Master d'informatique parcours informatique et interactions
À l'université côte d'azur

RAPPORT DE TER

Typing the higher-order polyadic μ-calculus

Camille Bonnin

supervisée par Mme. Cinzia Di Giusto et M. Etienne Lozes

Premier semestre de l'année 2020-2021

Introduction

Le μ-calcul a tout d'abord été introduit par Scott et de Bakker en 1969 [4], mais a été étendu vers sa forme actuelle la plus utilisée de nos jours par Kozen en 1983 [2]. Par la suite, plusieurs extensions au μ-calcul ont été introduites dont le μ-calcul polyadique, introduit par Andersen en 1994 [1], qui permet de manipuler des tuples d'états et le μ-calcul d'ordre supérieur, introduit par Viswanathan et Viswanathan en 2004 [5], qui permet les opérations sur des formules et non plus uniquement sur des états simples. Le μ-calcul polyadique d'ordre supérieur introduit par Lange, Lozes et Guzmán en 2014 [3] est une fusion de ces deux extensions.

Le μ-calcul polyadique d'ordre supérieur est une logique permettant d'exprimer une grande variété de formules. Il permet par exemple d'exprimer des relations sur les automates. Le μ-calcul polyadique d'ordre supérieur possède aussi un opérateur de point fixe ce qui permet de formuler des propriétés représentant des suites infinies d'états.

Cette logique permet d'exprimer un grand nombre de relations d'équivalence entre des processus différentes. Ce dernier point est particulièrement intéressant au niveau du model-checking car le même papier qui introduit cette logique [3], introduit également une méthode générale pour vérifier que deux processus sont équivalents valable pour toutes les relations d'équivalence qui peuvent être exprimées par le μ-calcul polyadique d'ordre supérieur. La nouveauté de cette méthode est qu'au lieu de définir une formule $F^P(Q)$ qui représente tous les processus Q équivalents à P, on a directement une formule $F(P, Q)$ qui est valable pour tous les P. Cela rends la méthode plus facilement adaptable quels que soient P et Q et marche même sans expression de P.

Le μ-calcul polyadique d'ordre supérieur est donc une logique qui mérite d'être étudiée. Mais avant d'utiliser une formule écrite avec cette logique, il faut savoir si cette formule est bien formée ou non d'où l'intérêt de savoir la typer et donc de ce TER. L'aspect polyadique du μ-calcul n'apportant pas de difficulté supplémentaire au niveau du typage (mais pas au niveau de l'interprétation), ce TER se focalise sur les formules du mu-calcul d'ordre supérieur sans l'aspect polyadique.

1 Le μ-calcul d'ordre supérieur

Le μ-calcul d'ordre supérieur est une logique modale qui permet le passage à l'ordre supérieur. Une logique d'ordre supérieur est une logique qui permet de décrire des formules dans lesquelles les variables sont d'autres formules ou des fonctions par opposition aux logiques du premier ordre qui décrivent uniquement les formules dont les variables sont des états seuls. Une logique modale quand à elle est une logique pour laquelle la véracité d'une formule est spécifiée par le biais de modalités. Par exemple, les logiques temporelles qui sont des logiques modales possèdent les modalités suivantes : "toujours" (noté $\Box$), "un jour" (noté $\Diamond$) et "jamais" (noté $\neg\Diamond$). Les modalités du μ-calcul (et du μ-calcul d'ordre supérieur) sont les modalités classiques de la logique modale : "nécessaire" (qui ne peut pas ne pas être vrai, noté $\Box$), "contingent" (qui peut être faux, noté $\neg\Box$), "possible" (qui peut être vrai, noté $\Diamond$) et "impossible" (qui ne peut pas ne pas être faux, noté $\neg\Diamond$) auxquelles on a rajouter les opérateurs de point fixe (notés μ et υ).

