Coccinelle

un outil de reconnaissance de motifs et de transformation de code C

Julien Brunel

Office National d'Études et de Recherches Aérospatiales
Département Traitement de l'Information et Modélisation

5 mai 2009

Plan

Outline

Coccinelle

Coccinelle

- Reconnaissance de motifs dans du code C
- Transformation de code
- Basé sur une extension de CTL (model checking sur le graphe de flot de contrôle)

Utilisation

- À l'origine, pour pallier le problème de l'*évolution collatérale* dans le noyau linux (modification d'API)
- Deuxième utilisation : détection et correction de bugs

Outline

SmPL : 'Semantic Patch Language'

- Comme le langage de patch (Unix), mais générique.
- On s'abstrait des détails inutiles
  - Espaces, indentations, commentaires
  - Choix du nom des variables (metavariables)
  - Autres variations dans la manière de coder («isomorphismes»)
- On raisonne sur les chemins du Graphe de Flot de Contrôle

Un patch sémantique peut modifier un grand nombre de fichiers.

Structure et syntaxe d'un patch sémantique

```
@regle_1@
//Déclaration de meta-variables;
expression x;
@@
//Descritpion d'un motif
x = kmalloc(...)
...
- kfree(x)

@regle_2@
//Déclaration de meta-variables;
expression y != regle_1.x;
@@
//Descritpion d'un motif
```

Recherche de fuites de mémoire

```
@leaks exists@
expression x, E, F;
statement S;

@@

* x = kmalloc(...)
...
if (x == NULL) S
... when != kfree(x)
    when != ( x = E | E = x )
* return F;
```

Recherche de fuites de mémoire

```
@leaks exists@
expression x, E;
statement S;
constant C;
@@

* x = kmalloc(...)
...
if (x == NULL) S
... when != kfree(x)
    when != ( x = E | E = x )
* return ( -C | NULL );
```

Recherche de fonctions qui retournent NULL

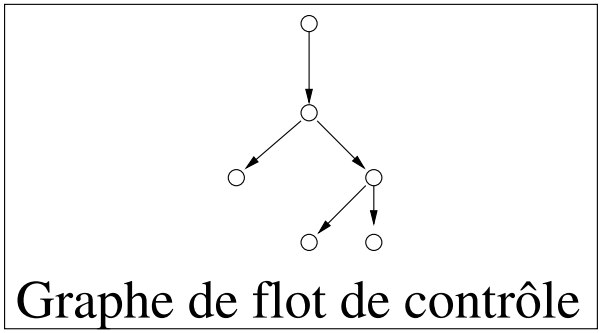
```
@returns_null exists@
identifier f;
expression E, E1;
@@
f(...) {
    ...
(
  return NULL;
|
  E = NULL;
   ... when != E = E1
  return E;
)
}
```

Correction des appels à usb_submit_urb

```
@@
expression lock, flags;
expression urb;
@@
  spin_lock_irqsave(lock, flags);
  <...
- usb_submit_urb(urb)
+ usb_submit_urb(urb, GFP_ATOMIC)
  ...>
  spin_unlock_irqrestore(lock, flags);
@@
expression urb;
@@
- usb_submit_urb(urb)
+ usb_submit_urb(urb, GFP_KERNEL)
```

Comment ça marche ?

Code C



Graphe de flot de contrôle

Patch sémantique

$A(\varphi_1 U \varphi_2) \wedge \ldots$

Formule de CTL–V

Algorithme de model checking
- **Identification des noeuds à modifier**

Modification du code C identifié

Outline

Exigences pour la sémantique

1. gérer les méta-variables
2. permettre aux méta-variables de prendre différentes valeurs dans des chemins différents du GFC
3. enregistrer les différentes valeurs des méta-variables
4. enregistrer les endroits où les transformations doivent être effectuées

