Analyse statique par interprétation abstraite de propriétés temporelles des programmes

Caterina Urban [1]

Accessit du prix de thèse Gilles Kahn 2015

Caterina Urban a soutenu sa thèse [2] en juillet 2015 à l'École normale supérieure de Paris, sous la direction de Radhia Cousot et Antoine Miné. Elle effectue actuellement un stage postdoctoral au sein du département Informatique de l'ETH Zurich, en Suisse.

Dans les dernières décennies, les logiciels sont devenus très importants dans toutes sortes de systèmes. Puisque nous comptons de plus en plus sur les logiciels, les conséquences d'une erreur de programmation peuvent être de plus en plus dramatiques, causant de grandes pertes financières et même humaines. Un exemple bien connu est l'échec spectaculaire du premier vol de la fusée Ariane-5 – voir Figure 1 – causé par un dépassement de capacité [3], qui a abouti à plus de 370 millions de dollars de perte.

Ma thèse a comme objectif général le développement de méthodes mathématiques correctes et efficaces en pratique pour prouver automatiquement la correction de logiciels. Plus précisément, ma thèse est fondée sur la théorie de l'Interprétation Abstraite [2], un cadre mathématique puissant pour l'approximation du comportement

1. http://people.inf.ethz.ch/caurban/
2. https://hal.inria.fr/tel-01176641
3. http://esamultimedia.esa.int/docs/esa-x-1819eng.pdf

FIGURE 1. Échec du vol 501 de la fusée Ariane le 4 juin 1996.

des programmes. En particulier, ma thèse se concentre sur la preuve des propriétés de vivacité des programmes, qui représentent des conditions qui doivent être réalisées ultimement ou de manière répétée pendant l'exécution d'un programme.

La *terminaison* des programmes est la propriété de vivacité la plus fréquement considérée. Les bugs de non-terminaison peuvent compromettre le fonctionnement des systèmes logiciels en les faisant boucler indéfiniment. Des exemples connus de bugs de non-terminaison sont le bug du Microsoft Zune [4] et, plus récemment, du service Microsoft Azure [5]. Les bugs de non-terminaison peuvent également être exploités pour des attaques par déni de service [6]. Prouver la terminaison est donc important pour garantir la fiabilité des logiciels.

La méthode traditionnelle pour prouver la terminaison des programmes [5] est basée sur les *fonctions de rang*, qui associent à chaque état de programme un élément d'un ensemble bien ordonné et dont la valeur décroît au cours de l'exécution du programme. Dans [3], Patrick Cousot et Radhia Cousot ont proposé l'inférence de fonctions de rang par interprétation abstraite. Intuitivement, on peut définir une fonction de rang de façon incrémentale : à partir des état finaux d'un programme, où

4. http://techcrunch.com/2008/12/31/zune-bug-explained-in-detail/

5. https://azure.microsoft.com/en-us/blog/update-on-azure-storage-service-interruption/

6. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1890

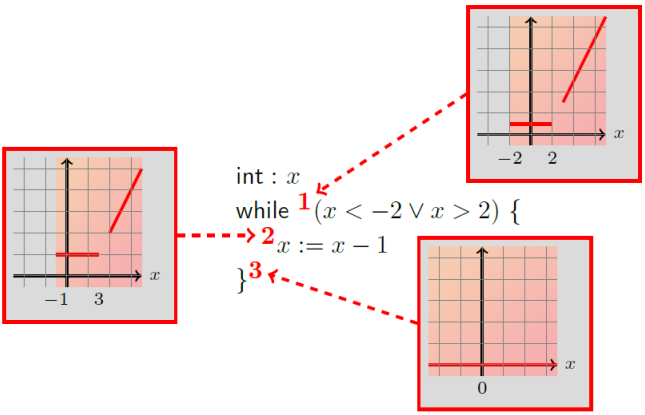

FIGURE 2. Fonctions de rang définies par morceaux dont le domaine fournit, pour chaque point du programme, des conditions suffisantes pour la terminaison et dont la valeur fournit une borne supérieure sur le nombre de pas d'exécution restants avant la terminaison.

