Théories de permutations avec Why3

Alain Giorgetti

Institut FEMTO-ST, UMR CNRS 6174
Univ. of Bourgogne Franche-Comté
16 route de Gray, F-25030 Besançon Cedex, France
alain.giorgetti@femto-st.fr

Résumé

Ce travail s'inscrit dans le cadre d'une étude générale de la pertinence de l'application des méthodes formelles du génie logiciel à la recherche en combinatoire. Dans cet article, nous expérimentons l'adéquation de l'outil de preuve formelle Why3 pour vérifier mécaniquement le contenu d'un article de combinatoire récent, sur des théories du premier ordre pour la notion de permutation sur un ensemble fini. La première théorie voit une permutation comme une fonction bijective, la seconde comme un couple d'ordres totaux stricts. Nous démontrons formellement avec Why3 certaines propriétés des modèles de ces deux théories. Nous introduisons ensuite une troisième théorie réduite à un ordre total strict sur un intervalle fini d'entiers. Nous démontrons formellement la propriété caractéristique de ses modèles, modulo un lemme que nous testons avec l'outil AutoCheck de test aléatoire et énumératif de propriétés OCaml et Why3.

1 Introduction

Les permutations sont des objets mathématiques fondamentaux, abondamment étudiés, notamment en combinatoire. En informatique, une permutation permet par exemple de spécifier qu'un programme qui trie une collection de données préserve son contenu. Très récemment, Michael Albert, Mathilde Bouvel et Valentin Féray ont étudié deux théories des permutations sur un ensemble fini, dans le cadre de la logique du premier ordre [ABF20]. Une permutation y est d'abord vue comme une bijection, puis comme un ordre total sur un ensemble fini lui-même totalement ordonné. Nous souhaitons étudier l'adéquation de l'environnement de vérification déductive Why3 [BFM+20] à la formalisation de ces théories et au raisonnement formel sur ses modèles.

Nous avons formalisé ces deux théories de permutations dans le langage WhyML de Why3, puis nous avons démontré formellement des propriétés de leurs modèles finis, avec Why3 et Coq [Coq20], comme résumé dans les parties 2 et 3. La partie 4 résume notre proposition de simplification de la seconde théorie, dans laquelle l'ensemble fini permuté est un intervalle fini d'entiers, naturellement ordonné, ce qui permet de ne considérer qu'un ordre total au lieu de deux. Comme modèles possibles de cette troisième théorie, nous avons défini deux fonctions qui devraient être des permutations mutuellement inverses. Dans la démonstration formelle de la propriété caractéristique des modèles de cette théorie, il ne reste plus qu'à démontrer qu'une de ces deux fonctions est inverse de l'autre. Nous avons testé cette conjecture à l'aide d'AutoCheck [EG20], un nouvel outil de test aléatoire et énumératif de propriétés OCaml et Why3.

Cette étude de cas de combinatoire certifiée [GM21] est détaillée dans un rapport de recherche de 17 pages [Gio20] qui rend la démarche et le code accessibles à tout étudiant ou combinatoricien, même peu familier avec la spécification et la preuve formelle, afin qu'il puisse s'en inspirer pour pratiquer ces méthodes. Peu de connaissances de Why3 sont requises pour comprendre le code présenté. En effet, chaque extrait de code est commenté, et la formalisation

est proche du texte mathématique initial. En complément du manuel de Why3, deux articles récents détaillent davantage le mécanisme de clonage de modules [FP20] et la notion de type avec invariant [Fil20] utilisés. Pour l'audience des JFLA, nous résumons ce travail (parties 2, 3 et 4), puis nous situons nos contributions par rapport à l'état de l'art (partie 5) et nous en présentons quelques perspectives (partie 6).

Le code est développé et vérifié avec Why3 1.3.3, Coq 8.9.0, AutoCheck 0.1.1 et les solveurs SMT Alt-Ergo 2.2.0, CVC4 1.6 et Z3 4.7.1. Le code, sa documentation, les démonstrations formelles et leurs statistiques sont accessibles à partir de la page `https://alaingiorgetti.github.io/autocheck/`.

