Promoting ECSS Category B Software to Category A

Reaching the highest software quality in ECSS

Fabian Schriever, Thomas Wucher, Christoph Weiß, Joan Roig, and Andoni Arregi

GTD GmbH, Markdorf, Germany

Do you REALLY know what software you are flying?

When your flight software was almost correct:

- Ariane-5 maiden flight was almost successful
- Mars Climate Orbiter was almost placed into Mars orbit
- Beresheet almost landed on Moon

Can you answer the following questions?

- What happens before `main()`?
- How many functions do you call without even knowing they exist?
- What's different between Host/Target/Optimizations compilations?

Don't Trust Your Toolchain!

You think your flight **software is qualified**, because you've used a **qualified operating system** and you have **qualified application software**?
NO! You will fly unqualified:

- standard C library functions (`memset`, ...)
- compiler library functions (`.udiv`, ...)
- linker introduced object code

Study Goals

To define a **methodology, guideline, and tooling** to promote ECSS Category B qualified software to **Category A** software.

To apply the assessment to real-life use-cases:

- LibmCS
- RTEMS 6

Main Tasks for Promoting Software to Category A

- Achievement of 100% MCDC structural coverage:
  - ▷ Can be achieved with open-source tools like GCov
  - ▷ NASA includes our approach in the Software Engineering and Assurance Handbook, NASA-HDBK-2203
- Verification of of compiler/toolchain-added object code:
  - ▷ Identifying hidden calls to library functions
  - ▷ Assessing different branching structure in (optimized) object code
  - ▷ Can also be achieved with open-source tools

Lessons Learned

- We conducted interviews to learn current Category A software challenges, involving ESA ESTEC Software and PA experts.
- The second completed European Category A software is based on wrong assumptions and flawed methods:
  - ▷ Atomic logical decision to avoid MCDC coverage, loosing the potential of MCDC error detection
  - ▷ Blind application of methods to trace object code to source code with wrong disassembly interpretation

Acknowledgements

The study is funded by the European Space Agency under contract 4000138220/22/NL/AS/adu

Availability and Contact

- MCDC Checker on GitLab:
  `https://gitlab.com/gtd-gmbh/mcdc-checker/mcdc-checker`
- GTD GmbH Information and Support:
  `https://gtd-gmbh.de/`
  `gtd@gtd-gmbh.de`

Tool 1 — MCDC Checker

Unoptimized object code implements decisions as BDDs, which coverage happens to be equivalent to MCDC when tree-like.

Our MCDC-Checker:

- walks the abstract syntax tree of your C/C++ source code
- checks each decision BDD whether it is tree-like
- proposes a reordering if that results in a tree-like BDD
- MCDC coverage can then be assessed by using GCov as usual

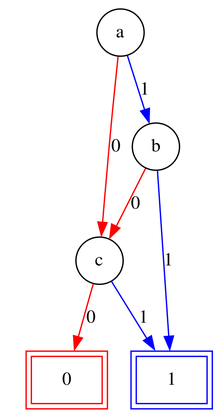


Figure: (a and b) or c

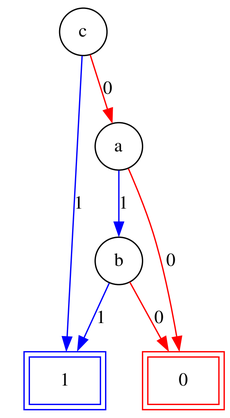


Figure: c or (a and b)

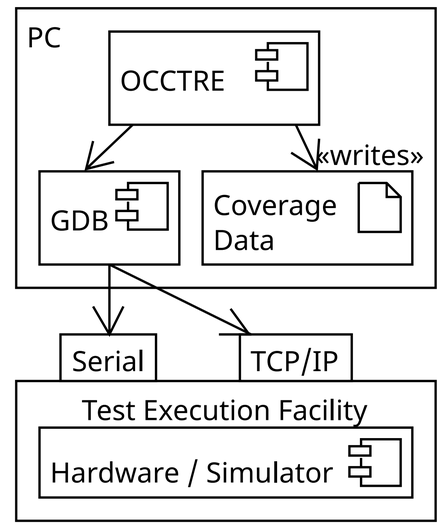
Tool 2 — ELF Checker

- Compares debug, stripped and binary versions of your application software
  ⇒ Instructions and data are equivalent
- Checks that sections from the debug version are correctly embedded into the flashable binary (correct address and content)
- Lists length, alignment, type and flags of all sections
- Shows location and alignment of all sections in the flashable binary

