Improving OS safety using the Coccinelle Program Matching and Transformation Tool

Gilles Muller (INRIA/LIP6),
Julia Lawall (DIKU),
Jesper Andersen, Julien Brunel, René Rydhof Hansen, Yoann Padioleau, and Nicolas Palix

http://coccinelle.lip6.fr

The problem: Dealing with Systems Code

- It's huge
- It's configuration polymorph
- It's (unfortunately) buggy
- It's often written in C
- It evolves continuously

Two Examples

- Bug finding (and fixing)
  - Search for patterns of wrong code
  - Systematically fix found wrong code
- Collateral evolutions
  - Evolution in a library interface entails lots of Collateral Evolutions in clients
    - Search for patterns of interaction with the library
    - Systematically transform the interaction code

The Coccinelle tool

- Program matching and transformation for unpreprocessed C code.
- Fits with the existing habits of Systems (Linux) programmers.

- Semantic Patch Language (SmPL):
  - Based on the syntax of patches,
  - Declarative approach to transformation
  - High level search that abstracts away from irrelevant details
  - A single small semantic patch can modify hundreds of files, at thousands of code sites

Using SmPL to abstract away from irrelevant details

- Differences in spacing, indentation, and comments
- Choice of the names given to variables (metavariables)
- Irrelevant code ('. . .', control flow oriented)
- Other variations in coding style (isomorphisms)

  e.g. `if(!y)` ≡ `if(y==NULL)` ≡ `if(NULL==y)`

Bug finding and fixing

- The "!&" bug

C allows mixing booleans and bit constants

```
if (!state->card->
    ac97_status & CENTER_LFE_ON)
        val &= ~DSP_BIND_CENTER_LFE;
```

In sound/oss/ali5455.c until Linux 2.6.18

(problem is over two lines)

A Simple SmPL Sample

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

- !E & C                    // !C is not a constant
+ !(E & C)
```

96 instances in Linux from 2.6.13 (August 2005) to v2.6.28 (December 2008)

58 in 2.6.20 (February 2007),

2 in Linux-next (19th September 2009)

Collateral Evolutions

Evolution

lib.c

becomes

```
int foo(int x){
int bar(int x, int y){
```

Legend: before / after

Collateral Evolutions (CE) in clients

client1.c

```
foo(1);
bar(1,?);

foo(2);
bar(2,?);
```

client2.c

```
foo(foo(2));
bar(bar(2,?),?);

if(foo(3)) {
if(bar(3,?)) {
```

clientn.c

CE in Linux device drivers

- Many libraries and many clients:
  - Lots of driver support libraries: one per device type, one per bus (pci library, sound library, …)
  - Lots of device specific code: Drivers make up more than 50% of Linux
- Many evolutions and collateral evolutions

  1200 evolutions in 2.6, some affecting 400 files, at over 1000 sites [EuroSys 2006] (summer 2005)

- Taxonomy of evolutions :

  Add argument, split data structure, getter and setter introduction, protocol change, change return type, add error checking, …

Example from Linux 2.5.71

- Evolution: scsi_get()/scsi_put() dropped from SCSI library
- Collateral evolutions: SCSI resource now passed directly to *proc_info* callback functions via a new parameter

```
int a_proc_info(int x
                ,scsi *y
                ) {
   scsi *y;
   ...
   y = scsi_get();
   if(!y) { ... return -1; }
   ...
   scsi_put(y);
   ...
   }
```

From local var to parameter

Delete calls to library

Delete error checking code

Legend: before / after

Semantic Patch

```
@@
function a_proc_info;
identifier x,y;
@@
   int a_proc_info(int x
+                  ,scsi *y
                           ) {
-    scsi *y;
     ...
-    y = scsi_get();
-    if(!y) { ... return -1; }
     ...
-    scsi_put(y);
     ...
   }
```

Affected Linux driver code

drivers/scsi/53c700.c