La plateforme de vérification déductive Why3 [BFM+20] utilisée est fondée sur une logique classique du premier ordre avec types. Elle propose le langage WhyML pour la spécification fonctionnelle de programmes impératifs et fonctionnels, avec des annotations formelles (préconditions, postconditions, assertions intermédiaires, invariants et variants de boucle, etc.). Le système utilise toutes ces annotations pour générer des *conditions de vérification*, aussi appelées *buts*, qui sont des formules WhyML. Ensuite, des prouveurs automatiques (comme les solveurs SMT) et des assistants de preuve (comme Coq) peuvent être utilisés pour prouver ces buts. De plus, Why3 propose de nombreuses transformations logiques pour simplifier ces buts, jusqu'à obtenir soit une preuve 100% interactive, soit des sous-buts assez simples pour être déchargés automatiquement. Why3 propose quatre *stratégies* de recherche de preuve, nommées Auto level 0 à 3, qui sont des combinaisons de transformations logiques et d'appels aux prouveurs automatiques, qui donnent souvent de bons résultats, dans une limite de temps croissante selon leur numéro. Nous dirons qu'une preuve avec Why3 est *automatique* si elle est obtenue par l'une de ces stratégies, et qu'elle est *interactive* sinon, lorsqu'au moins une transformation Why3 a été appliquée à la main. Enfin, Why3 offre un mécanisme d'extraction automatique des programmes vers divers langages cibles, dont OCaml, pour les exécuter.

2 Une permutation est une bijection

Dans [ABF20], la première théorie, nommée TOOB (T*heory* O*f* O*ne* B*ijection*), voit une permutation comme un symbole relationnel R binaire bijectif. Les axiomes de cette théorie ne sont pas précisés.

La librairie standard de Why3 ne contient pas de tels axiomes. Nous avons donc axiomatisé en WhyML qu'un prédicat binaire `rel` quelconque est fonctionnel, injectif, surjectif et bijectif, au sens des définitions suivantes : le *graphe* d'une fonction f de A dans B est l'ensemble $\{(x, f(x)) \mid x \in A\}$ des couples composés d'un élément x du *domaine* A de la fonction et de son *image* $f(x)$ par la fonction. Une relation binaire est dite *fonctionnelle* si elle est le graphe d'une fonction, et *injective*, *surjective* ou *bijective* si cette fonction l'est.

Ainsi, le code WhyML

```
module TOOB
  type t
  predicate rel t t
  clone export Bijectivity with type t = t, type u = t, predicate rel = rel, axiom .
end
```

formalise la théorie TOOB. Sa signature est réduite au prédicat binaire `rel`. Ses axiomes sont obtenus par instanciation de notre module `Bijectivity` qui caractérise une relation bijective entre un type `t` et un type `u`. La notation `axiom .` signifie que tous les axiomes de ce module sont admis.

Les auteurs de [ABF20] démontrent que les fonctions bijectives σ sur l'ensemble $[1..n]$ des n premiers entiers naturels non nuls sont des modèles de la théorie TOOB. Nous formalisons tout intervalle d'entiers relatifs $[l..u] = \{l, \ldots, u\}$ non vide ($l \leq u$) par un type WhyML d'entiers bornés `bint` (pour b*ounded* int*eger*) défini comme un enregistrement à un seul champ `to_int` du type `int` des entiers relatifs de Why3, soumis à l'invariant de type ($l \leq$ `to_int` $\leq u$). [Gio20, partie 2.3]. Nous formalisons ensuite σ par

```
val constant sigma : map bint bint
axiom Sigma_bij : bijective sigma
```

où `map` est le type polymorphe des applications, défini dans la librairie standard par

```
type map 'a 'b = 'a → 'b
```

et `bijective` est le prédicat que nous avons défini par

```
predicate injective (m: map 'a 'b) = ∀ i j: 'a. m[i] = m[j] → i = j
predicate surjective (m: map 'a 'b) = ∀ j: 'b. ∃ i: 'a. m[i] = j
predicate bijective (m: map 'a 'b) = injective m ∧ surjective m
```

