UNIVERSITÉ CÔTE D'AZUR

MASTER INFORMATIQUE

Rapport de TER

Typing the higher-order polyadic μ-calculus

Réalisé par
Camille Bonnin

Encadré par
Mme. Cinzia Di Giusto
M. Etienne Lozes

Master Informatique
à Université Côte d'Azur
Premier semestre de l'année 2020-2021

Table des matières

1 Introduction

Le μ-calcul a tout d'abord été introduit par Scott et de Bakker en 1969 (Scott et de Bakker, 1969), mais a été étendu vers sa forme actuelle la plus utilisée de nos jours par Kozen en 1983 (Dexter, 1983). Par la suite, plusieurs extensions au μ-calcul ont été introduites dont le μ-calcul polyadique, introduit par Andersen en 1994 (Andersen, 1994), qui permet de manipuler des tuples d'états et le μ-calcul d'ordre supérieur, introduit par Viswanathan et Viswanathan en 2004 (Viswanathan et Viswanathan, 2004), 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 (Lange *et al.*, 2014) 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 (Lange *et al.*, 2014), 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 μ-calcul d'ordre supérieur sans l'aspect polyadique.

2 Le μ-calcul d'ordre supérieur

2.1 Propriétés et opérateurs

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 ν).

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).

Avant d'énoncer la syntaxe du μ-calcul d'ordre supérieur, introduisons 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 les actions sont notées $a, b, \ldots$. Nous utilisons une version annotée du μ-calcul d'ordre supérieur d'où la présence d'indications de typage dans la syntaxe. Le type d'une variable ou d'une formule, décrit en 2.3, peut-être soit un type de base (noté $\bullet$), soit un type $\rightarrow$ (noté $\tau_1^v \rightarrow \tau_2$ où v est une variance, décrite en 2.2, et τ_1, τ_2 sont deux types quelconques).

Définition 1 (Syntaxe du μ-calcul d'ordre supérieur)**.** La syntaxe du μ-calcul d'ordre supérieur est la suivante :

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

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

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

Pour pouvoir donner le sens des opérateurs du μ-calcul d'ordre supérieur, il nous faut définir un système de transition d'états.

Définition 2 (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 $P \xrightarrow{a} Q$).

Pour des raisons de simplification, nous n'allons pas donner la sémantique formelle 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 définition 2.1 :

- $\top$ ("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 $P \xrightarrow{a} Q$;
- $\mu X : \tau.\Phi$ ("plus petit point fixe") : représente le plus petit point fixe, le plus petit élément el de type τ pour lequel $\Phi(el) = el$. Cet opérateur est utilisé pour représenter la propriété de vivacité ("quelque chose de bien va éventuellement arriver") ;
- $\lambda X^v : \tau.\Phi$ ("lambda abstraction") : permet de représenter les fonctions ;
- $\Phi\Psi$ ("application") : le résultat de l'application de Φ (fonction) à Ψ.

La définition suivante est une notion qui sera utile pour le typage des formule.

Définition 3 (Variable libre)**.** On dit qu'une variable X d'une formule Φ du μ-calcul d'ordre supérieur est libre si elle n'a pas été introduite par un opérateur μ ou λ.

Exemple 1. A faire

2.2 Variances

On peut voir la variance comme la monotonie de la fonction représentée par la variable ou la formule (rappelons que les formules sont d'ordre supérieur). 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 $\underline{\{\sqcap, \sqcup\}}$, $\underline{\{\sqcap\}}$, $\underline{\{\sqcup\}}$ et $\varnothing$, décroissantes pour les variables avec les variances duales de ces dernières ($\overline{\{\sqcap, \sqcup\}}$, $\overline{\{\sqcap\}}$, $\overline{\{\sqcup\}}$ et $\overline{\varnothing}$).

Définition 4 (Valeurs des variances)**.** Une variance v peut prendre ici une des dix valeurs suivantes : *none*, *any*, $\underline{\{\sqcap, \sqcup\}}$, $\underline{\{\sqcap\}}$, $\underline{\{\sqcup\}}$, $\varnothing$, $\overline{\{\sqcap, \sqcup\}}$, $\overline{\{\sqcap\}}$, $\overline{\{\sqcup\}}$ ou $\overline{\varnothing}$.

L'algorithme de model-checking pour le μ-calcul polyadique d'ordre 1 possède une complexité en EXPTIME, mais dans le cas où toutes les variance présentes dans tous les types τ présents dans les opérateurs $\mu X : \tau.\Phi$ remplissent une certaine propriété, alors il existe un algorithme de model-checking qui résout le problème avec une complexité de PSPACE (Lange *et al.*, 2014), d'où l'intérêt de regarder les variances des variables dans les formules lors de la phase de typage.

Une propriété importante des variance est l'additivité.

