Code Minimization Technology for SIL2LinuxMP – Qualifying Linux® for Functional Safety

Jul/13/2016

Hitachi, Ltd.

Taku Shimosawa

Hitachi India Pvt. Ltd.

Desai, Krishnaji

HITACHI
Inspire the Next

Contents

1. Functional Safety in OSS/Linux
2. SIL2LinuxMP Organization & Strategy
3. Static and Dynamic analysis & test
4. Minimization Technique
5. Conclusion and Future Prospects

1. Functional Safety in OSS/Linux

When Linux runs in control units in cars…

“Segmentation Fault”

in the brake system?

This should never happen !

Safety Assurance in OSS/Linux

Growing demands for OSS/Linux in Safety Critical domains.

- Automobiles
- Industrial Control Systems
- Traffic Management Systems
- …

?

How to prove safety in OSS ?

Problem:

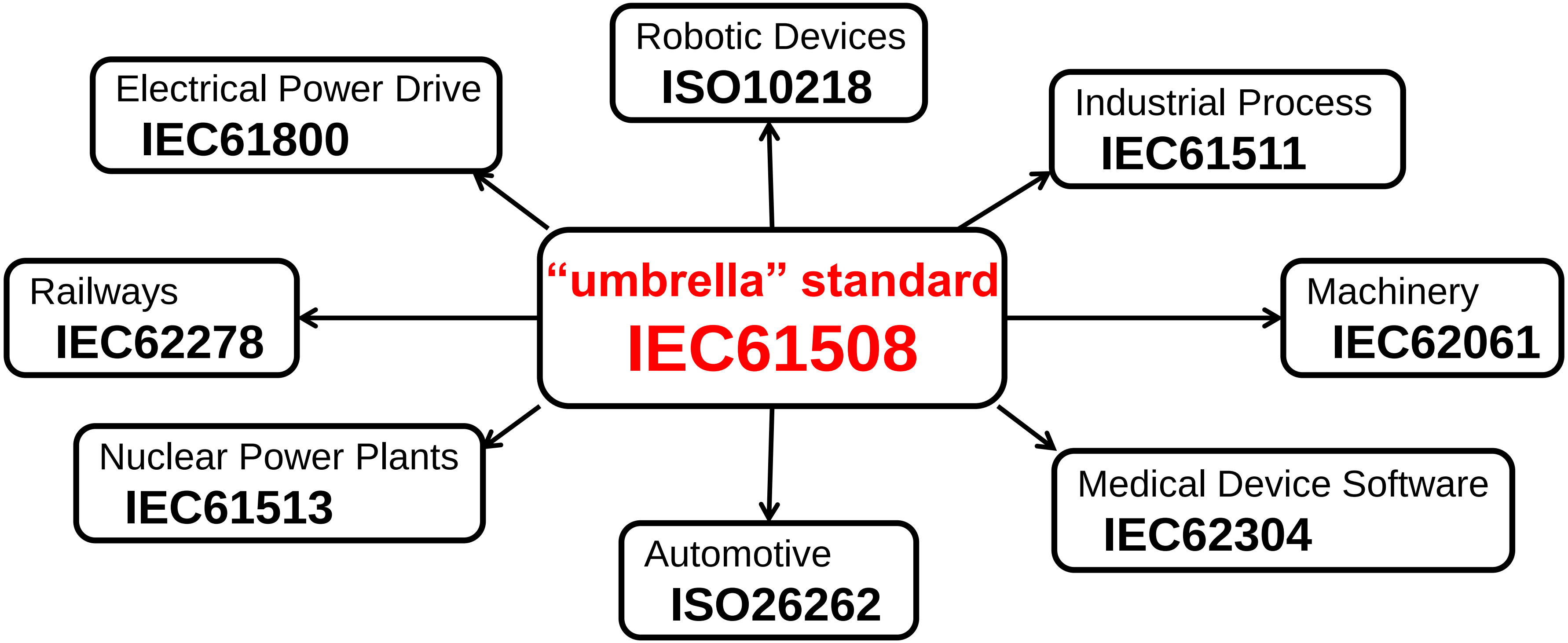
OSS project **does not guarantee** required safety level.

Insufficient development **evidences** for assessment.

Functional Safety Standards

“Compliance to the Standards” is becoming mandatory.

-> However, existing standards are hardly applicable to OSS.



“umbrella” standard
IEC61508

Robotic Devices
ISO10218

Electrical Power Drive
IEC61800

Industrial Process
IEC61511

Railways
IEC62278

Machinery
IEC62061

Nuclear Power Plants
IEC61513

Automotive
ISO26262

Medical Device Software
IEC62304

Challenge:

Establish a general certification process for OSS/Linux.

2. SIL2LinuxMP Organization & Strategy

OSADL & SIL2LinuxMP Project

OSADL: Open Source Automation Development Lab

SIL2LinuxMP Project:
Aims to establish a process to certify OSS/Linux with IEC61508.

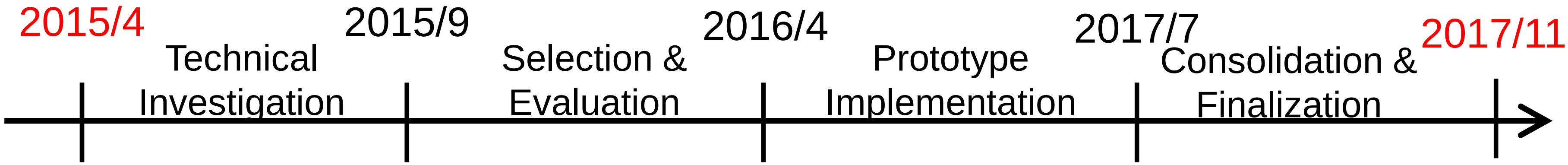
Target scope:
Linux Kernel, glibc, BusyBox.

Minimal configuration as to ease complexity in assessment

Participants:
Organizer: OSADL, OpenTech
4 Full Partners: BMW Car-IT, KUKA, A&R Tech, SensorTechnik
9 Reviewing Partners: **Hitachi**, Renesas, etc…
Consulting body : TÜV SÜD
Certification authority: TÜV Rheinland

Provides usecases as certification targets

Plan (Hitachi's View):



https://www.osadl.org/SIL2LinuxMP.sil2-linux-project.0.html

SIL2LinuxMP Strategy

How to comply IEC61508 with OSS/Linux ??

IEC61508 Part3 7.4.2.12:

Route 1_S: “Standard Compliant Development” ?

NO, OSS is not developed this way by itself.

Route 2_S:“Proven in Use” ?

NO, too time-consuming, too expensive strategy.

Also, vulnerable to even slight changes of SW/HW.

Route 3_S: “Compliant non-compliant Development” ?

YES, only this is the suitable way for OSS.
This route complies non-compliant software by compensating missing evidences

SIL2LinuxMP Strategy

Route3$_S$: Compliant non-compliant Development

Step 1:

Assess used pre-existing COTS* component,

identify missing evidences or non-compliant process.

- Ex: missing development document, untested codes etc.

Step 2:

Plan how to compensate **missing evidences** & compliant processes.

- Ex: automatic testing, metrics calculation & regressions

Step 3:

Apply the planned processes, review/**assess the outcome**.

- Ex: make arguments by obtained coverage metrics.

* COTS: Commercial off-the-shelf

SIL2LinuxMP Strategy

Route3$_S$: Compliant non-compliant Development

Step 1:

Assess used pre-existing COTS* component, **identify missing evidences** or non-compliant process.

- Ex: missing development document, untested codes etc.

"**Analysis Technique**" is the Key Factor !

Step 2:

Plan how to compensate **missing evidences** & compliant processes.

- Ex: automatic testing, metrics calculation & regressions

Step 3:

Apply the planned processes, review/**assess the outcome**.

- Ex: make arguments by obtained coverage metrics.

* COTS: Commercial off-the-shelf

3. Static and Dynamic analysis & test

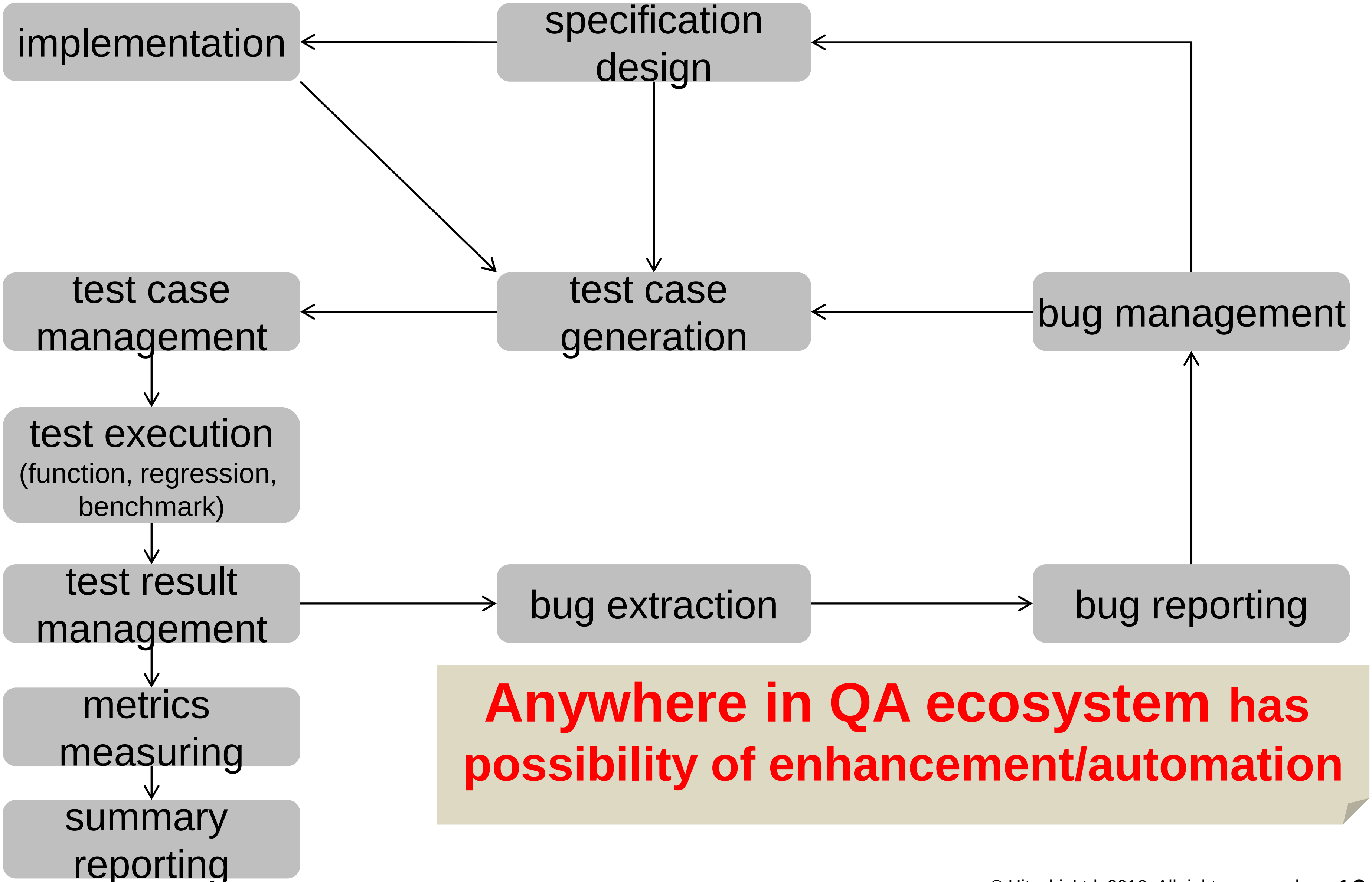
Tasks in analysis & test with V&V

Tool for Test Automation ??