Le modèle de TOOB associé à la permutation σ de $[1..n]$ est le couple (A^σ, R^σ) où $A^\sigma = [1..n]$ et R^σ est tel que $i\ R^\sigma\ j$ si et seulement si $\sigma(i) = j$ [ABF20, partie 2.2]. Cette correspondance entre σ et R^σ est formalisée par le prédicat

```
predicate rel_sigma (i j: bint) = map_rel sigma i j
```

où le prédicat `map_rel` défini par

```
predicate map_rel (m: map 'a 'b) (x: 'a) (y: 'b) = (m[x] = y)
```

associe son graphe à toute application. L'instanciation

```
clone TOOB with type t = bint, predicate rel = rel_sigma
```

du module `TOOB` formalise que le couple (A^σ, R^σ) est un modèle de la théorie TOOB. Ce clonage sans le mot-clé WhyML `axiom` exige la démonstration des instances correspondantes des axiomes du module `TOOB`. Ils sont prouvés automatiquement à l'aide d'Alt-Ergo, CVC4 ou Z3, sauf l'axiome de surjectivité, qui requiert une preuve interactive composée de deux transformations Why3.

Enfin, 17 lignes de code WhyML [Gio20, Listing 3] formalisent que tout modèle (A, R) de TOOB est isomorphe à un modèle de permutation (A^σ, R^σ) [ABF20, Proposition 2]. Cette propriété est démontrée interactivement avec Why3, à l'aide d'un lemme démontré en Coq.

3 Une permutation est un couple d'ordres totaux stricts

La théorie TOTO (T*heory* O*f* T*wo* O*rders*) de [ABF20] définit une permutation p de taille n comme un couple $(<_P, <_V)$ d'ordres totaux stricts sur le même ensemble $\{e_1, \ldots, e_n\}$ de n éléments, appelés *positions* (resp. *valeurs*) lorsqu'ils sont comparés avec $<_P$ (resp. $<_V$). L'ordre $<_P$ sur les positions est défini par $e_1 <_P \ldots <_P e_n$. L'ordre $<_V$ sur les valeurs est défini par $p(e_1) <_V \ldots <_V p(e_n)$.

Nous formalisons la théorie TOTO par

```
module TOTO
  type t               (* type for positions and values *)
  predicate ltP t t    (* strict total order on positions *)
  predicate ltV t t    (* strict total order on values *)
  clone relations.TotalStrictOrder with type t = t, predicate rel = ltP, axiom .
  clone relations.TotalStrictOrder as V with type t = t, predicate rel = ltV, axiom .
end
```

où les prédicats `ltP` et `ltV` formalisent respectivement les ordres totaux stricts $<_P$ et $<_V$. Le module `TotalStrictOrder` du fichier `relations.mlw` de la librairie standard de Why3 définit qu'une endorelation `rel` sur un type `t` est un ordre total strict si elle est transitive, asymétrique, et totale au sens de l'axiome suivant :

```
axiom Trichotomy : ∀ x y:t. rel x y ∨ rel y x ∨ x = y
```

Le modèle de la théorie TOTO associé à la permutation σ sur l'intervalle d'entiers $[l..u]$ est le triplet $(A^\sigma, <_P^\sigma, <_V^\sigma)$ où $A^\sigma = \{(i, \sigma(i)) \mid i \in [l..u]\}$ est l'ensemble des arcs du graphe de la fonction σ, et $<_P^\sigma$ (resp. $<_V^\sigma$) est l'ordre induit sur la première (resp. seconde) composante des éléments de A^σ par l'ordre naturel strict $<$ sur les entiers [ABF20]. Nous formalisons σ par

```
val constant m : map bint bint
```

et A^σ par un type `arrow` dont les habitants sont les arcs `(i, m[i])` du graphe de l'application `m`, pour tout habitant `i` du type `bint`.

Six lignes de code WhyML [Gio20, partie 4.2] suffisent pour formaliser la propriété que le triplet $(A^\sigma, <_P^\sigma, <_V^\sigma)$ est un modèle de la théorie TOTO. Les prédicats qui formalisent $<_P^\sigma$ et $<_V^\sigma$ doivent satisfaire les axiomes d'un ordre total strict. Ces démonstrations sont automatiques.