Le μ-calcul (et par extension le μ-calcul d'ordre supérieur) est une logique modale qui permet de décrire un grand nombre de formules. On peut extraire du μ-calcul des logiques temporelles comme CTL* (qui contient elle-même CTL et LTL, deux logiques très utilisées en model-checking).

1.1 Syntaxe

Pour pouvoir énoncer la syntaxe du μ-calcul d'ordre supérieur, commençons par poser quelques notations et définir un système de transition d'états.

Définition 1 (Système de transition d'états labellisé)**.** Un système de transition d'états labellisé est un triplet (Pr, Act, $\rightarrow$) où Pr est un ensemble d'états, Act un ensemble d'actions et $\rightarrow$: Pr $\times$ Act $\rightarrow$ Pr une relation de transition. Un système de transition d'états labellisé décrit les passages d'un éléments de Pr à un autre par le biais d'un élément de Act.

Supposons dans toute la suite que nous avons un système de transition d'états labellisé (Pr, Act, $\rightarrow$) où Pr $= \{P, Q, \ldots\}$ est un ensemble d'états, Act $= \{a, b, \ldots\}$ un ensemble d'actions et $\rightarrow$ la relation de transition entre deux états P et Q par l'action a (notée $P \xrightarrow{a} Q$).

Introduisons aussi les notations suivantes : les variables (ensemble Var) sont notées $X, Y, Z, \ldots$ ou $F, G, H, \ldots$, les formules sont notées Φ, Ψ, ... et le type d'une variable ou d'une formule peut-être soit un type de base (noté $\bullet$), soit un type flèche (noté $\tau_1^\nu \rightarrow \tau_2$ où ν est une variance (décrite en 1.3) et τ_1, τ_2 sont deux types quelconques).

On peut maintenant énoncer la syntaxe du μ-calcul polyadique d'ordre supérieur :

Définition 2 (Syntaxe du μ-calcul d'ordre supérieur)**.** La syntaxe du μ-calcul d'ordre supérieur est la suivante :

$$\Phi, \Psi ::= \top \mid \Phi \wedge \Psi \mid \neg\Phi \mid \langle a \rangle \Phi \mid X \mid \mu X : \tau.\Phi \mid \lambda X^\nu : \tau.\Phi \mid \Phi\Psi$$

Notation 1. Les cinq opérateurs suivants peuvent s'exprimer à partir des opérateurs précédents :

$$\begin{aligned}
\Phi \vee \Psi &::= \neg(\neg\Phi \wedge \neg\Psi) \\
\upsilon X.\Phi &::= \neg\mu X.\neg\Phi[\neg X/X] \text{ (remplacement de } X \text{ par } \neg X) \\
[a]\Phi &::= \neg\langle a \rangle\neg\Phi \\
\Phi \Rightarrow \Psi &::= \neg\Phi \vee \Psi \\
\Phi \Leftrightarrow \Psi &::= (\Phi \Rightarrow \Psi) \wedge (\Psi \Rightarrow \Phi)
\end{aligned}$$

1.2 Sens des opérateurs et exemples

Pour des raisons de simplification, nous n'allons pas donner la sémantique des opérateurs du μ-calcul d'ordre supérieur car elle n'est pas utile pour typer les formules, mais nous allons donner ici une intuition informelle du sens des opérateurs introduits en 1.1 :

