Coccinelle: Practical Program Transformation for the Linux Kernel

Julia Lawall (Inria/LIP6)

June 25, 2018

Motivation

Large, critical infrastructure-software code bases

- Linux kernel, OpenSSL, Qemu, Firefox, etc.
- Frequent changes to add features, improve performance, etc.
- Large, diverse developer bases.

Issues:

- How to impose API improvements on the entire code base? (Collateral evolutions)
- How to ensure that a bug found in one place is fixed everywhere?
- How to explore the code structure?

Motivation

Large, critical infrastructure-software code bases

- Linux kernel, OpenSSL, Qemu, Firefox, etc.
- Frequent changes to add features, improve performance, etc.
- Large, diverse developer bases.

Issues:

- How to impose API improvements on the entire code base? (Collateral evolutions)
- How to ensure that a bug found in one place is fixed everywhere?
- How to explore the code structure?

Motivation

Large, critical infrastructure-software code bases

- Linux kernel, OpenSSL, Qemu, Firefox, etc.
- Frequent changes to add features, improve performance, etc.
- Large, diverse developer bases.

Issues:

- How to impose API improvements on the entire code base? (Collateral evolutions)
- How to ensure that a bug found in one place is fixed everywhere?
- How to explore the code structure?

Example

Evolution: A new function: kzalloc

⟹ Collateral evolution: Merge kmalloc and memset into kzalloc

```
fh = kmalloc(sizeof(struct zoran_fh), GFP_KERNEL);
if (!fh) {
  dprintk(1,
    KERN_ERR
    "%s: zoran_open(): allocation of zoran_fh failed\n",
    ZR_DEVNAME(zr));
  return -ENOMEM;
}
memset(fh, 0, sizeof(struct zoran_fh));
```

Example

Evolution: A new function: kzalloc

⟹ Collateral evolution: Merge kmalloc and memset into kzalloc

```
fh = kzalloc(sizeof(struct zoran_fh), GFP_KERNEL);
if (!fh) {
  dprintk(1,
    KERN_ERR
    "%s: zoran_open(): allocation of zoran_fh failed\n",
    ZR_DEVNAME(zr));
  return -ENOMEM;
}
```

Example

Evolution: A new function: kzalloc

⟹ Collateral evolution: Merge kmalloc and memset into kzalloc

```
fh = kzalloc(sizeof(struct zoran_fh), GFP_KERNEL);
if (!fh) {
  dprintk(1,
    KERN_ERR
    "%s: zoran_open(): allocation of zoran_fh failed\n",
    ZR_DEVNAME(zr));
  return -ENOMEM;
}
```

Originally, hundreds of kmalloc and memset calls

Example

Bug: Reference count mismanagement

- `for_each` iterator increments the reference count of the current element and decrements the reference count of the previous one.
- `return` escapes, skipping the decrement.
- $\Longrightarrow$ A memory leak.

Example

```
  for_each_child_of_node(np, child) {
    ...
    ret = of_property_read_u32(child, "reg", &reg);
    if (ret) {
      dev_err(dev, "Failed to get reg property\n");
+     of_node_put(child);
      return ret;
    }
    if (reg >= MX25_NUM_CFGS) {
      dev_err(dev,
        "reg value is greater than the number of available ...\n");
+     of_node_put(child);
      return -EINVAL;
    }
    ...
}
```

56 instances in linux-next (June 22, 2018)

Example

Program structure:

- If a function is defined in a header files and used in multiple .c files, then its implementation is duplicated, wasting code space.
- Is this a problem in practice?

```
codel_impl.h: 202 lines:
   codel_vars_init: sz: 4 callers: 3
   codel_stats_init: sz: 4 callers: 3
   codel_params_init: sz: 7 callers: 3
   codel_dequeue: sz: 112 callers: 3
```

Existing tools

Collateral evolutions

- Refactoring tools in various IDEs
- Fixed set of semantics-preserving transformations

Bug finding

- Metal/Coverity [OSDI 2001], SLAM/SMV [SPIN 2001], Flawfinder, etc.
- Often black box, no support for bug fixing.

Visitors

- CIL, LLVM/Clang
- Configuration-specific, internal representation.

Coccinelle to the rescue!

What is Coccinelle?

- Pattern-based language (SmPL) for matching and transforming C code
- Under development since 2005. Open source since 2008.
- Allows code changes to be expressed using patch-like code patterns (semantic patches).

Semantic patches

- Like patches, but independent of irrelevant details (line numbers, spacing, variable names, etc.)
- Derived from code, with abstraction.
- Goal: fit with the existing habits of the Linux programmer.