Pour tout modèle $(A, <_P, <_V)$ de TOTO, il existe une permutation σ telle que $(A, <_P, <_V)$ et $(A^\sigma, <_P^\sigma, <_V^\sigma)$ sont isomorphes [ABF20]. Construire une démonstration formelle de cette propriété est un travail conséquent et délicat, notamment en raison des lourdeurs imposées par la co-existence de deux ordres totaux. Cet objectif est laissé en perspective et le problème est simplifié en limitant A à un intervalle d'entiers et l'ordre $<_P$ à l'ordre naturel sur les entiers, comme détaillé dans la partie suivante.

4 Une permutation est un ordre total strict

Considérons les modèles $(A, <_P, <_V)$ de la théorie TOTO dans lesquels A est un intervalle d'entiers I et $<_P$ est la restriction à I de l'ordre naturel $<$ sur les entiers, notée $<_I$. Dans ces modèles, l'ordre total strict $<_V$ suffit pour décrire une permutation sur I. Pour étudier ces modèles, nous considérons une théorie de permutations composée d'un unique ordre total strict sur un intervalle d'entiers, nommée STOI (pour *Strict Total Order on one integer Interval*) [1].

Le modèle de la théorie STOI associé à la permutation σ sur l'intervalle d'entiers $[l..u]$ est le couple $([l..u], <^\sigma)$ tel que $i <^\sigma j$ si et seulement si $\sigma(i) < \sigma(j)$. La définition

```
let predicate lt_map (m: map bint bint) (i j: bint) = lt_bint m[i] m[j]
```

du prédicat et de la fonction booléenne `lt_map` formalise cette correspondance entre une application `m`, de type `(map bint bint)`, et une relation binaire, ici formalisée par une fonction booléenne de type `(bint → bint → bool)`. Le prédicat et la fonction booléenne `lt_bint`, de type `(bint → bint → bool)`, formalise l'ordre $<_{[l..u]}$. Dans cette partie toutes les fonctions sont implémentées (et certains prédicats, sous forme de fonctions booléennes), afin de pouvoir tester certains lemmes avant de chercher à les démontrer formellement.

Le code WhyML

```
val constant sigma : map bint bint
axiom sigma_bij : bijective sigma
let predicate lt_sigma (i j: bint) = lt_map sigma i j
clone export relations.TotalStrictOrder with type t = bint, predicate rel = lt_sigma
```

1. Cette terminologie de "théorie" n'est pas tout à fait correcte, car en logique mathématique le domaine des symboles relationnels et fonctionnels d'une théorie n'est pas fixé dans la théorie, mais dans ses modèles. Nous commettons néanmoins cet abus de langage par souci d'homogénéité avec les deux parties précédentes.

formalise la définition de $<^\sigma$ et la propriété que $([l..u], <^\sigma)$ est un modèle de la théorie STOI, pour toute application bijective σ sur $[l..u]$. Why3 démontre automatiquement que le prédicat `lt_sigma` satisfait les axiomes d'un ordre total strict.

Réciproquement, une permutation peut être associée à tout modèle $([l..u], <)$ de la théorie STOI, selon la propriété caractéristique suivante des modèles de cette théorie.

Proposition 1. *Pour tout ordre total strict $<$ sur l'intervalle d'entiers $[l..u]$, il existe une permutation σ sur $[l..u]$ telle que $<=<^\sigma$.*

Pour caractériser cette permutation σ, nous définissons d'abord une fonction `rank` qui associe une application sur $[l..u]$ à toute relation binaire sur $[l..u]$. Ensuite, nous justifions la proposition 1 par la conjonction des deux propriétés suivantes : (P_1) l'image (`rank` $<$) d'un ordre total strict $<$ par la fonction `rank` est une application bijective – c'est la permutation σ de la proposition 1 ; (P_2) la fonction `lt_map` est un inverse à gauche de la fonction `rank` – c'est une autre formulation de l'égalité $<=<^\sigma$ de la proposition 1.

Pour tout ordre total strict $<$ sur un ensemble fini A à n éléments, le *rang* de $a \in A$, défini par $rank(a) = card(\{b \in A \mid b < a\}) + 1$, est le nombre d'éléments de A inférieurs à a selon $<$, augmenté de 1 pour que le rang soit toujours dans l'intervalle $[1..n]$ [ABF20, page 9]. Nous généralisons cette définition à tout intervalle d'entiers $[l..u]$, par une fonction WhyML `rank` telle que `(rank lt)`$(a) = card(\{b \in [l..u] \mid$ `lt` $b\ a\}) + l$, pour toute fonction booléenne binaire `(lt : bint → bint → bool)` [Gio20, partie 5.3].

