Ce document est un résumé en français de l'article “A Trustworthy Mechanized Formalization of R” [BDT18], soit « Une spécification digne de confiance de R dans un assistant de preuve ». Cet article a été publié à DLS (“Dynamic Languages Symposium”, soit le symposium international sur les langages dynamiques), une conférence internationale. L'article est accompagné d'un artefact disponible à l'adresse `https://github.com/Mbodin/CoqR/releases/tag/DLS2018`.

Contexte Le langage de programation R [R C15 ; IG96 ; R C] a reçu beaucoup d'attention ces derniers temps, ayant des millions d'utilisateurs dans des domaines très variés, allant de la biologie à la finance. Comme R est entièrement géré par sa communauté, le langage lui-même reflète cette diversité. R a été conçu pour avoir un grand pouvoir expressif à partir d'un nombre succin de caractères. Cela induit parfois un coût en compréhension et prédictibilité : la sémantique de R est subtile et truffée de cas spéciaux qui peuvent mener à des résultats imprévus.

Les raisons de ces cas spéciaux sont nombreuses — des décisions historiques (dette technique) au besoins spécifiques de certains programmeurs. Même une fonctionnalité aussi basique qu'un appel de fonction peut être source de surprises. Le programme R ci-dessous illustre différents modes d'appels. La fonction `f` concatène ses trois arguments. Elle peut être appelé en utilisant la position de ses arguments, en utilisant leurs noms, ou de manière mixte. Mais la fonction `f` peut aussi être appelée en indiquant non pas le nom de ses arguments, mais un préfixe (dans l'exemple `a` à la place de `abc`). S'il y a une ambigüité dans les préfixes (comme dans la dernière ligne de l'exemple), une erreur se produit. Cependant, l'avant dernière ligne de l'exemple ne produit pas d'erreur car il y a une correspondance exacte : bien que `ab` soit à la fois un préfixe de `ab` et `abc`, il correspond exactement à `ab` et n'est pas compté comme ambigu. Ainsi, bien que `a` soit à la fois préfixe de `ab` et `abc`, il n'est pas considéré comme ambigu dans l'avant dernière ligne car l'argument `ab` a déjà été associé.

```
f <- function (abc, ab, de) { c (abc, ab, de) }
f(1, 2, 3)           # Par position
f(de=3, abc=1, ab=2) # Par nom
f(1, d=3, 2)         # Les deux à la fois
f(3, a=1, ab=2)      # abc vaudra 1 lors de l'appel
f(a=3, 1, 2)         # Erreur à l'exécution
```

Cet exemple ne présente que quelques comportements surprenants des appels de fonction en R, qui possède d'autres cas spéciaux (arguments manquants, trop d'arguments, présence de l'argument spécial « `...` », évaluation paresseuse des arguments, etc.). De tels comportements spéciaux sont omniprésents dans R [Bur11]. Des outils de débogue existent [McP14], mais ils ne permettent pas de compenser de manière systématique la complexité de la sémantique de R. En conséquence, il est très difficile de détecter en amont la présence de bogues dans un programme R, et raisonner — même de manière informelle — sur de tels programmes est très difficile.

Les assistants de preuve tels que Coq [CHP+84] permettent de raisonner mathématiquement sur le comportement des programmes. Cependant, une telle preuve s'appuie invariablement sur une spécification du langage de programmation utilisé. Il existe une documentation de R [R C00], mais celle-ci n'est malheureusement pas adaptée au raisonnement. En effet, de nombreuses ambigüités sont présentes dans cette documentation. De plus, nous avons trouvé des différences entre ce que spécifie ce document et l'interpréteur de référence du langage, GNU R. Par exemple, la spécification indique que `if ("TRUE") 42` retourne une erreur alors que l'interpréteur de référence retourne `42` sans erreur. Il y a donc un besoin de spécification formelle du langage R, spécification qui pourra ensuite être utilisée pour raisonner sur les programmes R.

Contributions Cet article présente CoqR, une formalisation en Coq du langage R. Cette formalisation prend en compte tous les cas spéciaux de l'interpréteur, afin de pouvoir raisonner de manière complète sur le langage. En particulier cette sémantique est de taille conséquente. Les autres contributions de l'article consistent donc en différentes procédures pour s'assurer que cette sémantique est digne de confiance. Enfin, l'article montre que cette spécification est utilisable pour faire des preuves en montrant qu'il est possible de prouver en Coq des propriétés simples sur cette dernière.