test execution
(function, regression, benchmark)

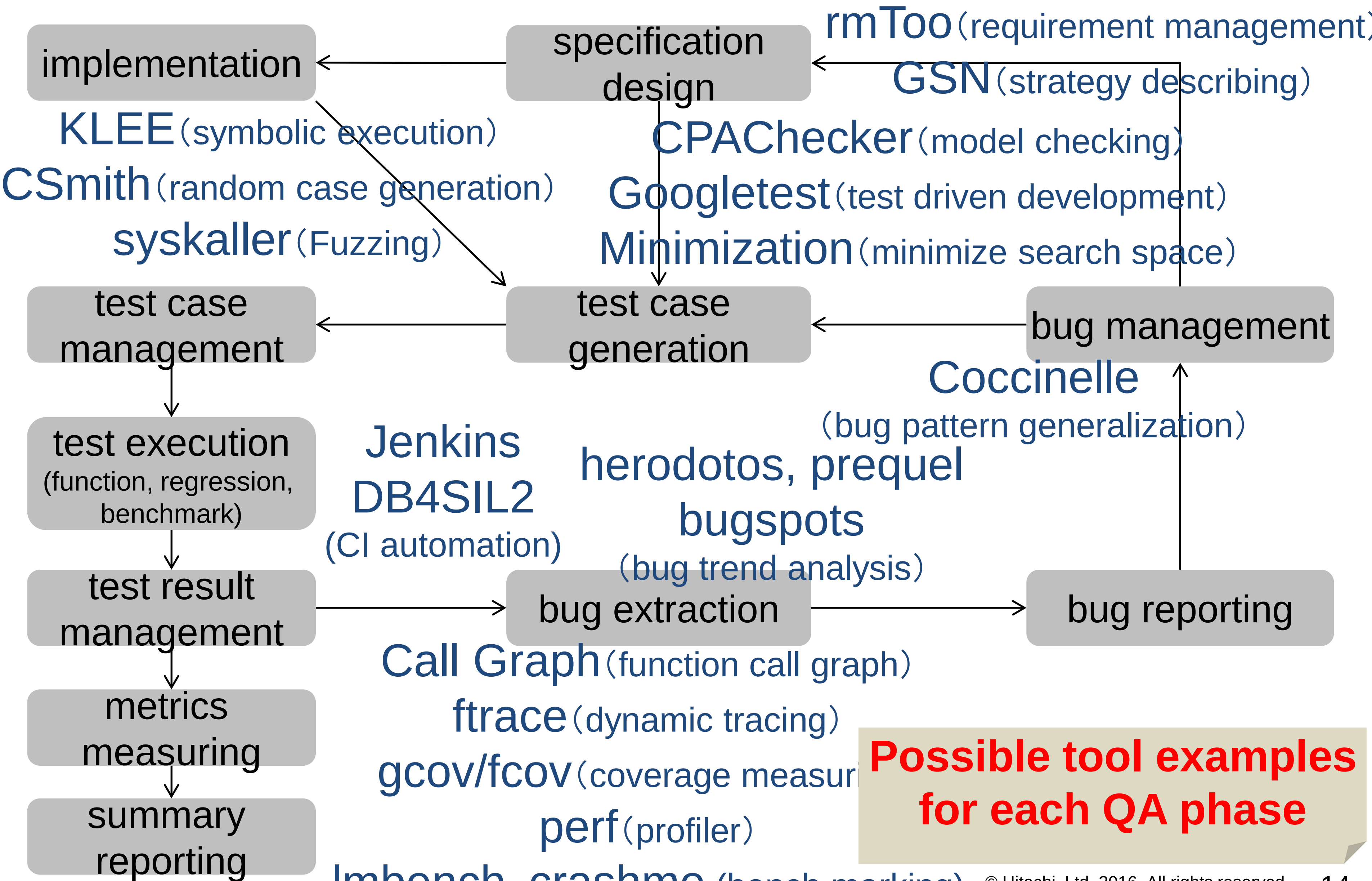
QA is not only about Test Execution !!

Tasks in analysis & test with V&V



Anywhere in QA ecosystem has possibility of enhancement/automation

Tools that help V&V and QA tasks



implementation

specification design

rmToo(requirement management

GSN(strategy describing)

KLEE(symbolic execution)

CSmith(random case generation)

syskaller(Fuzzing)

CPAChecker(model checking)

Googletest(test driven development)

Minimization(minimize search space)

test case management

test case generation

bug management

Coccinelle
(bug pattern generalization)

test execution
(function, regression, benchmark)

Jenkins
DB4SIL2
(CI automation)

herodotos, prequel
bugspots
(bug trend analysis)

test result management

bug extraction

bug reporting

Call Graph(function call graph)

ftrace(dynamic tracing)

gcov/fcov(coverage measuri

perf(profiler)

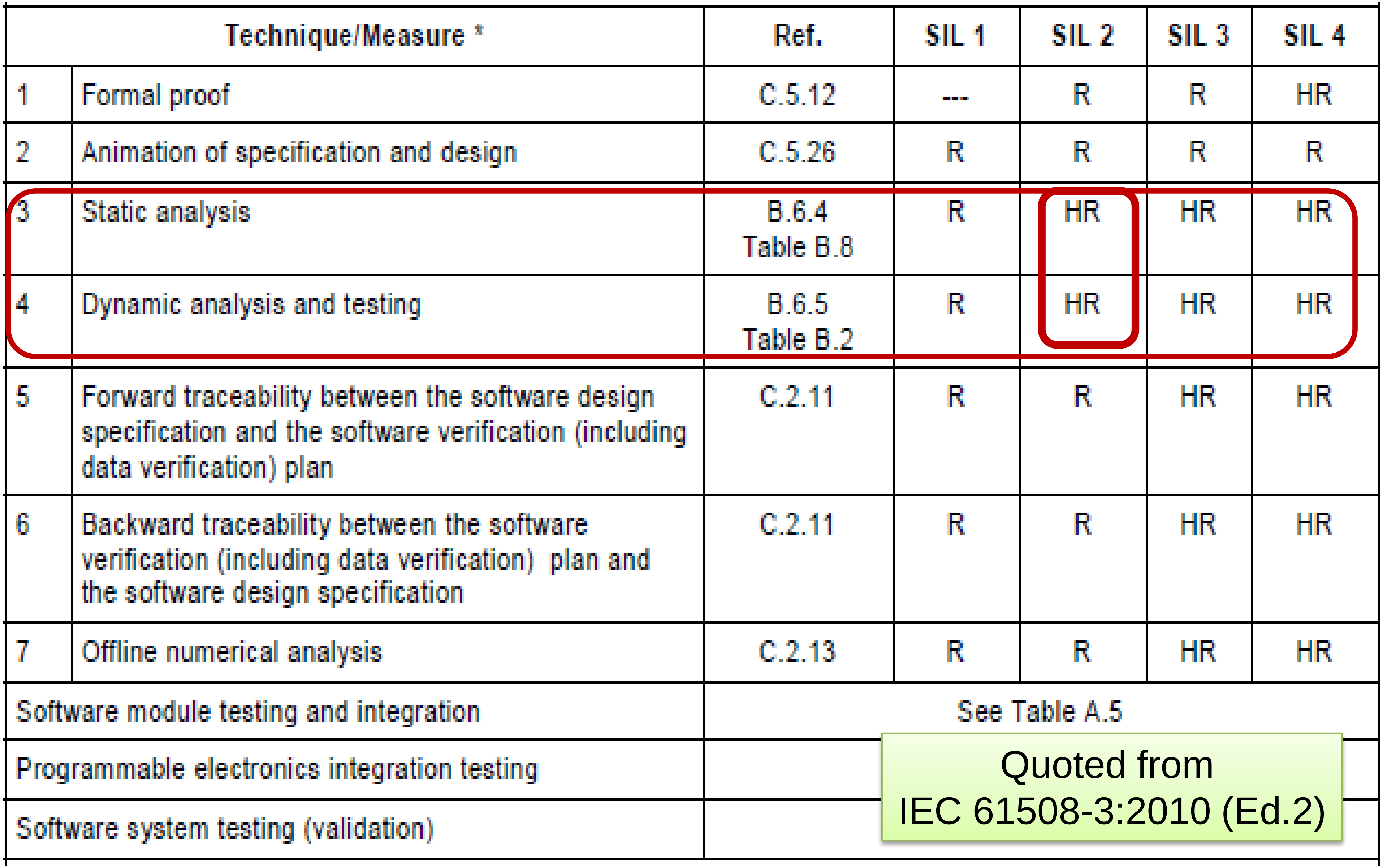
metrics measuring

summary reporting

lmbench, crashme (bench marking)

Possible tool examples for each QA phase

Standards Table A9 Software verification



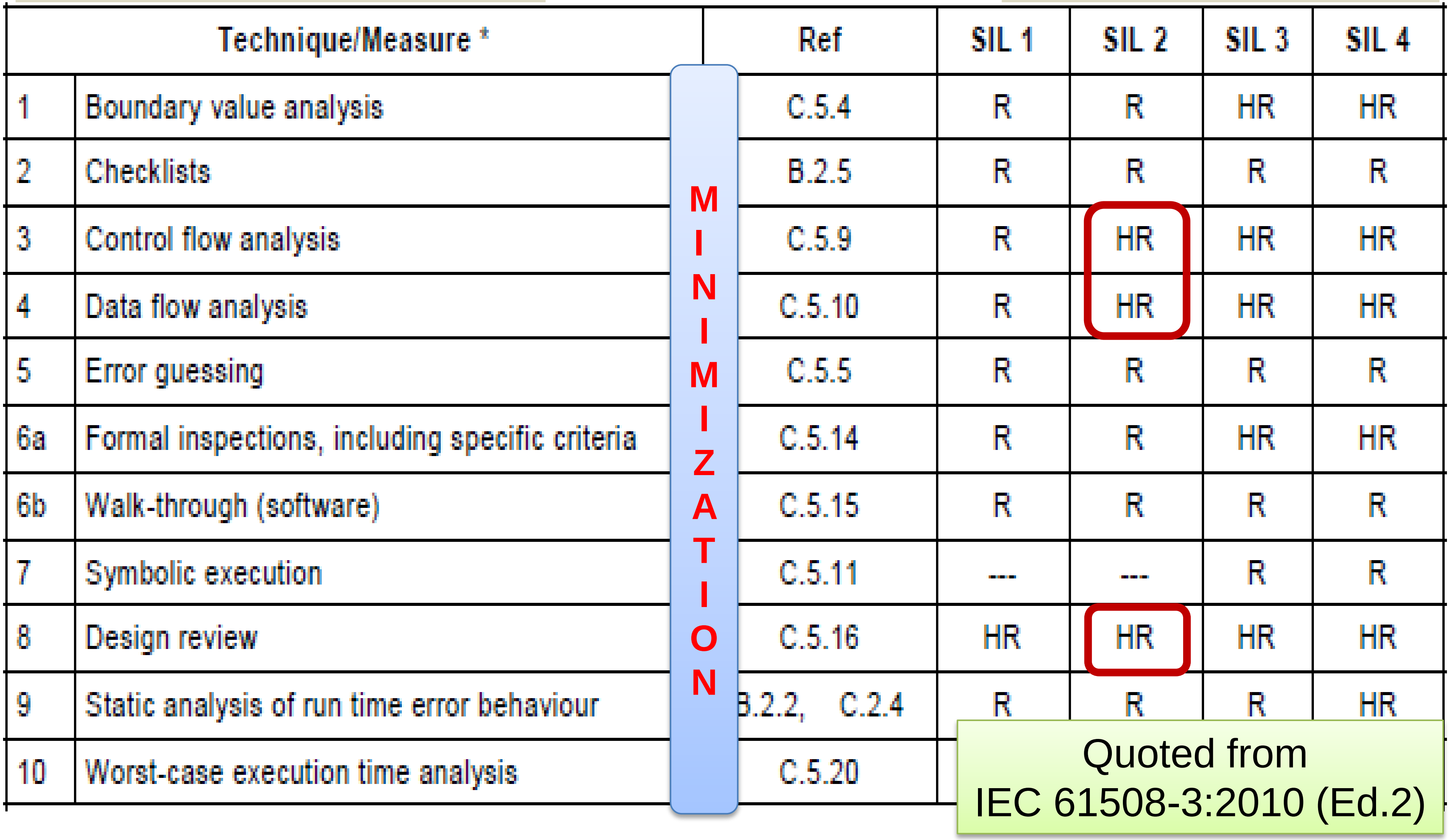
Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1	Formal proof	C.5.12	---	R	R	HR
2	Animation of specification and design	C.5.26	R	R	R	R
3	Static analysis	B.6.4 Table B.8	R	HR	HR	HR
4	Dynamic analysis and testing	B.6.5 Table B.2	R	HR	HR	HR
5	Forward traceability between the software design specification and the software verification (including data verification) plan	C.2.11	R	R	HR	HR
6	Backward traceability between the software verification (including data verification) plan and the software design specification	C.2.11	R	R	HR	HR
7	Offline numerical analysis	C.2.13	R	R	HR	HR
Software module testing and integration		See Table A.5				
Programmable electronics integration testing						
Software system testing (validation)						