```
int s53c700_info(int limit)
{
  char *buf;
  scsi *sc;
  sc = scsi_get();
  if(!sc) {
     printk(“error”);
     return -1;
  }
  wd7000_setup(sc);
  PRINTP(“val=%d”,
          sc->field+limit);
  scsi_put(sc);
  return 0;
}
```

drivers/scsi/pcmcia/nsp_cs.c

```
int nsp_proc_info(int lim)
{
  scsi *host;
  host = scsi_get();
  if(!host) {
     printk(“nsp_error”);
     return -1;
  }
  SPRINTF(“NINJASCSI=%d”,
          host->base);
  scsi_put(host);
  return 0;
}
```

Similar, but not identical

Applying the semantic patch

```
int s53c700_info(int limit)
{
  char *buf;
  scsi *sc;
  sc = scsi_get();
  if(!sc) {
     printk("error");
     return -1;
  }
  wd7000_setup(sc);
  PRINTP("val=%d",
          sc->field+limit);
  scsi_put(sc);
  return 0;
}
```

```
int nsp_proc_info(int lim)
{
  scsi *host;
  host = scsi_get();
  if(!host) {
     printk("nsp_error");
     return -1;
  }
  SPRINTF("NINJASCSI=%d",
           host->base);
  scsi_put(host);
  return 0;
}
```

proc_info.sp

```
@@
function a_proc_info;
identifier x,y;
@@
 int a_proc_info(int x
+                ,scsi *y
                    ) {
-   scsi *y;
    ...
-   y = scsi_get();
-   if(!y) { ... return -1; }
    ...
-   scsi_put(y);
      ...
  }
```

```
$ spatch –sp_file proc_info.sp
  –dir linux-next
```

Applying the semantic patch

```
int s53c700_info(int limit, scsi *sc)
{
  char *buf;




  wd7000_setup(sc);
  PRINTP("val=%d",
         sc->field+limit);

  return 0;
}
```

```
int nsp_proc_info(int lim, scsi *host)
{


  SPRINTF("NINJASCSI=%d",
          host->base);

  return 0;
}
```

proc_info.sp

```
@@
function a_proc_info;
identifier x,y;
@@
 int a_proc_info(int x
+                ,scsi *y
                    ) {
-   scsi *y;
    ...
-   y = scsi_get();
-   if(!y) { ... return -1; }
    ...
-   scsi_put(y);
      ...
  }
```

```
$ spatch –sp_file proc_info.sp
  –dir linux-next
```

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;
}
```

Constructing the semantic patch

- Eliminate irrelevant code

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

  ...

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

Constructing the 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 the semantic patch

