Val);
```
type Arr is array (Eggs_Number) of Natural;

for Num in Arr’Range loop ... 

procedure Set (X : out T);
procedure Get (X : in T);
procedure Get_And_Set (X : in out T);

type Pool_Ptr is access Integer;
type General_Ptr is access all Integer;
type Non_Null_Ptr is not null access Integer;

procedure Get (X : access Integer);
AdaCore Timeline

1992
GNAT
NYU/FSF

1994
AdaCore US

1996
AdaCore EU

GPS 2000
GNATbench
C

C++
GNATstack
GPRbuild
AJIS

2000

2009

2010

C

GtkAda

GVD

PolyORB

GNATcheck

...
AdaCore Business

Freely-licensed open-source products (FLOSS)
Renewable non-locked subscription
Subscription with Frontline support

60 Engineers
20 PhD
5 Professors
10 Consultants
AdaCore Customers
Outline

Ada & AdaCore

Dynamic Analysis Tools @ AdaCore

Static Analysis Tools @ AdaCore

Project Hi-Lite
Run-time Checking

**Constraint errors**

- Array access outside its bounds
- Range overflow
- Integer overflow (-gnato)
- etc.

**Validity checks**

```
pragma Initialize_Scalars;  X'Valid
```

**Assertions**

```
pragma Assert (test [, message]);
```
Run-time Checking

Constraint errors

Array access outside its bounds
Range overflow
Integer overflow (-gnato)

etc.

Validity checks

\texttt{-gnatVcefimoprst}

\texttt{pragma Initialize_Scalars;}

\texttt{x’Valid}

Assertions

\texttt{pragma Assert (test [, message]);}
Run-time Checking

Constraint errors

- Array access outside its bounds
- Range overflow
- Integer overflow (-gnato)
- etc.

Validity checks

-\texttt{-gnatVcefioprst}

\texttt{pragma Initialize_Scalars;}

\texttt{pragma Assert (test [, message]);}
Run-time Checking

**Constraint errors**

Array access outside its bounds
Range overflow
Integer overflow (-gnato)
*etc.*

**Validity checks**

```
-gnatVcefimoprst
```

```
pragma Initialize_Scalars;
```

```
pragma Assert (test [, message]);
```

X’Valid
Run-time Checking

### Constraint errors
- Array access outside its bounds
- Range overflow
- Integer overflow (-gnato)
- *etc.*

### Validity checks