Quoted from
IEC 61508-3:2010 (Ed.2)

Standards Table B.8 Static analysis

HITACHI
Inspire the Next

QA → Safety Assurance — requires → Evidence chain

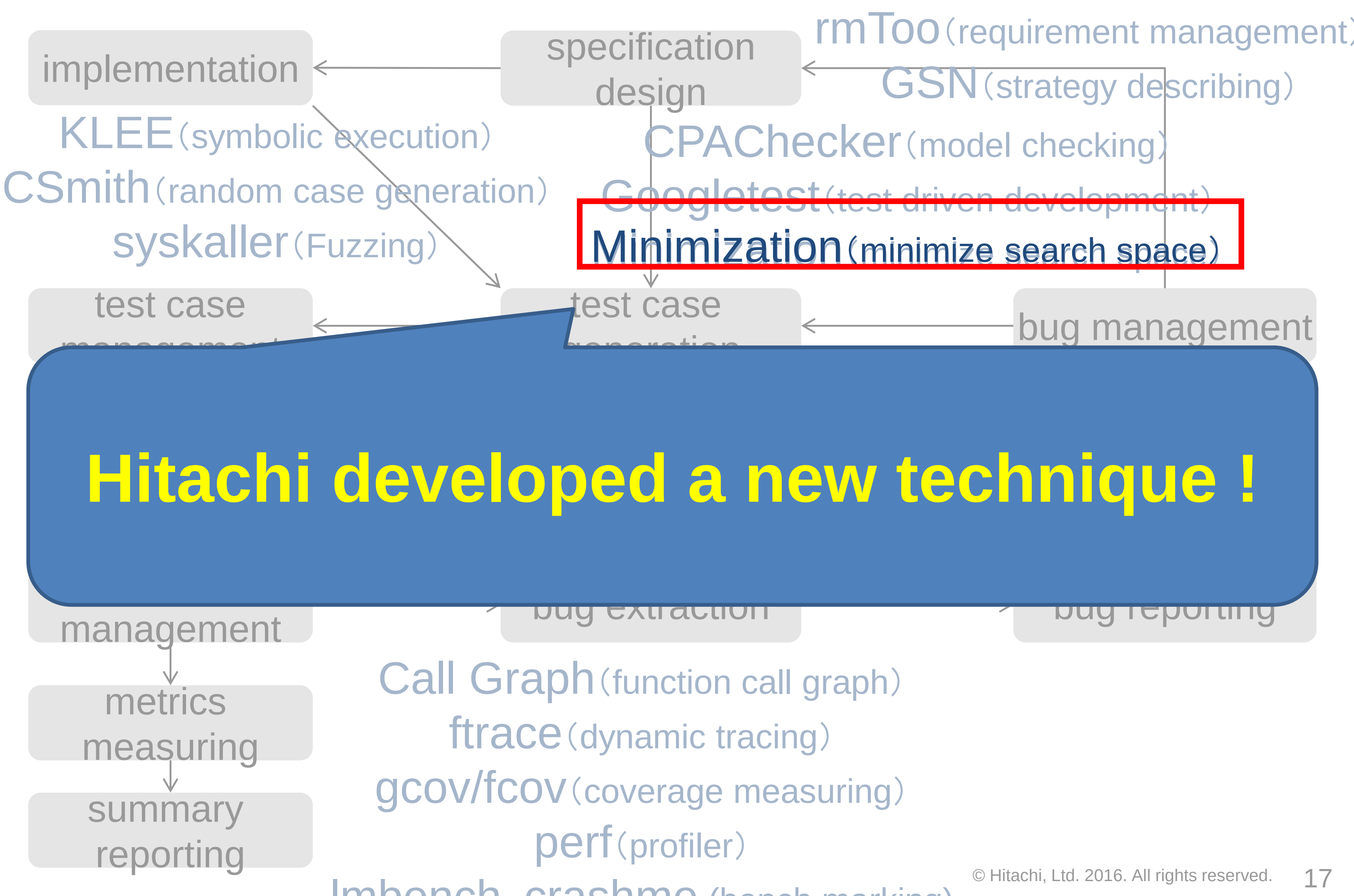


Technique/Measure *		Ref	SIL 1	SIL 2	SIL 3	SIL 4
1	Boundary value analysis	C.5.4	R	R	HR	HR
2	Checklists	B.2.5	R	R	R	R
3	Control flow analysis	C.5.9	R	HR	HR	HR
4	Data flow analysis	C.5.10	R	HR	HR	HR
5	Error guessing	C.5.5	R	R	R	R
6a	Formal inspections, including specific criteria	C.5.14	R	R	HR	HR
6b	Walk-through (software)	C.5.15	R	R	R	R
7	Symbolic execution	C.5.11	---	---	R	R
8	Design review	C.5.16	HR	HR	HR	HR
9	Static analysis of run time error behaviour	B.2.2, C.2.4	R	R	R	HR
10	Worst-case execution time analysis	C.5.20				

MINIMIZATION

Quoted from
IEC 61508-3:2010 (Ed.2)

Tools that help V&V and QA tasks


implementation
specification design
rmToo (requirement management)
GSN (strategy describing)
KLEE (symbolic execution)
CPAChecker (model checking)
CSmith (random case generation)
Googletest (test driven development)
syskaller (Fuzzing)
Minimization (minimize search space)
test case management
test case generation
bug management
management
bug extraction
bug reporting
metrics measuring
summary reporting
Call Graph (function call graph)
ftrace (dynamic tracing)
gcov/fcov (coverage measuring)
perf (profiler)
lmbench, crashme (bench marking)

Hitachi developed a new technique !

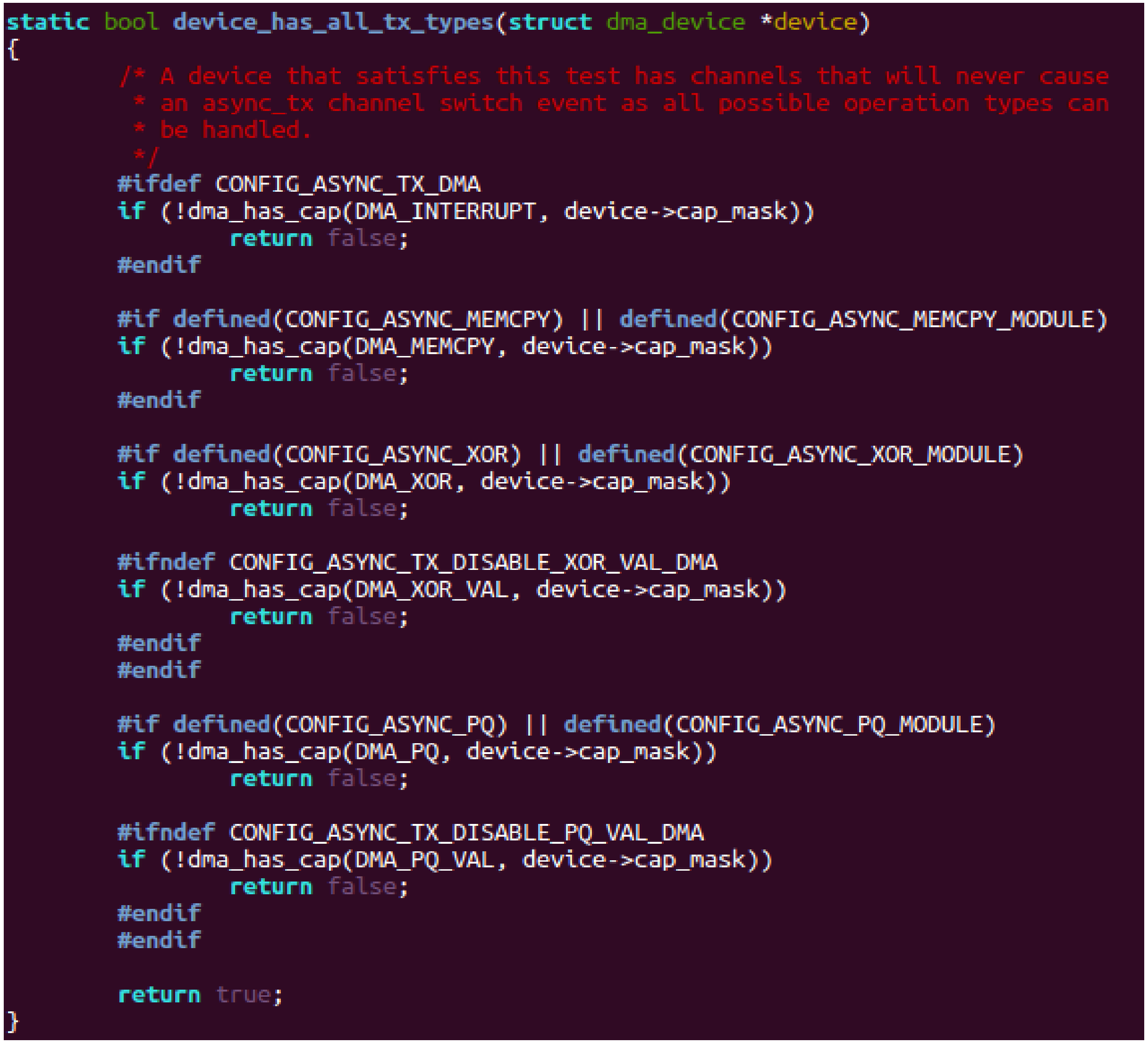
4. Minimization Technique

# “#ifdef disasters”

/drivers/dma/dmaengine.c

- The #ifdefs makes the code hard to:
  - Review
  - Debug
  - Maintain
  - Verify

