A Foundation for Flow-Based Program Matching

Using Temporal Logic and Model Checking

Julien Brunel, Onera, France
Damien Doligez, INRIA, France
René Rydhof Hansen, Aalborg University, Denmark
Julia L. Lawall, University of Copenhagen, Denmark
Gilles Muller, EMN, France

Coccinelle project

`http://www.emn.fr/x-info/coccinelle/`

Background

Our goal: Program matching and transformation

- Update API usage.
- Find and fix bugs.

Example: Reference count abuses (simplified Linux code):

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

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

Coccinelle, a program matching and transformation tool

Semantic patch

```
@@
  expression n;
@@

  n = get();
  ...
  put(n);
```

C code

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

Coccinelle, a program matching and transformation tool

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);
```

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?

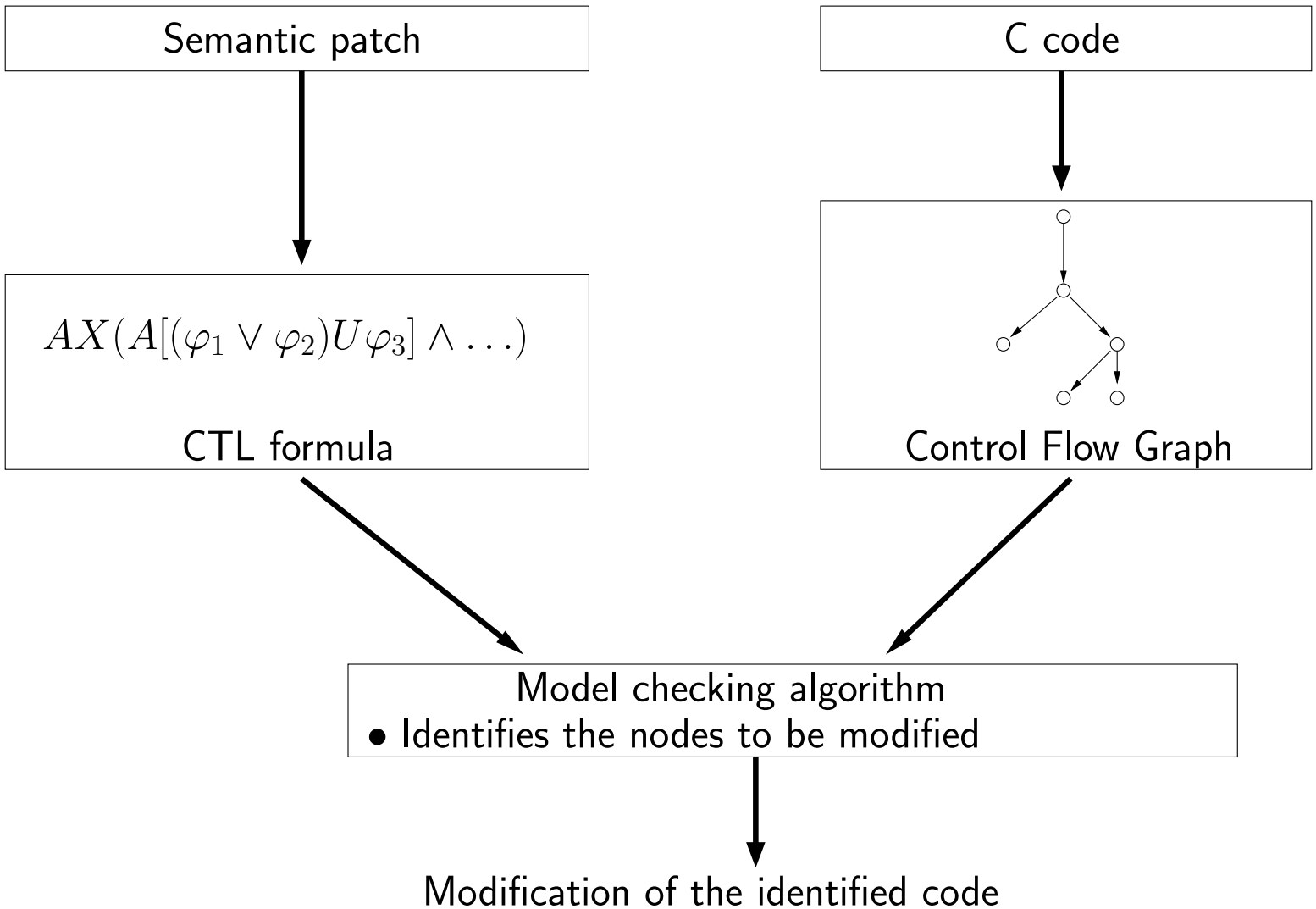
Requirements

- Reason about possible execution paths.
  - $\Rightarrow$ Suggests matching against paths in a control-flow graph.
  - $\Rightarrow$ 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.
  - $\Rightarrow$ Suggests matching against paths in a control-flow graph.
  - $\Rightarrow$ 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.
  - $\Rightarrow$ Our contribution (CTL-V model checking algorithm).
- Collect transformation information
  - Where to transform?
  - What transformation to carry out?
  - $\Rightarrow$ Our contribution (CTL-VW).

Coccinelle architecture



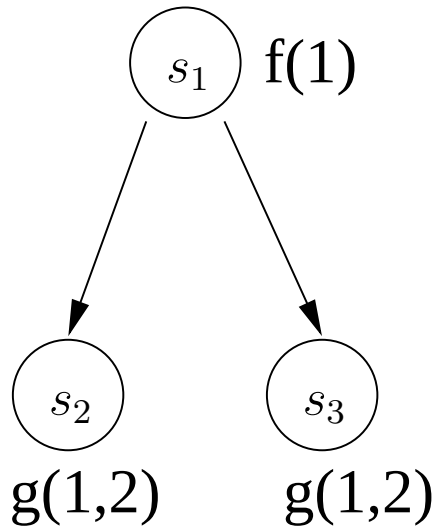
CTL translation and model checking

Semantic patch

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

CTL representation

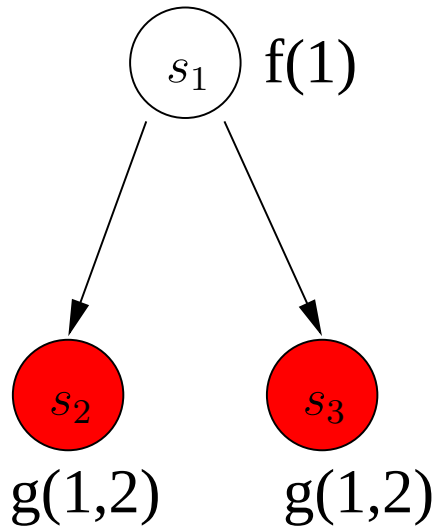
$$f(\ldots) \wedge \mathsf{AX}\, g(\ldots)$$



CTL translation and model checking