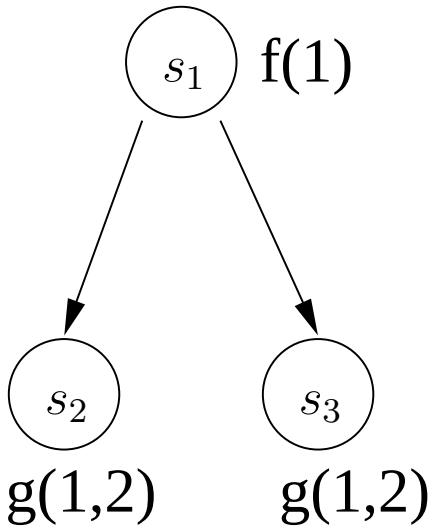
CTL translation and model checking

Semantic patch

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

CTL representation

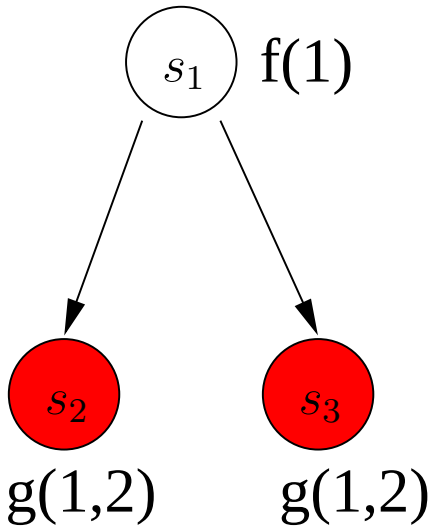
$$f(\ldots) \wedge \mathsf{AX}\, g(\ldots)$$



CTL translation and model checking

Semantic patch

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



CTL representation

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

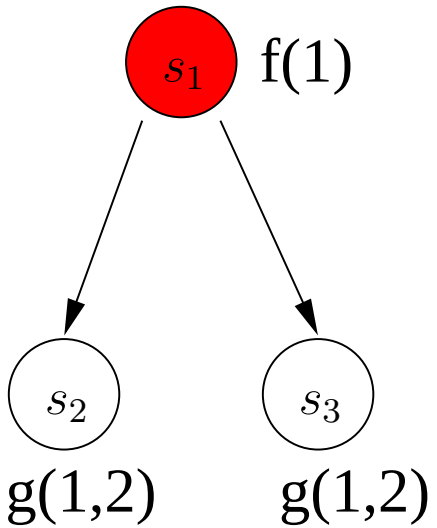
Model checking algorithm

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

CTL translation and model checking

Semantic patch

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



CTL representation

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

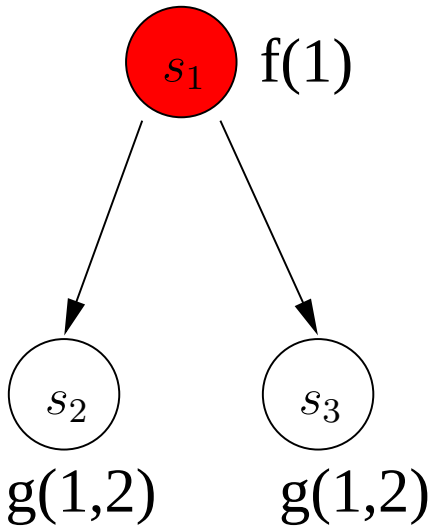
Model checking algorithm

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

CTL translation and model checking

Semantic patch

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



CTL representation

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

Model checking algorithm

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

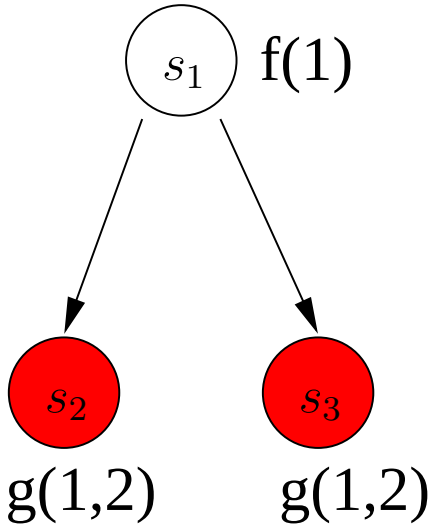
Adding an environment (CTL-FV)