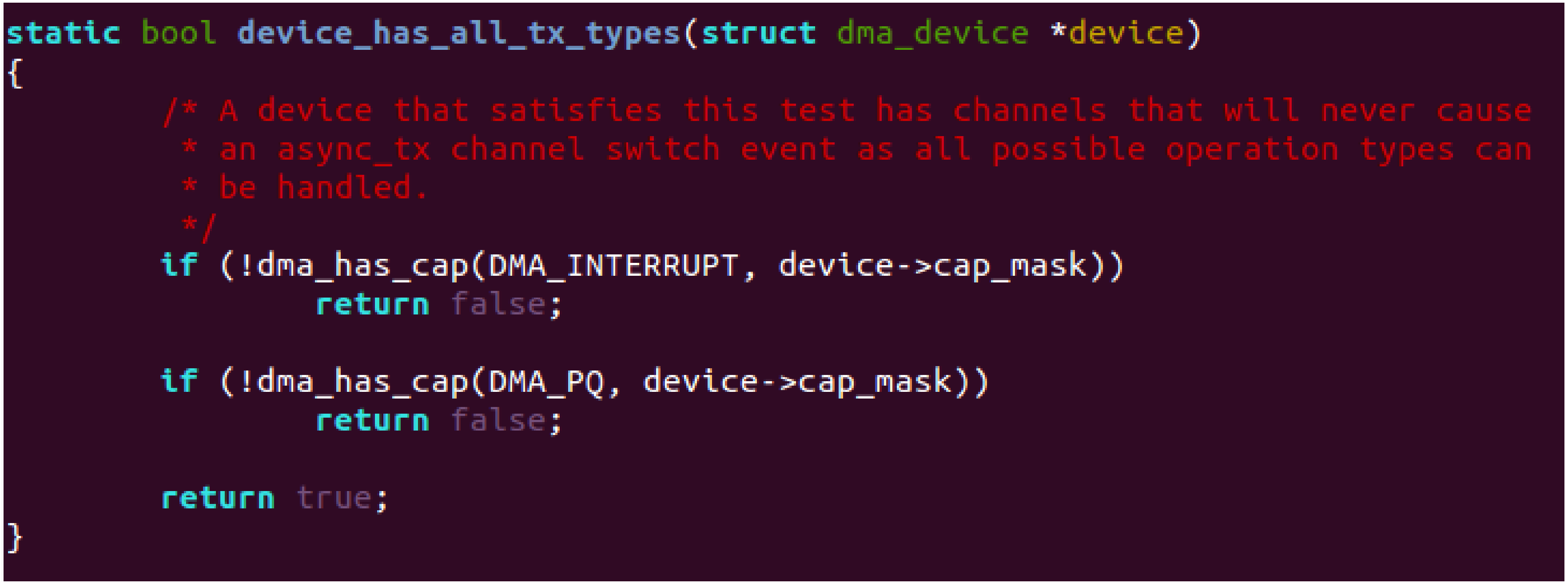
However,



```
static bool device_has_all_tx_types(struct dma_device *device)
{
        /* A device that satisfies this test has channels that will never cause
         * an async_tx channel switch event as all possible operation types can
         * be handled.
         */
        #ifdef CONFIG_ASYNC_TX_DMA
        if (!dma_has_cap(DMA_INTERRUPT, device->cap_mask))
                return false;
        #endif

        #if defined(CONFIG_ASYNC_MEMCPY) || defined(CONFIG_ASYNC_MEMCPY_MODULE)
        if (!dma_has_cap(DMA_MEMCPY, device->cap_mask))
                return false;
        #endif

        #if defined(CONFIG_ASYNC_XOR) || defined(CONFIG_ASYNC_XOR_MODULE)
        if (!dma_has_cap(DMA_XOR, device->cap_mask))
                return false;

        #ifndef CONFIG_ASYNC_TX_DISABLE_XOR_VAL_DMA
        if (!dma_has_cap(DMA_XOR_VAL, device->cap_mask))
                return false;
        #endif
        #endif

        #if defined(CONFIG_ASYNC_PQ) || defined(CONFIG_ASYNC_PQ_MODULE)
        if (!dma_has_cap(DMA_PQ, device->cap_mask))
                return false;

        #ifndef CONFIG_ASYNC_TX_DISABLE_PQ_VAL_DMA
        if (!dma_has_cap(DMA_PQ_VAL, device->cap_mask))
                return false;
        #endif
        #endif

        return true;
}
```

# “#ifdef disasters”

/drivers/dma/dmaengine.c

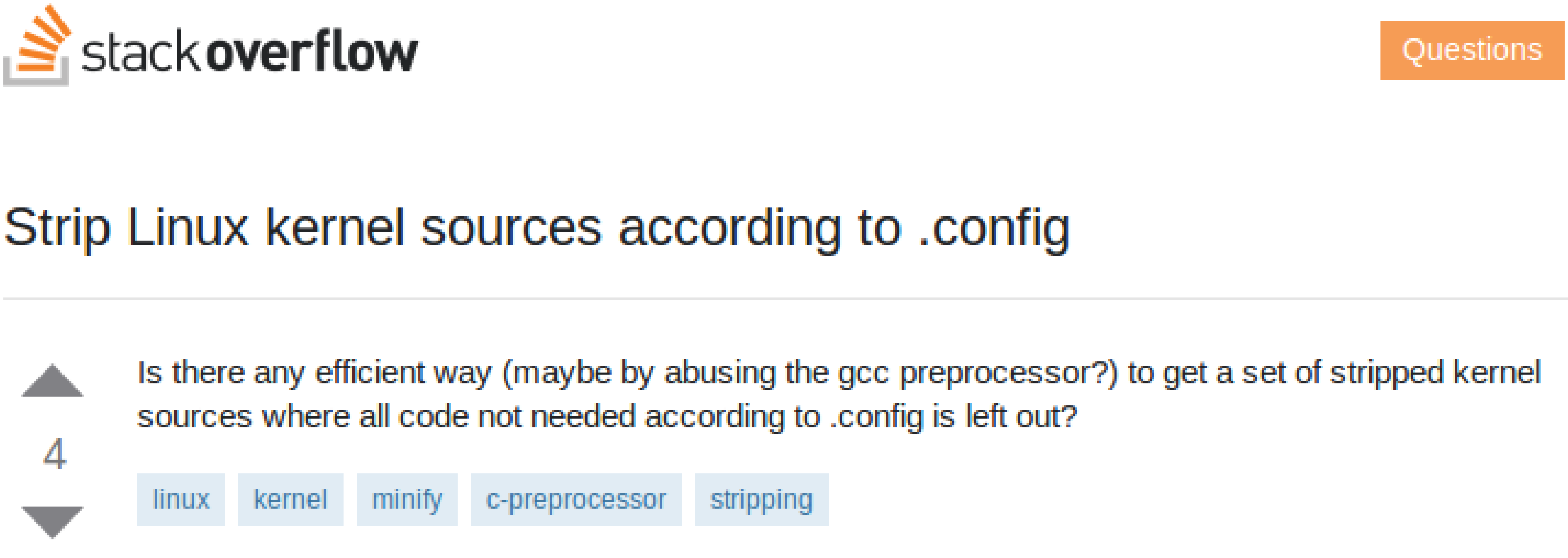


```
static bool device_has_all_tx_types(struct dma_device *device)
{
        /* A device that satisfies this test has channels that will never cause
         * an async_tx channel switch event as all possible operation types can
         * be handled.
         */
        if (!dma_has_cap(DMA_INTERRUPT, device->cap_mask))
                return false;

        if (!dma_has_cap(DMA_PQ, device->cap_mask))
                return false;

        return true;
}
```

If code is free from #ifdef blocks then, analysis shall be more effective.

Is there a way ?

Is there a way ??



Strip Linux kernel sources according to .config

Is there any efficient way (maybe by abusing the gcc preprocessor?) to get a set of stripped kernel sources where all code not needed according to .config is left out?

http://stackoverflow.com/questions/7353640/strip-linux-kernel-sources-according-to-config

✓ Possible if we tweak **`gcc`** preprocessor options.

How to do it for the whole source tree ??

The GREP Approach for minimization

HITACHI
Inspire the Next

http://stackoverflow.com/questions/7353640/strip-linux-kernel-sources-according-to-config

Use of GREP (Approach-I)

- Requires complete build in advance.
- Text parsing has to be acquired from build log.
- Source code modification to remove redundant code.

1
```
make KBUILD_VERBOSE=1 | tee build.log
```

2
```
grep 'gcc' build.log > gccbuild.log
```

3
```
sed 's/ -c -o /  /g' gccbuild.log > plainbuild.log
```

4
```
grep –v ‘#include’ <PATH> | gcc –E –fdirectives-only –undef  <GCC Stripped Code>
grep –v ‘^#’
```

5
```
mkdir -p ~/NewKernel/scripts/basic/
grep -v '#include' scripts/basic/fixdep.c| gcc -E -fdirectives-only -undef
gcc -Wp,-MD,scripts/basic/.fixdep.d -Wall -Wmissing-prototypes -Wstrict-
prototypes -O2 -fomit-frame-pointer -std=gnu89     -o scripts/basic/fixdep
scripts/basic/fixdep.c   - | grep -v '^#' > ~/NewKernel/scripts/basic/fixdep.c
```

LIMITATIONS

Too much user Involvement!!!

