A Foundation for Flow-Based Program Matching

Using Temporal Logic and Model Checking

Julia Lawall (University of Copenhagen)

Joint work with
Jesper Andersen, Julien Brunel, Damien Doligez,
René Rydhof Hansen,
Gilles Muller, Yoann Padioleau, and Nicolas Palix
DIKU-EMN

September 17, 2009

Overview

Goal: Describe and automate transformations on C code

1. Collateral evolutions.
2. Bug finding and fixing.

- Focus on open-source software, particularly Linux.

Our approach: Coccinelle

- Semantic patch language (SmPL).

Requirements for a SmPL implementation.

- CTL-based implementation.
- Extension of CTL with environments and witnesses.

Some theoretical and practical results.

Conclusions and future work.

Collateral evolutions

The collateral evolution problem:

- Library functions change.
- Client code must be adapted.
  - Change a function name, add an argument, etc.
- Linux context:
  - Many libraries: usb, net, etc.
  - Very many clients, including outside the Linux source tree.

Example

Evolution: New constants:

`IRQF_DISABLED`, `IRQF_SAMPLE_RANDOM`, *etc*.

⟹ Collateral evolution: Replace old constants by the new ones.

```
@@ -96,7 +96,7 @@ static int __init hp6x0_apm_init(void)
   int ret;

   ret = request_irq(HP680_BTN_IRQ, hp6x0_apm_interrupt,
-                    SA_INTERRUPT, MODNAME, 0);
+                    IRQF_DISABLED, MODNAME, 0);
   if (unlikely(ret < 0)) {
           printk(KERN_ERR MODNAME ": IRQ %d request failed",
                  HP680_BTN_IRQ);
```

Another 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));
```

Another 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;
}
```

Find and fix bugs

Example: Reference count abuses

```
a = get();
b = get();
if (x) return -1;
if (y) { put(a); return -2; }
put(a);
put(b);
```

- Find returns without `put`s.
- Fix by adding a `put` before such a `return`.

Our proposal: Coccinelle

Program matching and transformation for unpreprocessed C code.

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.

Example

```
@@ @@
(
- SA_INTERRUPT
+ IRQF_DISABLED
|
- SA_SAMPLE_RANDOM
+ IRQF_SAMPLE_RANDOM
|
- SA_SHIRQ
+ IRQF_SHARED
|
- SA_PROBEIRQ
+ IRQF_PROBE_SHARED
|
- SA_PERCPU_IRQ
+ IRQF_PERCPU
)
```

Constructing a semantic patch

```
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));
```

Constructing a semantic patch

Eliminate irrelevant code

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


...


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

Constructing a semantic patch

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));
```

Constructing a semantic patch

Abstract over subterms

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



@@

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

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

Constructing a semantic patch

Refinement

```
@@
expression x;
expression E1,E2,E3;
statement S;
identifier f;

@@

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

Constructing a semantic patch

Generalization

```
@@
expression x;
expression E1,E2,E3;
statement S;
identifier f;
type T,T2;
@@

- x = (T)kmalloc(E1,E2)
+ x = kzalloc(E1,E2)
  ...  when != ( f(...,x,...) | <+...x...+> = E3 )
       when != ( while(...) S | for(...;...;...) S )
- memset((T2)x, 0, E1);
```

Updates 355/564 files

Requirements

- Reason about possible execution paths.

- Keep track of different variables.
  - get/put for a, vs get/put for b.

- Collect transformation information
  - Where to transform?
  - What transformation to carry out?

