

TD Feuille 4 — Analyse statique des déréférencements de pointeurs nuls

Le but de cet exercice est de définir une analyse statique visant à vérifier que les pointeurs NULL ne sont jamais déréférencés. On considère initialement un fragment du C ne contenant que des variables de type `int *`, et pour simplifier des hypothèses similaires au cas de Java : il y a un `malloc`, mais pas de `free`, pas d'arithmétique de pointeurs et tous les pointeurs sont implicitement initialisés à NULL. De plus, on suppose qu'il n'y a pas de pointeurs de pointeurs (ce qui n'est pas vrai en Java). On considère par contre que `malloc` peut retourner un pointeur NULL (ou pas). On voudrait définir une analyse statique qui détecte :

- des accès mémoire qui, s'ils sont exécutés, provoqueront systématiquement un déréférencement d'un pointeur NULL (signalés en rouge)
- des accès mémoire qui ne provoqueront jamais un déréférencement d'un pointeur NULL (signalés en vert)

Comme il n'y a ni `free` et ni pointeur non initialisé, on admet que les pointeurs non nuls sont valides. On utilisera aussi les couleurs suivantes :

- la couleur orange pour signaler qu'on ne sait pas décider
- la couleur grise pour signaler du code mort.

En particulier le code mort peut apparaître si des vérifications ont déjà eu lieu. Par exemple :

```
o = NULL ;
*o = 3 ;    // en rouge déréférencement certain de NULL
*o = 3 ;    // en gris: inaccessible étant donné bug précédent
```

On coloriera tout le code inaccessible en gris (pas seulement les assertions).

Exercice 1 (Analyse manuelle). On considère chacun des deux fragments de code ci-dessous (après une initialisation implicite à NULL). Insérer les assertions RTE pour prévenir l'absence de déréférencement des pointeurs nuls. Puis colorier ces assertions, en vous basant sur votre compréhension intuitive du C.

1	p = malloc(sizeof(int)) ;		p = malloc(sizeof(int)) ;	1
2	*p = 4;		if (p != o) {	2
3	*o = 5 ;		*p = 4;	3
4	*o = *p ;		o = p;	4
			}	5
			if (o != NULL) {	6
			*o = 5 ;	7
			}	8
			*o = *p ;	9

Exercice 2 (Mise en place du treillis borné). On veut utiliser ici un domaine non-relationnel \mathbb{D} où les états abstraits sont des maps associant une valeur abstraite d'un domaine $\mathbb{V} \stackrel{\text{def}}{=} \{T, F\}^2$ à chaque variable x : si $A \in \mathbb{D}$ et x une variable du programme, on note $A(x)$ la valeur abstraite associée à x . De plus, un élément $v \in \mathbb{V}$ associé à une variable x est alors un couple de booléens $(v.Z, v.N)$ tel que :

- si $v.Z = F$ alors $x \neq \text{NULL}$;
- et si $v.N = F$ alors $x = \text{NULL}$.

► **Question 1.** Soit X l'ensemble des variables du programme et un état abstrait $A \in \mathbb{D}$. Définir $\gamma(A)$, la concrétisation de A , c'est-à-dire, dans le formalisme du cours, la proposition logique qui caractérise l'ensemble d'états représenté par A .

► **Question 2.** On considère l'assertion "assert ($p \neq \text{NULL}$) ;" dans le cas où on a une unique variable p . Donner la couleur dans laquelle il faut colorier l'assertion en fonction de la valeur abstraite v de p .

► **Question 3.** On considère la façon dont il faut définir le treillis \mathbb{V} pour que la coloriage de l'interpréteur générique du cours fonctionne comme attendu.