2 phases of GCC process

No integration with MakeFile

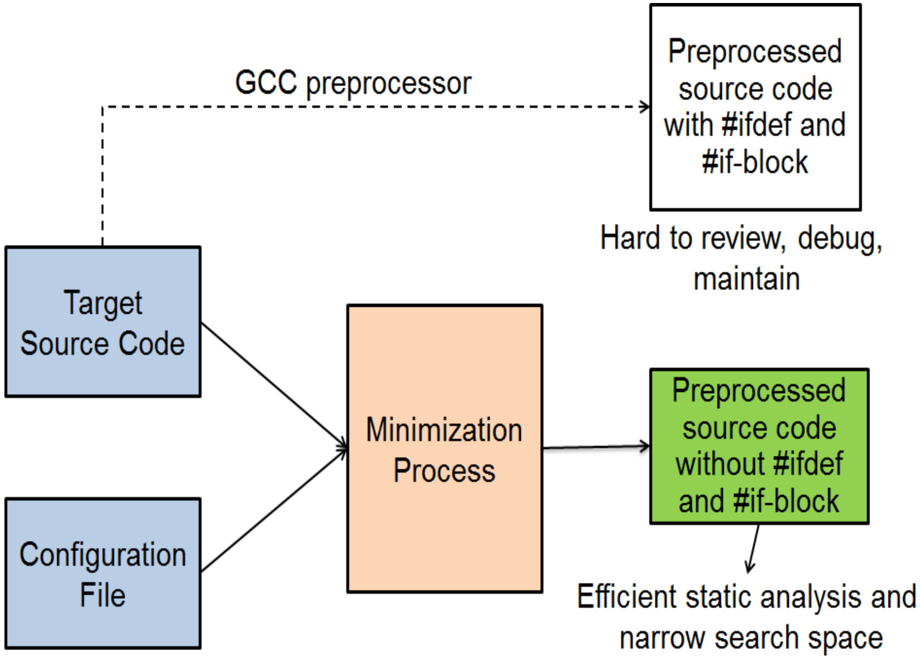
Expanded Headers persist

The Minimization Approach

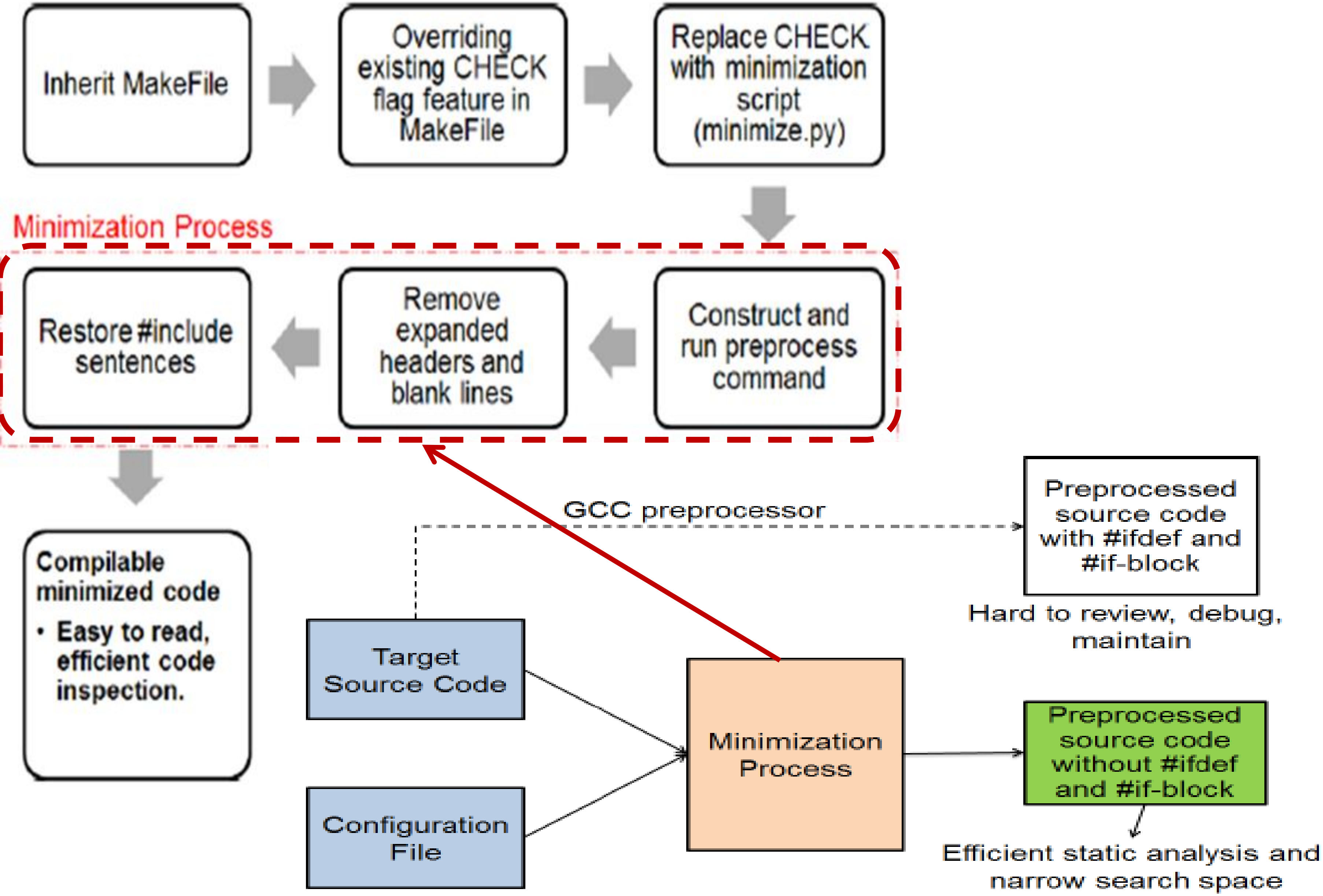
- The minimization approach tweaks integrated MakeFile options to produce compilable stripped code.

- Signifies efficient way to get a set of stripped kernel source code based on a .config file.

- Generate source tree where;
  - Unused #ifdef, #if blocks have been removed
  - #include and #define lines are preserved
  - Only used source files exist
  - Produces the same binary file as the original tree


GCC preprocessor
Preprocessed source code with #ifdef and #if-block
Hard to review, debug, maintain
Target Source Code
Configuration File
Minimization Process
Preprocessed source code without #ifdef and #if-block
Efficient static analysis and narrow search space

Minimization flow



Inherit MakeFile

Overriding existing CHECK flag feature in MakeFile

Replace CHECK with minimization script (minimize.py)

Minimization Process

Construct and run preprocess command

Remove expanded headers and blank lines

Restore #include sentences

Compilable minimized code

- Easy to read, efficient code inspection.

GCC preprocessor

Target Source Code

Configuration File

Minimization Process

Preprocessed source code with #ifdef and #if-block

Hard to review, debug, maintain

Preprocessed source code without #ifdef and #if-block

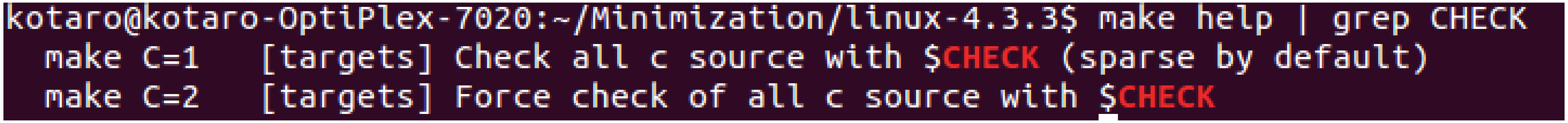
Efficient static analysis and narrow search space

Implementation

- `Makefile` integration
  - – Override existing `CHECK` flag feature
- Minimizing procedure
  - – Preprocess, expanded header restoration
- Binary verification
  - – Compare “minimized binary” and the original

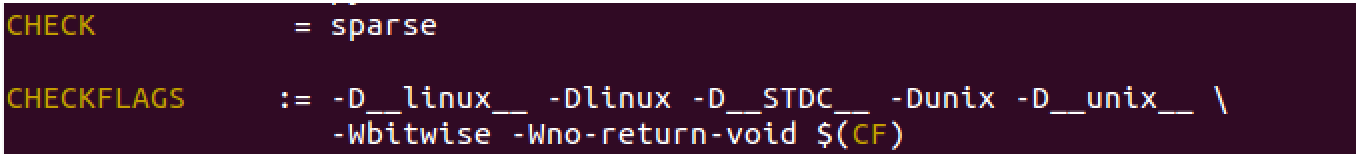
Makefile integration

- Override existing CHECK feature in kernel Makefile



```
kotaro@kotaro-OptiPlex-7020:~/Minimization/linux-4.3.3$ make help | grep CHECK
  make C=1   [targets] Check all c source with $CHECK (sparse by default)
  make C=2   [targets] Force check of all c source with $CHECK
```

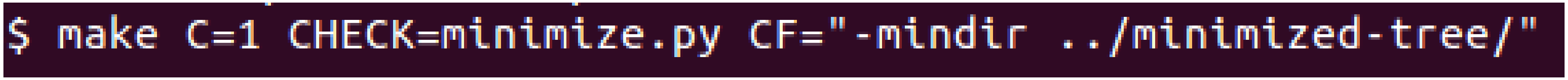
Makefile of the root directory:



```
CHECK            = sparse

CHECKFLAGS      := -D__linux__ -Dlinux -D__STDC__ -Dunix -D__unix__ \
                   -Wbitwise -Wno-return-void $(CF)
```

- Minimization script(minimize.py) usage:

Replace CHECK with minimize.py so make can process minimization



```
$ make C=1 CHECK=minimize.py CF="-mindir ../minimized-tree/"
```

In make process, “minimize.py” will receive the same option as the compile flags of each source file, plus $CHECKFLAGS variable.

ON THE FLY GENERATION (no post processing)

Minimization procedure

1. Preprocess the source files

   gcc –E –fdirectives-only

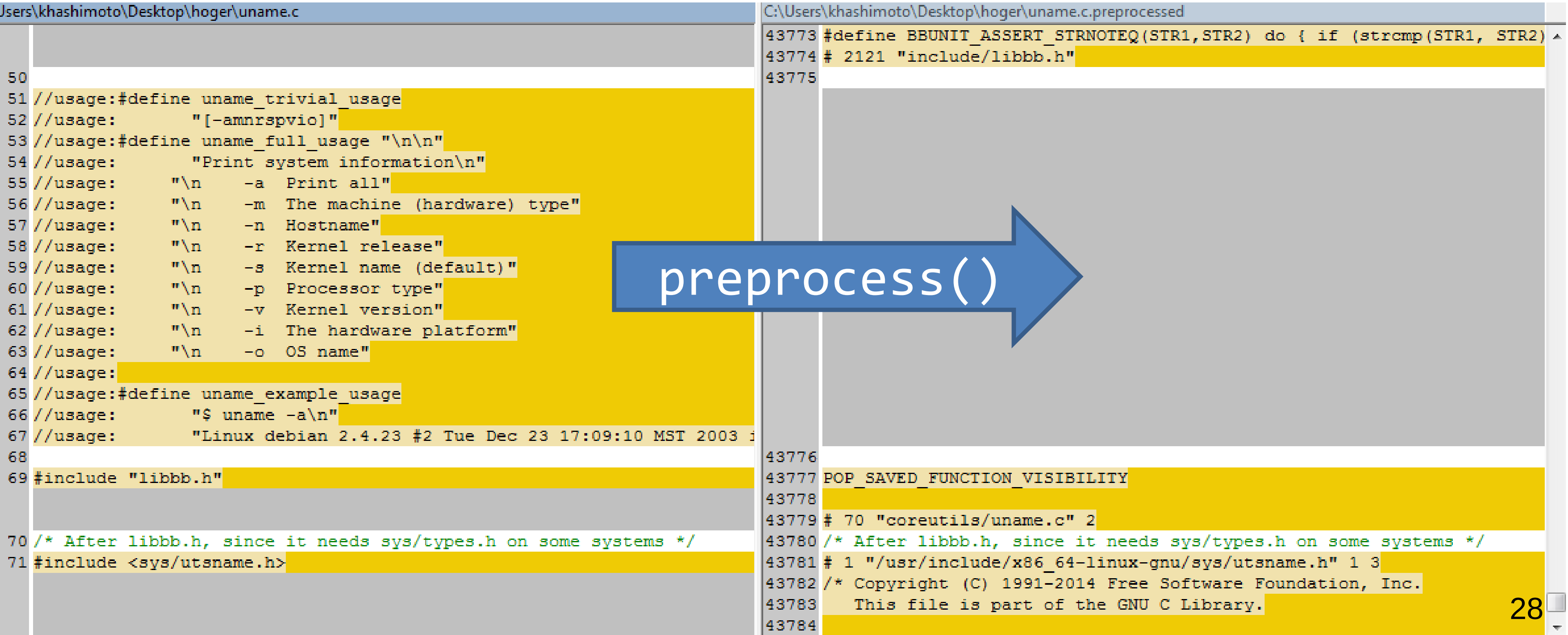
   ⇒ #ifdef block disappears, #include gets expanded,
   but #define macros are preserved.