Semantic patch example

```
@@
expression x,E1,E2;
@@
- x = kmalloc(E1,E2);
+ x = kzalloc(E1,E2);
  ...
- memset(x, 0, E1);
```

Creating a semantic patch: kmalloc → kzalloc

Start with an example

```
fh = kmalloc(sizeof(struct zoran_fh), GFP_KERNEL);
if (!fh) {
  dprintk(1,
    KERN_ERR
    "%s: zoran_open(): allocation of zoran_fh failed\n",
    ZR_DEVNAME(zr));
  return -ENOMEM;
}
memset(fh, 0, sizeof(struct zoran_fh));
```

Creating a semantic patch: kmalloc $\to$ kzalloc

Eliminate irrelevant code

```
fh = kmalloc(sizeof(struct zoran_fh), GFP_KERNEL);

...

memset(fh, 0, sizeof(struct zoran_fh));
```

Creating a semantic patch: kmalloc → kzalloc

Describe transformations

```
- fh = kmalloc(sizeof(struct zoran_fh), GFP_KERNEL);
+ fh = kzalloc(sizeof(struct zoran_fh), GFP_KERNEL);
  ...

- memset(fh, 0, sizeof(struct zoran_fh));
```

Creating a semantic patch: kmalloc $\rightarrow$ kzalloc

Abstract over subterms

```
@@
expression x;
expression E1,E2;

@@

- x = kmalloc(E1,E2);
+ x = kzalloc(E1,E2);
  ...

- memset(x, 0, E1);
```

One result

```
 int __init snd_seq_oss_create_client(void)
 {
         int rc;
         struct snd_seq_port_info *port;
         struct snd_seq_port_callback port_callback;

-        port = kmalloc(sizeof(*port), GFP_KERNEL);
+        port = kzalloc(sizeof(*port), GFP_KERNEL);
         if (!port) {
                 rc = -ENOMEM;
                 goto __error;
         }
         rc = snd_seq_create_kernel_client(NULL,
                                           SNDRV_SEQ_CLIENT_OSS, ...);
         if (rc < 0) goto __error;
         system_client = rc;
-        memset(port, 0, sizeof(*port));
         ...
}
```

Correct change

One result

```
int __init snd_seq_oss_create_client(void)
{
        int rc;
        struct snd_seq_port_info *port;
        struct snd_seq_port_callback port_callback;

-       port = kmalloc(sizeof(*port), GFP_KERNEL);
+       port = kzalloc(sizeof(*port), GFP_KERNEL);
        if (!port) {
                rc = -ENOMEM;
                goto __error;
        }
        rc = snd_seq_create_kernel_client(NULL,
                                          SNDRV_SEQ_CLIENT_OSS, ...);
        if (rc < 0) goto __error;
        system_client = rc;
-       memset(port, 0, sizeof(*port));
        ...
}
```

Correct change

Another result

```
- inblockdata = kmalloc(blocksize, gfp_mask);
+ inblockdata = kzalloc(blocksize, gfp_mask);
  if (inblockdata == NULL)
          goto err_free_cipher;
  ...
  inblock.data = (char *) inblockdata;
  inblock.len = blocksize;
  ...
  if (in_constant->len == inblock.len) {
          memcpy(inblock.data, in_constant->data, inblock.len);
  } else {
          krb5_nfold(in_constant->len * 8, in_constant->data,
                                 inblock.len * 8, inblock.data);
  }
  ...
- memset(inblockdata, 0, blocksize);
  kfree(inblockdata);
```

False positive

Another result

```
- inblockdata = kmalloc(blocksize, gfp_mask);
+ inblockdata = kzalloc(blocksize, gfp_mask);
  if (inblockdata == NULL)
          goto err_free_cipher;
  ...
  inblock.data = (char *) inblockdata;
  inblock.len = blocksize;
  ...
  if (in_constant->len == inblock.len) {
          memcpy(inblock.data, in_constant->data, inblock.len);
  } else {
          krb5_nfold(in_constant->len * 8, in_constant->data,
                     inblock.len * 8, inblock.data);
  }
  ...
- memset(inblockdata, 0, blocksize);
  kfree(inblockdata);
```

False positive

Creating a semantic patch: kmalloc → kzalloc

Refinement

```
@@
expression x;
expression E1,E2,E3;
identifier f;
type T;
@@

- x = kmalloc(E1,E2);
+ x = kzalloc(E1,E2);
  ...  when != E3 = (T)x
       when != (<+...x...+>) = E3
       when != f(...,x,...)
- memset(x, 0, E1);
```

Results

- Correctly updates 9 occurrences
  - No false positives

Results

- Correctly updates 9 occurrences
  - No false positives
- Other opportunities:
  - `acpi_os_allocate` → `acpi_os_allocate_zeroed`
  - `dma_pool_alloc` → `dma_pool_zalloc`
  - `dma_alloc_coherent` → `dma_zalloc_coherent`
  - `kmem_cache_alloc` → `kmem_cache_zalloc`
  - `pci_alloc_consistent` → `pci_zalloc_consistent`
  - `vmalloc` → `vzalloc`
  - `vmalloc_node` → `vzalloc_node`

Semantic patch example

```
@@
expression root,e,e1;
local idexpression child;
iterator name for_each_child_of_node;
@@

for_each_child_of_node(root, child) {
   ... when != of_node_put(child)
       when != e = child
(
   return <+...child...+>;
|
+  of_node_put(child);
?  return e1;
)
   ...
}
```

Semantic patch example

```
@h@
identifier f; position p; type t;
@@
(
inline t f(...) { ... }
|
t f@p(...) { ... }
)

@script:ocaml@
p << h.p;
f << h.f;
@@
if in_header p then add_def f p

@@
identifier f;
position p : script:ocaml(f) { in_c p && make_local f };
@@
f@p(...) { ... }

@@
identifier f;
position p : script:ocaml(f) { in_c p && add_call f p };
@@
f(...)@p
```

How does it work?