	Address	Length	Alignment	Address	Length	Alignment		Offset	Length	Alignment	
.comment	0x00000000	0x00000063	0x01	0x00000000	0x00000063	0x01	SHT_PROGBITS, MERGE, STRINGS	-	-	-	-
.debug_abbrev	0x00000000	0x000245C4	0x01	-	-	-	SHT_PROGBITS, -	-	-	-	-
.debug_aranges	0x00000000	0x00002930	0x08	-	-	-	SHT_PROGBITS, -	-	-	-	-
.debug_frame	0x00000000	0x00004554	0x04	-	-	-	SHT_PROGBITS, -	-	-	-	-
.debug_info	0x00000000	0x001143EC	0x01	-	-	-	SHT_PROGBITS, -	-	-	-	-
.debug_line	0x00000000	0x0004855E	0x01	-	-	-	SHT_PROGBITS, -	-	-	-	-
.debug_loc	0x00000000	0x00065F51	0x01	-	-	-	SHT_PROGBITS, -	-	-	-	-
.debug_ranges	0x00000000	0x00016970	0x01	-	-	-	SHT_PROGBITS, -	-	-	-	-
.debug_str	0x00000000	0x0000E75E	0x01	-	-	-	SHT_PROGBITS, MERGE, STRINGS	-	-	-	-
.gnu.attributes	0x00000000	0x00000010	0x01	0x00000000	0x00000010	0x01	SHT_GNU_ATTRIBUTES	-	-	-	-
.shstrtab	0x00000000	0x000000C3	0x01	0x00000000	0x0000004D	0x01	SHT_STRTAB	-	-	-	Data mismatch
.strtab	0x00000000	0x00005211	0x01	-	-	-	SHT_STRTAB	-	-	-	-
.symtab	0x00000000	0x00007440	0x04	-	-	-	SHT_SYMTAB	-	-	-	-
.text	0x00000000	0x000285C0	0x20	0x00000000	0x000285C0	0x20	SHT_PROGBITS	ALLOC, EXECINSTR	0x00000000	0x000285C0	0x20	Data mismatch
.rtemsroset	0x000285C0	0x00000060	0x08	0x000295C0	0x00000060	0x08	SHT_PROGBITS	ALLOC	0x000285C0	0x00000060	0x08	Addr mismatch
padding	-	-	-	-	-	-	-	-	0x00028620	0x00000020	-	Not Zero
.rtemsstack	0x00028640	0x00002000	0x40	0x00028640	0x00002000	0x40	SHT_NOBITS	ALLOC, WRITE	0x00028640	0x00002000	0x40	Flags mismatch
.data	0x0002A640	0x00003C80	0x08	0x0002A640	0x00003C80	0x08	SHT_PROGBITS	ALLOC, WRITE	0x0002A640	0x00003C80	0x08	Type mismatch
.bss	0x0002E2C0	0x00410C30	0x40	0x0002E2C0	0x00410C30	0x40	SHT_NOBITS	ALLOC, WRITE	-	-	-	-
padding	-	-	-	-	-	-	-	-	0x0002E2C0	0x00000060	-	Not Zero

Tool 3 — OCCTRE (Record Object Code Coverage)

- Executes the software image/binary step by step and records instruction information
- GDB based
- Generates processable object code coverage data like CSV, JSON or Markdown
- Highly configurable (platforms, facilities, disassemblers)
- Highly scriptable (python interface)



Address	Label(+off)	Instruction (hex)	Instruction	Opcode	Executed	Branch taken	Code File
0x0000f1a0	boot_card+0	9d e3 bf a0	save %sp, -96, %sp	save	1	set()	bsps/shared/start/bootcard.c:42
0x0000f1a4	boot_card+4	91 d0 20 09	ta 9	ta	1	{'0x0000f1a8'}	cpukit/score/cpu/sparc/include/rtems/score/sparc.h:429
0x0000f1a8	boot_card+8	03 00 00 61	sethi %hi(0x18400), %g1	sethi	1	set()	bsps/shared/start/bootcard.c:53
0x0000f1ac	boot_card+12	7f ff f9 d3	call 0xd8f8 <rtems_initialize>	call	1	set()	bsps/shared/start/bootcard.c:55
0x0000f1b0	boot_card+16	f0 20 62 70	st %i0, [%g1 + 0x270]	st	1	set()	bsps/shared/start/bootcard.c:55
0x0000d8f8	rtems_initialize_executive+0	9d e3 bf a0	save %sp, -96, %sp	save	1	set()	cpukit/sapi/src/exinit.c:130
0x0000d8fc	rtems_initialize_executive+4	3b 00 00 42	sethi %hi(0x10800), %i5	sethi	1	set()	cpukit/include/rtems/linkersets.h:151
0x0000d900	rtems_initialize_executive+8	39 00 00 42	sethi %hi(0x10800), %i4	sethi	1	set()	cpukit/sapi/src/exinit.c:134
0x0000d904	rtems_initialize_executive+12	ba 17 61 10	or %i5, 0x110, %i5	or	1	set()	cpukit/include/rtems/linkersets.h:152

Tool 4 — asm2cfg (Analyze Object Code Coverage)

- Generates the basic-block structure of investigated function
- Reads in object code coverage data (e.g. produced by OCCTRE)
- Generates human readable diagrams for missing test cases
- Highly configurable (platforms, facilities, disassemblers)
- Highly scriptable (python interface)