Sujet de stage de M2 : étude empirique, théorique et prospective du langage des expressions Nix

Lê Thành Dũng (Tito) Nguyễn* Ryan Lahfa†

Lieu : Laboratoire d'Informatique et des Systèmes, Aix-Marseille Université (campus de Luminy)

Période : Avril à août 2026 (4 mois et demi)

Contexte : Nix est un gestionnaire de paquets issu de la recherche académique [Dol06], servant de base à la distribution Linux NixOS [DLP10].

Dans la plupart des gestionnaires de paquets, l'état actuel est le résultat d'une succession de commandes impératives d'installation / suppression / mise à jour de paquets, ainsi que des effets de leurs scripts d'installation. Ces derniers sont loin d'être anodins, d'où notamment une thèse dédiée à la vérification des scripts d'installation de paquets Debian [Jea21]. Au contraire, le modèle de Nix est déclaratif : l'état du système est entièrement déterminé par le contenu de quelques fichiers de configuration centralisés. Cette approche inspirée de la programmation purement fonctionnelle permet notamment d'accroître la reproductibilité de la compilation [MZZ25].

Les fichiers de configuration de Nix sont en fait des programmes écrits dans un langage Turing-complet, parfois appelé « langage (des expressions) Nix ». Il s'agit d'un langage fonctionnel d'ordre supérieur à évaluation paresseuse (*call-by-need* [AF97 ; MOW98] avec partage de sous-expressions).

Récemment,[1] Broekhoff et Krebbers ont proposé une méthodologie générale pour formaliser la sémantique de langages de scripts, en l'appliquant au langage Nix [BK25]. Cependant, leur sémantique de Nix est *call-by-name*, ce qui la fait différer de l'implémentation officielle sur 5 cas de tests de Nix 2.25.0 [BK25, Section 5]. Il a été découvert plus tard [Lah25] que des interactions subtiles entre égalité de pointeurs de fonctions et partage de sous-expressions mènent à d'autres divergences entre *call-by-name* et évaluation paresseuse. En fait, ce comportement de l'égalité de pointeurs semble indésirable, puisqu'il rend observable dans le langage quelque chose qui devrait être de l'ordre du détail d'implémentation bas niveau.

Objectifs : Bien que le langage Nix soit Turing-complet (et même « morpion-complet » [Stü21]) en principe, nous pensons que son caractère *domain-specific* le rend suffisamment minimaliste pour se prêter à une description mathématique, renouant ainsi avec une vieille ambition de la sémantique des langages de programmation [MTH90 ; MHR20 ; Ast19].

Dans un premier temps, nous tenterons de définir une sémantique opérationnelle aussi fidèle que possible au langage Nix « réel ». Cette fidélité sera mesurée à l'aide de la suite de tests de Nix. Un objectif est donc d'avoir une sémantique traduisible de façon transparente — c'est-à-dire en minimisant le risque d'introduire des bugs — en code exécutable (pas forcément efficace). Pour cela, une approche potentielle serait d'adapter une *machine abstraite* pour le call-by-need, par

*Chargé de recherche CNRS au LIS, équipe Logique, Sémantique et Catégories `https://lsc.lis-lab.fr/`
†Membre de la *core team* de Lix `https://lix.systems/team/`

[1] Notons aussi un travail récent sur la sémantique comparative des gestionnaires de paquets [Gib+26].

exemple [Acc+25, Section 7]. Un défi est qu'une grande partie de la littérature théorique sur le call-by-need concerne des λ-calculs purement fonctionnels, alors que nous aurons besoin de travailler dans un cadre à effets.

Dans un second temps, nous utiliserons cette base pour réfléchir à des améliorations du langage Nix (nourissant ainsi une réflexion déjà menée au sein des projets Lix et Snix[2]). Il s'agirait de trouver des modifications minimales à la sémantique du langage assurant des propriétés souhaitables. Par exemple, pour exclure des détails d'implémentation observables comme mentionné plus haut, on peut tenter d'établir une équivalence entre une sémantique opérationnelle bas niveau, et une autre sémantique plus haut niveau.

Étant donné la durée limitée du stage, nous n'essaierons pas de formaliser nos sémantiques dans un assistant de preuve, contrairement à [BK25]. Cependant, nous tenterons de garder des approches qui se prêteraient à une formalisation ultérieure en Rocq — potentiellement en collaboration avec Cyril Cohen (Inria Lyon) qui a manifesté un grand intérêt pour l'étude du langage Nix (communication personnelle).