Semantic patch

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

CTL representation

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



Model checking algorithm

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

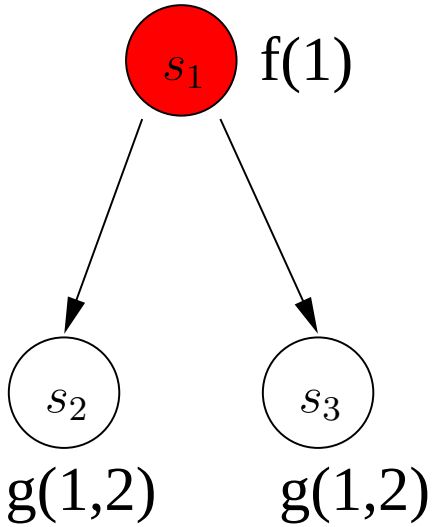
Adding an environment (CTL-FV)

Semantic patch

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

CTL representation

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



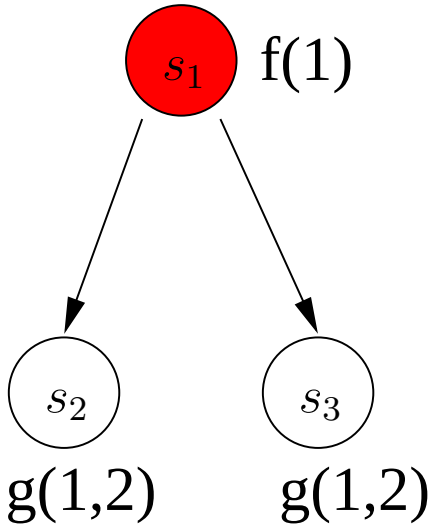
Model checking algorithm

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

Adding an environment (CTL-FV)

Semantic patch

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



CTL representation

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

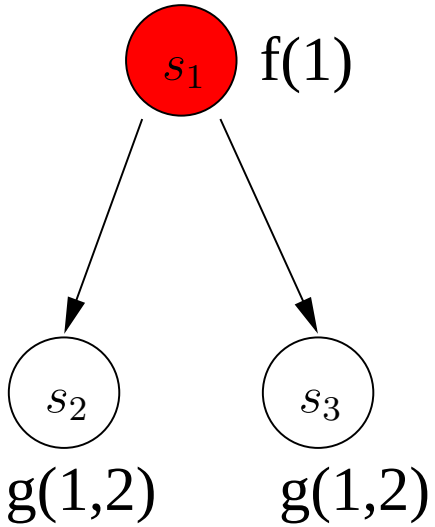
Model checking algorithm

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

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

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

Problem : y has to be the same everywhere.

Example

Semantic patch

```
@leaks exists@
expression x, E;
statement S;
constant C;
@@

* x = kmalloc(...)
...
if (x == NULL) S
... when != kfree(x)
    when != ( x = E | E = x )
* return ( -C | NULL );
```

C code

```
a = kmalloc (...);
if (!a) {...}
if (x) return -1;
if (y) return -2;
kfree (a);
```

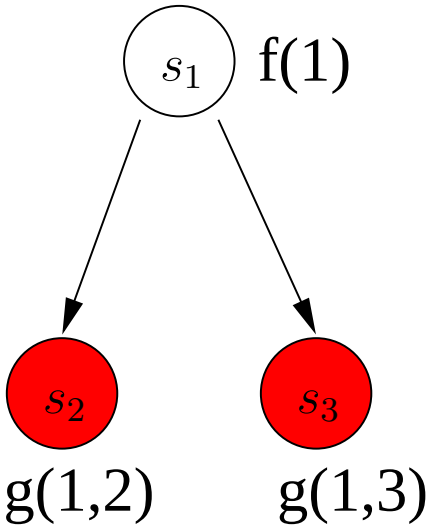
Adding existential quantification (CTL-V)