1. Quel est son $\top_{\mathbb{V}}$? Quel est son $\perp_{\mathbb{V}}$? Exprimer son $\sqcup_{\mathbb{V}}$ et son $\sqcap_{\mathbb{V}}$ à l'aide d'opérateurs booléens (logiques) usuels. De même exprimer $\sqsubseteq_{\mathbb{V}}$ à partir d'opérateurs logiques. Dessiner le treillis borné de \mathbb{V} . Quelle est sa hauteur ?
2. Retrouver comment sont définis le \top , \sqcup , \sqcap et \sqsubseteq du domaine non-relationnel \mathbb{D} sur l'ensemble de variables X à partir de ceux de son domaine \mathbb{V} de valeurs abstraites.
3. Soit $A \in \mathbb{D}$, avec $A(x) = \perp_{\mathbb{V}}$. Que peut-on dire sur $\gamma(A)$? Comment définir \perp ?

Exercice 3 (Interprétation abstraite des affectations). Dans un code comme celui de l'exo 1, l'interprétation abstraite des affectations sur le contenu des pointeurs " $*p = \dots$ " correspond juste à un assert : l'effet de bord est ignoré (le contenu des pointeurs est ignoré par le domaine abstrait). Définir l'interprétation de " $A[p_1 := T]$ " uniquement pour les 3 cas suivants : $T \in \{\text{NULL}, \text{malloc}(\text{sizeof}(\text{int})), p_2\}$ où p_1 et p_2 sont des variables.

On notera $A \oplus \{p_1 \mapsto v\}$ la mise à jour de A en p_1 avec la valeur abstraite v .

Exercice 4 (Interprétation abstraite des gardes). Définir les gardes suivantes :

1. $A \sqcap (p == \text{NULL})$
2. $A \sqcap (p \neq \text{NULL})$
3. $A \sqcap (p_1 == p_2)$
4. $A \sqcap (p_1 \neq p_2)$

Exercice 5. Exécutions de l'interpréteur

► **Question 1.** Dériver une règle de l'interpréteur abstrait pour la commande gardée **if** C **then** S_1 **else** S_2 . Puis calculer les états abstraits pour chaque ligne des exemples de l'exo 1 (après insertion des asserts RTE et en simplifiant les négations dans les conditions). Le coloriage de l'interpréteur correspond-il à celui donné à l'exo 1 ?

► **Question 2.** Dans l'exemple de droite de l'exo 1, on remplace la ligne 6 par $p \neq \text{NULL}$. Calculer les états abstraits pour chaque ligne de code. Commenter le coloriage des assertions sur cet exemple. Qu'en déduisez-vous sur la complétude de cette analyse statique ?

► **Question 3.** Comme on est sur un treillis de hauteur finie, on prend $\nabla \stackrel{def}{=} \perp$. Quel est l'invariant de boucle trouvé par l'interpréteur dans le code ci-dessous? En combien d'itérations de la boucle? De quelle couleur colorie-t-il les assert? On commence toujours dans un état initial où toutes les variables sont à NULL.

```
1  p=malloc(sizeof(int));
2  if (p!=NULL) {
3      while (o1 == o4) {
4          o1=o2;
5          o2=o3;
6          o3=p;
7      }
8      assert (o1 != NULL);
9      assert (p != NULL);
10     assert (o2 != NULL);
11 }
```

Exercice 6 (Extension avec pointeurs non-initialisés). On relâche l'hypothèse que les pointeurs sont bien initialisés. En conséquence, au niveau des assert RTE, il faut maintenant garantir que les déréférencements ont bien lieu sur des pointeurs non nuls ayant été initialisés avec un malloc. Quel ∇ faudrait-il prendre pour conserver un analyseur correct? Quelle serait la valeur abstraite retournée par un malloc?

Exercice 7 (Extension avec free). À la suite de l'exercice précédent, on envisage de gérer "free(p)" comme une désinitialisation du pointeur p sous la précondition que p est initialement initialisé et non-nul (c'est-à-dire que le pointeur p sera alors considéré dans la suite de l'analyse comme non-initialisé). Est-ce que cette analyse statique reste correcte? Justifier la réponse.