Définition 5 ($\sqcap$-additivité et $\sqcup$-additivité)**.** Une fonction $f : \mathrm{A} \to \mathrm{B}$ est $\sqcap$-additive (resp $\sqcup$-additive) si $\forall x, y \in \mathrm{A}^2$, $f(x \sqcap y) = f(x) \sqcap f(y)$ (resp $f(x \sqcup y) = f(x) \sqcup f(y)$).

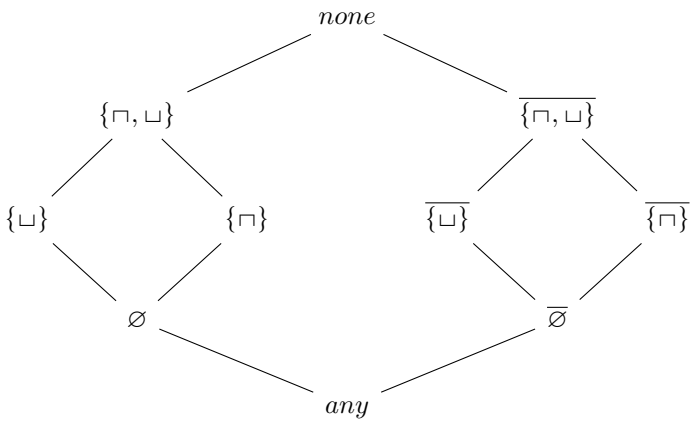

FIGURE 1 – Treillis des variances.

Définition 6 ($\preccurlyeq$)**.** 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 v et v', on note $v \preccurlyeq v'$ si v et v' sont comparables et que v est plus petite que (ou égale à) v' selon le treillis.

Notation 2 ($v_1 \wedge v_2$)**.** On note $v_1 \wedge v_2 = v$ la plus grande variance v telle que $v \preccurlyeq v_1$ et $v \preccurlyeq v_1$. Par exemple, $\{\sqcap, \sqcup\} \wedge \{\sqcap\} = \{\sqcap\}$, $\{\sqcap\} \wedge \{\sqcup\} = \varnothing$ et $\overline{\{\sqcap, \sqcup\}} \wedge \{\sqcap\} = any$.

La variance possède également plusieurs propriétés.

Définition 7 (Propriétés de la variance)**.** Soit v une variance et $f : \mathrm{A} \to \mathrm{B}$ une fonction :

- si f est monotone, $\sqcap$-additive et $\sqcup$-additive, alors elle possède la variance $v = \{\sqcap, \sqcup\}$;
- si f est monotone et $\sqcap$-additive, alors elle possède la variance $v = \{\sqcap\}$;
- si f est monotone et $\sqcup$-additive, alors elle possède la variance $v = \{\sqcup\}$;
- si f est monotone, alors elle possède la variance $v = \varnothing$;
- si la fonction $\neg f$ définie par $(\neg f)(x) = \neg f(x)$ possède la variance v, alors f possède la variance $\overline{v}$;
- si f est constante, alors elle possède la variance *none* ;
- f possède toujours la variance *any* quelque soit sa définition.

Les variances possèdent aussi les deux opérations suivantes, le dual et la composition.

Définition 8 (Dual d'une variance)**.** Le dual d'une variance v, noté $dual(v)$ est défini de la façons suivante :

$$dual(v) = \begin{cases} v & \text{si } v \in \{any, none, \varnothing, \{\sqcap, \sqcup\}\} \\ \{\sqcup\} & \text{si } v = \{\sqcap\} \\ \{\sqcap\} & \text{si } v = \{\sqcup\} \\ \overline{dual(v')} & \text{si } v = \overline{v'} \end{cases}$$

Définition 9 (Composition de deux variances)**.** La composition de deux variances v_1 et v_2, notée $v_1 \circ v_2$ est définie de la façons suivante :

$$v_1 \circ v_2 = \begin{cases} none & \text{si } none \in \{v_1, v_2\} \\ any & \text{si } none \notin \{v_1, v_2\} \text{ et } any \in \{v_1, v_2\} \\ v_1 \cap v_2 & \text{si } v_1, v_2 \subseteq \{\sqcap, \sqcup\} \\ \overline{\overline{v_1} \cap v_2} & \text{si } \overline{v_1}, v_2 \subseteq \{\sqcap, \sqcup\} \\ \overline{dual(v_1) \cap \overline{v_2}} & \text{si } v_1, \overline{v_2} \subseteq \{\sqcap, \sqcup\} \\ dual(\overline{v_1}) \cap \overline{v_2} & \text{si } \overline{v_1}, \overline{v_2} \subseteq \{\sqcap, \sqcup\} \end{cases}$$

On peut faire les trois observations suivantes :

- Si une fonction f possède une variance v_2 et qu'il existe une variance v_1 telle que $v_1 \preccurlyeq v_2$, alors f possède aussi la variance f.
- Si une fonction $f : \mathrm{A} \to \mathrm{B}$ possède une variance v, alors la fonction $dual(f) : \mathrm{A} \to \mathrm{B}$ telle que $(dual(f))(x) = \neg f(\neg x)$ possède la variance $dual(v)$.
- Soient deux fonctions $f_1 : \mathrm{B} \to \mathrm{C}$ et $f_2 : \mathrm{A} \to \mathrm{B}$, si f_1 possède la variance v_1 et f_2 possède la variance v_2, alors $f_1 \circ f_2$ possède la variance $v_1 \circ v_2$.