2. Identify & delete the expanded header contents
   - Use clues(linemarkers) that exist in the preprocessed file
   - Example of linemarkers: # 30 “/usr/include/sys/stsname.h” 2

3. Restore #include sentences
   - Copy relevant #include lines from the original source

Preprocess the source files

- preprocess() function in minimize.py
  - Takes gcc options passed via Makefile
  - Appends “-E –fdirectives-only” flags
  - Perform preprocess for the target C file



Jsers\khashimoto\Desktop\hoger\uname.c

```
//usage:#define uname_trivial_usage
//usage:       "[-amnrspvio]"
//usage:#define uname_full_usage "\n\n"
//usage:       "Print system information\n"
//usage:     "\n    -a  Print all"
//usage:     "\n    -m  The machine (hardware) type"
//usage:     "\n    -n  Hostname"
//usage:     "\n    -r  Kernel release"
//usage:     "\n    -s  Kernel name (default)"
//usage:     "\n    -p  Processor type"
//usage:     "\n    -v  Kernel version"
//usage:     "\n    -i  The hardware platform"
//usage:     "\n    -o  OS name"
//usage:
//usage:#define uname_example_usage
//usage:       "$ uname -a\n"
//usage:       "Linux debian 2.4.23 #2 Tue Dec 23 17:09:10 MST 2003 i

#include "libbb.h"
/* After libbb.h, since it needs sys/types.h on some systems */
#include <sys/utsname.h>
```

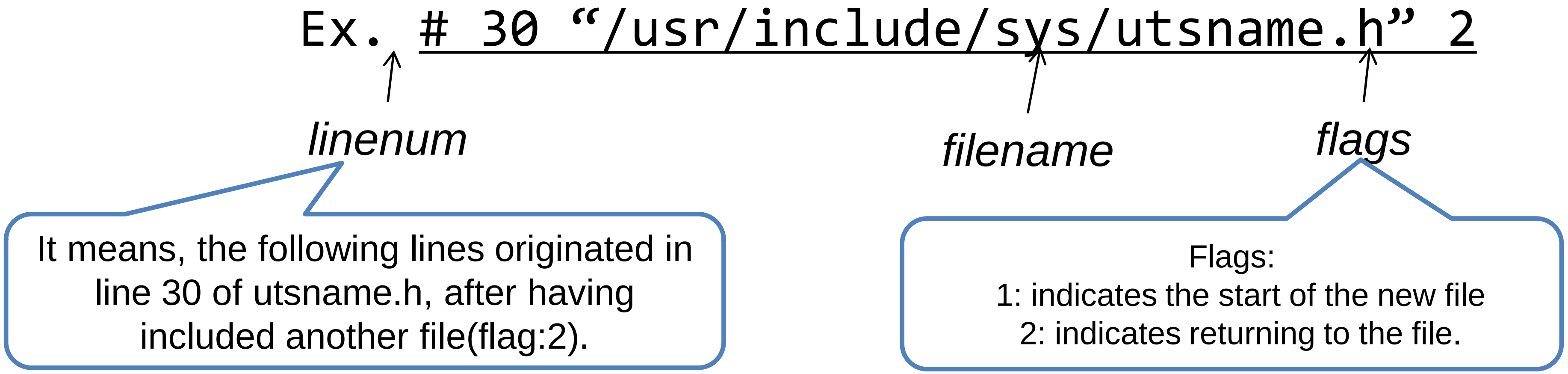
C:\Users\khashimoto\Desktop\hoger\uname.c.preprocessed

```
#define BBUNIT_ASSERT_STRNOTEQ(STR1,STR2) do { if (strcmp(STR1, STR2)
# 2121 "include/libbb.h"


POP_SAVED_FUNCTION_VISIBILITY

# 70 "coreutils/uname.c" 2
/* After libbb.h, since it needs sys/types.h on some systems */
# 1 "/usr/include/x86_64-linux-gnu/sys/utsname.h" 1 3
/* Copyright (C) 1991-2014 Free Software Foundation, Inc.
   This file is part of the GNU C Library.

```

Identify & delete the expanded headers

- stripHeaders() function in minimize.py
  - Takes preprocessed C file
  - Search Preprocessor Output relevant to #include lines
  - Delete included contents guided by the *linemarkers*

Included file name and line number information is conveyed in the preprocessor output; *linemarkers*



Ex. # 30 “/usr/include/sys/utsname.h” 2

linenum

filename

flags

It means, the following lines originated in line 30 of utsname.h, after having included another file(flag:2).

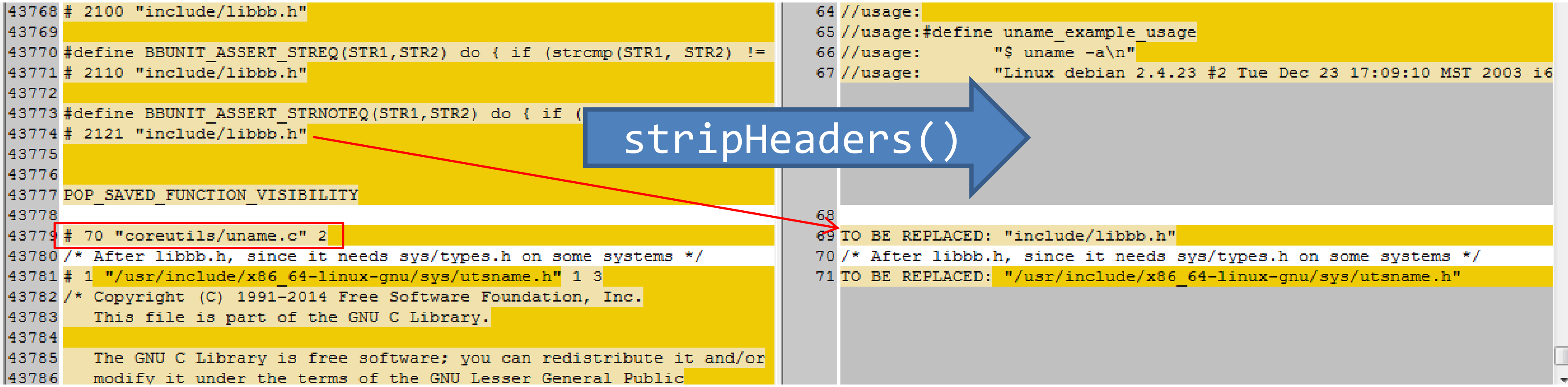
Flags:
1: indicates the start of the new file
2: indicates returning to the file.

https://gcc.gnu.org/onlinedocs/cpp/Preprocessor-Output.html

Identify & delete the expanded headers

- `stripHeaders()` algorithm
  - Find linemarkers (starting with '`# `*`number "filename"`*')
  - If *`filename`* is the target C file:
    - copy the following lines
    - And if *`flag`* in the linemarker is 2:
      - Mark "TO BE REPLACED" that means "there is #include line"

Flag 2 indicates returning to the file (after having included another file).

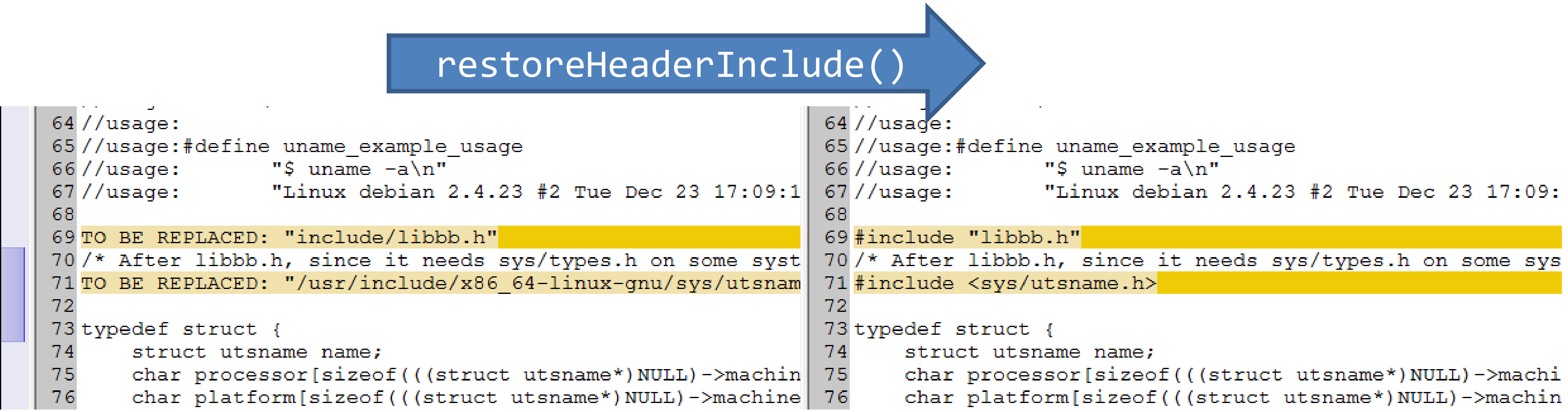


```
# 2100 "include/libbb.h"

#define BBUNIT_ASSERT_STREQ(STR1,STR2) do { if (strcmp(STR1, STR2) !=
# 2110 "include/libbb.h"

#define BBUNIT_ASSERT_STRNOTEQ(STR1,STR2) do { if (
# 2121 "include/libbb.h"


POP_SAVED_FUNCTION_VISIBILITY

# 70 "coreutils/uname.c" 2
/* After libbb.h, since it needs sys/types.h on some systems */
# 1 "/usr/include/x86_64-linux-gnu/sys/utsname.h" 1 3
/* Copyright (C) 1991-2014 Free Software Foundation, Inc.
   This file is part of the GNU C Library.

   The GNU C Library is free software; you can redistribute it and/or
   modify it under the terms of the GNU Lesser General Public
```

```
//usage:
//usage:#define uname_example_usage
//usage:       "$ uname -a\n"
//usage:       "Linux debian 2.4.23 #2 Tue Dec 23 17:09:10 MST 2003 i6

TO BE REPLACED: "include/libbb.h"
/* After libbb.h, since it needs sys/types.h on some systems */
TO BE REPLACED: "/usr/include/x86_64-linux-gnu/sys/utsname.h"
```

Restore #include sentences

- `restoreHeaderInclude()` function in `minimize.py`
  - Takes header-stripped preprocessed file
  - Look for “TO BE REPLACED” marks
  - Compare with the original C file, copy original #include lines

