TP n° 7 – Interprétation abstraite [1]

etienne.jl.simon@gmail.com thomas.h@ens.fr
thomash.fr/teaching/

1 Oups !

Yorel Reivax a écrit le code suivant pour le pilote automatique de la fusée Ariane 5 :

```
let derive x = int_of_float (7. ** x)
```

37 secondes après le décollage, la fusée explose [2]. Pourquoi ?

Afin d'éviter de telles erreurs qui peuvent être très coûteuses [3], on veut prouver statiquement que ces erreurs ne peuvent pas arriver.

2 Domaine des signes

On veut avoir pour chaque variable et pour chaque instant du programme l'ensemble des valeurs possibles pour cette variable afin de vérifier que seules les valeurs autorisées sont possibles. Cependant ces ensembles sont le plus souvent trop complexes pour être représentés en mémoire, on en calcule donc un approximation dans ce qu'on appelle le domaine abstrait.

Nous allons ici considérer le type `int` (le domaine concret correspondant est l'ensemble des parties de $\mathbb{Z}$) et lui associer le domaine abstrait $\{\emptyset, \mathbb{N}^*, -\mathbb{N}^*, \{0\}, \mathbb{N}, -\mathbb{N}, \mathbb{Z}\}$ que l'on peut repésenter en Caml par :

```
type sign = Empty | Positive | Negative | Zero | Non_negative | Non_positive | Int
```

On travaille sur un mini langage où toutes les variables sont des entiers de précision arbitraire. Un programme dans ce langage est une suite de fonctions mutuellement récursives. Chaque fonction a un argument entier et un corps qui est une expression entière.

Exemple de programme :

```
def power2(x) =
  ifpositive x then 2 * power2(x-1) else 1
def f(x) =
  zero(x) - power2(x)
def zero(x) =
  ifpositive x then zero(x-1) else 0
```

On utilise les types suivants pour représenter les programmes en Caml :

```
type binop = Add | Sub | Mul | Div
type expr =
| Const of int
| Var
| Binop of binop * expr * expr
| Ifpositive of expr * expr * expr
| Call of string * expr
type def = { name : string; body : expr; }
```

Question 1. Donner le signe des fonctions du programme d'exemple.

1. Inspiré d'un examen de Jean-Christophe Filliâtre : `https://www.lri.fr/~filliatr/ens/compil/examen/janvier-12.pdf`
2. Histoire vraie : `http://en.wikipedia.org/wiki/Cluster_(spacecraft)`
3. 370 millions de dollars dans le cas d'Ariane 5

Question 2. Écrire la fonction `sign_binop : binop -> sign -> sign -> sign` qui étant donné un opérateur et le signe de deux expressions renvoie le signe de l'opérateur appliqué à ces deux expressions.

Question 3. Écrire une fonction `sign_expr : (string -> sign) -> expr -> sign` qui détermine le signe d'une expression, le premier argument donnant le signe de chaque fonction du programme.

Question 4. Écrire une fonction qui prend un programme (de type `def list`) en argument et renvoie une table donnant le signe de chacune de ses fonctions.

Question 5. Dans certains cas, observer les valeurs possibles d'une variable indépendemment des autres variables ne permet pas de conclure, il faut alors regarder les relations entre les variables.

Modifier la fonction de la question précedente pour qu'elle donne le signe d'une fonction en fonction du signe de son argument.

Donner un exemple de programme pour lequel la nouvelle fonction permet de prouver l'absence de division par zéro alors que l'ancienne fonction n'y arrive pas.

3 Domaine des intervalles

On considère maintenant le domaine abstrait composé des intervalles (représentés en Caml par des couples).

Question 6. Écrire la fonction `interval_binop : binop -> int * int -> int * int -> int * int`.

Question 7. * On vous fournit pour chaque fonction, l'intervalle de valeurs sur lesquelles l'utilisateur peut appliquer cette fonction (la fonction peut toujours être appliquer sur des nombres plus grands ou plus petits par les autres fonctions, l'intervalle des valeurs d'entrées peut donc être agrandi en cours d'analyse).

Écrire une fonction qui étant donnée cette information calcule les intervalles de retour de chaque fonction.

Question 8. * Votre algorithme finit-il à coup sûr ? Si non, proposez une solution.