

# Introduction to Coccinelle

Julia Lawall (Inria/LIP6)

<http://coccinelle.lip6.fr>

June 25, 2018

## Common programming problems

- Programmers don't really understand how C works.
  - `!e1 & e2` does a bit-and with 0 or 1.
- A simpler API function exists, but not everyone uses it.
  - Mixing different functions for the same purpose is confusing.
- A function may fail, but the call site doesn't check for that.
  - A rare error case will cause an unexpected crash.
- Etc.

Need for pervasive code changes.

## Example: Bad bit-and

```
if (!dma_cntrl & DMA_START_BIT) {  
    BCMLOG(BCMLOG_DBG, "Already Stopped\n");  
    return BC_STS_SUCCESS;  
}
```

From `drivers/staging/crystalhd/crystalhd_hw.c`

## Example: Inconsistent API usage

drivers/mtd/nand/r852.c:

```
if (!bounce) {
    dev->phys_dma_addr =
        pci_map_single(dev->pci_dev, (void *)buf, R852_DMA_LEN,
            (do_read ? PCI_DMA_FROMDEVICE : PCI_DMA_TODEVICE));

    if (pci_dma_mapping_error(dev->pci_dev, dev->phys_dma_addr))
        bounce = 1;
}
```

drivers/mtd/nand/denali.c:

```
denali->buf.dma_buf =
    dma_map_single(&dev->dev, denali->buf.buf, DENALI_BUF_SIZE,
        DMA_BIDIRECTIONAL);
if (dma_mapping_error(&dev->dev, denali->buf.dma_buf)) ...
pci_set_master(dev); // looks like a pci function...
...
ret = pci_request_regions(dev, DENALI_NAND_NAME);
```

Why not pci functions in all cases? Or dma functions in all cases?

## Example: Missing error check

`kmalloc` returns `NULL` on insufficient memory.

```
alloc = kmalloc(sizeof *alloc, GFP_KERNEL);
INIT_LIST_HEAD(&intmem_allocations);
intmem_virtual = ioremap(MEM_INTMEM_START + RESERVED_SIZE,
                        MEM_INTMEM_SIZE - RESERVED_SIZE);
initiated = 1;
alloc->size = MEM_INTMEM_SIZE - RESERVED_SIZE;
```

From `arch/cris/arch-v32/mm/intmem.c`

## Our goals

- Automatically **find** code containing bugs or defects, or requiring collateral evolutions.
- Automatically **fix** bugs or defects, and perform collateral evolutions.
- Provide a system that is **accessible** to software developers.

## Requirements for automation

The ability to abstract over irrelevant information:

- `if (!dma_cntrl & DMA_START_BIT) { ... }`:  
    `dma_cntrl` is not important.

The ability to match scattered code fragments:

- `kmalloc` may be far from the first dereference.

The ability to transform code fragments:

- Replace `pci_map_single` by `dma_map_single`, or vice versa.

# Coccinelle

Program matching and transformation for unpreprocessed C code.

Fits with the existing habits of C programmers.

- C-like, patch-like notation

Semantic patch language (SmPL):

- **Metavariables** for abstracting over subterms.
- “...” for abstracting over code sequences.
- Patch-like notation ( $-/+$ ) for expressing transformations.



## The !& problem

**The problem:** Combining a boolean (0/1) with a constant using & is usually meaningless:

```
if (!dma_cntrl & DMA_START_BIT) {  
    BCMLOG(BCMLOG_DBG, "Already Stopped\n");  
    return BC_STS_SUCCESS;  
}
```

**The solution:** Add parentheses.

**Our goal:** Do so automatically for any expression  $E$  and constant  $C$ .

## A semantic patch for the !& problem

@@

expression E;

constant C;

@@

- !E & C

+ !(E & C)

Two parts per rule:

- Metavariable declaration
- Transformation specification

A semantic patch can contain multiple rules.

# Metavariable types

Surrounded by @@ @@.

- expression, statement, type, constant, local idexpression
- A type from the source program
- iterator, declarer, iterator name, declarer name, typedef

## Transformation specification

- - in the leftmost column for something to remove
- + in the leftmost column for something to add
- \* in the leftmost column for something of interest
  - Cannot be used with + and -.
- Spaces, newlines irrelevant.

## Exercise 1

1. Create a file `ex1.cocci` containing the following:

```
@@  
expression E;  
constant C;  
@@
```

```
- !E & C  
+ !(E & C)
```

2. Run `spatch`: `spatch --sp-file ex1.cocci --dir linux-3.2/drivers/staging/crystalhd`
3. Did your semantic patch do everything it should have?
4. Did it do something it should not have?