- Abstract over subterms

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

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

Constructing the semantic patch

- Refinement

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

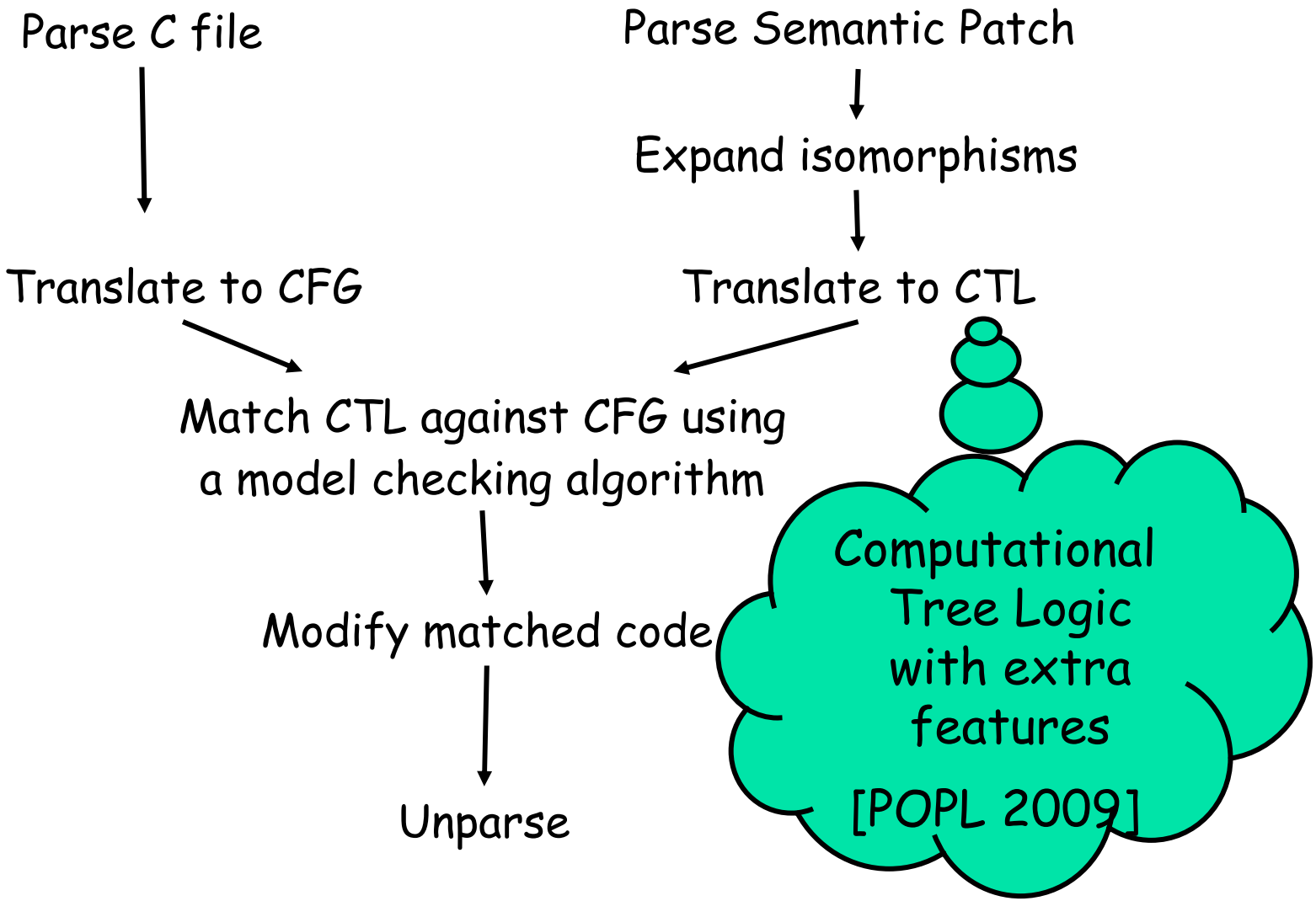
Constructing the semantic patch

- Generalization

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

How does the Coccinelle tool work?

Transformation engine



Other issues

- Need to produce readable code
  - Keep space, indentation, comments
  - Keep CPP instructions as-is. Also programmer may want to transform some #define,*iterator* macros (e.g. list_for_each)

Very different from most other C tools

- Interactive engine, partial match
- Implementation of isomorphisms
  - Rewriting the Semantic patch (not the C code),
  - Generate disjunctions

60 000 lines of OCaml code

Evaluation on Collateral Evolutions [Eurosys 2008]

Experiments

- Methodology
  - Detect past collateral evolutions in Linux 2.5 and 2.6 using the `patchparse` tool [Eurosys'06]
  - Select representative ones
    - Test suite of over 60 CEs
  - Study them and write corresponding semantic patches
    - Note: we are not kernel developers
- Going "back to the future". Compare:
  - what Linux programers did manually
  - What Coccinelle, given our SPs, does automatically

Test suite

- 20 Complex CEs : bugs introduced by the programmers
  - In each case 1-16 errors + misses
- 23 Mega CEs : affect over 100 sites on Linux between 2.6.12 and 2.6.20
  - 22-1124 files affected
  - Up to 39 human errors
  - Up to 40 people for up to two years
- 26 CEs for the bluetooth directory update from 2.6.12 to 2.6.20
  - Median case