Pour tout ordre total strict `lt`, nous avons démontré formellement que la fonction `(rank lt)` est à valeurs dans l'intervalle $[l..u]$. L'ajout de trois lemmes rend cette preuve automatique avec Why3. Nous avons aussi élaboré une preuve interactive Coq pour démontrer que la fonction `(rank lt)` est injective. Pour établir qu'elle est surjective, donc bijective (propriété P_1), nous introduisons une fonction `unrank` [Gio20, partie 5.4] de telle sorte que les fonctions `(rank lt)` et `(unrank lt)` soient mutuellement inverses, au sens des deux lemmes d'inverse à gauche

```
lemma   rank_ltK: ∀ lt. totalStrictOrder lt → ∀ i. rank lt (unrank lt i) = i
lemma unrank_ltK: ∀ lt. totalStrictOrder lt → ∀ i. unrank lt (rank lt i) = i
```

où le prédicat `totalStrictOrder` spécifie que son paramètre, de type `('a → 'a → bool)`, implémente un ordre total strict. Le second lemme se démontre automatiquement avec Why3 ou interactivement avec Coq, à partir du premier lemme et de l'injectivité de `(rank lt)`, prouvée avec Coq. La démonstration du premier lemme est laissée en perspective. Ce lemme est testé avec AutoCheck [EG20], par énumération de tous les ordres totaux stricts sur l'intervalle [1..6] [Gio20, partie 5.5]. AutoCheck est un prototype d'outil de test automatique de propriétés OCaml et WhyML, par génération aléatoire et énumérative de données de test.

La propriété P_2 est formalisée par le lemme

```
lemma lt_mapK: ∀ lt. totalStrictOrder lt → lt_map (rank lt) = lt
```

qui est prouvé avec Why3, par la transformation split_vc suivie d'un appel au prouveur Z3. Ensuite, la proposition 1 est prouvée interactivement par 8 transformations Why3.

5 Travaux connexes

Dans un environnement de preuve formelle, la notion de permutation peut apparaître dans une spécification de l'existence d'une permutation entre deux structures (partie 5.1), dans des structures représentant une permutation (partie 5.2) ou dans une axiomatisation de la notion de permutation (partie 5.3). Nous détaillons ces occurrences de la notion de permutation dans

les environnements Why3 et Coq que nous utilisons, tout en positionnant nos contributions dans cette classification.

5.1 Existence d'une permutation entre deux structures

Dans la librairie standard de Coq, les prédicats `Sorting.PermutSetoid.permutation` et `Sorting.Permutation` caractérisent l'existence d'une permutation entre deux listes. Le premier est fondé sur l'égalité des contenus des listes, vus comme des multi-ensembles, pour toute égalité décidable entre leurs éléments. Le second est fondé sur la composition de transpositions.

Dans la librairie standard de Why3, les prédicats `list.Permut.permut`, `map.MapPermut.permut` et `seq.Permut.permut` caractérisent l'existence d'une permutation, respectivement entre deux listes, applications et séquences. Ils sont définis indépendamment les uns des autres et sont utilisés pour d'autres structures, par exemple dans le prédicat `array.ArrayPermut.permut` sur les tableaux mutables. Tous ces prédicats sont fondés sur l'égalité du nombre d'occurrences de chaque élément dans les deux structures.

Ces prédicats permettent par exemple de spécifier que le résultat du tri d'une structure est une permutation de la structure initiale. En complément, la librairie Coccinelle de Coq introduit une notion de permutation relative à une endorelation binaire quelconque, puis caractérise par un prédicat inductif simple l'existence d'une telle permutation entre deux listes, pour faciliter la modélisation de certains ordres pour la réécriture de termes [Con07].

Notre contexte et nos objectifs n'étant ni la vérification de tris ni la modélisation de la réécriture, mais la vérification formelle de théorèmes et de programmes de la combinatoire, nous n'apportons aucune contribution à ces caractérisations.