Requirements

- Reason about possible execution paths.
  - ⇒ Suggests matching against paths in a control-flow graph.
  - ⇒ Define the language by translation to CTL [Lacey et al POPL'03].
- Keep track of different variables.
  - Multiple `kmallocs` before any `memset`.
  - `for_each` iterators can be nested.
  - ⇒ Our contribution (CTL-V model checking algorithm).
- Collect transformation information
  - Where to transform?
  - What transformation to carry out?
  - ⇒ Our contribution (CTL-VW).

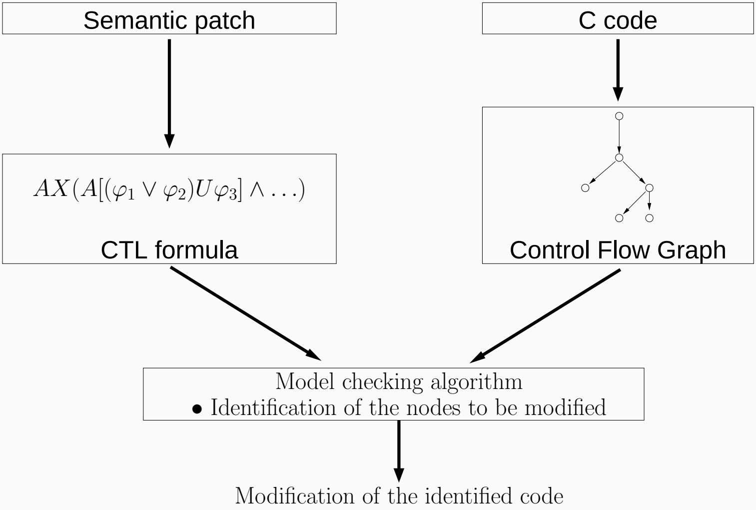
Requirements

- Reason about possible execution paths.
  - ⇒ Suggests matching against paths in a control-flow graph.
  - ⇒ Define the language by translation to CTL [Lacey et al POPL'03].
- Keep track of different variables.
  - Multiple `kmallocs` before any `memset`.
  - `for_each` iterators can be nested.
  - ⇒ Our contribution (CTL-V model checking algorithm).
- Collect transformation information
  - Where to transform?
  - What transformation to carry out?
  - ⇒ Our contribution (CTL-VW).

Requirements

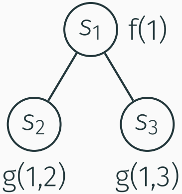
- Reason about possible execution paths.
  - ⇒ Suggests matching against paths in a control-flow graph.
  - ⇒ Define the language by translation to CTL [Lacey et al POPL'03].
- Keep track of different variables.
  - Multiple `kmallocs` before any `memset`.
  - `for_each` iterators can be nested.
  - ⇒ Our contribution (CTL-V model checking algorithm).
- Collect transformation information
  - Where to transform?
  - What transformation to carry out?
  - ⇒ Our contribution (CTL-VW).

Coccinelle architecture



Semantic patch

```
  f(...);
- g(...);
```



CTL representation

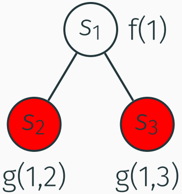
$$f(\ldots) \wedge \mathrm{AX}\ g(\ldots)$$

Model checking algorithm

$$\mathrm{SAT}(g(\ldots)) = \{s_2, s_3\}$$
$$\mathrm{SAT}(\mathrm{AX}\ g(\ldots)) = \{s_1\}$$
$$\mathrm{SAT}(f(\ldots) \wedge \mathrm{AX}\ g(\ldots)) = \{s_1\}$$

Semantic patch

```
  f(...);
- g(...);
```



CTL representation

$$f(\ldots) \wedge \mathsf{AX}\ g(\ldots)$$

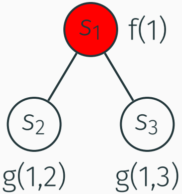
Model checking algorithm

$$\begin{aligned}
\mathrm{SAT}(g(\ldots)) &= \{s_2, s_3\} \\
\mathrm{SAT}(\mathsf{AX}\ g(\ldots)) &= \{s_1\} \\
\mathrm{SAT}(f(\ldots) \wedge \mathsf{AX}\ g(\ldots)) &= \{s_1\}
\end{aligned}$$

CTL translation and model checking

Semantic patch

```
   f(...);
 - g(...);
```



CTL representation

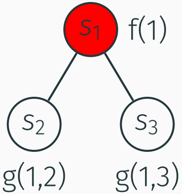
$$f(\ldots) \wedge \mathrm{AX}\ g(\ldots)$$

Model checking algorithm

$$\begin{aligned}
\mathrm{SAT}(g(\ldots)) &= \{s_2, s_3\} \\
\mathrm{SAT}(\mathrm{AX}\ g(\ldots)) &= \{s_1\} \\
\mathrm{SAT}(f(\ldots) \wedge \mathrm{AX}\ g(\ldots)) &= \{s_1\}
\end{aligned}$$

