AdaCore

GNU tools Cauldron 2012
2012-07-12

Flow control preservation in GCC for use in safety critical developments

Eric Botcazou, Olivier Hainque

Preamble

Not a super-gcc-technical talk :-)

**More of an introduction
to a fairly technical project that proved successful
in use and that we'd like to contribute**

This is about two new options, that ...

- dump tables describing source constructs of relevance to certification activities, coverage analysis in particular (-fdump-scos)
- control optimizers to allow non intrusive coverage analysis and facilitate source to object traceability (-fpreserve-control-flow)

Outline

- Motivation
- Implementation principles
- Current status

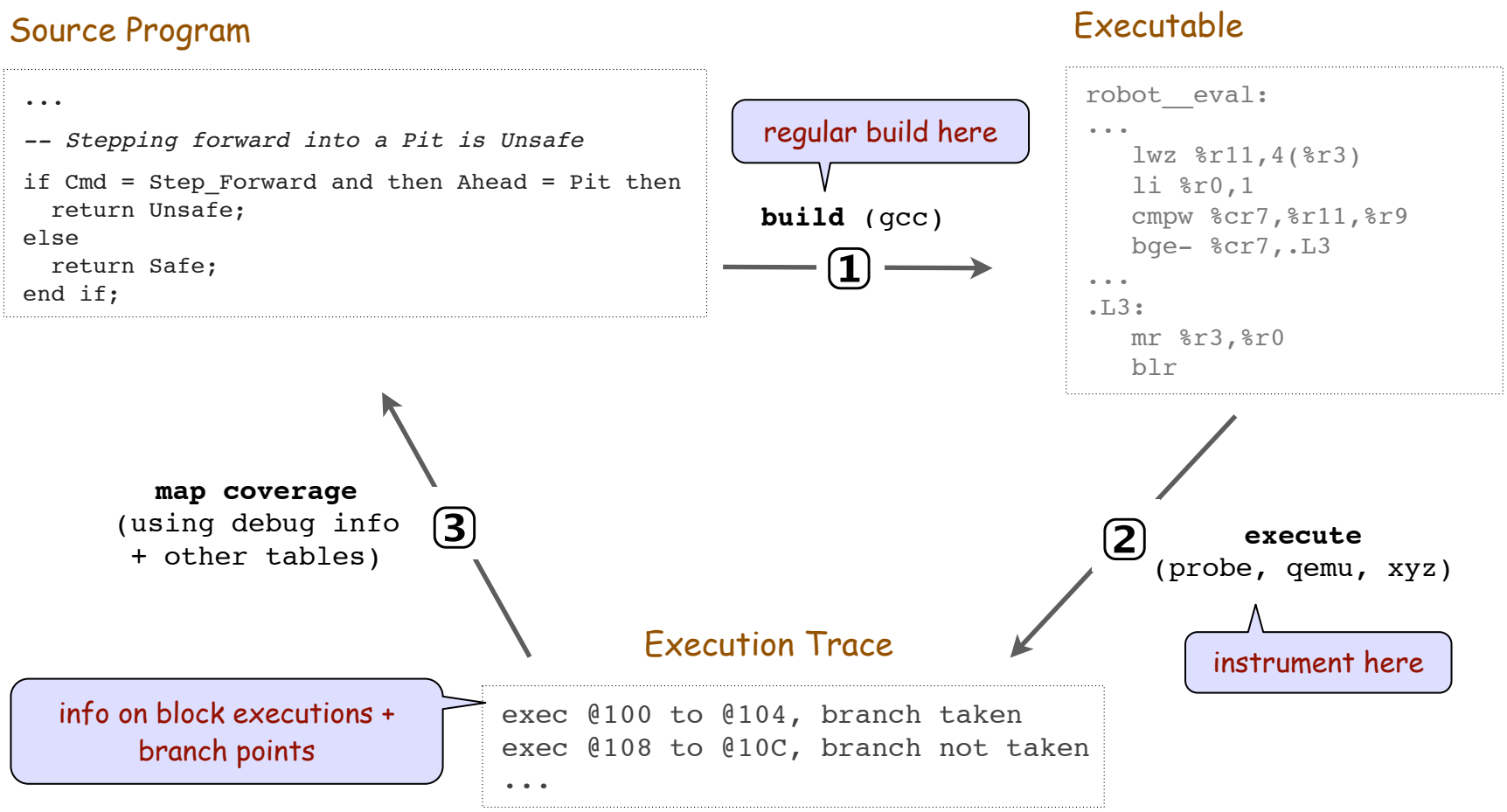
Motivation

Certification processes for safety critical software (e.g. DO178B)

- **Non-intrusive coverage analysis**
- **Source to object traceability**

Motivation : Non intrusive coverage analysis

- **Infer coverage from execution traces**, out of hardware probe or emulator



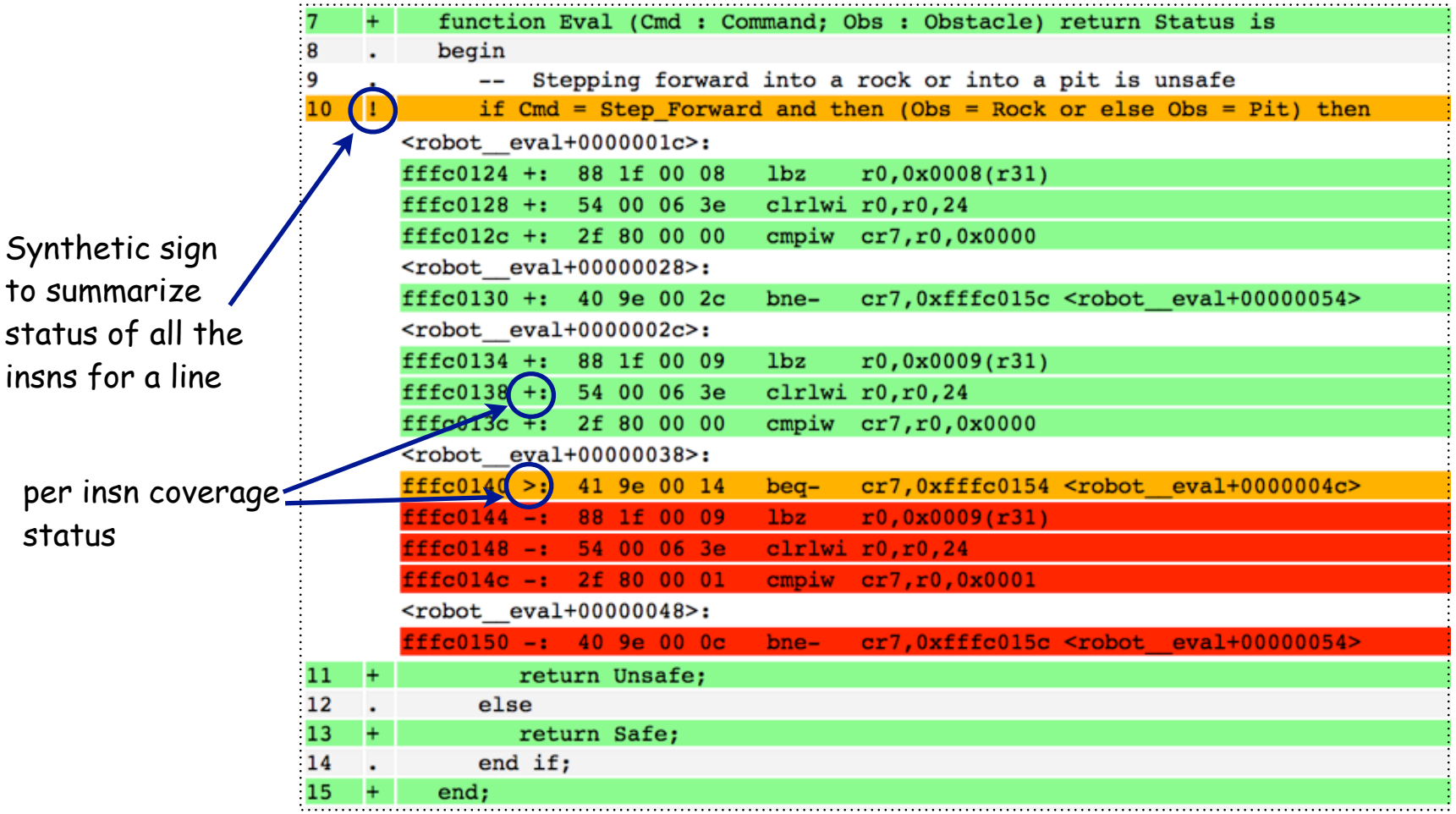
Motivation : Non intrusive coverage analysis

