Comment améliorer l'efficacité de l'analyse statique des programmes ?

Mamy Razafintsialonina

1. Contexte

◇ **Frama-C** est une plateforme open-source pour l'analyse de programmes C.

◇ Composée d'un ensemble d'analyseurs, dont **EVA**, basé sur l'**interprétation abstraite.**

◇ Dans le développement de **logiciels de grande taille**, le code source devient volumineux et complexe.

◇ Des **modifications fréquentes** nécessitent des analyses régulières.

◇ Les changements sont souvent **petits** et **très localisés**.

◇ La **similarité** entre les versions du code permet une **analyse plus rapide**, réduisant le temps de vérification.

2. Réutilisation des résumés de fonction

◇ Les analyses d'EVA sont **sensibles aux contextes** et **modulaires** grâce à une technique de **résumé de fonction** existante.

◇ C'est un **cache** qui stocke les états d'entrée $\mathbf{I}^\sharp$ et sortie $\mathbf{O}^\sharp$ des fonctions.

◇ Cela permet d'**éviter de réanalyser** le corps de la fonction.

◇ Le cache des fonctions non modifiées est **rechargé** pour l'**analyse incrémentale.**

◇ Cela réduit le temps d'analyse pour les changements qui :

- ◇ Ne modifient pas plusieurs fonctions.
- ◇ N'affectent pas une fonction profondément imbriquée.

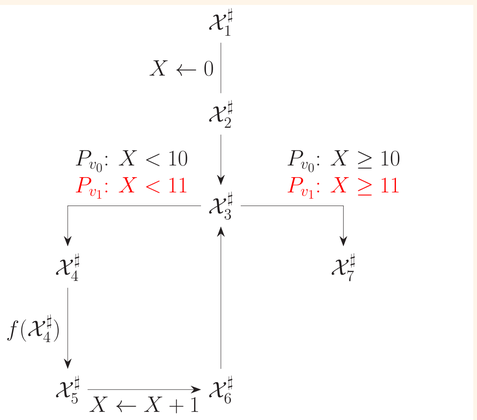

Figure 1: CFG d'un programme initial P_{v_0}, et un programme modifié P_{v_1}. Une itération complète jusqu'à convergence de la boucle de P_{v_1} aboutira à une seule réanalyse de f.

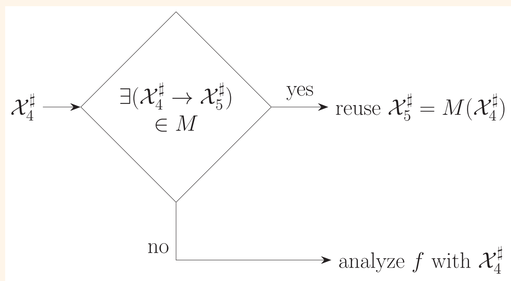

Figure 2: Recherche de la clé $\mathcal{X}_4^\sharp$ dans $M : I^\sharp \to O^\sharp$.

3. Réutilisation des invariants de boucle

◇ Si une fonction est modifiée, son cache est invalidé.

◇ Néanmoins, l'**analyse des boucles** de la fonction peut être **accélérée**.

◇ Des **itérations** peuvent être **évitées** en commençant l'itération à partir de l'ancien invariant $\mathbf{X^{old}_3}$.

◇ Cette approche reste **sûre** même pour un invariant X arbitraire, mais peut générer une **perte de précision**.

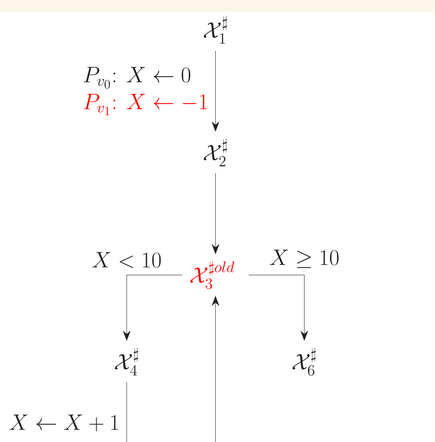

Figure 3: CFG de la boucle du programme initial P_{v_0} et modifié P_{v_1}.

$$\mathcal{X}_3^{\sharp 1} = (\mathcal{X}_3^{\sharp old} \sqcup^\sharp \mathcal{X}_3^{\sharp 0}) \ \triangledown^\sharp \ (\mathcal{X}_3^{\sharp old} \sqcup^\sharp \mathcal{X}_2^{\sharp 0} \sqcup^\sharp \mathcal{X}_5^{\sharp 0})$$
$$\mathcal{X}_3^{\sharp n} = \mathcal{X}_3^{\sharp n-1} \ \triangledown^\sharp \ (\mathcal{X}_2^{\sharp n-1} \sqcup^\sharp \mathcal{X}_5^{\sharp n-1})$$

Figure 4: Soit l'invariant $\mathcal{X}_j^{\sharp i}$, avec i le nombre d'itérations, et j un point du programme. $\mathcal{X}_3^{\sharp old}$ est l'invariant de la boucle du programme P_{v_0}.