- $\top$ ("top") : constante, représente n'importe quel état ;
- $\Phi \wedge \Psi$ ("conjonction") : est vraie si Φ et Ψ sont toutes les deux vraies ;
- $\neg\Phi$ ("négation") : est vraie si Φ est fausse ;
- $\langle a \rangle \Phi$ ("diamant") : représente le possible, est vraie dans un état P de Pr s'il existe au moins un état Q de Pr pour lequel Φ est vraie et que le système de transitions d'états labellisés (Pr, Act, $\rightarrow$) comporte une transition $P \xrightarrow{a} Q$;
- X : variable, représente la variable X, si X est une formule, possède la même valeur de vérité que X ;
- $\mu X : \tau.\Phi$ ("plus petit point fixe") : représente le plus petit point fixe, est vraie dans tout état faisant partie de l'intersection de tous les ensembles d'états Ens pour lesquels lorsque l'on remplace X par Ens dans Φ, alors les états où Φ est vraie font partie de Ens. Cet opérateur est utilisé pour représenter la propriété de vivacité ("quelque chose de bien va éventuellement arriver") ;
- $\lambda X^\nu : \tau.\Phi$ ("lambda abstraction") : permet de rajouter un niveau d'abstraction à Φ en "transformant" Φ en une fonction pour laquelle X est une variable dont la variance doit être ν et le type τ ;
- $\Phi\Psi$ ("application") : le résultat de Φ pour laquelle on a remplacé les variables introduites par un opérateur λ par Ψ.

Le sens des cinq opérateurs introduits dans la notation 1. est le suivant :

- $\Phi \vee \Psi$ ("disjonction") : est vraie si Φ est vraie ou si Ψ est vraie ;

- $\upsilon X.\Phi$ ("plus grand point fixe") : représente le plus petit point fixe, est vraie dans tout état faisant partie de l'union de tous les ensembles d'états Ens pour lesquels lorsque l'on remplace X par Ens dans Φ, alors Φ est vraie dans tous les états de Ens. Cet opérateur est utilisé pour représenter la propriété de sureté ("quelque chose de mal n'arrive jamais") ;
- $[a]\Phi$ ("boîte") : représente le nécessaire, est vraie dans un état P de Pr si Φ est vraie dans tous les états Q de Pr pour lesquels le système de transitions d'états labellisés (Pr, Act, $\rightarrow$) comporte une transition $P \xrightarrow{a} Q$;
- $\Phi \Rightarrow \Psi$ ("implication") : est vraie si Ψ est vraie si Φ est vraie (on ne peut pas avoir Ψ est fausse et Φ vraie en même temps) ;
- $\Phi \Leftrightarrow \Psi$ ("équivalence") : est vraie si Ψ est vraie si et seulement si Φ est vraie (Ψ et Φ ont la même valeur de vérité).

Exemple 1. A faire

1.3 Variances

L'algorithme de model-checking pour le μ-calcul polyadique d'ordre 1 possède une complexité en EXPTIME, mais dans le cas où les variances de toutes les variables possède une propriété particulière, alors il existe un algorithme de model-checking qui résout le problème avec une complexité de PSPACE [3], d'où l'intérêt de regarder les variances des variables dans les formules lors de la phase de typage.

1.3.1 Intuition

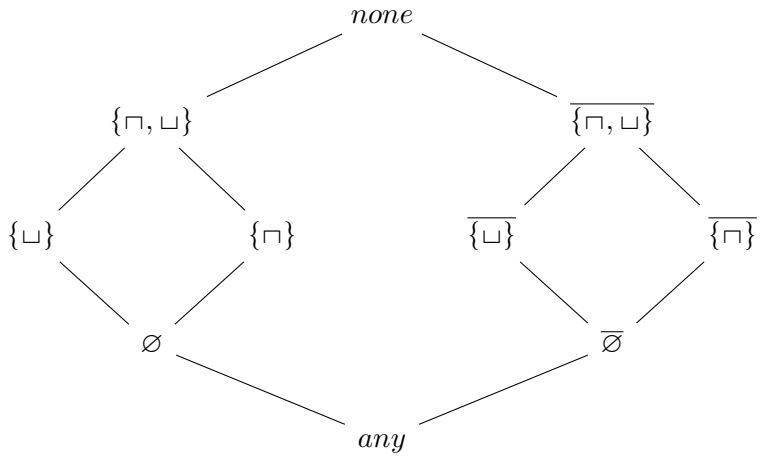

FIGURE 1 – Treillis des variances.