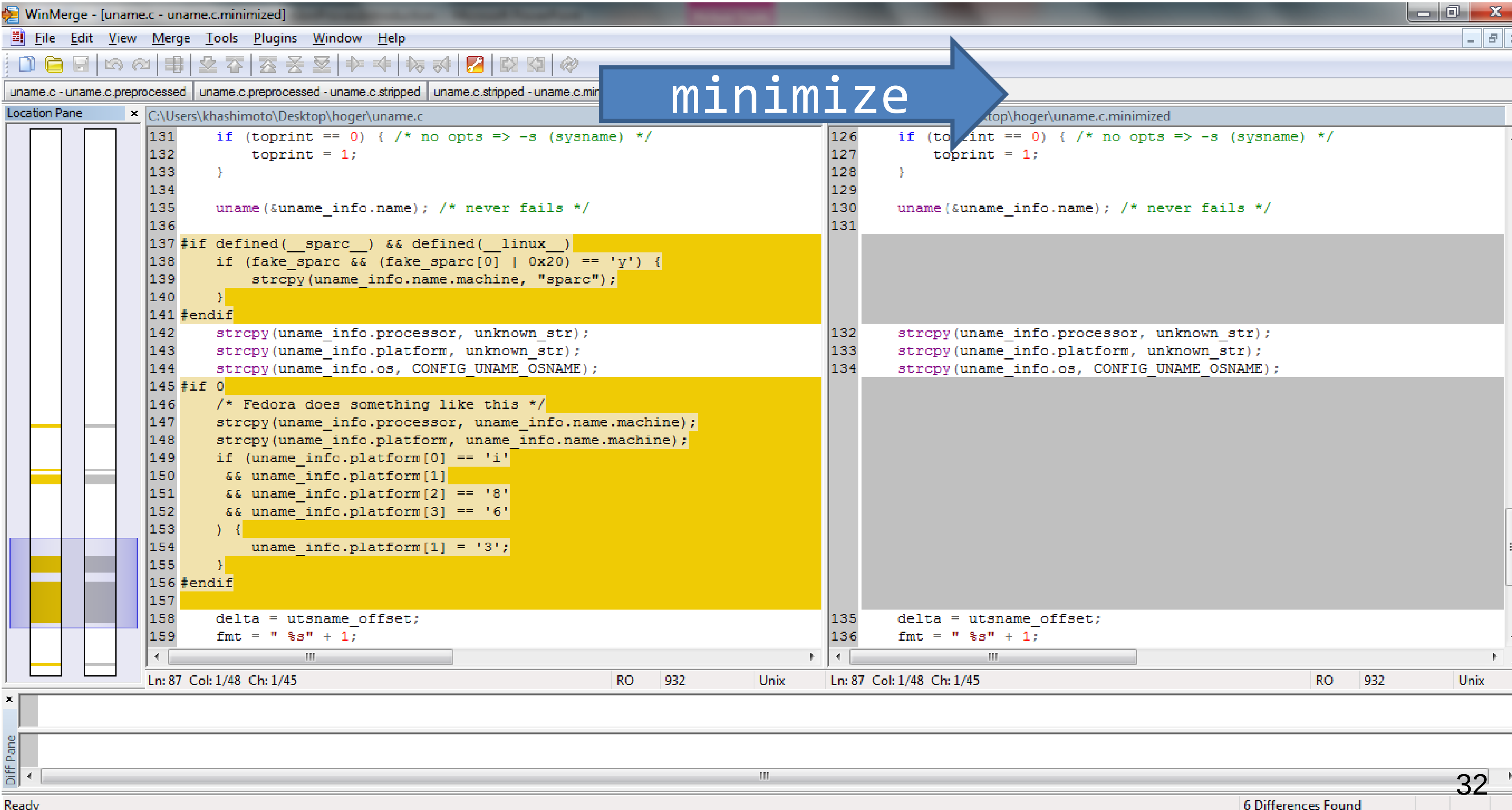


```
//usage:
//usage:#define uname_example_usage
//usage:        "$ uname -a\n"
//usage:        "Linux debian 2.4.23 #2 Tue Dec 23 17:09:1

TO BE REPLACED: "include/libbb.h"
/* After libbb.h, since it needs sys/types.h on some syst
TO BE REPLACED: "/usr/include/x86_64-linux-gnu/sys/utsnam

typedef struct {
    struct utsname name;
    char processor[sizeof(((struct utsname*)NULL)->machin
    char platform[sizeof(((struct utsname*)NULL)->machine
```

```
//usage:
//usage:#define uname_example_usage
//usage:        "$ uname -a\n"
//usage:        "Linux debian 2.4.23 #2 Tue Dec 23 17:09:

#include "libbb.h"
/* After libbb.h, since it needs sys/types.h on some sys
#include <sys/utsname.h>

typedef struct {
    struct utsname name;
    char processor[sizeof(((struct utsname*)NULL)->machi
    char platform[sizeof(((struct utsname*)NULL)->machin
```

Minimizing procedure

- Finally, diff result is only deletions of the unused code.
  - Without changing #include, #define lines.



WinMerge - [uname.c - uname.c.minimized]
File Edit View Merge Tools Plugins Window Help
uname.c - uname.c.preprocessed
uname.c.preprocessed - uname.c.stripped
uname.c.stripped - uname.c.mir
Location Pane
C:\Users\khashimoto\Desktop\hoger\uname.c
top\hoger\uname.c.minimized
Diff Pane

minimize

```
    if (toprint == 0) { /* no opts => -s (sysname) */
        toprint = 1;
    }

    uname(&uname_info.name); /* never fails */

#if defined(__sparc__) && defined(__linux__)
    if (fake_sparc && (fake_sparc[0] | 0x20) == 'y') {
        strcpy(uname_info.name.machine, "sparc");
    }
#endif
    strcpy(uname_info.processor, unknown_str);
    strcpy(uname_info.platform, unknown_str);
    strcpy(uname_info.os, CONFIG_UNAME_OSNAME);
#if 0
    /* Fedora does something like this */
    strcpy(uname_info.processor, uname_info.name.machine);
    strcpy(uname_info.platform, uname_info.name.machine);
    if (uname_info.platform[0] == 'i'
     && uname_info.platform[1]
     && uname_info.platform[2] == '8'
     && uname_info.platform[3] == '6'
    ) {
        uname_info.platform[1] = '3';
    }
#endif

    delta = utsname_offset;
    fmt = " %s" + 1;
```

```
    if (toprint == 0) { /* no opts => -s (sysname) */
        toprint = 1;
    }

    uname(&uname_info.name); /* never fails */

    strcpy(uname_info.processor, unknown_str);
    strcpy(uname_info.platform, unknown_str);
    strcpy(uname_info.os, CONFIG_UNAME_OSNAME);
    delta = utsname_offset;
    fmt = " %s" + 1;
```

Ln: 87 Col: 1/48 Ch: 1/45 RO 932 Unix
Ln: 87 Col: 1/48 Ch: 1/45 RO 932 Unix
Ready
6 Differences Found

4 ½ Results & Evaluation

Minimization Results

Linux Kernel Tree

- allnoconfig: 64684 unused lines were removed → **22% LoC reduced**.
- defconfig: 103144 unused lines were removed → **5% LoC reduced**.

BusyBox Tree

- allnoconfig: 51 out of 112 compiled C files have been minimized 5945 lines unused lines were removed → **34% LoC reduced**
- defconfig: 296 out of 505 compiled C files have been minimized. 20453 lines unused lines were removed → **11% LoC reduced**

ARCTIC Core source code

- Statistics shows approximately **5.5 times higher chances of eliminating unused #ifdef switches** compared to Linux Kernel.

Minimization Evaluation

HITACHI
Inspire the Next

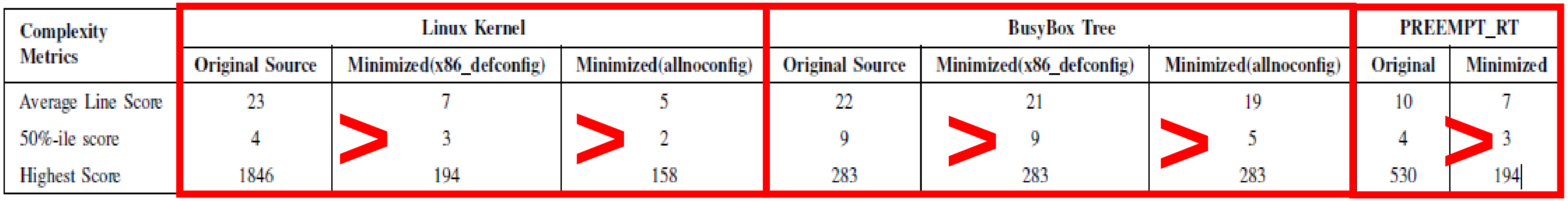
Complexity Statistics reduced

- To analyze the complexity of "C" program function.
- Linux with PREEMPT_RT patch, Linux Kernel source, BusyBox tree as shown in table below.
- Complexity (a GNU utility) tool has been used.

Disassembled code("objdump –d") matches

- Between the binaries built from minimized source and original one.
- Confirmed configuration & target:
  - BusyBox-1.24.1: defconfig, allnoconfig
    - busybox (executable)
  - Linux kernel 4.4.1: allnoconfig
    - vmlinux.o

Minimized code is compilable and produces same binary



Complexity Metrics	Linux Kernel					BusyBox Tree					PREEMPT_RT		
	Original Source		Minimized(x86_defconfig)		Minimized(allnoconfig)	Original Source		Minimized(x86_defconfig)		Minimized(allnoconfig)	Original		Minimized
Average Line Score	23		7		5	22		21		19	10		7
50%-ile score	4	>	3	>	2	9	>	9	>	5	4	>	3
Highest Score	1846		194		158	283		283		283	530		194

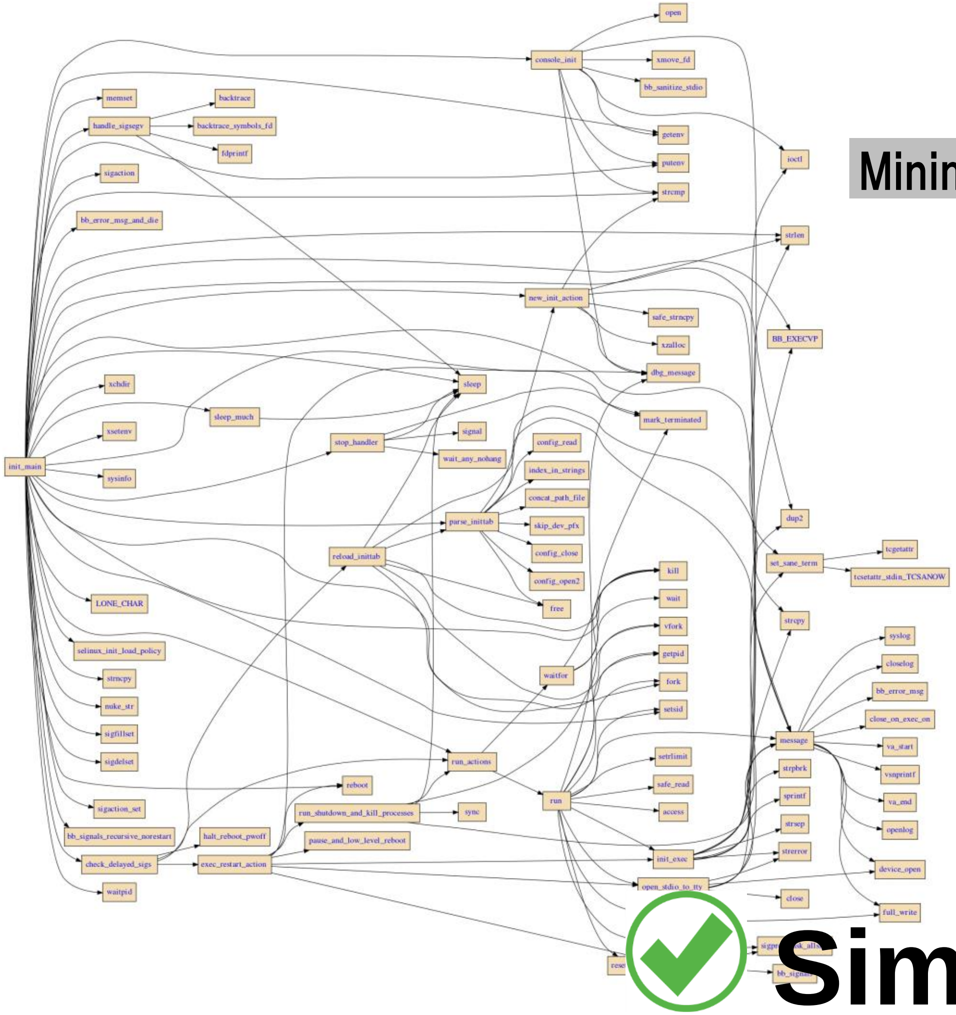
Measured complexity in terms of average line score, 50%-ile score and highest score.