Semantic patch

```
  f(...);
- g(...);
```



CTL representation

$$f(\ldots) \wedge \mathsf{AX}\ g(\ldots)$$

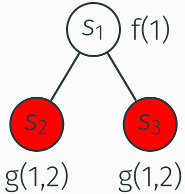
Model checking algorithm

$$\begin{aligned}
\mathrm{SAT}(g(\ldots)) &= \{s_2, s_3\} \\
\mathrm{SAT}(\mathsf{AX}\ g(\ldots)) &= \{s_1\} \\
\mathrm{SAT}(f(\ldots) \wedge \mathsf{AX}\ g(\ldots)) &= \{s_1\}
\end{aligned}$$

Adding an environment (CTL-FV)

Semantic patch

```
  f(x);
- g(x,y);
```



CTL representation

$$f(x) \wedge \mathrm{AX}\ g(x,y)$$

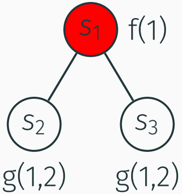
Model checking algorithm

$$
\begin{aligned}
\mathrm{SAT}(g(x,y)) &= \{(s_2,[x \mapsto 1, y \mapsto 2]);(s_3,[x \mapsto 1, y \mapsto 2])\} \\
\mathrm{SAT}(\mathrm{AX}\ g(x,y)) &= \{(s_1,[x \mapsto 1, y \mapsto 2])\} \\
\mathrm{SAT}(f(x)) &= \{(s_1,[x \mapsto 1])\} \\
\mathrm{SAT}(f(x) \wedge \mathrm{AX}\ g(x,y)) &= \{(s_1,[x \mapsto 1, y \mapsto 2])\}
\end{aligned}
$$

Problem: y has to be the same everywhere.

Semantic patch

```
  f(x);
- g(x,y);
```



CTL representation

$$f(x) \wedge \mathrm{AX}\ g(x,y)$$

Model checking algorithm

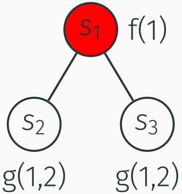
$$\begin{aligned}
\mathrm{SAT}(g(x,y)) &= \{(s_2,[x \mapsto 1, y \mapsto 2]); (s_3,[x \mapsto 1, y \mapsto 2])\}\\
\mathrm{SAT}(\mathrm{AX}\ g(x,y)) &= \{(s_1,[x \mapsto 1, y \mapsto 2])\}\\
\mathrm{SAT}(f(x)) &= \{(s_1,[x \mapsto 1])\}\\
\mathrm{SAT}(f(x) \wedge \mathrm{AX}\ g(x,y)) &= \{(s_1,[x \mapsto 1, y \mapsto 2])\}
\end{aligned}$$

Problem: y has to be the same everywhere.

Adding an environment (CTL-FV)

Semantic patch

```
  f(x);
- g(x,y);
```



CTL representation

$$f(x) \wedge \mathrm{AX}\ g(x,y)$$

Model checking algorithm

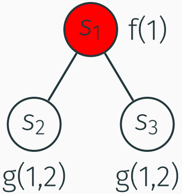
$$
\begin{aligned}
\mathrm{SAT}(g(x,y)) &= \{(s_2,[x \mapsto 1, y \mapsto 2]); (s_3,[x \mapsto 1, y \mapsto 2])\} \\
\mathrm{SAT}(\mathrm{AX}\ g(x,y)) &= \{(s_1,[x \mapsto 1, y \mapsto 2])\} \\
\mathrm{SAT}(f(x)) &= \{(s_1,[x \mapsto 1])\} \\
\mathrm{SAT}(f(x) \wedge \mathrm{AX}\ g(x,y)) &= \{(s_1,[x \mapsto 1, y \mapsto 2])\}
\end{aligned}
$$

Problem: y has to be the same everywhere.

Adding an environment (CTL-FV)

Semantic patch

```
  f(x);
- g(x,y);
```



CTL representation

$$f(x) \wedge \mathrm{AX}\ g(x, y)$$

Model checking algorithm

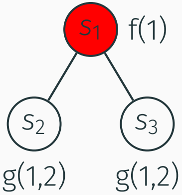
$$
\begin{aligned}
\mathrm{SAT}(g(x,y)) &= \{(s_2, [x \mapsto 1, y \mapsto 2]); (s_3, [x \mapsto 1, y \mapsto 2])\} \\
\mathrm{SAT}(\mathrm{AX}\ g(x,y)) &= \{(s_1, [x \mapsto 1, y \mapsto 2])\} \\
\mathrm{SAT}(f(x)) &= \{(s_1, [x \mapsto 1])\} \\
\mathrm{SAT}(f(x) \wedge \mathrm{AX}\ g(x,y)) &= \{(s_1, [x \mapsto 1, y \mapsto 2])\}
\end{aligned}
$$

Problem: y has to be the same everywhere.