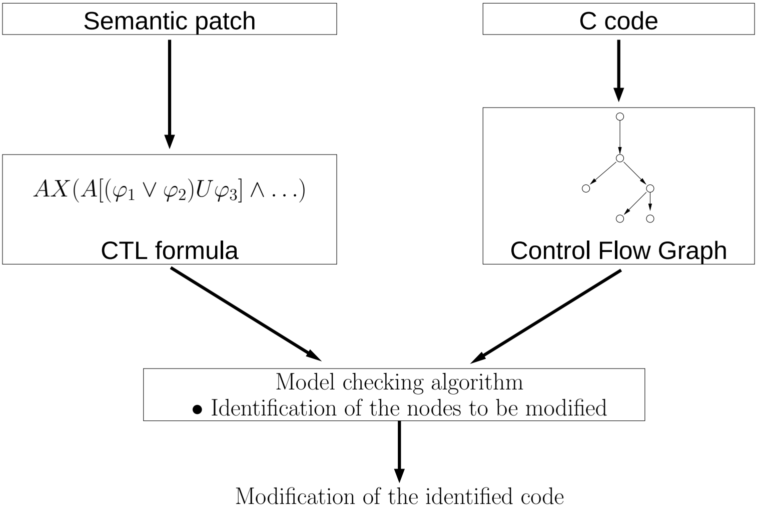
Requirements

- 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.
  - – get/put for a, vs get/put for b.
- Collect transformation information
  - – Where to transform?
  - – What transformation to carry out?

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.
  - – get/put for a, vs get/put for b.
  - ⇒ 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

C code

$AX(A[(\varphi_1 \vee \varphi_2)U\varphi_3] \wedge \ldots)$

CTL formula

Control Flow Graph

Model checking algorithm

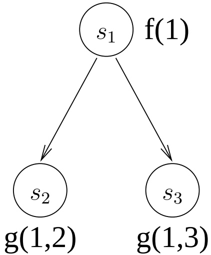
- Identification of the nodes to be modified

Modification of the identified code

CTL translation and model checking

Semantic patch

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



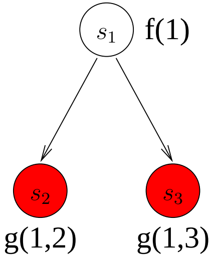
CTL representation

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

CTL translation and model checking

Semantic patch

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



CTL representation

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

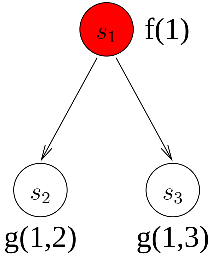
Model checking algorithm

$$\mathrm{SAT}(g(\ldots)) \quad = \quad \{s_2, s_3\}$$

CTL translation and model checking

Semantic patch

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



CTL representation

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

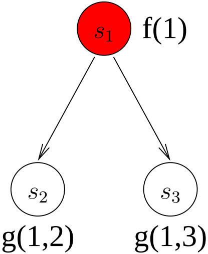
Model checking algorithm

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

CTL translation and model checking

Semantic patch

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



CTL representation

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

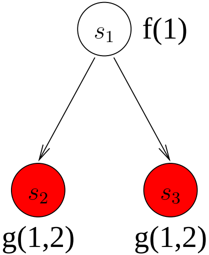
Model checking algorithm

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

Adding an environment (CTL-FV)

Semantic patch

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



CTL representation

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

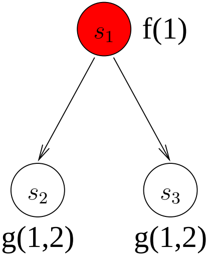
Model checking algorithm

$$\mathrm{SAT}(g(x,y)) \quad = \quad \{(s_2,[x \mapsto 1, y \mapsto 2]);(s_3,[x \mapsto 1, y \mapsto 2])\}$$

Adding an environment (CTL-FV)

Semantic patch

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



CTL representation

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

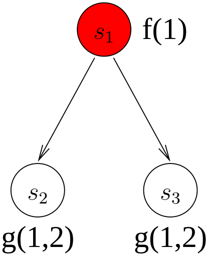
Model checking algorithm

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

Adding an environment (CTL-FV)

Semantic patch

```
  f(X);
- g(X,Y);
```



CTL representation

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

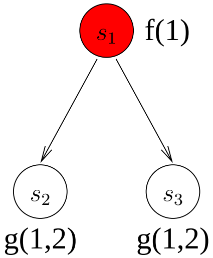
Model checking algorithm

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

Adding an environment (CTL-FV)

Semantic patch

```
  f(X);
- g(X,Y);
```



CTL representation

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

Model checking algorithm

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

Problem: y has to be the same everywhere.