"Complexity" reduced after Minimization !!

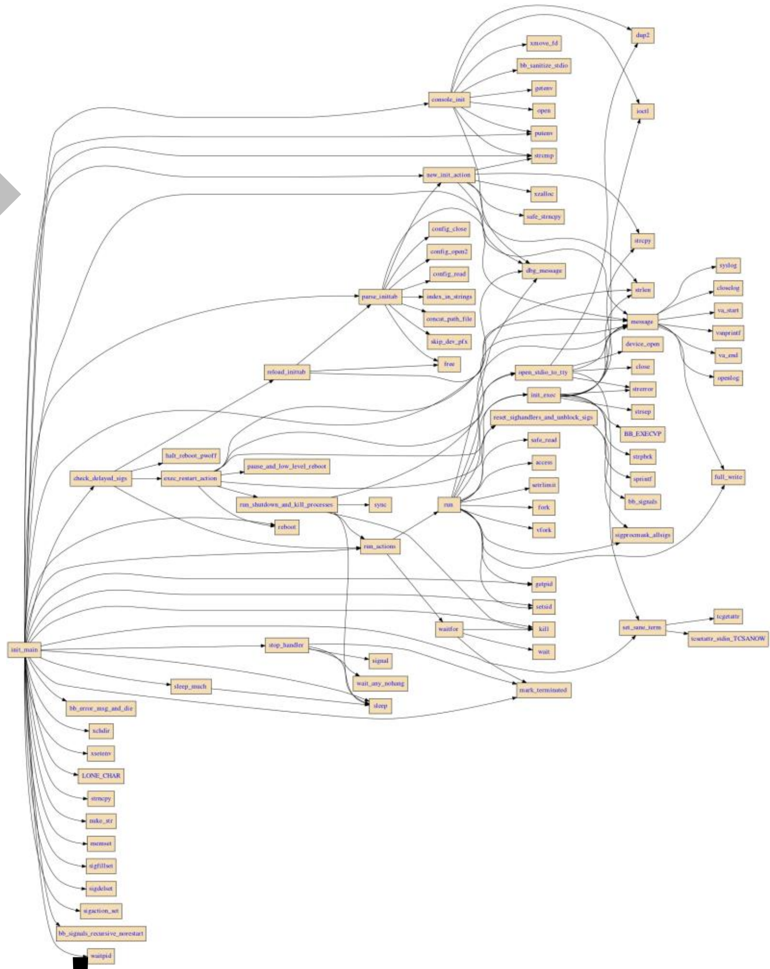
Benefits

- Verification time and cost improvement
  - Static analysis through Coccinelle
  - Executed a semantic patch for detecting functions have different return type values
  - Statistics
    - Comparison of **execution time** and minimization was **faster**.
    - **12[s] and 2.24[s]** for **original and minimized** kernel respectively.
- False positive reduction
  - Wrong Coccinelle indication about presence of particular condition.
  - Statistics
    - **Original** kernel source: **126**
    - **Minimized** kernel source: **82**
- Pruning function call graph
  - Analysis requires every possible call path to establish and trace relationship between program and subroutines.
  - Call graph is a directed graph that represents this relationship.

Benefits



Minimization



Simpler !

No. of nodes: 94
No. of edges: 140

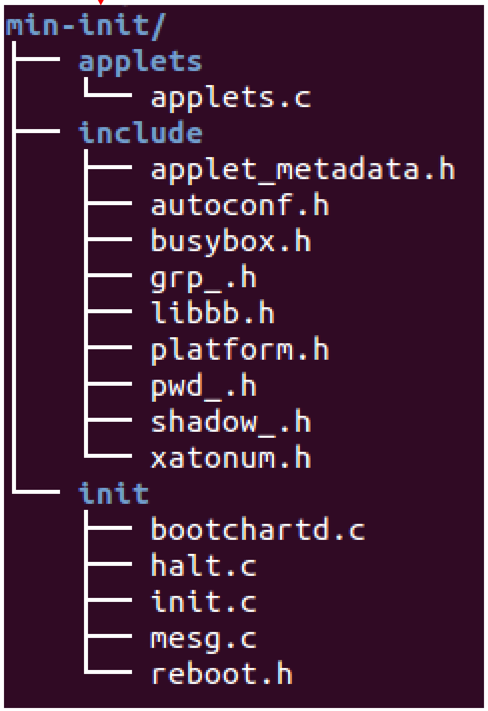
No. of nodes: 85
No. of edges: 123

Benefits

Extracting Minimal Subtarget Sources

```
$ cd busybox-1.24.1
$ make init C=2 CHECK=minimize.py CF="-mindir ../min-init"
```

If subtarget is specified in the minimized command,
Only the used source files will be extracted.



```
min-init/
├── applets
│   └── applets.c
├── include
│   ├── applet_metadata.h
│   ├── autoconf.h
│   ├── busybox.h
│   ├── grp_.h
│   ├── libbb.h
│   ├── platform.h
│   ├── pwd_.h
│   ├── shadow_.h
│   └── xatonum.h
└── init
    ├── bootchartd.c
    ├── halt.c
    ├── init.c
    ├── mesg.c
    └── reboot.h
```

Depended *.c files in minimized form.
Actually included *.h files

✓ Easy to identify which files are used

✓ Helps efficient software walk-through

5. Conclusion and Future prospects

Conclusion

- To get Linux certified with the functional safety standard, code analysis tools are mandatory to be applied to OSS/Linux

- Minimization widens possibility of products with OSS/Linux certified to functional safety standard making the code analysis and review on them more applicable.
  - Minimized code also have minimized search spaces in which such tools explore.

Future work

- Extend the Minimization technique to support other source codes that do not use Kbuild-like build system.
  - *Linux kernel* and *busybox* both use Kbuild.
  - *libc* should be addressed
  - *Automake*, *Cmake* support will broaden the supported applications.

- Evaluate the technique in the practical tools to be used for real certification in SIL2LinuxMP.

Open challenges

- To prove minimized tree is “equal” to original one

  How to formally verify equivalence ??

- To find out more application targets for Minimization ??

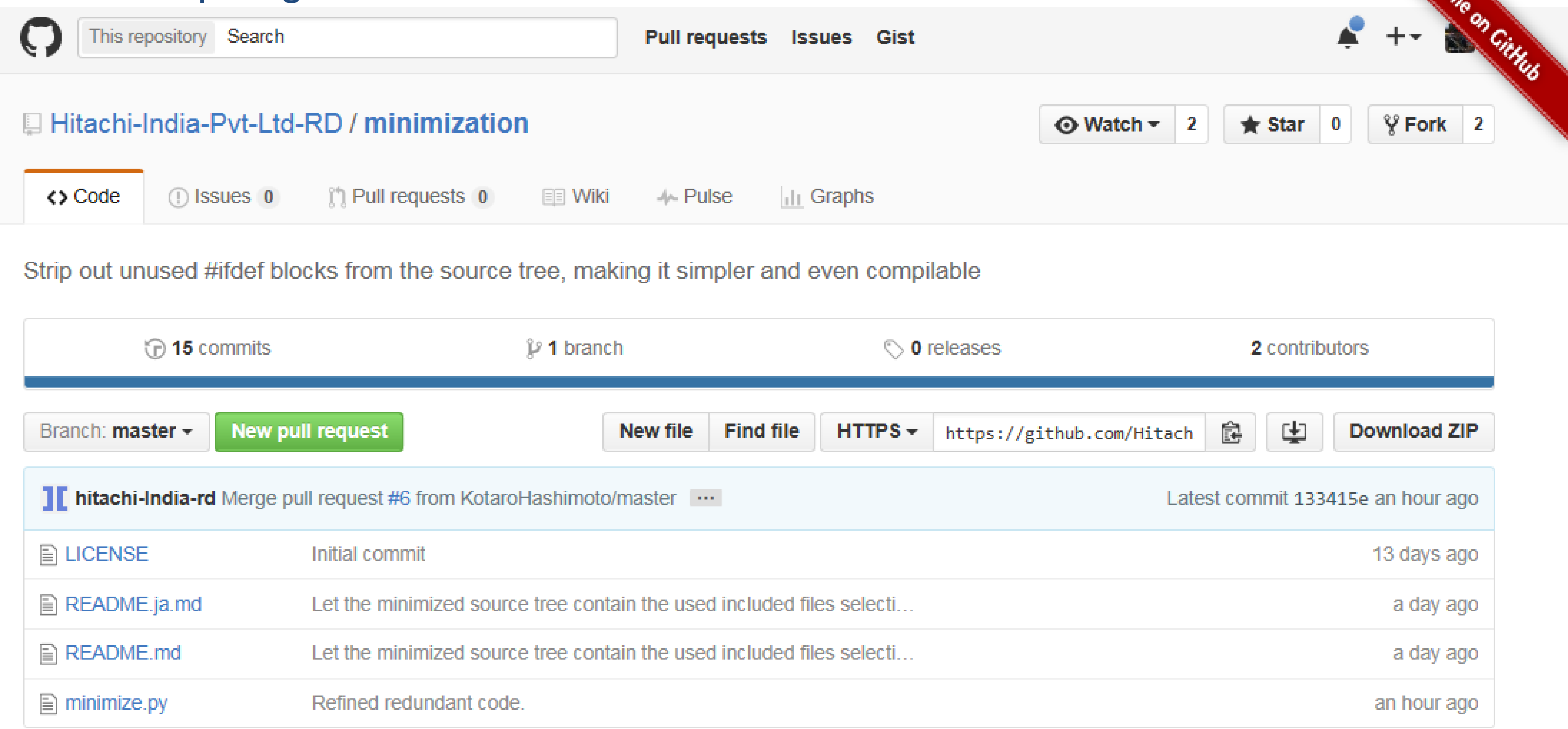
  Something that enhances existing tools / techniques

Minimization tool available in:

https://github.com/Hitachi-India-Pvt-Ltd-RD/minimization

Try it out !

https://github.com/Hitachi-India-Pvt-Ltd-RD/minimization



Hitachi-India-Pvt-Ltd-RD / **minimization**

Strip out unused #ifdef blocks from the source tree, making it simpler and even compilable

15 commits | **1** branch | **0** releases | **2** contributors

hitachi-India-rd Merge pull request #6 from KotaroHashimoto/master … Latest commit 133415e an hour ago

LICENSE	Initial commit	13 days ago
README.ja.md	Let the minimized source tree contain the used included files selecti…	a day ago
README.md	Let the minimized source tree contain the used included files selecti…	a day ago
minimize.py	Refined redundant code.	an hour ago

Please suggest useful applications !

Legal Information

- Linux is a registered trademark of Linus Torvalds.
- All other trademarks and registered trademarks are the property of their respective holders.

END

Code Minimization Technology for SIL2LinuxMP – Qualifying Linux for Functional Safety

Jul/13/2016

Taku Shimosawa
Krishnaji Desai

HITACHI

Inspire the Next