Adding an environment (CTL-FV)

Semantic patch

```
  f(x);
- g(x,y);
```



CTL representation

$$f(x) \wedge \mathrm{AX}\ g(x, y)$$

Model checking algorithm

$$\begin{aligned}
\mathrm{SAT}(g(x,y)) &= \{(s_2, [x \mapsto 1, y \mapsto 2]); (s_3, [x \mapsto 1, y \mapsto 2])\} \\
\mathrm{SAT}(\mathrm{AX}\ g(x,y)) &= \{(s_1, [x \mapsto 1, y \mapsto 2])\} \\
\mathrm{SAT}(f(x)) &= \{(s_1, [x \mapsto 1])\} \\
\mathrm{SAT}(f(x) \wedge \mathrm{AX}\ g(x,y)) &= \{(s_1, [x \mapsto 1, y \mapsto 2])\}
\end{aligned}$$

Problem: *y* has to be the same everywhere.

Example

Semantic patch

```
@@
expression root,e,e1;
local idexpression child;
iterator name for_each_child_of_node;
@@

for_each_child_of_node(root, child) {
   ... when != of_node_put(child)
       when != e = child
(
   return <+...child...+>;
|
+  of_node_put(child);
?  return e1;
)
   ...
}
```

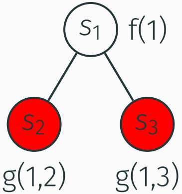
C code

```
for_each_child_of_node(np, child) {
  ...
  ret = of_property_read_u32(child, "reg",
                             &reg);
  if (ret) {
    dev_err(dev, "Failed to get reg property");
    return ret;
  }
  if (reg >= MX25_NUM_CFGS) {
    dev_err(dev,
      "reg value is greater than the ...");
    return -EINVAL;
  }
  ...
}
```

Adding existential quantification (CTL-V)

Semantic patch

```
  f(x);
- g(x,y);
```



CTL representation

$$\exists x.(f(x) \wedge \mathrm{AX}\ \exists y.\ g(x,y))$$

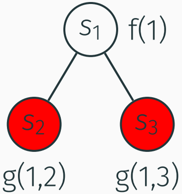
Model checking algorithm

$$
\begin{aligned}
\mathrm{SAT}(g(x,y)) &= \{(s_2,[x \mapsto 1, y \mapsto 2]);(s_3,[x \mapsto 1, y \mapsto 3])\}\\
\mathrm{SAT}(\exists y.g(x,y)) &= \{(s_2,[x \mapsto 1]);(s_3,[x \mapsto 1])\}\\
\mathrm{SAT}(\mathrm{AX}\ \exists y.g(x,y)) &= \{(s_1,[x \mapsto 1])\}\\
\mathrm{SAT}(f(x) \wedge \mathrm{AX}\ \exists y.g(x,y)) &= \{(s_1,[x \mapsto 1])\}\\
\exists x.(\mathrm{SAT}(f(x) \wedge \mathrm{AX}\ \exists y.g(x,y))) &= \{(s_1,\emptyset)\}
\end{aligned}
$$

Adding existential quantification (CTL-V)

Semantic patch

```
  f(x);
- g(x,y);
```



CTL representation

$$\exists x.(f(x) \wedge \mathrm{AX}\ \exists y.\ g(x,y))$$

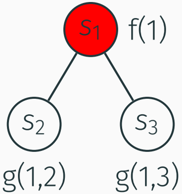
Model checking algorithm

$$
\begin{aligned}
\mathrm{SAT}(g(x,y)) &= \{(s_2,[x \mapsto 1, y \mapsto 2]); (s_3,[x \mapsto 1, y \mapsto 3])\} \\
\mathrm{SAT}(\exists y.g(x,y)) &= \{(s_2,[x \mapsto 1]); (s_3,[x \mapsto 1])\} \\
\mathrm{SAT}(\mathrm{AX}\ \exists y.g(x,y)) &= \{(s_1,[x \mapsto 1])\} \\
\mathrm{SAT}(f(x) \wedge \mathrm{AX}\ \exists y.g(x,y)) &= \{(s_1,[x \mapsto 1])\} \\
\exists x.(\mathrm{SAT}(f(x) \wedge \mathrm{AX}\ \exists y.g(x,y))) &= \{(s_1,\emptyset)\}
\end{aligned}
$$

Adding existential quantification (CTL-V)

Semantic patch

```
  f(x);
- g(x,y);
```



CTL representation

$$\exists x.(f(x) \wedge \mathrm{AX}\ \exists y.\ g(x,y))$$

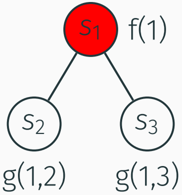
Model checking algorithm

$$
\begin{aligned}
\mathrm{SAT}(g(x,y)) &= \{(s_2,[x \mapsto 1, y \mapsto 2]); (s_3,[x \mapsto 1, y \mapsto 3])\} \\
\mathrm{SAT}(\exists y.g(x,y)) &= \{(s_2,[x \mapsto 1]); (s_3,[x \mapsto 1])\} \\
\mathrm{SAT}(\mathrm{AX}\ \exists y.g(x,y)) &= \{(s_1,[x \mapsto 1])\} \\
\mathrm{SAT}(f(x) \wedge \mathrm{AX}\ \exists y.g(x,y)) &= \{(s_1,[x \mapsto 1])\} \\
\exists x.(\mathrm{SAT}(f(x) \wedge \mathrm{AX}\ \exists y.g(x,y))) &= \{(s_1,\emptyset)\}
\end{aligned}
$$