On peut voir la variance comme la monotonie de la fonction représentée par la variable ou la formule (rappelons que les formules comme les variables peuvent avoir le type $\rightarrow$). On peut voir une variance *none* comme une fonction constante et une variable *any* comme une fonction quelconque. Les variables avec les autres variances sont des fonctions monotones, croissantes pour les variables avec les variances $\{\sqcap, \sqcup\}$, $\{\sqcap\}$, $\{\sqcup\}$ et $\varnothing$, décroissantes pour les variables avec les variances duales à ces dernières.

1.3.2 Définitions et propriétés

A finir

Définition 3 ($\sqcap$-additivité et $\sqcup$-additivité)**.** Une fonction $f : \mathrm{A} \rightarrow \mathrm{B}$ est $\sqcap$-additive (resp $\sqcup$-additive) si $\forall x, y \in \mathrm{A}^2$, $f(x \sqcap y) = f(x) \sqcap f(y)$ (resp $f(x \sqcup y) = f(x) \sqcup f(y)$).

Notation 2 ($\preccurlyeq$)**.** On utilise le symbole $\preccurlyeq$ pour représenter la relation d'ordre partielle donnée par le treillis des variances (figure 1.) dans lequel *any* est le plus petit élément et *none* le plus grand. Soit deux variances ν et ν', on note $\nu \preccurlyeq \nu'$ si ν et ν' sont comparables et que ν est plus petite que ν' selon le treillis.

Définition (sémantique) :

1.3.3 Exemples

Exemples de calculs et du formules valides et non valides à cause de la variance (cf papier).

1.4 Règles de typage

1.4.1 Définitions

Gamma + Delta.

1.4.2 Enoncé

Enoncer les nouvelles règles du papier + commentaires (explications). Pour règle du μ, commenter que l'on doit avoir une variance qui correspond à une variation croissante pour avoir un point fixe.

1.4.3 Exemples

Donner des exemples (reprendre les précédents + prendre des exemples des formules pour les résultats), faire les arbres + commenter.

2 (1^{ere} ?) Implémentation des règles de typage

On a enlevé la règle {i ⟵ j} substitution. (explications).

2.1 Travail récupéré (titre à revoir)

Diviser en plusieurs parties.
Ce qui a été fait par Thomas (syntaxe, "desugar", calculs sur les variances).
Expliquer les choix pour ce qui n'a pas été gardé ou changé.

2.2 Fonction de typage

2.2.1 Implémentation des environnements de typage

Représentation des environnements de typage complets et incomplets + des assignements.

2.2.2 La fonction de typage

Pseudo-code de la fonction "typing". 1 section par cas ? (à voir en fonction de la taille).
Expliquer le code en le liant aux règles de typage réécrites.

2.3 Fonctions annexes

Calcul de "inter", Gamma "rond" variance, etc : pseudo-code + explications.

3 (Piste(s) d')Améliorations

3.1 Les limites de l'implémentation actuelle

Le Gamma.

3.2 (Piste(s) d')Améliorations

A voir.

4 Résultats

4.1 Tableau des résultats

Faire un tableau avec les résultats de tests.

4.2 Commentaires

Commenter les résultats, peut-être faire des arbres de typage pour certains.

Conclusion

Bref résumé + avis personnel + piste(s) d'approfondissement (adapter avec la partie 3).

Références

[1] Andersen, H.R. : A polyadic modal μ-calculus (1994)

[2] Kozen, D. : Results on the propositional μ-calculus. Theoretical computer science **27**(3), 333–354 (1983)

[3] Lange, M., Lozes, E., Guzmán, M.V. : Model-checking process equivalences. Theoretical Computer Science **560**, 326–347 (2014)

[4] Scott, D., de Bakker, J.W. : A theory of programs. Unpublished manuscript, IBM, Vienna (1969)

[5] Viswanathan, M., Viswanathan, R. : A higher order modal fixed point logic. In : International Conference on Concurrency Theory. pp. 512–528. Springer (2004)