Absence of program instrumentation is a plus
in safety critical developments

- Easy to repeat analysis for different criteria
- Analysis on code close to what will go embedded, if not identical
- Tradeoffs to find, room for lots of future work as we will see ...

Motivation : Non intrusive coverage analysis

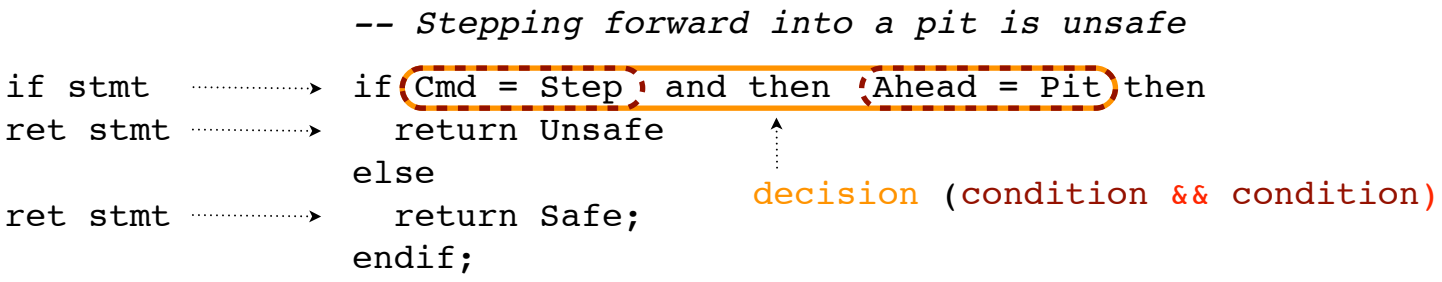
- Execution traces + basic line debug info provide
  machine insn & branch coverage info mapped to source lines



- Useful but not sufficient for typical certification purposes ...

Motivation : safety critical certification processes (e.g. DO178B/C)

- Care about **source level coverage criteria**, reasoning on ***statements***, ***decisions*** (boolean expressions), and ***conditions*** (operands)