Semantic patch

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



CTL representation

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

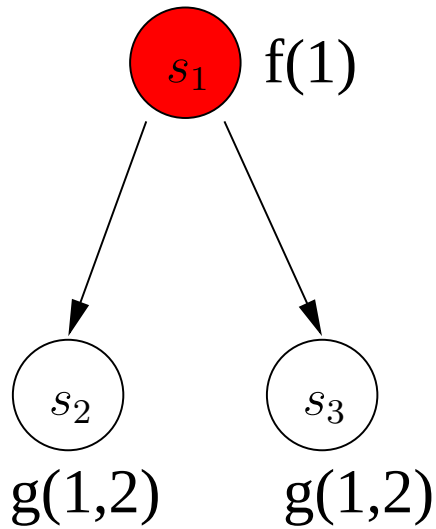
Model checking algorithm

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

CTL translation and model checking

Semantic patch

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



CTL representation

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

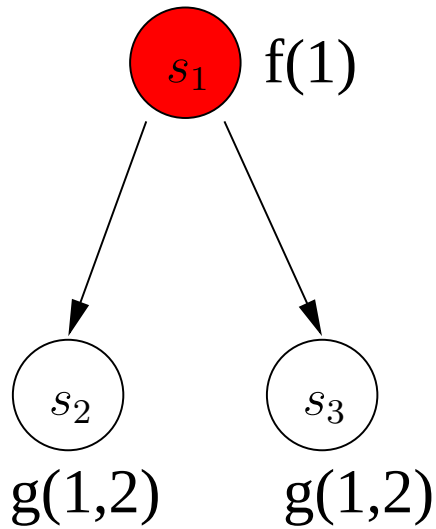
Model checking algorithm

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

CTL translation and model checking

Semantic patch

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



CTL representation

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

Model checking algorithm

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

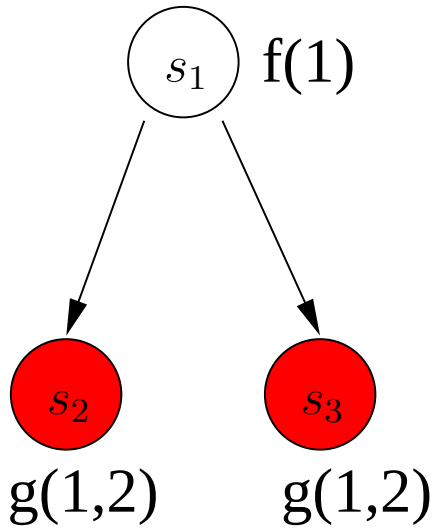
Adding an environment (CTL-FV)

Semantic patch

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

CTL representation

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



Model checking algorithm

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

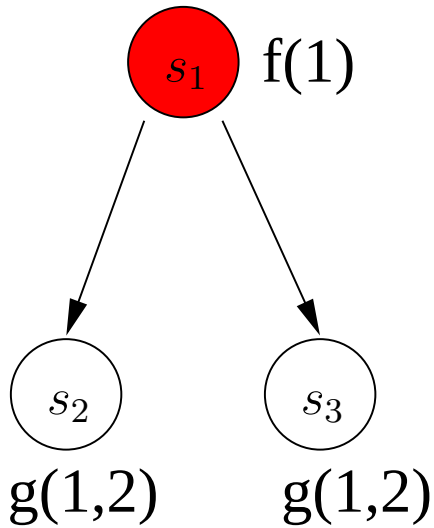
Adding an environment (CTL-FV)

Semantic patch

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

CTL representation

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



Model checking algorithm

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

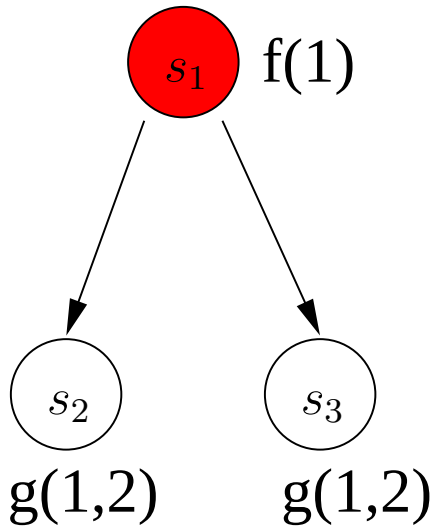
Adding an environment (CTL-FV)

Semantic patch

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

CTL representation

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



Model checking algorithm

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

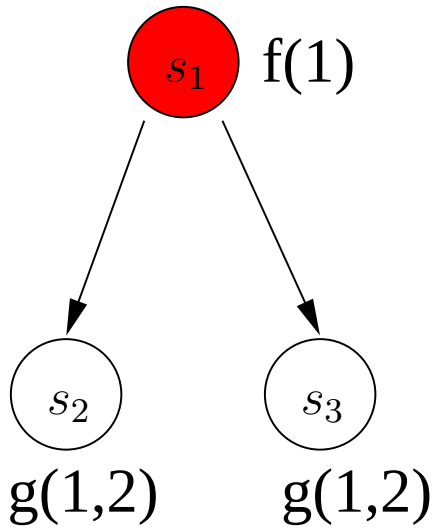
Adding an environment (CTL-FV)

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);
```