La formalisation CoqR ne formalise cependant pas la totalité du langage : devant la complexité de R, fournir une couverture totale du langage et de ses bibliothèques aurait été irréaliste. Le but premier de cette formalisation est donc de fournir des bases solides pour tout projet formel sur le langage R. Nous nous sommes concentrés sur le noyau du langage, ainsi que 120 fonctionnalités. Cela nous a permis de nous concentrer sur ces fonctionnalités et de fournir une couverture conséquente sur ces dernières. Notre implémentation du langage passe ainsi un nombre significatif de tests provenant de projets conséquents dans la communauté R. Le projet CoqR contient plus de 18 000 lignes de Coq. Pour donner une comparaison, JSRef [Bod+14], qui est elle-même considérée comme une formalisation conséquente, contient environ 12 500 lignes de Coq.

Un des points principaux de l'article consiste en l'identification de sources de confiance pour le langage. Il a été choisi de se comparer à l'interpréteur GNU R car ce dernier est l'interpréteur de référence du langage R : tout interpréteur alternatif est comparé avec ce dernier. Du fait de la taille de CoqR, une approche similaire à JSCert [Bod+14] a été choisie : la formalisation Coq a été d'une part grandement testée, et d'autre part, a été comparée ligne à ligne avec la source de confiance. Nous pensons que cette double vérification permet de donner une grande confiance en la formalisation CoqR.

Afin de tester l'interpréteur CoqR, une architecture de tests a été conçue. Un soin particulier a été mis en place afin que cette architecture de tests, en plus de détecter les différences de comportement entre CoqR et GNU R, puisse classifier ces différences. Cette classification nous a permis d'accélérer le processus de formalisation en guidant quelles étaient les fonctionnalités à implémenter ou déboguer en priorité. Nous avons réutilisé les suites de tests de deux pro-

jets conséquents : d'une part celles de GNU R, et d'autre part celles de FastR [Kal+14]. Ces suites de tests contiennent plus de 17 000 expressions. Nous avons de plus conçu notre propre suite de tests afin d'augmenter la couverture de test. Nous avons ainsi ajouté plus de 3 000 tests afin de tester spécifiquement les cas spéciaux de R. Sur ces 20 000 expressions, de nombreuses expressions utilisent malheureusement des fonctionnalités non présentes dans l'interpréteur CoqR : la plupart finissent avec un message spécifique indiquant que CoqR n'implémente pas une fonction utilisée dans ce test. Si l'on réduit les tests à ceux que CoqR peut passer (et passe effectivement), on obtient plus 6 000 tests.

Afin de pouvoir comparer ligne à ligne notre interpréteur CoqR avec l'interpréteur de référence GNU R, nous nous sommes basés sur le système de notations de Coq. Cette approche est différente de JSCert, qui se basait sur des règles de dérivation. Les règles de dérivation n'étant pas exécutables en Coq, l'approche de JSCert demande ainsi un double-effort supplémentaire pour pouvoir tester la sémantique : d'une part un interpréteur doit être définit, d'autre part ce dernier doit être prouvé correct vis-à-vis de la spécification à base de règles de dérivation. L'approche par notation permet ainsi de ne pas dupliquer le travail de formalisation, mais vient avec un travail en amont d'ingénierie de notations Coq.

CoqR a été la première spécification formelle de R. Elle a été construite pour être de confiance en étant à la fois testée et en étant proche textuellement de l'interpréteur de référence GNU R. Elle a de plus été conçue en Coq dans un format qui permet de montrer des propriétés de programme. CoqR n'est qu'un premier pas dans l'exploration formelle de R, mais il pose de solides fondations pour cette direction de recherche.

Références

[BDT18] Martin Bodin, Tomás Diaz et Éric Tanter. “A Trustworthy Mechanized Formalization of R”. In : *DLS*. 2018.

[Bod+14] Martin Bodin et al. “A Trusted Mechanised JavaScript Specification”. In : *POPL*. 2014.

[Bur11] Patrick Burns. *The R Inferno*. 2011.

[CHP+84] Thierry Coquand, Gérard Huet, Christine Paulin et al. *the Coq Proof Assistant*. 1984. url : `https://coq.inria.fr/`.

[IG96] Ross Ihaka et Robert Gentleman. “R : a Language for Data Analysis and Graphics”. In : *Journal of Computational and Graphical Statistics* (1996).

[Kal+14] Tomas Kalibera et al. “A Fast Abstract Syntax Tree Interpreter for R”. In : *Virtual Execution Environments*. 2014.

[McP14] Jonathan McPherson. “Debugging in R”. In : *The R User Conference, UseR !* 2014.

[R C] R Core Team. *The Comprehensive R Archive Network*. URL : `https://cran.r-project.org/` (visité le 2018).

[R C00] R Core Team. “R Language Definition”. In : *R Foundation for Statistical Computing* (2000).

[R C15] R Core Team. *R : A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. 2015. URL : `https://www.R-project.org/`.