```
 pragma Vcexoopprst
```

```
pragma Initialize_Scalars;  X'Valid
```

### Assertions
```
pragma Assert (test [, message]);
```
procedure Linear_Search
  (Table : in IntArray;
   Value : in Integer;
   Found : out Boolean;
   Index : out Integer);
pragma Precondition (Counter < Integer 'Last);
pragma Postcondition (not Found or else
  (Table(Index) = Value and then
   Counter = Counter 'Old + 1));

procedure Linear_Search (...) is
begin
  ...
  for J in Integer range Table 'Range loop
    pragma Assert (Found = False and
      Counter < Integer 'Last and
      Counter = Counter 'Old);
  ...
end loop;
end Linear_Search;
<table>
<thead>
<tr>
<th>Memory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gnatmem/valgrind</td>
<td>memory management</td>
</tr>
<tr>
<td>memory pools</td>
<td>callbacks on (de-)allocation</td>
</tr>
<tr>
<td>debug pools</td>
<td>callback on dereference</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stack</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-fstack-check</td>
<td>stack overflow detection and recovery</td>
</tr>
<tr>
<td>gnatbind -u</td>
<td>post-execution analysis</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exceptions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exception traces</td>
<td>trace all exceptions</td>
</tr>
<tr>
<td>exception actions</td>
<td>callback on exceptions</td>
</tr>
</tbody>
</table>
### Memory
- gnatmem/valgrind: memory management
- memory pools: callbacks on (de-)allocation
- debug pools: callback on dereference

### Stack
- -fstack-check: stack overflow detection and recovery
- gnatbind -u: post-execution analysis

### Exceptions
- exception traces: trace all exceptions
- exception actions: callback on exceptions
## Memory, Stack & Exceptions

### Memory
- **gnatmem/valgrind**: memory management
- **memory pools**: callbacks on (de-)allocation
- **debug pools**: callback on dereference

### Stack
- **-fstack-check**: stack overflow detection and recovery
- **gnatbind -u**: post-execution analysis

### Exceptions
- **exception traces**: trace all exceptions
- **exception actions**: callback on exceptions
Outline

Ada & AdaCore

Dynamic Analysis Tools @ AdaCore

Static Analysis Tools @ AdaCore

Project Hi-Lite
GNAT Warnings

-`gnatwc.cdfh.ijklm.op.pr.rtu.w.x`

Conditional expression known to be true or false at compile-time

Hiding declaration

Variable could be constant

Unused entity
GNAT Style Checks

**Pragma Restrictions**

- Maximum line length
- Boolean operators
- Separate specs
- Indentation level

**Pragma Profile**

- -gnaty3aAbBcdefhilknOprssStuxoM80
GNAT Style Checks

-\texttt{-gnaty3aAbBcdefhilkInOprsStuxoM80}

\begin{itemize}
    \item \texttt{pragma Restrictions}
    \item \texttt{pragma Profile}
\end{itemize}
### GNATmetric: Metrics Computation

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>average_lines_in_bodies</td>
<td>9.00</td>
</tr>
<tr>
<td>all_decs</td>
<td>13</td>
</tr>
<tr>
<td>lloc</td>
<td>19</td>
</tr>
<tr>
<td>public_subprograms</td>
<td>2</td>
</tr>
<tr>
<td>all_subprograms</td>
<td>2</td>
</tr>
<tr>
<td>public_types</td>
<td>1</td>
</tr>
<tr>
<td>all_types</td>
<td>1</td>
</tr>
<tr>
<td>all_stmts</td>
<td>6</td>
</tr>
<tr>
<td>average_complexity</td>
<td>1.50</td>
</tr>
<tr>
<td>blank_lines</td>
<td>12</td>
</tr>
<tr>
<td>all_lines</td>
<td>42</td>
</tr>
<tr>
<td>code_lines</td>
<td>26</td>
</tr>
<tr>
<td>comment_lines</td>
<td>4</td>
</tr>
<tr>
<td>eol_comments</td>
<td>0</td>
</tr>
<tr>
<td>comment_percentage</td>
<td>13.33</td>
</tr>
</tbody>
</table>

```ada
package body Simple is
  function Threshold (Not_Null : Boolean) return Sensor_Value is
    begin
      if Not_Null then
        Result := 43;
      else
        Result := 276;
      end if;
      return Result;
    end Threshold;

