ÉLÉMENT DE PORTFOLIO 03

Logiciel ou bibliothèque logicielle

1 DÉFINITION DE CET ÉLÉMENT

Titre de l'élément : La plateforme MOPSA pour l'analyse statique par interprétation abstraite

URL de l'élément : `https://gitlab.com/mopsa/mopsa-analyzer`

2 MOTIVATIONS DU CHOIX DE CET ÉLÉMENT

MOPSA (*Modular Open Platform for Static Analysis*) est une plateforme logicielle pour la vérification formelle de la correction des programmes par interprétation abstraite. Nous l'avons incluse dans ce portfolio car il s'agit d'un nouveau logiciel, initié en 2017 dans le cadre du projet ERC Consolidator MOPSA [1] (2016–2021), avec Antoine Miné d'APR comme *Principal Investigator*, et elle est un élément central de la recherche en interprétation abstraite dans l'équipe.

Cette plate-forme a une conception originale qui diffère des analyseurs statiques classiques (comme Astrée [2], Frama-C [3] ou Infer [4]) par son architecture très modulaire et extensible. Son rôle est d'aider la recherche, l'enseignement et la diffusion en analyse statique auprès des étudiants, des chercheurs et des industriels. Elle est diffusée publiquement en logiciel libre depuis 2020 sur GitLab. [2]

Au sein de l'équipe, les travaux de recherche ayant mené à la conception de MOPSA et les résultats de recherche originaux obtenus grâce à MOPSA ont fait l'objet de trois thèses soutenues, de trois thèse en cours, d'un travail d'ingénieur sur cinq ans (2016–2021) et de deux post-docs achevés. Une thèse soutenue et une thèse en cours sont réalisées en collaboration industrielle avec les entreprises Airbus et Nomadic Labs. MOPSA a un rôle structurant et agit comme un tremplin pour la recherche en vérification par interprétation abstraite dans APR. Elle continuera à alimenter sa recherche dans les années futures.

Cette réalisation montre que le travail théorique réalisé par l'équipe en interprétation abstraite (définition de sémantiques formelles, d'abstractions, preuves de correction, etc.) est complété par des réalisations pratiques (implantation, expérimentation) démontrant la réelle application des techniques développées sur des langages et des programmes réalistes. Elle illustre également le souhait d'aller au-delà de la réalisation de prototypes académiques pour diffuser dans la communauté des outils robustes, bien documentés et directement exploitables.

3 PRÉSENTATION DE CET ÉLÉMENT

MOPSA est un logiciel d'analyse statique par interprétation abstraite. Les analyseurs statiques sont des outils pour la vérification des logiciels permettant de détecter les erreurs dès la compilation. L'interprétation abstraite, proposée par Patrick et Radhia Cousot dans les années 70 [5], est un cadre formel pour concevoir des analyseurs sûrs, automatiques, basés sur une sémantique mathématique, couvrant intégralement l'espace des exécutions des programmes et sans faux négatif. Ce type d'outil entre donc dans la catégorie des méthodes formelles et permet de prouver des propriétés de correction sur les programmes avec des garanties mathématiques. L'interprétation abstraite résout les problèmes d'indécidabilité et de complexité inhérents en surapproximant les calculs sémantiques dans un univers abstrait, plus simple. Ces surapproximations sont sûres mais parfois imprécises et causent des fausses alarmes. Il est par ailleurs souvent difficile de définir les calculs abstraits pour les constructions des langages modernes, souvent très complexes (par exemple pour les langages dynamiques, comme Python). Le développement d'abstractions sûres, suffisamment précises et efficaces demeure l'enjeu premier de l'interprétation abstraite.

Pour aider dans cette recherche, MOPSA propose une architecture très modulaire et extensible. Comme les analyseurs de l'état de l'art (Astrée [2], Frama-C [3]), MOPSA réalise un calcul sémantique grâce à une multitude