More than 5800 driver files

Results

- SP are on average 106 lines long (6-369)
- SPs often 100 times smaller than "human-made" patches. A measure of time saved:
  - Not doing manually the CE on all the drivers
  - Not reading and reviewing big patches, for people with drivers outside source tree
- Correct and complete automated evolutions for 93% of the files
  - Problems on the remaining 7%: We miss code sites
    - CPP issues, lack of isomorphisms (data-flow and inter-procedural)
    - We are not kernel developers … don't know how to specify
- Average processing time of 0.7s per file

Sometimes the tool was right and the human wrong

Impact on the Linux kernel

- **Collateral evolution related SPs**
  - Over 11 semantic patches
  - Over 52 patches

- **SPs for bug-fixing and bad programming practices**
  - Over 57 semantic patches
  - Over 148+20 patches

>> 400 patches in total

Current/Future Work
Coccinelle *in the large*

- Protocol-based bug detection in Linux [DSN2009]

- Management of conflicts between Linux kernel and services (detection, solving)
- Version consistency
- Collaborative design of rules
  - Rule ranking
  - Collaborative refinements

Conclusion

- SmPL: a declarative language for program matching and transformation
- Looks like a patch; fits with Systems (Linux) programmers' habits
- Quite "easy" to learn; already accepted by the Linux community

- A transformation engine based on model checking technology

Questions?

CocciCheck your code, it's free....

http://coccinelle.lip6.fr

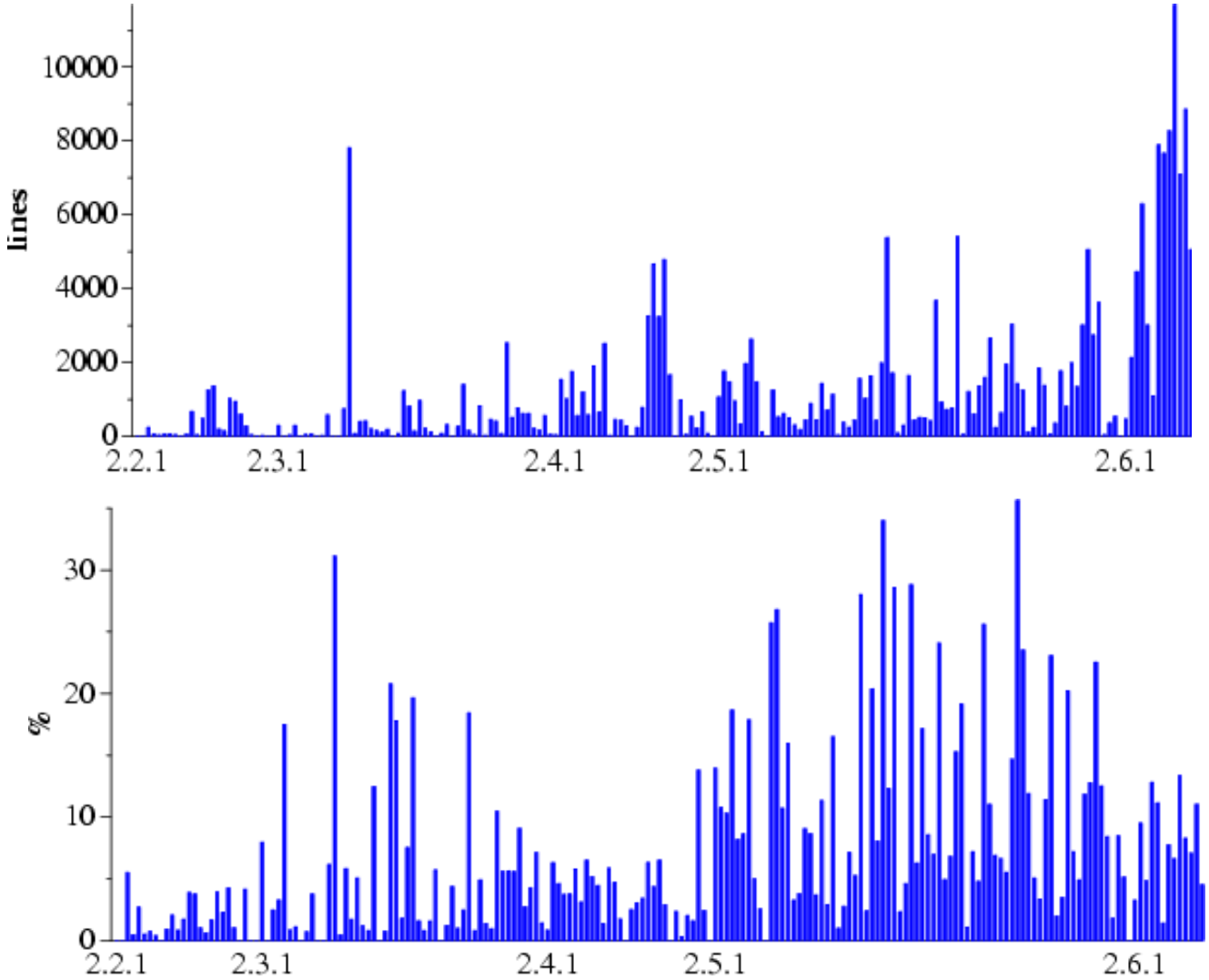
Why Coccinelle ?