5.2 Représentations formelles d'une permutation

Dans un outil de vérification de théorèmes et de programmes, une *représentation formelle* d'une permutation doit permettre de spécifier des propriétés universelles et existentielles des permutations, voire d'implémenter des programmes qui agissent sur des permutations. Dans un environnement typé comme Why3 ou Coq, chaque représentation peut être définie par un type de données, éventuellement obtenu par sous-typage d'un type prédéfini, à l'aide d'un prédicat caractéristique.

Par exemple, le prédicat `math_permut` de [Con07] et le type `permut` de [DG18] formalisent la définition d'une permutation sur $[0..n-1]$ comme une endofonction bijective, à partir du type des fonctions sur les entiers naturels de Coq. En WhyML, les études de cas `inverse_in_place`[2] et [GDL19] définissent un prédicat qui caractérise les tableaux d'entiers (type prédéfini `array int`) de longueur n dont les valeurs sont dans l'intervalle $[0..n-1]$ de leurs indices et deux à deux distinctes. Quoique moins concret, le modèle d'application bijective de la présente étude – composé du type `(map bint bint)` et du prédicat caractéristique `bijective` des applications injectives et surjectives, défini dans la partie 2 – est aussi une représentation formelle possible des permutations en Why3.

5.3 Axiomatisations formelles de la notion de permutation

Dans une logique adaptée, toute caractérisation ou représentation formelle d'une permutation, dont celles des parties 5.1 et 5.2, peut inspirer une théorie de permutations, voire plusieurs.

2. http://toccata.lri.fr/gallery/inverse_in_place.en.html

Cependant, à notre connaissance, aucun travail antérieur n'aborde cette question selon la perspective des théories et de leurs modèles. De plus, aucun code public Why3 ne formalise de théorie de permutations comme nous le faisons, uniquement avec un ou deux symboles relationnels binaires.

6 Conclusion et perspectives

Nous avons présenté une formalisation originale de théories de permutations dans le langage WhyML de la plateforme de vérification déductive Why3. Les deux premières théories sont issues d'un article de combinatoire, qui étudie ensuite leur expressivité. La troisième théorie est une restriction calculatoire de la deuxième théorie, créée directement en WhyML par l'auteur. Des propriétés des modèles de ces théories sont énoncées et démontrées formellement, souvent de manière automatique, sinon de manière interactive, par des transformations Why3 ou des tactiques Coq. Un outil de test automatique est appliqué à une propriété dont la démonstration est moins immédiate.

Le code produit est proche du texte mathématique. Peu d'éléments techniques l'alourdissent, en partie grâce à la plasticité du langage WhyML. Si le code est parfois plus long que le texte mathématique, c'est surtout parce qu'il explicite des détails laissés implicites dans ce dernier, un phénomène bien connu en formalisation des mathématiques. En levant toute ambiguïté sur chaque notion, le code peut même faciliter la compréhension du texte par un non-spécialiste. Les premières propositions, considérées comme élémentaires dans le texte mathématique, ont été démontrées facilement avec Why3, automatiquement ou après une courte interaction. Les propositions suivantes, plus profondes, requièrent des efforts de spécification et de preuve formelle plus conséquents. Globalement, cette expérimentation encourage l'étude directe de certains sujets combinatoires de manière formelle, en recourant au test automatique quand la preuve devient trop chronophage, afin de ne pas trop freiner l'élan créatif.

Ce travail a également des retombées positives sur la plateforme Why3. C'est d'abord un exemple supplémentaire d'utilisation de fonctionnalités de Why3, comme son mécanisme de clonage de modules, sa notion de type avec invariant et son langage de preuve interactive. L'étude a aussi donné lieu à des développements plus généraux que son propre cadre, ré-utilisables dans d'autres applications. Enfin, l'étude contribue à l'amélioration de Why3, en identifiant des limitations et en suggérant des extensions de sa librairie standard.

Ces premiers résultats nous encouragent à poursuivre ce travail de recherche dans diverses directions. Nous envisageons d'abord de formaliser une plus grande proportion du texte mathématique de référence. Nous souhaitons aussi étendre l'étude à d'autres points de vue sur la notion de permutation, dont ceux suggérés dans la partie 5 ou ceux des produits de cycles disjoints et des nombres factoriels, aussi appelés codes de permutations [GDL19, partie 4]. Il s'agit non seulement de formaliser plusieurs théories et représentations de permutations, mais également de les relier entre elles, d'étudier leurs combinaisons et de démontrer formellement leurs propriétés. Ceci fait, il serait utile d'évaluer l'impact de chaque théorie et représentation sur le degré d'automaticité qu'elle procure pour démontrer des théorèmes ou des propriétés de programmes sur les permutations.