Semantic patch

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

CTL representation

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



Model checking algorithm

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

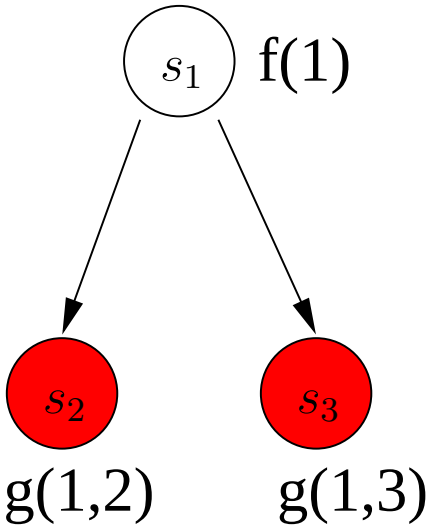
Adding existential quantification (CTL-V)

Semantic patch

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

CTL representation

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



Model checking algorithm

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

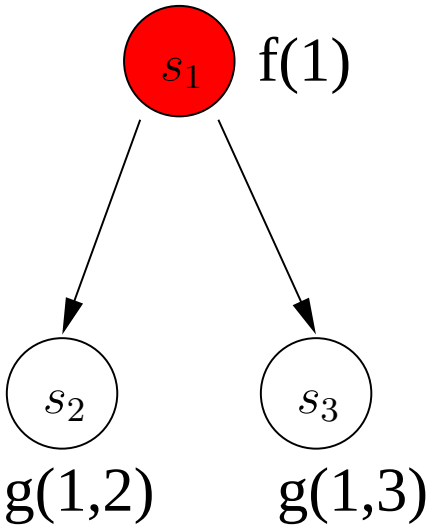
Adding existential quantification (CTL-V)

Semantic patch

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

CTL representation

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



Model checking algorithm

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

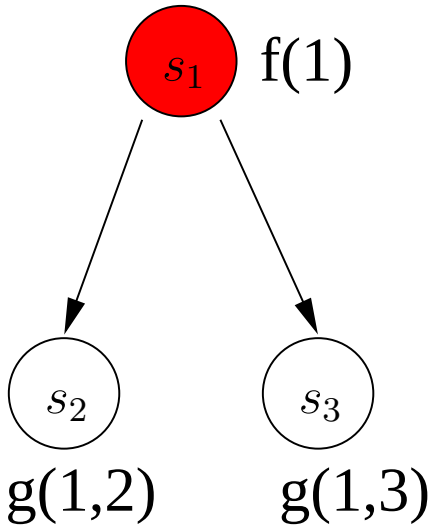
Adding existential quantification (CTL-V)

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)

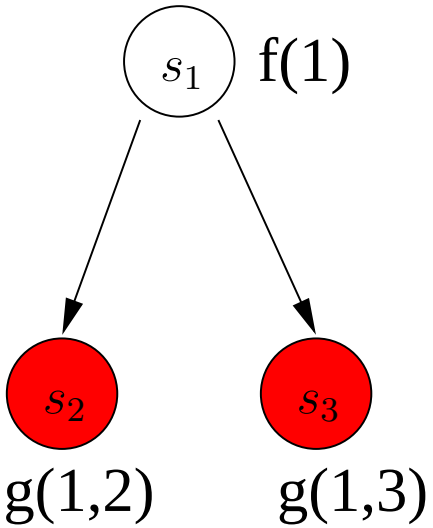
Goal : collect information about what and where to transform

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)

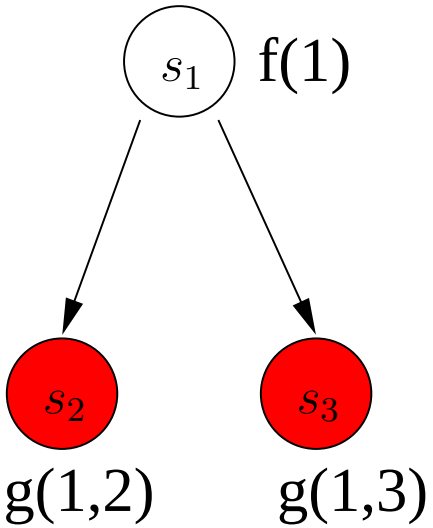
Goal : collect information about what and where to transform

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)