    procedure Normalize (Level : in out Sensor_Value) is
      begin
        Not_Null := Boolean;
        Not_Null := Level /= 0;
        Level := Level + Threshold (Not_Null);
      end Normalize;
  end Simple;
```

Subversion:153623 (Locally modified) | Insert | Writable | Unmodified | 1:1
GNATcheck: Coding Standard Checker
CodePeer: Modular Static Analysis

Warnings + {Contracts}
## CodePeer Warnings

### CodePeer report

<table>
<thead>
<tr>
<th>Entity</th>
<th>+/-</th>
<th>High base deltas</th>
<th>now</th>
<th>Medium base deltas</th>
<th>now</th>
<th>Low base deltas</th>
<th>now</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test</td>
<td></td>
<td>16</td>
<td>16</td>
<td>80</td>
<td>80</td>
<td>160</td>
<td>160</td>
</tr>
<tr>
<td>admintoken.adb</td>
<td></td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>admintoken.readandcheck</td>
<td></td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>admintoken.readandcheck.checkauthcert</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>admintoken.readandcheck.makedescription</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>admintoken.readandcheck.checkidcertok</td>
<td></td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>tcpip.adb</td>
<td></td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>usertoken.adb</td>
<td></td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>tokenreader.adb</td>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>cert-id.adb</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>enclave.adb</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>clock-interface.adb</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>userentry.adb</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>configdata.adb</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

### Message history

- added
- unchanged
- removed
- suppressed
- informational
- low
- medium
- high

### Message ranking

- array index check
- call too complex - analysis skipped
- dead code
- dead code continues
- overflow check
- precondition
- range check
- subp not available
- suspicious precondition
- test always false
- test always true
- unused assignment
- unused assignment in callee
- validity check

### Message categories

<table>
<thead>
<tr>
<th>Message category</th>
<th>High</th>
<th>Medium</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>dead code</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>test always false</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>validity check</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
CodePeer Contracts
Counter : Natural := 0;

procedure Linear_Search
  (Table : in IntArray;
   Value : in Integer;
   Found : out Boolean;
   Index : out Integer);

—# global in out Counter;

—# derives Counter from Counter, Table, Value;

—# pre Counter < Integer 'Last;
—# post Found → (Table(Index) = Value and
—#     Counter = Counter~ + 1);
procedure Linear_Search (...) is
begin
  Found := False;
  Index := 0;

  for J in Integer range Table 'Range loop
    —# assert Found = False and
    —#         Counter < Integer 'Last and
    —#         Counter = Counter ~;

    if Table(J) = Value then
      Counter := Counter + 1;
      Found := True;
      Index := J;
      exit;
    end if;
  end loop;
end Linear_Search;
Outline

Ada & AdaCore

Dynamic Analysis Tools @ AdaCore

Static Analysis Tools @ AdaCore

Project Hi-Lite
Big Picture

Testing

Static Analysis

Hi-Lite

Formal Verification
Common Language for Properties

Executable Annotation Language

- User Input
- Inferred by Static Analysis
- Generated with Code from Model
- Testing
- Static Analysis
- Formal Verification
## State-of-the-art Free Software Tools

<table>
<thead>
<tr>
<th>Software</th>
<th>Category</th>
<th>Experts</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNAT Pro</td>
<td>compiler</td>
<td>AdaCore</td>
<td>GNU GPL</td>
</tr>
<tr>
<td>CodePeer</td>
<td>analyser</td>
<td>AdaCore</td>
<td>GNU GPL</td>
</tr>
<tr>
<td>Examiner</td>
<td>verifier and VC generator</td>
<td>Praxis</td>
<td>GNU GPL</td>
</tr>
<tr>
<td>Simplifier</td>
<td>prover</td>
<td>Praxis</td>
<td>GNU GPL</td>
</tr>
<tr>
<td>Why</td>
<td>VC generator</td>
<td>ProVal</td>
<td>GNU LGPL</td>
</tr>
<tr>
<td>Alt-Ergo</td>
<td>prover</td>
<td>ProVal</td>
<td>CeCILL-C</td>
</tr>
<tr>
<td>Frama-C</td>
<td>analyser and verifier</td>
<td>CEA LIST and ProVal</td>
<td>GNU LGPL</td>
</tr>
</tbody>
</table>
Workflow Between Tools
Many Possible Uses
Challenges

Inferring more precise annotations
Conditional contracts instead of “soft” contracts
Non-overlapping of reference/pointer parameters
Top-down propagation of calling contexts

Verification of properties on containers
Standard library of containers in SPARK
Expressing quantification over containers
Automatic proof of such properties

Improved user interaction
Modular interaction at different levels
Path highlighting (warnings, verification conditions)
Traceability of results
Challenges

Inferring more precise annotations

- Conditional contracts instead of “soft” contracts
- Non-overlapping of reference/pointer parameters
- Top-down propagation of calling contexts

Verification of properties on containers

- Standard library of containers in SPARK
- Expressing quantification over containers
- Automatic proof of such properties

Improved user interaction

- Modular interaction at different levels
- Path highlighting (warnings, verification conditions)
- Traceability of results
Challenges

Inferring more precise annotations
Conditional contracts instead of “soft” contracts
Non-overlapping of reference/pointer parameters
Top-down propagation of calling contexts

Verification of properties on containers
Standard library of containers in SPARK
Expressing quantification over containers
Automatic proof of such properties

Improved user interaction
Modular interaction at different levels
Path highlighting (warnings, verification conditions)
Traceability of results
Beyond Formal Verification

The Tokeneer Project
A hands on look at an NSA funded, highly secure biometric software system.

Copy-paste error

1 \textbf{if} Some\_Var \textbf{then}
2 ... 
3 \textbf{if} Some\_Var \textbf{then}

Dead defensive code

1 \texttt{X := F ( ... );}
2 \texttt{case X in}
3 ... 
4 \texttt{Invalid\_Value \Rightarrow ...}

Refactoring error

1 \texttt{X : T;}
2 \texttt{function F (Y : T) is}
3 \texttt{begin}
4 \texttt{ Use (X);}
5 \texttt{end;}
6 \texttt{F (X);}

Ada run-time errors

bound check, null string, uninitialized scalar