Adding existential quantification (CTL-V)

Semantic patch

```
  f(x);
- g(x,y);
```



CTL representation

$$\exists x.(f(x) \wedge \text{AX } \exists y.\ g(x,y))$$

Model checking algorithm

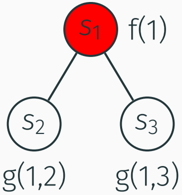
$$\begin{aligned}
\mathrm{SAT}(g(x,y)) &= \{(s_2, [x \mapsto 1, y \mapsto 2]); (s_3, [x \mapsto 1, y \mapsto 3])\} \\
\mathrm{SAT}(\exists y.g(x,y)) &= \{(s_2, [x \mapsto 1]); (s_3, [x \mapsto 1])\} \\
\mathrm{SAT}(\text{AX } \exists y.g(x,y)) &= \{(s_1, [x \mapsto 1])\} \\
\mathrm{SAT}(f(x) \wedge \text{AX } \exists y.g(x,y)) &= \{(s_1, [x \mapsto 1])\}
\end{aligned}$$

$$\exists x.(\mathrm{SAT}(f(x) \wedge \text{AX } \exists y.g(x,y))) = \{(s_1, \emptyset)\}$$

Adding existential quantification (CTL-V)

Semantic patch

```
  f(x);
- g(x,y);
```



CTL representation

$$\exists x.(f(x) \wedge \mathrm{AX}\ \exists y.\ g(x,y))$$

Model checking algorithm

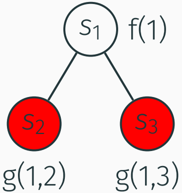
$$
\begin{aligned}
\mathrm{SAT}(g(x,y)) &= \{(s_2,[x \mapsto 1, y \mapsto 2]);(s_3,[x \mapsto 1, y \mapsto 3])\}\\
\mathrm{SAT}(\exists y.g(x,y)) &= \{(s_2,[x \mapsto 1]);(s_3,[x \mapsto 1])\}\\
\mathrm{SAT}(\mathrm{AX}\ \exists y.g(x,y)) &= \{(s_1,[x \mapsto 1])\}\\
\mathrm{SAT}(f(x) \wedge \mathrm{AX}\ \exists y.g(x,y)) &= \{(s_1,[x \mapsto 1])\}\\
\exists x.(\mathrm{SAT}(f(x) \wedge \mathrm{AX}\ \exists y.g(x,y))) &= \{(s_1,\emptyset)\}
\end{aligned}
$$

Adding witnesses (CTL-VW)

Goal: collect information about how and where to transform

Semantic patch

```
  f(x);
- g(x,y);
```



CTL representation

$$\exists x.(f(x) \wedge AX\ \exists y.\ g(x,y))$$

Model checking algorithm

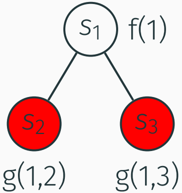
$$\begin{aligned}
\mathrm{SAT}(g(x,y)) &= \{(s_2,[x\mapsto 1, y\mapsto 2],()); (s_3,[x\mapsto 1, y\mapsto 3],())\} \\
\mathrm{SAT}(\exists y.g(x,y)) &= \{(s_2,[x\mapsto 1],(\langle s_2,[y\mapsto 2],()\rangle)); \\
&\quad\ (s_3,[x\mapsto 1],(\langle s_3,[y\mapsto 3],()\rangle))\} \\
\mathrm{SAT}(AX\ \exists y.g(x,y)) &= \{(s_1,[x\mapsto 1],(\langle s_2,[y\mapsto 2],()\rangle,\langle s_3,[y\mapsto 3],()\rangle))\}
\end{aligned}$$

Adding witnesses (CTL-VW)

Goal: collect information about how and where to transform

Semantic patch

```
  f(x);
- g(x,y);
```



CTL representation

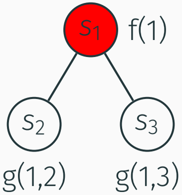
$$\exists x.(f(x) \wedge \mathrm{AX}\ \exists y.\ g(x,y))$$

Model checking algorithm

$$
\begin{aligned}
\mathrm{SAT}(g(x,y)) &= \{(s_2,[x \mapsto 1, y \mapsto 2],()); (s_3,[x \mapsto 1, y \mapsto 3],())\} \\
\mathrm{SAT}(\exists y.g(x,y)) &= \{(s_2,[x \mapsto 1],(\langle s_2,[y \mapsto 2],()\rangle)); \\
&\quad\ (s_3,[x \mapsto 1],(\langle s_3,[y \mapsto 3],()\rangle))\} \\
\mathrm{SAT}(\mathrm{AX}\ \exists y.g(x,y)) &= \{(s_1,[x \mapsto 1],(\langle s_2,[y \mapsto 2],()\rangle,\langle s_3,[y \mapsto 3],()\rangle))\}
\end{aligned}
$$