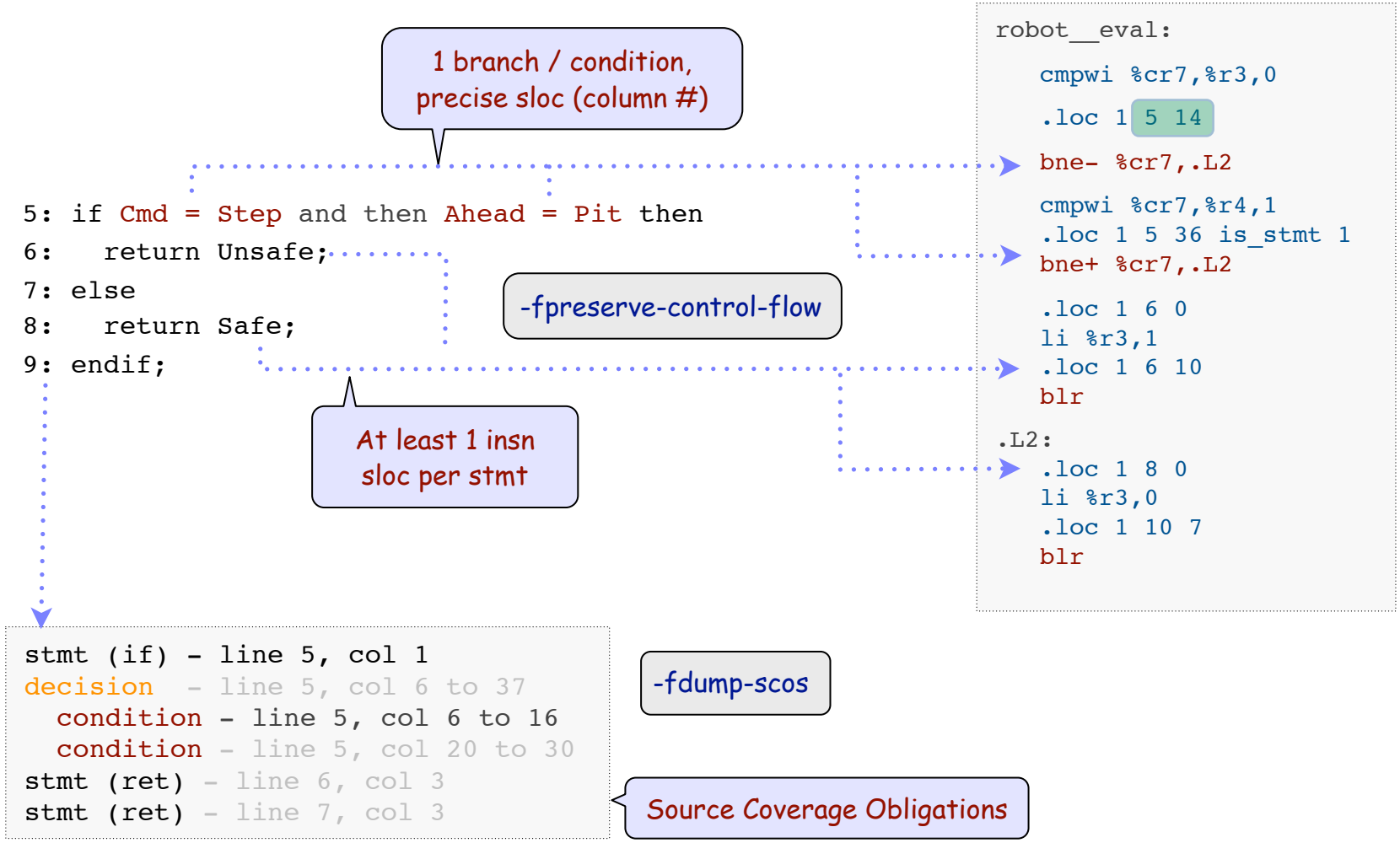


- e.g. ***decision coverage*** : tests shall exercise every decision both ways
- ***mcdc*** : decision coverage + every condition is shown to have “independent influence” on the decision

How can we infer these from execution traces ?

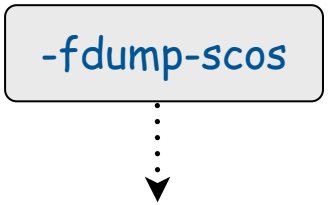
Motivation : inferring decision or condition coverage from execution traces

- **One central idea : fetching condition/decision values from branch traces**
- **Multiple basic needs, new control options ...**

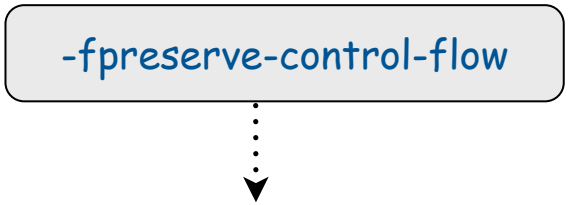


Wrapup, the basic point is ...