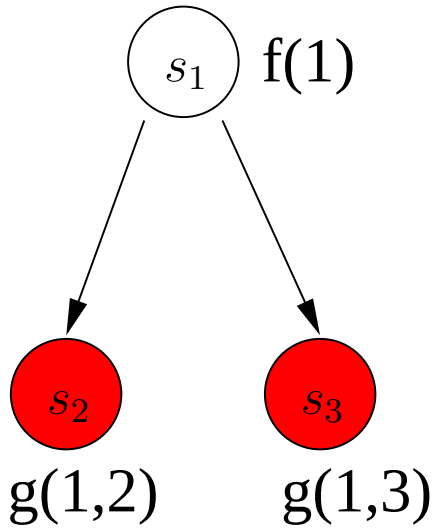
Adding existential quantification (CTL-V)

Semantic patch

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

CTL representation

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



Model checking algorithm

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

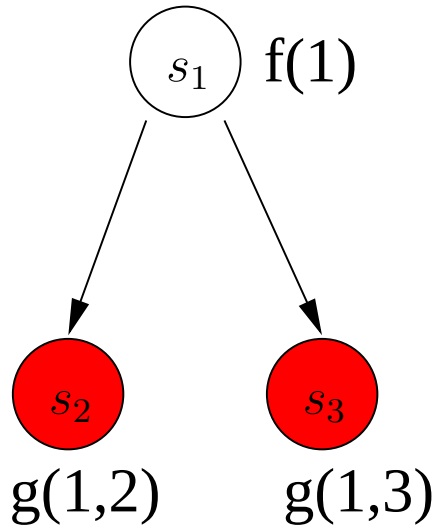
Adding existential quantification (CTL-V)