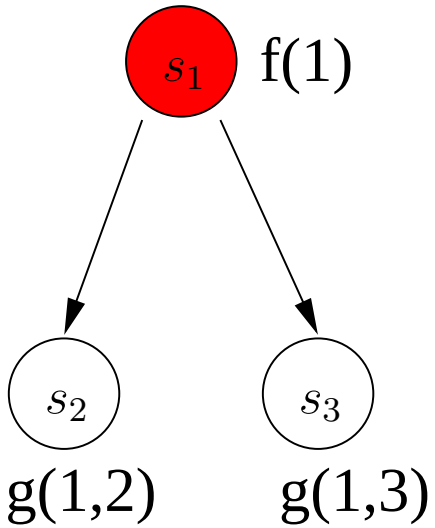
Goal : collect information about what and where to transform

Semantic patch

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

CTL representation

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



Model checking algorithm

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

Outline

Definition (Syntaxe de CTL-VW)

$$\phi \quad ::= \quad p(\overline{x}) \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \exists x.\phi \mid$$

$$\mathsf{AX}\phi \mid \mathsf{EX}\phi \mid \mathsf{A}[\phi\,\mathsf{U}\,\phi] \mid \mathsf{E}[\phi\,\mathsf{U}\,\phi]$$

Modèle $(S, \rightarrow, \text{Label})$

- S : ensemble d'états
- $\rightarrow \subseteq S \times S$ relation de transition
- Label : $S \rightarrow \mathcal{P}(\text{Atom})$ fonction de label

Environnement θ

$\theta : \text{MetaVar} \rightarrow (\text{Val} + \mathcal{P}(\text{Val}))$

$$\theta = [x \mapsto 1, y \not\mapsto \{3, 4\}]$$

Témoin ω, forêt Ω

$\omega \in S \times \text{MetaVar} \times (\text{Val} + \mathcal{P}(\text{Val})) \times \text{ForetTemoins}$
Ω : (multi)ensemble de témoins

$$\Omega = \{\langle s_0, y, 2, \emptyset\rangle, \langle s_3, x, 1, \{\langle s_4, y, 3, \emptyset\rangle\}\rangle\}$$

Definition (Semantique de CTL-VW)

$$
\begin{array}{lcl}
s \models_{\theta,\emptyset} p(\overline{x}) & \Leftrightarrow & p(\theta(\overline{x})) \in \mathsf{Label}(s) \\
s \models_{\theta,\Omega_1 \uplus \Omega_2} \phi_1 \wedge \phi_2 & \Leftrightarrow & s \models_{\theta,\Omega_1} \phi_1 \wedge s \models_{\theta,\Omega_2} \phi_2 \\
s \models_{\theta,\Omega} \phi_1 \vee \phi_2 & \Leftrightarrow & s \models_{\theta,\Omega} \phi_1 \vee s \models_{\theta,\Omega} \phi_2 \\
s \models_{\theta,\emptyset} \neg\phi & \Leftrightarrow & \forall \Omega.\ \ s \not\models_{\theta,\Omega} \phi \\
\\
s \models_{\theta,\{\langle s,x,v,\Omega\rangle\}} \exists x.\phi & \Leftrightarrow & s \models_{\theta[x\mapsto v],\Omega} \phi \\
\\
s \models_{\theta,\Omega} \mathsf{AX}\phi & \Leftrightarrow & \exists (\Omega_{s'})_{s' \in \mathsf{next}(s)} \cdot \\
\multicolumn{3}{r}{\biguplus_{s' \in \mathsf{next}(s)} \Omega_{s'} = \Omega \ \wedge\ \forall s' \in \mathsf{next}(s).\ s' \models_{\theta,\Omega_{s'}} \phi} \\
s \models_{\theta,\Omega} \mathsf{EX}\phi & \Leftrightarrow & \exists s' \in \mathsf{next}(s).\ s' \models_{\theta,\Omega} \phi
\end{array}
$$