Adding witnesses (CTL-VW)

Goal: collect information about how and where to transform



Semantic patch

```
  f(x);
- g(x,y);
```

CTL representation

$$\exists x.(f(x) \wedge \mathrm{AX}\ \exists y.\ g(x,y))$$

Model checking algorithm

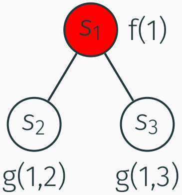
$$\begin{aligned}
\mathrm{SAT}(g(x,y)) &= \{(s_2,[x \mapsto 1, y \mapsto 2],()); (s_3,[x \mapsto 1, y \mapsto 3],())\} \\
\mathrm{SAT}(\exists y.g(x,y)) &= \{(s_2,[x \mapsto 1],(\langle s_2,[y \mapsto 2],()\rangle)); \\
&\quad\ (s_3,[x \mapsto 1],(\langle s_3,[y \mapsto 3],()\rangle))\} \\
\mathrm{SAT}(\mathrm{AX}\ \exists y.g(x,y)) &= \{(s_1,[x \mapsto 1],(\langle s_2,[y \mapsto 2],()\rangle,\langle s_3,[y \mapsto 3],()\rangle))\}
\end{aligned}$$

Adding witnesses (CTL-VW)

Goal: collect information about how and where to transform

Semantic patch

```
  f(x);
- g(x,y);
```



CTL representation

$$\exists x.(f(x) \wedge \text{AX}\ \exists y.\ g(x,y))$$

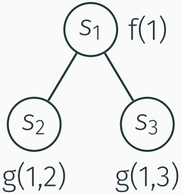
Model checking algorithm

$$\text{SAT}(\exists x.(f(x) \wedge \text{AX}\ \exists y.\ g(x,y))) =$$
$$\{(s_1, \emptyset, (\langle s_1, [x \mapsto 1], (\langle s_2, [y \mapsto 2], ()\rangle, \langle s_3, [y \mapsto 3], ()\rangle)\rangle))\}$$

Witnessing transformations

Semantic patch

```
  f(x);
- g(x,y);
```



CTL representation

$$\exists x.(f(x) \ \wedge \ \mathrm{AX}\ (\exists y.\ g(x,y) \ \wedge \ \exists v.\ \mathrm{matches}("g(x,y)^{-}", v)))$$

Model checking

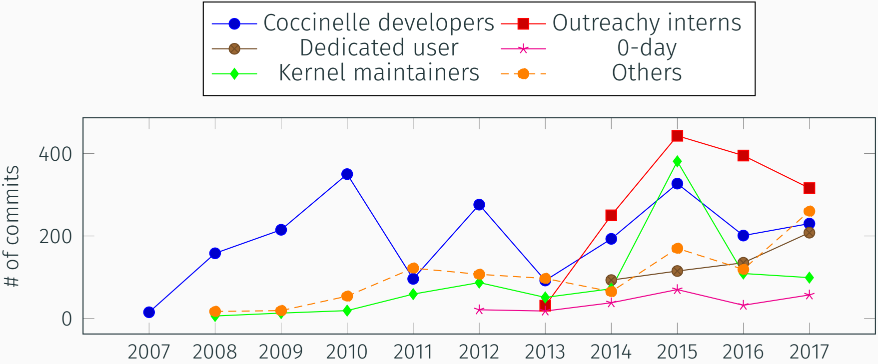
$$\begin{aligned}\mathrm{SAT}(g(x,y) \wedge \exists v.\ \mathrm{matches}("g(x,y)^{-}", v)) \ = \\ \{(s_2, [x \mapsto 1, y \mapsto 2], (\langle s_2, [v \mapsto "g(x,y)^{-}"], ()\rangle)); \\ s_3, [x \mapsto 1, y \mapsto 3], (\langle s_3, [v \mapsto "g(x,y)^{-}"], ()\rangle))\}\end{aligned}$$

Result

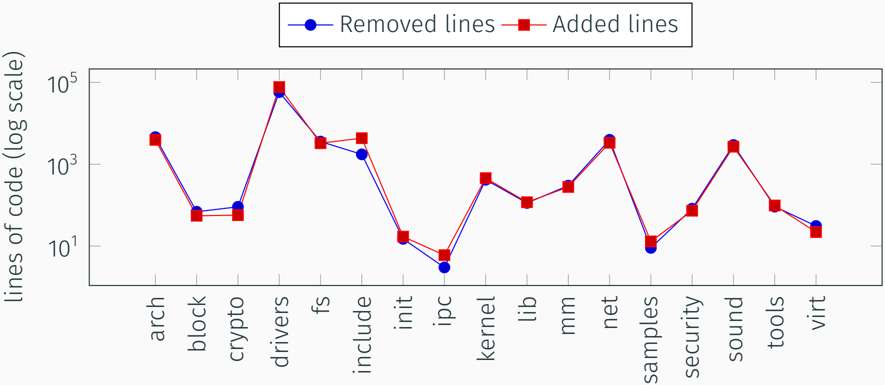
$$\begin{aligned}\{(s_1, \emptyset, (\langle s_1, [x \mapsto 1], (\langle s_2, [y \mapsto 2], (\langle s_2, [v \mapsto "g(x,y)^{-}"], ()\rangle)\rangle, \\ \langle s_3, [y \mapsto 3], (\langle s_3, [v \mapsto "g(x,y)^{-}"], ()\rangle)\rangle)\rangle))\}\end{aligned}$$