1. `https://mopsa.lip6.fr`
2. `https://gitlab.com/mopsa/mopsa-analyzer`

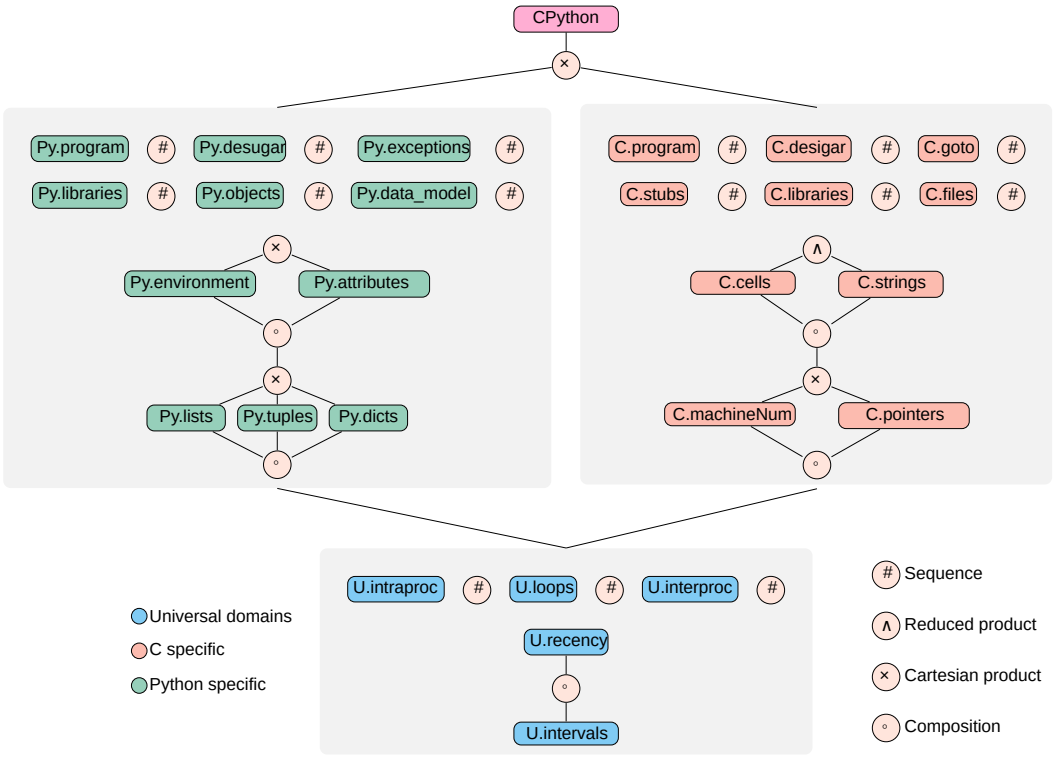

FIGURE 1 – Composition de domaines abstraits dans MOPSA pour l'analyse de valeurs et d'exceptions de programmes Python appelant des fonctions C natives et exploitant la bibliothèque C standard.

d'abstractions communiquantes, mais il possède une architecture beaucoup moins rigide que ces outils : il fait l'hypothèse que les abstractions de tous types de données (numériques, de pointeurs, d'objets, etc.) et les itérateurs sur les constructions syntaxiques du programme sont des abstractions obéissant à une signature commune, permettant d'exprimer des relations complexes entre valeurs et peuvent être combinées sans couplage. Il enrichit également les mécanismes existants de communication entre domaines avec une réécriture dynamique d'expressions.

MOPSA est écrit en OCaml. Son architecture, décrite dans [10], occupe 13 KLOC et est complétée par une bibliothèque d'abstractions génériques (abstraction du tas, intervalles, itérateurs de boucles, etc.) de 25 KLOC. Nous avons implanté dans MOPSA une première analyse, assez classique, des erreurs à l'exécution de programmes C (11 KLOC) [9] et défini un langage de modélisation de bibliothèques [14]. Cela nous a permis d'analyser des programmes C système de taille modeste, comme ceux de Coreutils. Afin de permettre la reproductibilité, ces analyses « benchmark » sont diffusées sous forme de projets compagnons sur Gitlab, et nous participons également à la compétition SV-COMP 2023. Nous avons également mis à contribution MOPSA pour soutenir la recherche en interprétation abstraite au-delà de l'état de l'art, vers des analyses de nouveaux langages et de nouvelles propriétés. Nous avons ainsi développé une analyse de type, de valeur et d'exception pour Python [11, 12] (13 KLOC). Nous avons également développé une analyse originale pour la vérification de programmes multi-langages, mêlant C et Python [13]. La figure 1 montre la combinaison des abstractions utilisées pour cette analyse : outre les domaines génériques (en bas) utilisés dans la plus part des analyses, nous avons inclus les domaines développés indépendemment et spécifiquement pour l'analyse du C (à droite) et de Python (à gauche). Un seul nouveau domaine (en haut), très court (2,5 KLOC), a dû être développé pour gérer les interactions entre les deux langages. Cela démonte la capacité de l'architecture à s'adapter à de nouveaux problèmes tout en réutilisant le travail déjà fourni. Des travaux en cours de finalisation ciblent l'analyse de *smart contracts* sur la *blockchain* Tezos [1], l'analyse de portabilité [6, 8] et de patchs en C [7]. Ils devraient être prochainement intégrés dans la branche principale de MOPSA.