Definition (*AU* and *EU* semantics)

$$
\begin{array}{l}
s \models_{\theta,\Omega} \mathsf{A}[\phi_1 \,\mathsf{U}\, \phi_2] \quad \Leftrightarrow \quad \exists \Sigma \in \Pi(s).\ \exists (\Omega_{s'})_{s' \in \overline{\Sigma}}. \\
\forall \pi \in \Sigma.\ \pi[|\pi|] \models_{\theta,\Omega_{\pi[|\pi|]}} \phi_2 \wedge \forall 0 \leqslant j < |\pi|.\ \pi[j] \models_{\theta,\Omega_{\pi[j]}} \phi_1 \\
\qquad\qquad \wedge \biguplus_{s' \in \overline{\Sigma}} \Omega_{s'} = \Omega
\end{array}
$$

$$
\begin{array}{l}
s \models_{\theta,\Omega} \mathsf{E}[\phi_1 \,\mathsf{U}\, \phi_2] \quad \Leftrightarrow \quad \exists \Sigma \in \Pi(s).\ \exists \pi \in \Sigma. \exists (\Omega_i)_{i \in 0..|\pi|}. \\
\qquad \pi[|\pi|] \models_{\theta,\Omega_{|\pi|}} \phi_2 \wedge \forall 0 \leqslant j < |\pi|.\ \pi[j] \models_{\theta,\Omega_j} \phi_1 \\
\qquad\qquad \wedge \biguplus_{j \leqslant |\pi|} \Omega_j = \Omega
\end{array}
$$

SmPL vers CTL-VW

f (*E1*)

. . .

g (*E2*)

f(*E1*)
∧
AX(A[. . .
U
g (*E2*)])

SmPL vers CTL-VW

f (*E1*)

. . .

g (*E2*)

$$\exists E1.(f(E1) \wedge \mathrm{AX}(\mathrm{A}[\ldots \mathrm{U}\ \exists E2.\mathtt{g}\ (E2)\ ]))$$

SmPL vers CTL-VW

f (*E1*)

. . .

g (*E2*)

$$\exists E1.(f(E1) \wedge \mathsf{AX}(\mathsf{A}[\neg(f(E1) \vee \exists E2.g(E2)) \ \mathsf{U} \ \exists E2.\mathtt{g}\,(E2)\,]))$$

Outline

Algorithme de model checking

Fonction SAT :

$$
\begin{aligned}
\mathrm{SAT}(p(\overline{x})) &= \{(s,\theta,\emptyset) \mid p(\theta(\overline{x})) \in \mathsf{Label}(s) \wedge \mathsf{dom}(\theta) = \mathsf{fv}(p(\overline{x}))\} \\
\mathrm{SAT}(\phi_1 \wedge \phi_2) &= \mathsf{conj}(\mathrm{SAT}(\phi_1), \mathrm{SAT}(\phi_2)) \\
\mathrm{SAT}(\phi_1 \vee \phi_2) &= \mathrm{SAT}(\phi_1) \cup \mathrm{SAT}(\phi_2) \\
\mathrm{SAT}(\neg\phi) &= \mathsf{neg}(\mathrm{SAT}(\phi)) \\
\mathrm{SAT}(\exists x.\phi) &= \{\mathsf{existsone}(x,t) \mid t \in \mathrm{SAT}(\phi)\} \\
\mathrm{SAT}(\mathsf{AX}\,\phi) &= \mathsf{pre}_\forall(\mathrm{SAT}(\phi)) \\
\mathrm{SAT}(\mathsf{EX}\,\phi) &= \mathsf{pre}_\exists(\mathrm{SAT}(\phi)) \\
\mathrm{SAT}(\mathsf{A}[\phi_1\,\mathsf{U}\,\phi_2]) &= \mathrm{SAT}_{\mathrm{AU}}(\mathrm{SAT}(\phi_1), \mathrm{SAT}(\phi_2)) \\
\mathrm{SAT}(\mathsf{E}[\phi_1\,\mathsf{U}\,\phi_2]) &= \mathrm{SAT}_{\mathrm{EU}}(\mathrm{SAT}(\phi_1), \mathrm{SAT}(\phi_2))
\end{aligned}
$$

Algorithme de model checking (2)