Practical application

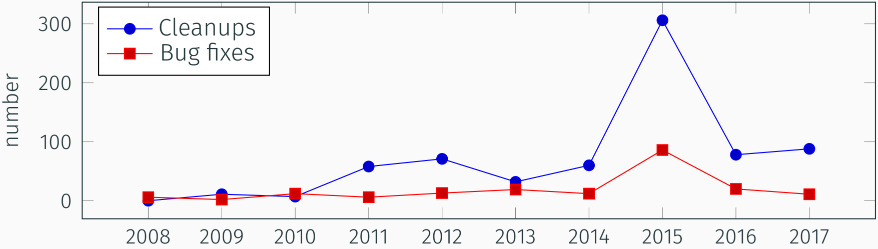
Usage in the Linux kernel



Impact: Changed lines



Impact: Maintainer use



45% of maintainers who have at least one commit touching at least 100 files have at some point used Coccinelle.

Impact: Maintainer use examples

TTY. Remove an unused function argument.

- 11 affected files.

DRM. Eliminate a redundant field in a data structure.

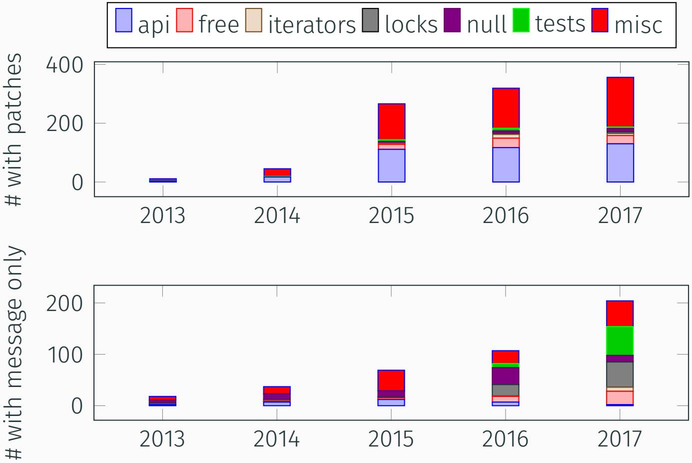
- 54 affected files.

Interrupts. Prepare to remove the irq argument from interrupt handlers, and then remove that argument.

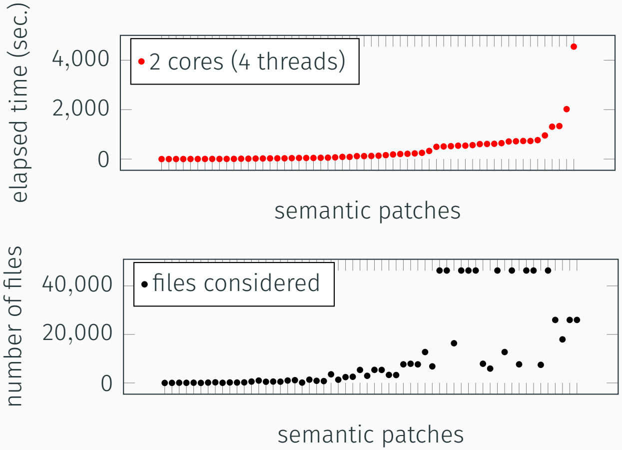
- 188 affected files.

Impact: Intel's 0-day build-testing service

59 semantic patches in the Linux kernel with a dedicated make target.



Performance



Based on the 59 semantic patches in the Linux kernel.

Coccinelle community

25 contributors

- Most at Inria, due to use of OCaml and PL concepts.
- Active mailing list.

Availability

- Packaged for many Linux distros.

Use outside Linux

- RIOT, systemd, qemu, etc.

Summary: Theoretical results

Semantics and model checking algorithm for CTL with environments (CTL-V)

- Soundness and completeness proved.
- Proof validated with Coq.

Semantics and model checking algorithm for CTL with environments and witnesses (CTL-VW)

- Soundness and completeness proved.
- Not yet validated with Coq.

More details in POPL 2009.

Summary: Practical results

Collateral evolutions

- Semantic patches for over 60 collateral evolutions.
- Applied to over 5800 Linux files from various versions, with a success rate of 100% on 93% of the files.

Bug finding

- Generic bug types:
  - Null pointer dereference, initialization of unused variables, `!x&y`, etc.
- Bugs in the use of Linux APIs:
  - Incoherent error checking, memory leaks, etc.

∼6000 patches created using Coccinelle accepted into Linux

Used by other Linux kernel developers

Conclusion

A patch-like program matching and transformation language

Simple and "efficient" extension of CTL that is useful for this domain

Accepted by Linux developers

Future work

- Programming languages other than C
- Semantic patch inference

Coccinelle is publicly available

http://coccinelle.lip6.fr/