## Exercise 2

Some code contains a cast on the result of `kmalloc`. For example:

```
info->RegsBuf = (unsigned char *)  
    kmalloc(sizeof(info->ATAREgs), GFP_KERNEL);
```

If the cast just converts to a different pointer type, the cast is not needed.

1. Complete the following semantic patch to remove this unnecessary cast.

```
@@ expression e; expression arg1, arg2; type T; @@
```

[fill it in]

2. Test your semantic patch on the code in `linux-3.2/drivers/isdn`
3. Are you satisfied with the appearance of the results? If not, try to improve it.

## Practical issues

To check that your semantic patch is valid:

```
spatch --parse-cocci mysp.cocci
```

To run your semantic patch:

```
spatch --sp-file mysp.cocci file.c  
spatch --sp-file mysp.cocci --dir directory
```

To understand why your semantic patch didn't work:

```
spatch --sp-file mysp.cocci file.c --debug
```

If you don't need to include header files:

```
spatch --sp-file mysp.cocci --dir directory  
--no-includes --include-headers
```

## More practical issues

Put the interesting output in a file:

```
spatch ... > output.patch
```

Omit the uninteresting output:

```
spatch --very-quiet ...
```



## Inconsistent API usage

Do we need this function?

```
static inline dma_addr_t
pci_map_single(struct pci_dev *hwdev, void *ptr, size_t size,
               int direction)
{
    return dma_map_single(hwdev == NULL ? NULL : &hwdev->dev, ptr,
                          size, (enum dma_data_direction)direction);
}
```

# The use of `pci_map_single`

The code:

```
dev->phys_dma_addr =  
    pci_map_single(dev->pci_dev, (void *)buf, R852_DMA_LEN,  
        (do_read ? PCI_DMA_FROMDEVICE : PCI_DMA_TODEVICE));
```

would be more uniform as:

```
dev->phys_dma_addr =  
    dma_map_single(&dev->pci_dev->dev, (void *)buf, R852_DMA_LEN,  
        (do_read ? DMA_FROM_DEVICE : DMA_TO_DEVICE));
```

Issues:

- Change function name.
- Add field access to the first argument.
- Rename the fourth argument.

## pci\_map\_single: Example and definitions

Commit b0eb57cb

```
- rbi->dma_addr = pci_map_single(adapter->pdev,  
+ rbi->dma_addr = dma_map_single(  
+     &adapter->pdev->dev,  
      rbi->skb->data, rbi->len,  
      PCI_DMA_FROMDEVICE);
```

### PCI constants

```
/* This defines the direction arg  
to the DMA mapping routines. */  
#define PCI_DMA_BIDIRECTIONAL 0  
#define PCI_DMA_TODEVICE 1  
#define PCI_DMA_FROMDEVICE 2  
#define PCI_DMA_NONE 3
```

### DMA constants

```
enum dma_data_direction {  
    DMA_BIDIRECTIONAL = 0,  
    DMA_TO_DEVICE = 1,  
    DMA_FROM_DEVICE = 2,  
    DMA_NONE = 3,  
};
```

## pci\_map\_single: First attempt

Outline of a semantic patch, including the patch example:

@@

@@

```
- rbi->dma_addr = pci_map_single(adapter->pdev,  
+ rbi->dma_addr = dma_map_single(  
+     &adapter->pdev->dev,  
     rbi->skb->data, rbi->len,  
     PCI_DMA_FROMDEVICE);
```

## pci\_map\_single: First attempt

Eliminate irrelevant code:

@@

@@

```
- pci_map_single(adapter->pdev,  
+ dma_map_single(  
+     &adapter->pdev->dev,  
     rbi->skb->data, rbi->len,  
     PCI_DMA_FROMDEVICE)
```

## pci\_map\_single: First attempt

Abstract over subterms:

@@

expression E1,E2,E3;

@@

```
- pci_map_single(E1,  
+ dma_map_single(  
+   &E1->dev,  
    E2, E3,  
    PCI_DMA_FROMDEVICE)
```

## pci\_map\_single: First attempt

Rename the fourth argument:

@@

expression E1,E2,E3;

@@

```
- pci_map_single(E1,  
+ dma_map_single(  
+   &E1->dev,  
   E2, E3,  
-   PCI_DMA_FROMDEVICE)  
+   DMA_FROM_DEVICE)
```

## pci\_map\_single: Second attempt

Need to consider all direction constants.

```
@@ expression E1,E2,E3; @@  
- pci_map_single(E1,  
+ dma_map_single(&E1->dev,  
    E2, E3,  
-    PCI_DMA_FROMDEVICE)  
+    DMA_FROM_DEVICE)
```

```
@@ expression E1,E2,E3; @@  
- pci_map_single(E1,  
+ dma_map_single(&E1->dev,  
    E2, E3,  
-    PCI_DMA_TODEVICE)  
+    DMA_TO_DEVICE)
```

Etc. Four rules in all.