Semantic patch

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

CTL representation

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



Model checking algorithm

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

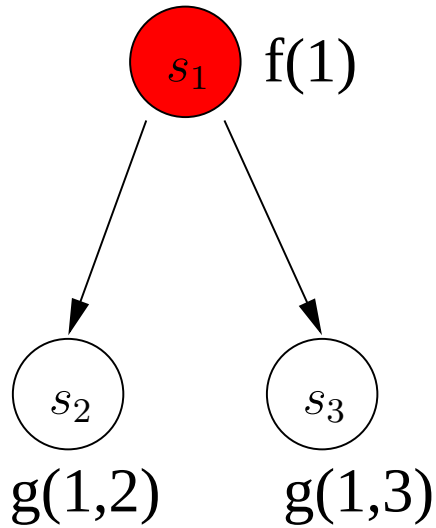
Adding existential quantification (CTL-V)

Semantic patch

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

CTL representation

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



Model checking algorithm

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

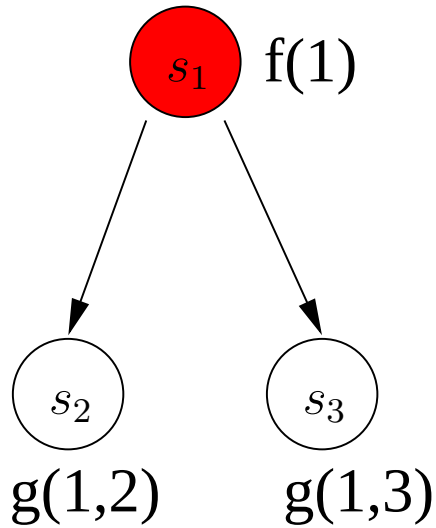
Adding existential quantification (CTL-V)

Semantic patch

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

CTL representation

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



Model checking algorithm

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

Adding witnesses (CTL-VW)

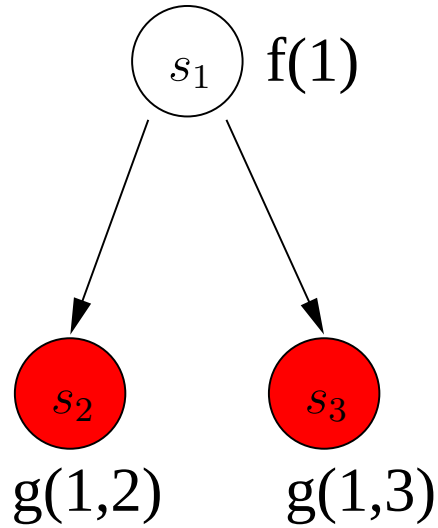
Goal: collect information about what and where to transform

Semantic patch

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

CTL representation

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



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)

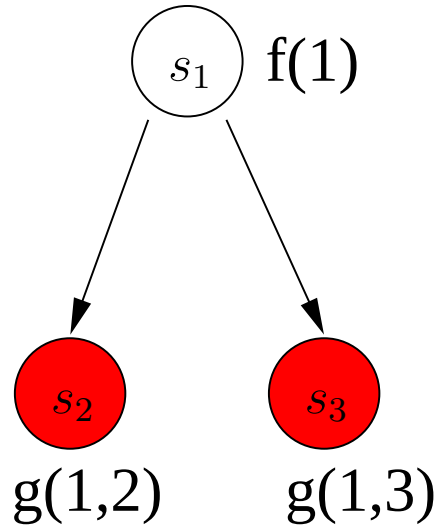
Goal: collect information about what and where to transform

Semantic patch

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

CTL representation

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



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)

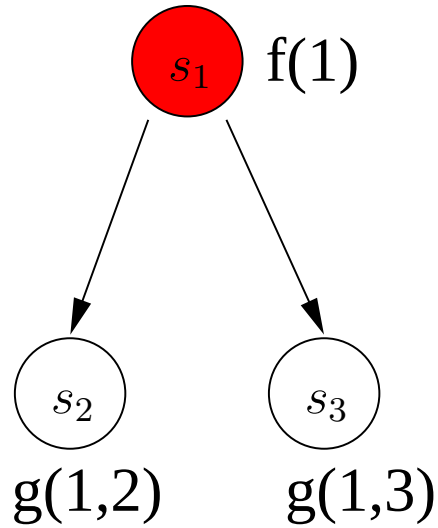
Goal: collect information about what and where to transform

Semantic patch

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

CTL representation

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



Model checking algorithm

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

In the paper

- Semantics of our extension CTL-VW.
- Theorems of soundness and completeness for CTL-VW (proof validation with Coq for CTL-V)
- Syntax of a simple semantic patch language, semantics by translation into CTL-VW
- Optimizations
- A few results of bug finding experiments in the Linux kernel

Conclusion

Simple and efficient extension of CTL that is useful for program matching and transformation

Over 180 patches accepted into Linux using Coccinelle

Future work

- Extension to other kinds of logics for other kinds of matching effects
- Working on other programming languages than C
- Handling dataflow and interprocedural analysis

Coccinelle is publicly available

http://www.emn.fr/x-info/coccinelle/

http://lwn.net/Articles/315686/