- Allow code generation suitable for non-intrusive coverage analysis aimed at safety-critical certification processes



- "Source Coverage Obligation" Tables



- 1 branch per condition, precise sloc info
- At least 1 insn sloc per statement

- Helps source to machine code traceability as well

- Not so easy with optimizations, major challenge actually ...
- Tricky balance between satisfying the functional constraints and the ability to embed (efficiency),
- We have to care about allowing as much optimization as possible

Implementation principles & Current Status

Implementation Principles & Current Status / -fdump-scos

- A lot of information to convey
  (statements, decisions, conditions, operators, dominance)
- Need compact representation
- Implementation for Ada wired in the GNAT parser, where we have precise & highlevel visiblity on the syntactic items
- Implementation for C as a pre-gimplification standalone pass, ~450 lines, not exercised much yet

Implementation Principles & Current Status / -preserve-control-flow

- We need to address “decisions” in general, not only part of source control flow constructs (if, while, for, ...), e.g `r = x && y`
- We rely on short-circuit operators to produce branches, so to define conditions in addition to explicit control-flow

```
if (x) {
   ...
}
```

1 condition

```
if (z && t) {
   ...
}
```

2 conditions

2 conditions

```
r = x && y; gimplified into (x && y) ? true : false;
```

- Non short-circuit logical operators are just computational

```
if ((x|y) && (z|t)) {
   ...
}
```

2 conditions here, not 4

Implementation Principles & Current Status / -preserve-control-flow

- Prior to exec trace analysis, static analysis phase that matches the machine code CFG (from binary) against the source one (from SCOs)

```
5: if Cmd = Step and then Ahead = Pit then
6:   return Unsafe;
7: else
8:   return Safe;
9: endif;
```

-fdump-scos

```
stmt (if) - line 5, col 1
decision  - line 5, col 4 to 40
  and-then  - line 5, col 14
  condition - line 5, col 4 to 13
  condition - line 5, col 23 to 40
...
```

```
robot__eval:
   cmpwi %cr7,%r3,0
   .loc 1 5 8
   bne- %cr7,.L2
   cmpwi %cr7,%r4,1
   .loc 1 5 30 is_stmt 1
   bne+ %cr7,.L2
   ...
```

-fpreserve-control-flow

- Sophisticated heuristics, need to sort out the relevant branches :

```
if ((x > y) != (z < t)) {
   ...
}
```

1 condition here (!=), multiple branches

- Optimization is a challenge that we need to address to allow analysis on code as embedded, as much as we can

Implementation Principles & Current Status / -preserve-control-flow

- First results with gcc 4.3, Ada only, -O0 only
- Ported to gcc 4.5, support of -O1 + -fdump-scos for C
- Three patchsets essentially:

disconnect the incompatible optimization circuits

propagate slocs through the optimization stream

arrange to have column # in branch slocs, attached to operands - not operators

- A few challenges on the way. VTA was a great help in keeping track of statements optimized away, inlining included
- Allowed a definite leap in -O support :-) Now we need other ideas !!

Implementation Principles & Current Status / -preserve-control-flow

- Two new options: -fdump-scos & -fpreserve-control-flow
- Allow non-intrusive coverage analysis up to the strictest DO178B safety-critical certification criteria and alike
- Base of our GNATcoverage technology, operational up to -O1
  - powerpc & sparc with an instrumented qemu,
  - x86-linux using valgrind
- Ongoing port to 4.7, would love to contribute to mainline
- Allowing more optimizations would be great - we need ideas ...

AdaCore

Thank you for your attention