



# Exécution symbolique pour la génération de tests ciblant des labels

---

Saïd ZUHAIR · Léo ANDRÈS · Nicolas BERTHIER · Steven DE OLIVEIRA

JFLA 2026

# Contexte



# Objectif: Génération automatique de tests

- Le test de code est nécessaire pour s'assurer de la qualité d'un programme
- Les tests écrits manuellement :
  - prennent beaucoup de temps à écrire ;
  - vérifient généralement les cas courants.

# Critères de couverture

- Moyen de mesurer la couverture de code
- Quelques critères souvent utilisés:
  - FC (Function Coverage)
  - SC ou STMT (Statement Coverage)
  - DC (Decision Coverage)
  - CC (Condition Coverage)
  - MCC (Multiple-Condition Coverage)
  - WM (Weak Mutation)
- Peut être imposé comme exigence de conformité

## Exemple CC(Condition Coverage)

Chaque condition atomique d'une expression booléenne doit être évaluée au moins une fois comme vraie et au moins une fois comme fausse.

```
int numPos(int a) {  
    int n = 0;  
    if(a > 0 && a < 20) n++;  
    return n;  
}
```

- $a > 0$ ,  $\neg(a > 0)$
- $a < 20$ ,  $\neg(a < 20)$

E.g. : -1 et 21

# Les labels

- Une méthode générique d'expression de tous types de critères de couverture

```
int numPos(int a) {  
    int n = 0;  
    if(a > 0 && a < 20) n++;  
    return n;  
}
```

LAnnotate[1]



```
int numPos(int a) {  
    int n = 0;  
    pc_label(a > 0,1,"CC");  
    pc_label(! (a > 0),2,"CC");  
    pc_label(a < 20,3,"CC");  
    pc_label(! (a < 20),4,"CC");  
    if (a > 0 && a < 20) n ++;  
    return n;  
}
```

- Les outils de génération de tests peuvent viser les labels pour cibler les critères de couvertures

[1] Bardin, Sébastien and Chebaro, Omar and Delahaye, Mickaël and Kosmatov, Nikolai, «An all-in-one toolkit for automated white-box testing», in International Conference on Tests and Proofs

# Génération automatique de test pour C avec Seacoral

- Orchestrateur d'outils de génération de test pour C
  - Construit autour de la notion des labels
  - Permet la collaboration et le partage de tests entre les outils
  - Supporte actuellement divers outils :
    - libfuzzer
    - CBMC
    - KLEE
- Contribution : Ajout du support d'Owi

**<https://github.com/OCamlPro/SeaCoral/>**

# L'exécution symbolique avec Owi

- Owi
  - Boîte à outils pour **WebAssembly**
  - Traite les programmes C, C++, Rust, Go et Zig
  - Contient un moteur d'exécution symbolique
- Exécution symbolique
  - Typiquement utilisé pour du bug finding
  - Exploration de tous les branchements d'un programme
  - Accumulation de prédicats pour construire une « condition de chemin »

**<https://github.com/OCamlPro/owi/>**

# L'exécution symbolique avec Owi

## Programme en entrée

```
int process_data(int x, int y){
    int result = x + 10;
    if (result > 15) {
        return 100 / (y - 5);
    }
    return result * 2;
}
```