Références

[Acc+25] Beniamino Accattoli, Francesco Magliocca, Loïc Peyrot et Claudio Sacerdoti Coen. « The Cost of Skeletal Call-By-Need, Smoothly ». In : *10th International Conference on Formal Structures for Computation and Deduction (FSCD 2025)*. Sous la dir. de Maribel Fernández. T. 337. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany : Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025, 5 :1-5 :22. ISBN : 978-3-95977-374-4. DOI : 10.4230/LIPIcs.FSCD.2025.5 (cf. p. 2).

[AF97] Zena M. Ariola et Matthias Felleisen. « The Call-By-Need lambda Calculus ». In : *Journal of Functional Programming* 7.3 (1997), p. 265-301. DOI : 10.1017/S0956796897002724 (cf. p. 1).

[Ast19] Troy Astarte. « Formalising Meaning : a History of Programming Language Semantics ». Thèse de doct. Newcastle upon Tyne, 2019. URL : http://homepages.cs.ncl.ac.uk/troy.astarte/res/pdf/TK_Astarte_Formalising_Meaning_2019.pdf (cf. p. 1).

[BK25] Rutger Broekhoff et Robbert Krebbers. « Verified Interpreters for Dynamic Languages with Applications to the Nix Expression Language ». In : *Proceedings of the ACM on Programming Languages* 9.ICFP (2025), p. 917-946. DOI : 10.1145/3747537 (cf. p. 1, 2).

[DLP10] Eelco Dolstra, Andres Löh et Nicolas Pierron. « NixOS : A purely functional Linux distribution ». In : *Journal of Functional Programming* 20.5-6 (2010), p. 577-615. DOI : 10.1017/S0956796810000195 (cf. p. 1).

[Dol06] Eelco Dolstra. « The purely functional software deployment model ». Thèse de doct. Utrecht University, Netherlands, 2006. URL : http://dspace.library.uu.nl/handle/1874/7540 (cf. p. 1).

[Gib+26] Ryan Gibb, Patrick Ferris, David Allsopp, Thomas Gazagnaire et Anil Madhavapeddy. *Package Managers à la Carte : A Formal Model of Dependency Resolution*. 2026. arXiv : 2602.18602 [cs.PL]. URL : https://arxiv.org/abs/2602.18602 (cf. p. 1).

[Jea21] Nicolas Jeannerod. « Verification of Shell scripts performing file hierarchy transformations ». Thèse de doct. Université Paris Cité, mars 2021. URL : https://theses.hal.science/tel-03917971 (cf. p. 1).

[2]Respectivement https://lix.systems/ et https://snix.dev/

[Lah25] Ryan Lahfa. *Lix Wiki — Pointer Equality*. Nov. 2025. URL : https://wiki.lix.systems/books/development/page/pointer-equality (cf. p. 1).

[MHR20] David MacQueen, Robert Harper et John H. Reppy. « The history of Standard ML ». In : *Proceedings of the ACM on Programming Languages* 4.HOPL (2020), 86 :1-86 :100. DOI : 10.1145/3386336 (cf. p. 1).

[MOW98] John Maraist, Martin Odersky et Philip Wadler. « The Call-by-Need Lambda Calculus ». In : *Journal of Functional Programming* 8.3 (1998), p. 275-317. DOI : 10.1017/S0956796898003037 (cf. p. 1).

[MTH90] Robin Milner, Mads Tofte et Robert Harper. *The Definition of Standard ML*. MIT Press, 1990. ISBN : 978-0-262-63132-7 (cf. p. 1).

[MZZ25] Julien Malka, Stefano Zacchiroli et Théo Zimmermann. « Does Functional Package Management Enable Reproducible Builds at Scale ? Yes ». In : *22nd IEEE/ACM International Conference on Mining Software Repositories, MSR@ICSE 2025, Ottawa, ON, Canada, April 28-29, 2025*. IEEE, 2025, p. 775-787. DOI : 10.1109/MSR66628.2025.00115 (cf. p. 1).

[Stü21] Terru Stübinger. *Are Nix Expressions Pacman-Complete ?* Nov. 2021. URL : https://stuebinm.eu/posts/nix-tic-tac-toe-complete.html (cf. p. 1).