4 RÉFÉRENCES BIBLIOGRAPHIQUES

[1] G. Bau, A. Miné, V. Botbol, and M. Bouaziz. Abstract interpretation of Michelson smart-contracts. In *Proc. of the 11th ACM SIGPLAN Int. Workshop on the State Of the Art in Program Analysis (SOAP'22)*, pages 36–43. ACM, Jun. 2022.

[2] J. Bertrane, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, and X. Rival. Static analysis and verification of aerospace software by abstract interpretation. In *AIAA Infotech@Aerospace*, number 2010-3385, pages 1–38. AIAA, Apr. 2010.

[3] S, Blazy, D. Bühler, and B. Yakobowski. Structuring abstract interpreters through state and value abstractions. In *Verification, Model Checking, and Abstract Interpretation*, pages 112–130. Springer, 2017.

[4] C. Calcagno, D. Distefano, J. Dubreil, D. Gabi, P. Hooimeijer, M. Luca, P. O'Hearn, I. Papakonstantinou, J. Purbrick, and D. Rodriguez. Moving fast with software verification. In *NFM*, pages 3–11. Springer, 2015.

[5] P. Cousot and R. Cousot. Abstract interpretation : A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of POPL'77*, pages 238–252. ACM, Jan. 1977.

[6] D. Delmas. *Static analysis of program portability by abstract interpretation*. PhD thesis, Sorbonne Université, Dec. 2022.

[7] D. Delmas and A. Miné. Analysis of software patches using numerical abstract interpretation. In *Proc. of the 26th Int. Static Analysis Symp. (SAS'19)*, volume 11822 of *LNCS*, pages 225–246. Springer, Oct. 2019.

[8] D. Delmas, A. Ouadjaout, and A. Miné. Static analysis of endian portability by abstract interpretation. In *Proc. of the 28th Int. Static Analysis Symp. (SAS'21)*, volume 12913 of *LNCS*, pages 102–123. Springer, Oct. 2021.

[9] M. Journault. *Precise and modular static analysis by abstract interpretation for the automatic proof of program soundness and contracts inference*. PhD thesis, Sorbonne Université, Nov. 2019.

[10] M. Journault, A. Miné, R. Monat, and A. Ouadjaout. Combinations of reusable abstract domains for a multilingual static analyzer. In *Proc. of the 11th Working Conf. on Verified Software : Theories, Tools, and Experiments (VSTTE'19)*, volume 12031 of *LNCS*, pages 1–18. Springer, Jul. 2019.

[11] R. Monat. *Static type and value analysis by abstract interpretation of Python programs with native C libraries*. PhD thesis, Sorbonne Université, Nov. 2021.

[12] R. Monat, A. Ouadjaout, and A. Miné. Static type analysis by abstract interpretation of Python programs. In *Proc. of the 34th European Conf. on Object-Oriented Programming (ECOOP'20)*, volume 166 of *Leibniz Int. Proceedings in Informatics (LIPIcs)*, pages 17 :1–17 :29. Dagstuhl Publishing, Jul. 2020.

[13] R. Monat, A. Ouadjaout, and A. Miné. A multilanguage static analysis of Python programs with native C extensions. In *Proc. of the 28th Int. Static Analysis Symp. (SAS'21)*, volume 12913 of *LNCS*, pages 323–345. Springer, Oct. 2021.

[14] A. Ouadjaout and A. Miné. A library modeling language for the static analysis of C programs. In *Proc. of the 27th Int. Static Analysis Symp. (SAS'20)*, volume 12389 of *LNCS*, pages 223–246. Springer, Nov. 2020.