	P_{v_0}			P_{v_1}	
l	$\mathcal{X}_l^{\sharp 0}$	$\mathcal{X}_l^{\sharp 1}$	$\mathcal{X}_l^{\sharp 2}$	$\mathcal{X}_l^{\sharp 0}$	$\mathcal{X}_l^{\sharp 1}$
1	$\top$	$\top$	$\top$	$\top$	$\top$
2	$\bot$	0	0	$\bot$	-1
3 $\triangledown$	$\bot$	0	$[0, +\infty[$	$\bot$	$[0, +\infty[\sqcup^\sharp -1$
4	$\bot$	0	$[0, 9]$	$\bot$	$[-1, 9]$
5	$\bot$	1	$[1, 10]$	$\bot$	$[0, 10]$
6	$\bot$	$\bot$	$[0, +\infty[$	$\bot$	$[-1, +\infty[$

Table 1: Itérations de la boucle des programmes P_{v_0} et P_{v_1}. On atteint le point-fixe de P_{v_1} avec moins d'itérations.

4. Résultats

◇ Les analyses sont effectuées successivement sur une intervalle de commits de **PolarSSL** et **Monocypher**.

◇ Pour un programme **P** avec des versions $\mathbf{v_i}$, nous lançons une analyse EVA initiale de $\mathbf{v_0}$, et ensuite une analyse EVA incrémentale de toutes les $\mathbf{v_i}$ en réutilisant les résultats de $\mathbf{v_{i-1}}$.

◇ Les résultats incluent le temps de parsing, chargement du cache, calcul de la différence des ASTs, et l'analyse avec EVA.

◇ Par rapport à une analyse normale de **PolarSSL** (resp. **Monocypher**) :

- ◇ L'analyse avec réutilisation des résumés de fonction est **4x** (resp. **1,7x**) **plus rapide**.
- ◇ L'analyse en **combinant** les deux approches est **6x** (resp. **1,7x**) **plus rapide**.

◇ Les pertes de précision sont négligeables, une alarme de plus pour la version $\mathbf{v_3}$ (resp. $\mathbf{v_4}$), même nombre d'alarmes pour le reste.

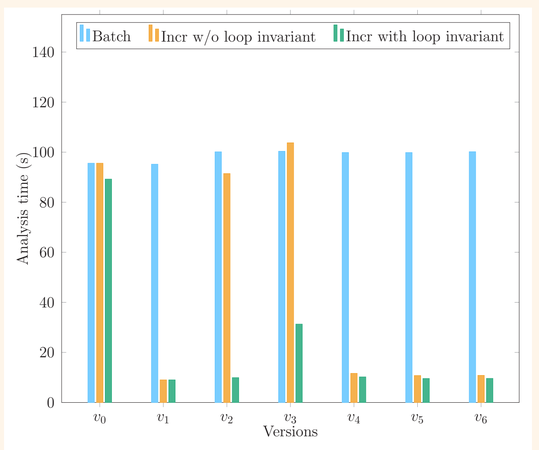

Figure 5: Temps d'analyse des versions successives de **PolarSSL**. Nous comparons le temps d'analyse incrémentale de V_i avec V_{i-1} par rapport au temps d'analyse normale de V_i.

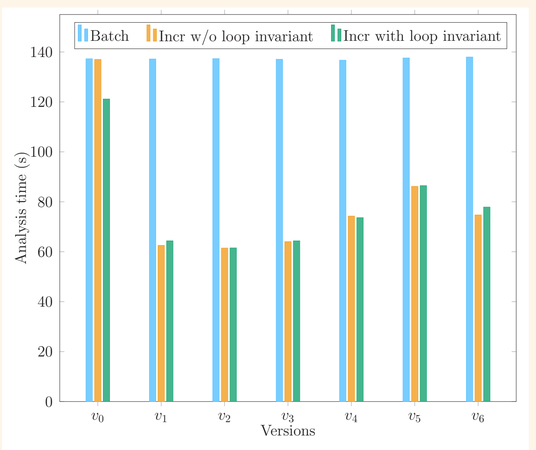

Figure 6: Temps d'analyse des versions successives de **Monocypher**. Nous comparons le temps d'analyse incrémentale de V_i avec V_{i-1} par rapport au temps d'analyse normale de V_i.

v_i	Diff	LoC	Alarms (Batch)	Alarms (Incr)
v_0	10+/10-	28784	92	92
v_1	2+/1-	28784	92	92
v_2	6+/1-	28784	92	92
v_3	10+/0-	28799	92	93
v_4	6+/1-	28801	92	92
v_5	7+/0-	20533	92	92
v_6	10+/10-	20437	92	92

Table 2: Taille du code par version, patch appliqué et nombre d'alarmes émises pour l'analyse de **PolarSSL**.

v_i	Diff	LoC	Alarms (Batch)	Alarms (Incr)
v_0	6+/6-	28227	178	178
v_1	1937+/1-	29491	178	178
v_2	1+/1-	29491	178	178
v_3	57+/73-	29431	178	178
v_4	7+/7-	29431	178	179
v_5	30+/33-	29429	178	178
v_6	5+/0-	29431	178	178

Table 3: Taille du code par version, patch appliqué et nombre d'alarmes émises pour l'analyse de **Monocypher**.

5. Perspectives

- Exploration des solutions pour améliorer la précision.
- Support des analyses où le **paramétrage** est se fait **incrémentalement**.
- Utilisation du **cache** au niveau des **boucles.**
- Rechargement **partiel** ou **différé** des caches précédents.

ny-andrianinamamy.razafintsialonina@cea.fr