A Coccinelle (ladybug) is a bug that eats smaller bugs

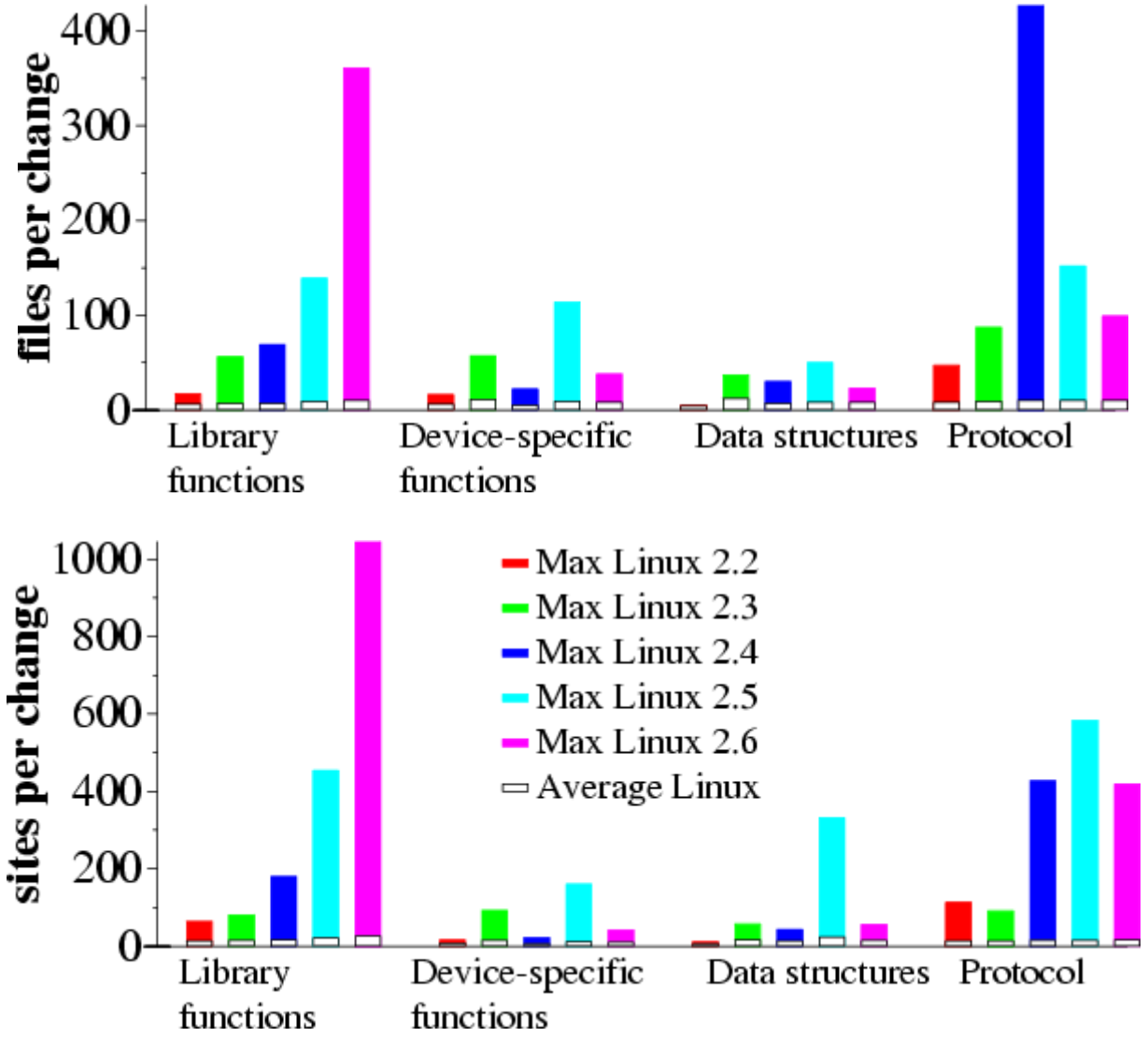
Kill bugs before they hatch!!!