Conjonction

$$\text{conj}(T_1, T_2) = \{(s, \theta_1 \sqcap \theta_2, \Omega_1 \uplus \Omega_2) \mid (s, \theta_1, \Omega_1) \in T_1) \wedge (s, \theta_2, \Omega_2) \in T_2\}$$

Quantificateur existentiel

$$\begin{aligned}
&\text{existsone}(x, (s, \theta, \Omega)) = \\
&\quad (s, \theta - [x \mapsto v], \{\langle s, x, v, \Omega\rangle\}), \text{ if } \theta^+(x) = v \\
&\quad (s, \theta - [x \not\mapsto V], \{\langle s, x, V, \Omega\rangle\}), \text{ if } \theta^-(x) = V \\
&\quad (s, \theta, \{\langle s, x, \emptyset, \Omega\rangle\}), \text{ otherwise}
\end{aligned}$$

Correction et complétude

On voudrait montrer

$$(s, \theta, \Omega) \in \mathrm{SAT}(\phi) \quad \Longleftrightarrow \quad s \models_{\theta,\Omega} \phi$$

Difficulté due aux «bindings» négatifs

- Un environnement $[x \not\mapsto 2]$ «correspond» à plusieurs environnements positifs $[x \mapsto 1], [x \mapsto 3], \ldots$
- Un résultat (s, θ, Ω) de l'algorithme de model checking peut «correspondre» à plusieurs triplets pour la sémantique

Correction et complétude (2)

Relation d'ordre

On définit

- Une relation d'ordre sur les «bindings», qui induit une relation d'ordre sur les environnements et sur les témoins.

$$[x \mapsto 2] \quad \sqsubseteq \quad [x \nrightarrow \{3,4\}] \quad \sqsubseteq \quad [x \nrightarrow \{3\}]$$

- Pour $T \subseteq S \times \mathrm{Env} \times \mathrm{WitForest}$, $\eta_\phi(T) =$

$$\{(s,\theta,\Omega) \mid \exists (s',\theta',\Omega') \in T.\ s = s' \ \wedge\ \theta \sqsubseteq \theta' \ \wedge\ \Omega \sqsubseteq \Omega'\}$$

Theorem (Correction et complétude)

$$s \models_{\theta,\Omega} \phi \iff (s,\theta,\Omega) \in \eta_\phi(\mathrm{SAT}(\phi))$$

Outline

Résultats expérimentaux

Démarche de la détection et correction de bugs

- Spécification d'un motif aussi «correct» que possible («correct» = conforme à ce que l'on recherche)
- Incorrect $\Rightarrow$ existence de *faux positifs* (fausses alarmes)
- Le taux de faux positifs est diminué en raffinant le patch sémantique

Noyau Linux

- environ 25 000 fichiers
- 250 M de code
- + de 8 millions de lignes de code

$\Rightarrow$ + de 180 patches acceptés dans le noyau

Benchmarks

Tests sur un 8 processeurs, 16GB de RAM, 3GHz

patch sémantique	temps	patch sémantique	temps
leaks	3 mn 30	categorize functions	11 mn
returns_null	4 mn 20	detect bad call of a function	2 s
usb_submit	1mn 40	correct bad call of a function	2 s
		bad pointer check	9 mn

Perspectives

Perspectives

- Utiliser coccinelle sur du code critique
- Patches et reconfiguration de code à grande échelle
- Tester le respect de standards de codage
- Étendre coccinelle à d'autres langage que C
- Gérer les flots de données

Plus d'informations

Documentation, téléchargement, exemples, etc. sur le site du projet :

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

Paraticipation à Coccinelle, points de contact

- Université de Copenhague (Julia Lawall `julia@diku.dk`)
- ONERA (Julien Brunel `julien.brunel@onera.fr`)
- Université de Aalborg
- École des Mines de Nantes

Références

- Détection et correction de bugs

 ***WYSIWIB : A declarative approach to finding protocols and bugs in Linux code***, **DSN'09**, Lisbonne, Portugal, 2009

- Sémantique de Coccinelle, model checking de CTL-VW

 ***A foundation for flow-based program matching using temporal logic and model checking***, **POPL'09**, Savannah, Géorgie, 2009