Une perspective majeure est d'élargir l'étude aux théories de deux permutations, voire d'ensembles de permutations, en démontrant le cas échéant leur structure de groupe. Jusqu'ici, nous n'avons considéré que des théories pour **une** permutation. Or, il est fréquemment utile de considérer conjointement une bijection et son inverse. Nous avons conçu et utilisé des variantes de modules de la librairie standard de Why3 qui axiomatisent qu'une fonction f est un inverse à gauche d'une fonction g, en déduisent que g est injective et f est surjective, puis que les

deux fonctions sont bijectives et mutuellement inverses si de plus g est un inverse à gauche de f [Gio20, partie 2.4]. Nous proposons de considérer ce code comme une théorie de deux fonctions mutuellement inverses, nommée TOTIF (T*heory* O*f* T*wo* I*nverse* F*unctions*), puis de spécialiser TOTIF en une théorie de **deux** permutations inverses.

Remerciements. L'équipe de développement de Why3 et les relecteurs anonymes sont chaleureusement remerciés pour leurs précieux conseils. Ce travail a été soutenu par l'EIPHI Graduate School (contrat ANR-17-EURE-0002).

Références

[ABF20] Michael Albert, Mathilde Bouvel, and Valentin Féray. Two first-order logics of permutations. *Journal of Combinatorial Theory, Series A*, 171 :105158, April 2020. https://arxiv.org/abs/1808.05459v2.

[BFM+20] François Bobot, Jean-Christophe Filliâtre, Claude Marché, Guillaume Melquiond, and Andrei Paskevich. *The Why3 Platform*, 2020. http://why3.lri.fr/manual.pdf.

[Con07] Evelyne Contejean. Modeling Permutations in Coq for Coccinelle. In *Rewriting, Computation and Proof : Essays Dedicated to Jean-Pierre Jouannaud on the Occasion of His 60th Birthday*, volume 4600 of *Lecture Notes in Computer Science*, pages 259–269. Springer, 2007.

[Coq20] The Coq Proof Assistant, 2020. http://coq.inria.fr.

[DG18] Catherine Dubois and Alain Giorgetti. Tests and proofs for custom data generators. *Formal Aspects of Computing*, 30 :659–684, Jul 2018.

[EG20] Clotilde Erard and Alain Giorgetti. AutoCheck, 2020. https://alaingiorgetti.github.io/autocheck/.

[Fil20] Jean-Christophe Filliâtre. Simpler proofs with decentralized invariants. *Journal of Logical and Algebraic Methods in Programming*, March 2020. To appear. See http://why3.lri.fr/spdi/.

[FP20] Jean-Christophe Filliâtre and Andrei Paskevich. Abstraction and genericity in Why3. In *9th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, volume 12476 of *Lecture Notes in Computer Science*. Springer, October 2020. http://why3.lri.fr/isola-2020/.

[GDL19] Alain Giorgetti, Catherine Dubois, and Rémi Lazarini. Combinatoire formelle avec Why3 et Coq. In Nicolas Magaud and Zaynah Dargaye, editors, *Journées Francophones des Langages Applicatifs. JFLA 2019*, pages 139–154, 2019. https://hal.inria.fr/hal-01985195.

[Gio20] Alain Giorgetti. Formalisation et vérification de théories de permutations. Rapport de recherche RR-1715, UBFC (Université de Bourgogne Franche-Comté) and FEMTO-ST, December 2020. https://hal.archives-ouvertes.fr/hal-03033416.

[GM21] Alain Giorgetti and Nicolas Magaud. Combinatoire certifiée. In Mireille Blay-Fornarino, Catherine Dubois, and Pierre-Etienne Moreau, editors, *GdR Génie de la Programmation et du Logiciel, Défis 2030*, pages 61–65, 2021. https://hal.archives-ouvertes.fr/hal-03097727.