Example

Semantic patch

```
@@
  expression n, E;
@@

  n = get();
  ... when != put(n)
(
  put(n);
|
+ put(n);
  return E;
)
```

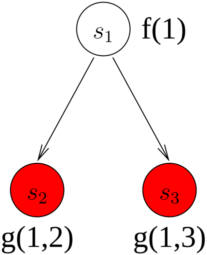
C code

```
a = get();
b = get();
if (x) return -1;
if (y) { put(a); return -2; }
put(a);
put(b);
```

Adding existential quantification (CTL-V)

Semantic patch

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



CTL representation

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

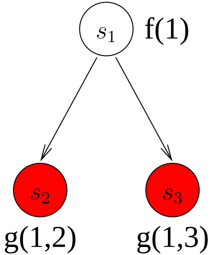
Model checking algorithm

$$\mathrm{SAT}(g(x,y)) \quad = \quad \{(s_2, [x \mapsto 1, y \mapsto 2]); (s_3, [x \mapsto 1, y \mapsto 3])\}$$

Adding existential quantification (CTL-V)

Semantic patch

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



CTL representation

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

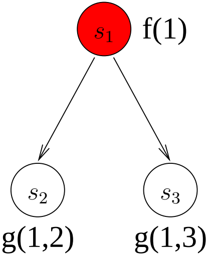
Model checking algorithm

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

Adding existential quantification (CTL-V)

Semantic patch

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



CTL representation

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

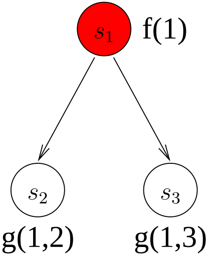
Model checking algorithm

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

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

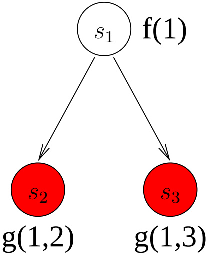
$$\begin{aligned}
\mathrm{SAT}(g(x,y)) &= \{(s_2,[x \mapsto 1, y \mapsto 2]);(s_3,[x \mapsto 1, y \mapsto 3])\}\\
\mathrm{SAT}(\exists y.g(x,y)) &= \{(s_2,[x \mapsto 1]);(s_3,[x \mapsto 1])\}\\
\mathrm{SAT}(\mathrm{AX}\ \exists y.g(x,y)) &= \{(s_1,[x \mapsto 1])\}\\
\mathrm{SAT}(f(x) \wedge \mathrm{AX}\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

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

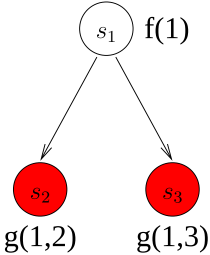
$$\mathrm{SAT}(g(x,y)) \quad = \quad \{(s_2,[x \mapsto 1, y \mapsto 2],()); (s_3,[x \mapsto 1, y \mapsto 3],())\}$$

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

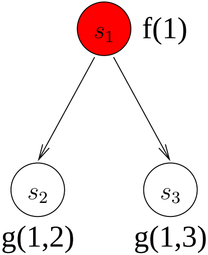
$$\begin{array}{lcl}
\mathrm{SAT}(g(x,y)) & = & \{(s_2, [x \mapsto 1, y \mapsto 2], ()); (s_3, [x \mapsto 1, y \mapsto 3], ())\} \\
\mathrm{SAT}(\exists y.g(x,y)) & = & \{(s_2, [x \mapsto 1], (\langle s_2, [y \mapsto 2], ()\rangle)); \\
 & & \ (s_3, [x \mapsto 1], (\langle s_3, [y \mapsto 3], ()\rangle))\}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

Adding witnesses (CTL-VW)

Goal: collect information about how and where to transform

Semantic patch

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



CTL representation

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

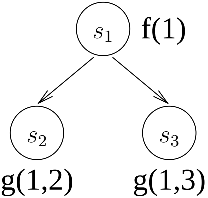
Model checking algorithm

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

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

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

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.

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.

Over 350 patches created using Coccinelle accepted into Linux

Starting to be used by other Linux developers

Probable bugs found in gcc, postgresql, vim, amsn, pidgin, mplayer

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

- Extension to other kinds of logics for other kinds of matching effects
- Programming languages other than C
- Dataflow and interprocedural analysis

Coccinelle is publicly available

`http://coccinelle.lip6.fr/`