COCCINELLE

Collateral Evolutions in Linux patches (summer 2005)



Sites and files affected by a collateral evolution



Computational Tree Logic (CTL)

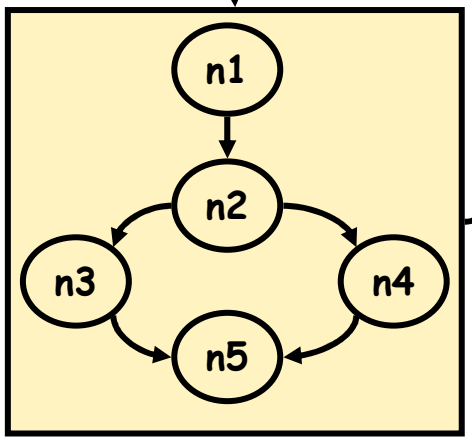
- CTL formula =~ regexp on tree
  - p,q,r, true, false, $\wedge$, $\vee$, $\neg$: as in propositional logic
  - X : next node
  - A[f U g], E [f U g] : forall/exist f until g
- Why CTL ?
  - Our CFG-oriented approach requires a formalism that works on graphs
  - Has been essential in prototyping

Example

C file

```
f(1);
if(exp) g(3);
else    g(4);
```

CFG



Semantic patch

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

CTL

$$\exists X.\, f(X);\ \&\ \mathbf{AX}\ \mathbf{A}[\mathbf{true}\ \mathbf{U}\ \exists Y.\exists v.\, g^{-}(^{-}Y^{-})^{-};^{-+g(X,Y);_{v}}]$$

match

Witness tree

Formula matches model at node 1 with witness tree:

- $X \to 1$
  - $Y \to 3,\ v \to (n3,\ g^{-}(^{-}Y^{-})^{-};^{-+g(X,Y)})$
  - $Y \to 4,\ v \to (n4,\ g^{-}(^{-}Y^{-})^{-};^{-+g(X,Y)})$

CTL and Model checking

- Model checking a CTL formula against a model answers just yes/no (with counter example).
- We do program transformations, not just "pattern checking":
  - Bind metavariables and remember their value
  - Remember where we have matched sub-formulas

- We have extended CTL : existential variables and program transformation annotations

Dissemination through patch

```
@@ -246,7 +246,8  @@
 - int wd7000_info(int x) {
 + int wd7000_info(int x, scsi y) {
   float z;
 - scsi *y;
   z = x + 1;
 - y = scsi_get();
 - if(!y) {
 -     kprintf("error");
 -     return -1;
 - }
   kprintf("val = %d", y->field + z);
 - scsi_put(y);
   return 0;
 }
```

line location in original file

"plus" line

"context" line

"minus" lines

- Result of `diff -u old.c new.c`
- Specific to a single file, to a code site
- Line-oriented