## Sortie d'Owi

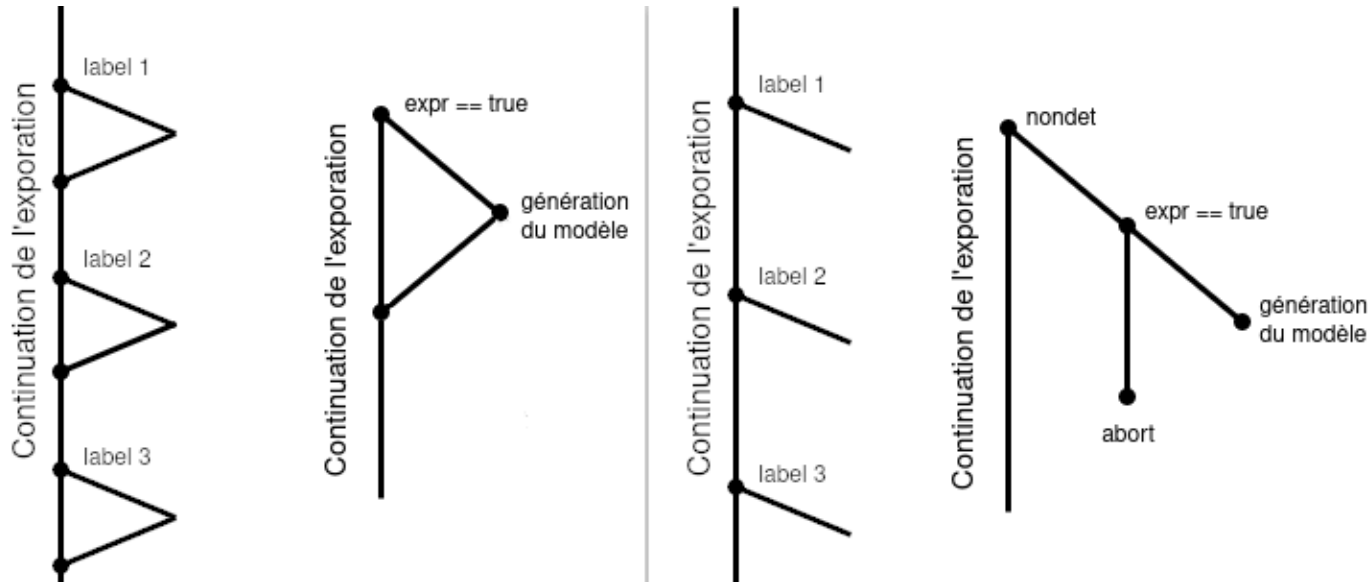
```
owi: [ERROR] Trap: integer
divide by zero
model {
    symbol symbol_0 i32 6
    symbol symbol_1 i32 5
}
```

# Gestion des labels pour Owi

---

# Gestion des labels pour KLEE

- Étudié dans le papier pour Klee4labels [1] (inspiré d'un autre papier sur PathCrawler [2])



[1] N. Berthier, S. De Oliveira, N. Kosmatov, D. Longuet, et R. Soulat, « An efficient black-box support of advanced coverage criteria for Klee », in *Proceedings of the 38th ACM/SIGAPP symposium on applied computing*, 2023, p. 1706-1715.

[2] S. Bardin, N. Kosmatov, et F. Cheyner, « Efficient leveraging of symbolic execution to advanced coverage criteria », in *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*, 2014, p. 173-182.

# Gestion des labels pour Owi

```
#define pc_label(expr, id, name)
do {
    if (!owi_label_is_covered(id) && (expr)) {
        owi_assert (0);
    }
} while (0)
```

```
\
\  
\
\  
\
```

Évaluation paresseuse de l'expression de label

# Gestion des labels pour Owi

## Ajout d'un non déterministe

```
#define pc_label(expr, id, name)
do {
    int nondet = owi_invisible_bool();
    if (nondet) {
        if (!owi_label_is_covered(id) && (expr)) {
            owi_assert (0);
        }
        abort();
    }
} while (0)
```