la fonction a valeur 0, on ajoute des états au domaine de la fonction en remontant le programme en arrière et en comptant le nombre de pas du programme exécutés en tant que valeur de la fonction. Toutefois, cette fonction de rang n'est évidemment pas calculable. Par conséquent, ma thèse conçoit de nouvelles approximations afin de synthétiser automatiquement des fonctions de rang définies par morceaux, qui fournissent des *conditions suffisantes* pour la terminaison et ainsi des bornes supérieures sur le temps d'exécution avant la terminaison [7, 9, 10] – voir Figure 2 : la fonction au point de contrôle **3**, par exemple, est définie partout et ainsi garantit la terminaison pour toutes les valeurs de x à ce point de contrôle ; au contraire, le domaine de la fonction au point de contrôle **1** fournit la condition $-2 \leqslant x$ pour la terminaison. L'analyse est correcte, c'est-à-dire que toutes les exécutions du programme qui respectent ces conditions suffisantes terminent et une exécution qui ne respecte pas ces conditions peut ne pas terminer. Les approximations sont paramétrées par le choix entre l'expressivité et le coût des approximations sous-jacentes, qui gèrent des informations sur l'ensemble des valeurs possibles des variables du programme [1] ainsi que les relations numériques possibles entre elles [4, 6].

Ma thèse développe également un cadre d'interprétation abstraite pour prouver des *propriétés de vivacité*, qui vient comme une généralisation du cadre proposé par Patrick Cousot et Radhia Cousot pour la terminaison. En particulier, le cadre est

dédié à des propriétés de vivacité exprimées dans la logique temporelle, qui sont utilisées pour s'assurer qu'un événement souhaitable se produit une fois ou une infinité de fois au cours de l'exécution du programme. Comme pour la terminaison, des fonctions de rang définies par morceaux sont utilisées pour déduire des conditions suffisantes pour ces propriétés et fournir des bornes supérieures sur le temps d'exécution avant un événement souhaitable [11].

Finalement, les résultats présentés dans ma thèse ont été mis en œuvre dans un prototype d'analyseur [7]. Les résultats expérimentaux montrent qu'il donne de bons résultats sur une grande variété de programmes et qu'il est compétitif avec l'état de l'art car il est capable d'analyser des programmes qui sont hors de la portée des méthodes existantes. Le prototype a également participé à la compétition SV-COMP [8] en 2014 et 2015 [8] avec des résultats encourageants.

Références

[1] Patrick Cousot and Radhia Cousot. Static Determination of Dynamic Properties of Programs. In *Symposium on Programming*, pages 106–130, 1976.

[2] Patrick Cousot and Radhia Cousot. Abstract Interpretation : a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *POPL*, pages 238–252, 1977.

[3] Patrick Cousot and Radhia Cousot. An Abstract Interpretation Framework for Termination. In *POPL*, pages 245–258, 2012.

[4] Patrick Cousot and Nicolas Halbwachs. Automatic Discovery of Linear Restraints Among Variables of a Program. In *POPL*, pages 84–96, 1978.

[5] Robert W. Floyd. Assigning Meanings to Programs. *Proceedings of Symposium on Applied Mathematics*, 19 :19–32, 1967.

[6] Antoine Miné. The Octagon Abstract Domain. *Higher-Order and Symbolic Computation*, 19(1) :31–100, 2006.

[7] Caterina Urban. The Abstract Domain of Segmented Ranking Functions. In *SAS*, pages 43–62, 2013.

[8] Caterina Urban. FuncTion : An Abstract Domain Functor for Termination (Competition Contribution). In *TACAS*, pages 464–466, 2015.

[9] Caterina Urban and Antoine Miné. An Abstract Domain to Infer Ordinal-Valued Ranking Functions. In *ESOP*, pages 412–431, 2014.

[10] Caterina Urban and Antoine Miné. A Decision Tree Abstract Domain for Proving Conditional Termination. In *SAS*, pages 302–318, 2014.

[11] Caterina Urban and Antoine Miné. Proving Guarantee and Recurrence Temporal Properties by Abstract Interpretation. In *VMCAI*, pages 190–208, 2015.

7. http://www.di.ens.fr/~urban/FuncTion.html
8. http://sv-comp.sosy-lab.org