## pci\_map\_single: Third attempt

Avoid code duplication: Use a disjunction.

```
@@ expression E1,E2,E3; @@  
- pci_map_single(E1,  
+ dma_map_single(&E1->dev,  
    E2, E3,  
(  
-     PCI_DMA_BIDIRECTIONAL  
+     DMA_BIDIRECTIONAL  
|  
-     PCI_DMA_TODEVICE  
+     DMA_TO_DEVICE  
|  
-     PCI_DMA_FROMDEVICE  
+     DMA_FROM_DEVICE  
|  
-     PCI_DMA_NONE  
+     DMA_NONE_DEVICE  
)  
)
```

## pci\_map\_single: Fourth attempt

```
@@ expression E1,E2,E3,E4; @@
```

```
- pci_map_single(E1,  
+ dma_map_single(&E1->dev,  
    E2, E3, E4)
```

```
@@ expression E1,E2,E3; @@
```

```
dma_map_single(E1, E2, E3,  
(  
-     PCI_DMA_BIDIRECTIONAL  
+     DMA_BIDIRECTIONAL  
|  
-     PCI_DMA_TODEVICE  
+     DMA_TO_DEVICE  
|  
-     PCI_DMA_FROMDEVICE  
+     DMA_FROM_DEVICE  
|  
-     PCI_DMA_NONE  
+     DMA_NONE_DEVICE  
)  
)
```

## Exercise 3

1. Implement some version of the semantic patch for converting calls to `pci_map_single` to calls to `dma_map_single`.
2. Test your implementation on the directory `linux-3.2/drivers/net/ethernet`.
3. Implement both the third version and the fourth version. Compare the results.
4. Other PCI functions replicate DMA behavior, e.g., `pci_unmap_single`. For example, commit `b0eb57cb` contains:

```
- pci_unmap_single(pdev, tbi->dma_addr, tbi->len,  
+ dma_unmap_single(&pdev->dev, tbi->dma_addr, tbi->len,  
PCI_DMA_TODEVICE);
```

Extend your semantic patch to implement this transformation. Try to minimize the number of rules.

# Dots

## Issue:

- Sometimes it is necessary to search for multiple related code fragments.

## Goals:

- Specify patterns consisting of fragments of code separated by arbitrary execution paths.
- Specify constraints on the contents of those execution paths.

## Example: Inadequate error checking of kmalloc

kmalloc returns NULL on insufficient memory.

Good code:

```
block = kmalloc(WL12XX_HW_BLOCK_SIZE, GFP_KERNEL);  
if (!block)  
    return;
```

Bad code:

```
g = kmalloc (sizeof (*g), GFP_KERNEL);  
g->next = chains[r_sym].next;
```

## More bad code

```
alloc = kmalloc(sizeof *alloc, GFP_KERNEL);
INIT_LIST_HEAD(&intmem_allocations);
intmem_virtual = ioremap(MEM_INTMEM_START + RESERVED_SIZE,
                        MEM_INTMEM_SIZE - RESERVED_SIZE);
initiated = 1;
alloc->size = MEM_INTMEM_SIZE - RESERVED_SIZE;
```

The `kmalloc` and the dereference are not necessarily contiguous.

# Using dots

Start with a typical example of code

```
alloc = kcalloc(sizeof *alloc, GFP_KERNEL);
INIT_LIST_HEAD(&intmem_allocations);
intmem_virtual = ioremap(MEM_INTMEM_START + RESERVED_SIZE,
                        MEM_INTMEM_SIZE - RESERVED_SIZE);
initiated = 1;
alloc->size = MEM_INTMEM_SIZE - RESERVED_SIZE;
```

# Using dots

Highlight what is wanted

```
* alloc = kcalloc(sizeof *alloc, GFP_KERNEL);  
INIT_LIST_HEAD(&intmem_allocations);  
intmem_virtual = ioremap(MEM_INTMEM_START + RESERVED_SIZE,  
MEM_INTMEM_SIZE - RESERVED_SIZE);  
  
initiated = 1;  
* alloc->size = MEM_INTMEM_SIZE - RESERVED_SIZE;
```



## Using dots

Replace the irrelevant statements by ...

```
* alloc = kmalloc(sizeof *alloc, GFP_KERNEL);
```

```
...
```

```
* alloc->size = MEM_INTMEM_SIZE - RESERVED_SIZE;
```

# Using dots

Abstract over irrelevant subterms.