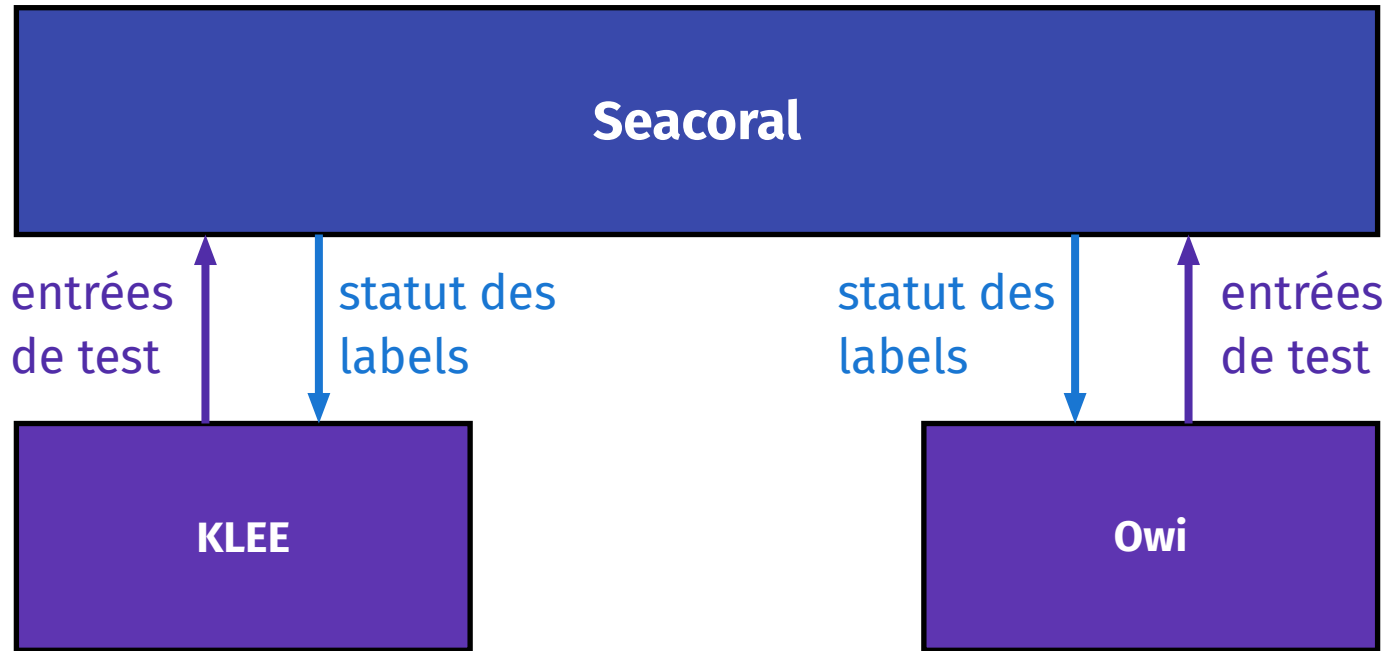
\  
\  
\  
\  
\  
\  
\  
\  
\

ne reboucle pas donc pas d'explosion de chemin

# L'intégration d'Owi dans Seacoral

---

# Optimisation du parcours d'Owi



Les outils envoient leurs données à Seacoral. Seacoral met à jour le statut des labels dans la base de donnée partagée.

# Génération du harnais de test

```
int numPos(int a) {
    int n = 0;
    if (a > 0 && a < 20)
        n++;
    return n;
}

// declaration des types
struct __numPos_inputs { int a; };
// declaration des fonctions d'initialisation
void __sc_init_typ__numPos_inputs (
    struct __numPos_inputs (* ptr), const unsigned int depth
);
// implémentation des fonctions d'initialisation
void __sc_init_typ__numPos_inputs (
    struct __numPos_inputs (* ptr), const unsigned int depth
) {
    (*ptr).a = owi_int();
}
int main () {
    struct __numPos_inputs test_struct;
    __sc_init_typ__numPos_inputs(&test_struct, 0);
    (void) numPos(test_struct.a);
    return 0;
}
```

# Traitement des sorties des outils

- Lecture des modèles générés par les outils
- Validation par un rejeu concret
  - Exécution instrumentée pour détecter les erreurs
  - Mise à jour du statut des labels

# Exemple d'un test généré

## Programme en entrée

```
int numPos(int a) {  
    int n = 0;  
    if(a > 0 && a < 20) n++;  
    return n;  
}
```

## Test généré en C

```
int main (){  
    int a = -2147483647;  
    (void) numPos (a);  
    return 0;  
}
```

# Évaluations expérimentales

---

# Évaluations : Résultats

	Critère	timeout=1s			timeout=5s			timeout=60s		
		Owi1	Owi8	KLEE	Owi1	Owi8	KLEE	Owi1	Owi8	KLEE
Total	DC	195	211	198	198	210	223	198	212	223
	CC	264	287	261	270	287	287	271	287	296
	MCC	244	260	233	250	260	249	250	260	273
	WM	654	713	533	703	714	681	702	715	722
	All	1357	1470	1224	1422	1472	1440	1420	1473	1514

- Moyenne sur 10 lancés du nombre de labels couverts
- Owi1 et Owi8 correspondent à Owi lancé avec 1 et 8 domaines respectivement
- Base de test de Klee4labels, contenant 10 programmes

# Évaluations : Interprétation

- Owi couvre beaucoup de labels simples rapidement
- KLEE atteint une meilleure couverture avec plus du temps
- Le parallélisme dans Owi peut lui permettre de parcourir les labels simples plus rapidement
- Owi1 n'est pas aussi efficace que Owi8 avec 8 fois plus du temps

## **Conclusion**

- Les labels sont un bon moyen de représenter les critères de couverture
- S'inspirer des travaux sur PathCrawler et KLEE nous a permis de concevoir une intégration pour Owi en ~600 loc qui est efficace.
- Bon potentiel de collaboration entre Owi et les autres outils

## **Travaux Futurs**

- Ajouter le support des critères MC/DC
- Ma thèse :)