Projet PAG-WWW

Émilie THOMÉ

October 2020

Introduction

Ce document fait office de rapport pour le projet PAG-WWW. Il concerne la réalisation d'abstractions sur des intervalles de variables. Le code ainsi que les tests effectués ont été envoyés avec ce bilan. Dans une première partie, les opérations et les conditions implémentées seront rapidement mises en revue. Ensuite les différents tests seront présentés et commentés sur leur pertinance. Enfin, les points faibles de l'abstraction ici présentée seront une mis en lumière.

1 Représentation de l'interval

Les intervalles ont été représentés comme étant un couple de `ConstFlattened` comme vu dans le TD1. Le $\top$ de l'interval est $\mathbf{R} = [-\infty; +\infty]$ et $\bot$ est un interval vide $[a; b]$ avec $a > b$ donc en particulier $[+\infty; -\infty]$. Ces limites sont représentées en `ConstFlattened` par les $\top/\bot$. Donc $\top_{interval} = (\bot, \top)$ et $\bot_{interval} = (\top, \bot)$. Je me suis rendue compte après implémentation que PAG-WWW mettait $\bot_{interval} = (\bot, \bot)$ automatiquement, cela a eu des répercution lors de tests mais cela a été réglé : si $I = (a, b)$ avec $b = \bot$ alors I est nécessairement un $\bot_{interval}$.

2 Opérations & Conditions

Les opérations implémentées sont les quatres opérations élémentaire : + ; - ; * ; /. Elles ont été conçues pour résister aux différents cas d'intervalles : $[a; b]$, $[-\infty; b]$, $[a; +\infty]$ ou $[-\infty; +\infty]$.

Les opérations les plus simples étaient l'additions et la soustractions (vues en cours), je m'en suis occupé en premier. Les plus pénibles étaient la multiplication et la division.

La multiplication a été la troisième opération à avoir été implémentée. La simple multiplication des maximum et des minimum entre eux n'est pas la solution de cet opération, il faut regardé les différents cas tels que celui avec un minimum négatif, un maximum positif etc... Mais à cela s'ajoute encore la

présence des $+/-\infty$ représentés par $\top/\bot$. Donc j'en suis venue à réaliser une fonction `findmin` et une `findmax` qui, prenant un premier `ConstFlattened` et une liste de `ConstFlattened`, renvoie respectivement le min et le max de la liste en utilisant le premier élément pour comparer à la première itération. Elle m'a ensuite été utile pour la jonction, l'intersection et la division d'intervalles.

La division a été de loin la plus pénible, car il y a non pas le cas 0 mais les cas $0 \in I$! Que le 0 soit au bord, dans, "vers" les négatifs ou les positifs... Tout a été pris en compte, ce qui finalement n'est pas nécessaire je pense.

Pour les conditions, il n'y avait qu'à faire une fonction branch traitant les différents types de conditions. Suivant les indications du cours j'ai implémenté le cas inférieur et inférieur ou égal (les cas supérieur et supérieur ou égal se déduisent de ses deux fonctions en inversant le terme de gauche et le terme de droite), le cas d'égalité et l'inégalité. Pour l'inégalité, j'ai choisi de rendre l'état actuel car l'abstraction par intervalles ne seraient changée que dans le cas où l'un des bords est le nombre en question (et donc pour le singleton ce serait les deux bords). Je n'ai aussi pas traiter en profondeur le cas où plusieurs variables sont d'un même côté de l'expression, je ne fait que prendre l'interval de ce côté pour faire une condition plus large sur l'autre côté de l'expression.

Afin de traiter ces différents cas j'ai réaliser une fonction intersection, lower, lowerequal et testequal ainsi qu'une fonction equalConst qui ne sert qu'à evaluer une conditions qui n'est posée que sur des constante (et dont on peut tirer un booléen).

Une fonction join et join_int ont été écrite, comme demandé dans le TD2.

3 Tests

Les tests opérés sur l'analyseur ainsi créé sont :
testoperations.cgi : Premier test basique sur les opérations avec des variables simples.
testoperationsinterval.cgi : Teste les opérations sur les intervalles.
testifetdivisionslimites.cgi : Certains différents cas de divisions sont testés ici.
testintervaljoin.cgi : Teste la manière dont les intervalles sont joins.
testif.cgi : Il montre comment se comporte l'analyseur avec un if, comment les différentes branches se regroupent en fin de if.
testifpluscomplique.cgi : Ce test if est plus complexe que le précedent, on peut y voir une imbrications de plusieurs variables dans un test if (et donc la limitation que j'ai posé à cette situation que je trouve plus complexe de $x = 12/y + z$).
testwhile.cgi : Le fonctionnement de l'analyseur en cas de boucle while est testé ici.
test_while_trueloop.cgi : Deuxième test avec une boucle while qui ne s'arrêterait pas, on vérifie donc que l'invariant de boucle soit reconnu.
testameliorations.cgi : Ce test sert à montrer les limites de l'analyseur et à donner une piste d'amélioration.

Je vous ai mis les différents tests ainsi que les images des résultats obtenus en parallèle du rapport.

4 Pistes d'amélioration

On peut voir lors des tests de nombreux points faibles quant à cet analyseur :

- Le cas du test de différence n'est pas restrictif, il ne fait que passer l'état. Alors qu'un cas comme $x! = 0$ alors que l'abstraction de x est $(0, 0)$ doit être impérativement considéré.
- Les expressions à plusieurs variables dans les conditions ne sont pas traité jusqu'à l'obtention d'un point fixe mais ne relève que d'une évaluation de l'opération afin de faire une condition sur l'autre variable. Par exemple, dans le cas $x <= 12/y + z$, le terme $12/y + z$ est évalué avec `expAEval` puis la condition est appliquée à x. Ce qui peut être gênant lorsqu'on a la condition $x + y < 3$ car ici il n'y a aucune restriction apportée, même pas une vérification que l'intersection de $x + y$ et $]-\infty, 2]$ n'est pas vide.
- Et de plus, lors d'un cas d'évaluation booléenne resultant sur un vrai ou faux, la branche fausse est quand même visitée et reportée lors de l'union des deux branches. Cependant, l'état est réinitialisé à bottom pour toutes les variables mais seulement au début de la branche fausse, ce qui veut dire que si des variables changent de valeur dans la branche fausse ces valeurs seront reportée.

Conclusion

L'abstraction par interval est un moyen puissant pour vérifier certaines propriétés de programme comme la non nullité d'une variable tout au long de l'execution. Cependant il faut veiller à ce que tous les cas soient pris en compte afin de ne pas se retrouver avec résultats étonnant. Ce TD m'a permise d'implémenter une version de cette abstraction, perfectible en tout point mais qui permet d'avoir les intervalles des variables de programme contenant les opérations élémentaires et des conditions.