- May use ....

```
@@ expression e; identifier f; @@  
* e      = kmalloc(...);  
...  
  
* e->f
```

# Using dots

Check properties of the matched statement sequence

```
@@ expression e; identifier f; @@
```

```
* e      = kmalloc(...);
```

```
... when != e == NULL
```

```
    when != e != NULL
```

```
* e->f
```

# Using dots

## Sanity check

```
@@ expression e, e1; identifier f; @@  
* e      = kmalloc(...);  
  ... when != e == NULL  
      when != e != NULL  
      when != e = e1  
  
* e->f
```

## Results: 18 kmallocs in 12 files

Real bug: linux-3.2/arch/cris/arch-v32/mm/intmem.c

```
- alloc = kmalloc(sizeof *alloc, GFP_KERNEL);  
  INIT_LIST_HEAD(&intmem_allocations);  
  intmem_virtual = ioremap(MEM_INTMEM_START + RESERVED_SIZE,  
                           MEM_INTMEM_SIZE - RESERVED_SIZE);  
  
  initiated = 1;  
- alloc->size = MEM_INTMEM_SIZE - RESERVED_SIZE;
```

## Results: 18 kmallocs in 12 files

Real bug: linux-3.2/arch/cris/arch-v32/mm/intmem.c

```
- alloc = kmalloc(sizeof *alloc, GFP_KERNEL);
  INIT_LIST_HEAD(&intmem_allocations);
  intmem_virtual = ioremap(MEM_INTMEM_START + RESERVED_SIZE,
                          MEM_INTMEM_SIZE - RESERVED_SIZE);

  initiated = 1;
- alloc->size = MEM_INTMEM_SIZE - RESERVED_SIZE;
```

False positive! linux-3.2/net/ipv4/syncookies.c

```
- ireq->opt = kmalloc(opt_size, GFP_ATOMIC);
- if (ireq->opt != NULL && ip_options_echo(&ireq->opt->opt, skb)) {
    kfree(ireq->opt);
    ireq->opt = NULL;
}
```

## False positives

```
ireq->opt != NULL && ip_options_echo(&ireq->opt->opt, skb)
```

“...” matches complete statements.

- `ireq->opt != NULL` is not seen as being before `&ireq->opt->opt`.

**Solution:** stop at NULL tests or bad dereference (disjunction).

- `e == NULL`: OK
- `e != NULL`: OK
- `e->f`: Bug

## Revised version

```
@@ expression e,e1; identifier f; @@  
  e = kmalloc(...);  
  ... when != e = e1  
(  
  e == NULL || ...  
  |  
  e != NULL && ...  
  |  
  * e->f  
)
```

Shortest path property:

- “...” matches everything except what is on either side.

Matches 11 files, eliminating the false positive.



## Exercise 4

The following code allocates a region of memory and then clears it:

```
state = kmalloc(sizeof(struct drxd_state), GFP_KERNEL);
if (!state)
    return NULL;
memset(state, 0, sizeof(*state));
```

The function `kzalloc` does both, *i.e.*, we could write:

```
state = kzalloc(sizeof(struct drxd_state), GFP_KERNEL);
if (!state)
    return NULL;
```

1. Write a semantic patch to make this transformation.
2. Test your semantic patch on `linux-3.2/drivers/net/wireless`.
3. Are there any files where your semantic patch should not transform the code, but it does?

## Exercise 5

One of the results for the `kmalloc` with no NULL test example is the following (`linux-3.2/drivers/macintosh/via-pmu.c`):

```
- pp = kmalloc(sizeof(struct pmu_private), GFP_KERNEL);  
  if (pp == 0)  
      return -ENOMEM;  
- pp->rb_get = pp->rb_put = 0;
```

The code will not crash, but it is not as nice as it could be. Write a semantic patch to replace such bad uses of 0 by NULL.

### Hints:

- A metavariable of type “`expression *`” matches any pointer typed expression.
- This exercise has nothing to do with dots.

# Summary

SmPL features seen so far:

- Metavariables for abstracting over arbitrary expressions.
- Metavariables restricted to particular types.
- Disjunctions.
- Multiple rules.
- Dots.

## Some other features

### Nests:

- Match 0 or more occurrences of `b` between `a` and `c`.  
`b(); <... a(); ...> c();`

### Isomorphisms:

- Write `x == NULL`. Match `!x`.

### Python, OCaml interface:

- Print warning messages.
- Position metavariables give access to position information.

### Rule ordering:

- Later rules see the results of earlier rules.

## Conclusion

- Coccinelle provides a declarative language for program matching and transformation.
- Coccinelle semantic patches look like patches; fit with Linux programmers' habits.
- Quite “easy” to learn; already accepted by the Linux community.
- Future work will build on Coccinelle to develop tools motivated by problems observed in Linux development.

<http://coccinelle.lip6.fr>