Exemple 2. Exemples de calculs et du formules valides et non valides à cause de la variance (cf papier).

L'algorithme de model-checking pour le μ-calcul polyadique d'ordre 1 possède une complexité en EXPTIME, mais dans le cas où toutes les variances présentes dans tous les types τ présents dans les opérateurs $\mu X : \tau.\Phi$ sont additives, alors il existe un algorithme de model-checking qui résout le problème avec une complexité de PSPACE (Lange *et al.*, 2014), d'où l'intérêt de regarder les variances des variables dans les formules lors de la phase de typage.

2.3 Règles de typage

Maintenant que l'on a vu la syntaxe du μ-calcul d'ordre supérieur et que l'on a introduit les variances, il ne nous reste plus que à définir un système de typage pour pouvoir énoncer les règles de typage du μ-calcul d'ordre supérieur qui ont été utilisées pour faire l'algorithme de typage décrit en section 3. En effet, afin de pouvoir utiliser la récursivité, il a fallu réécrire les règles de typage présentes dans l'article de Lange, Lozes et Guzmán (Lange *et al.*, 2014) et introduire une nouvelle notion, l'environnement de typage sans variances.

Définition 10 (Système de typage)**.** Un système de typage est un système logique qui comprend des règles qui attribuent un type aux variables et aux formules d'un langage. Ici on attribuera aussi une variance aux variables.

Définition 11 (Jugement de typage)**.** Un jugement de typage, noté $\Delta : \Phi \vdash (\Gamma, \tau)$ est un triplet où Δ est un environnement de typage sans variances, Φ une formule du *mu*-calcul d'ordre supérieur, Γ un environnement de typage et τ un type. Cela signifie que dans un contexte qui remplit toutes les variables libres de Φ présentes dans Δ ont les types qui leur sont attribués dans Δ, alors Φ a le type τ et l'on se trouve dans l'environnement de typage Γ (les variables libres de Φ ont les types et variances décrits dans Γ).

Dans la suite on utilisera la notation $type(\Delta, \Phi) = (\Gamma, \tau)$ pour $\Delta : \Phi \vdash (\Gamma, \tau)$ afin de faciliter l'implémentation.

On va maintenant définir les notions d'environnement de typage sans variances, de type et d'environnement de typage. On a introduit la notion d'environnement de typage sans variances pour pouvoir utiliser des règles de typage récursives. En effet, comme on accepte les formules d'ordre supérieur, si l'on ne connaît pas le type des variables libres à l'avance, on n'a pas de cas de base autre que $\top$ pour pouvoir effectuer la récurrence alors qu'il faudrait aussi avoir les variables étant donné qu'elles ne peuvent pas non plus se décomposer en d'autres formules.

Définition 12 (Environnement de typage sans variances Δ)**.** Un environnement de typage sans variances, noté Δ est un ensemble d'assignements de typage sans variance de la forme $X : \tau$ où X est une variable et τ son type (qui peut être de la forme $\bullet$ ou $\tau_1^{v_1} \to \tau_2$).

Notation 3 ($vars(\Delta)$)**.** Soit un environnement de typage sans variances $\Delta = \{X_1 : \tau_1, \ldots, X_n : \tau_n\}$, on note $vars(\Delta) = \{X_1, \ldots, X_n\}$ l'ensemble des variables présentes dans Δ.

Définition 13 (Type)**.** Le type d'une variable ou d'une formule peut prendre deux formes. Il peut être soit un type de base (comme un booléen, cela dépend du langage mais n'a pas d'importance ici), on le note alors $\bullet$, soit un type $\to$ représentant une fonction, on le note alors $\tau_1^v \to \tau_2$ avec v, une variance et τ_1 et τ_2, deux types quelconques.

Définition 14 (Environnement de typage Γ)**.** Un environnement de typage, noté Γ est un ensemble d'assignements de typage de la forme $X^v : \tau$ où X est une variable, v sa variance et τ son type (qui peut être de la forme $\bullet$ ou $\tau_1^{v_1} \to \tau_2$).
On note $\emptyset$ l'environnement de typage vide.

Notation 4 ($vars(\Gamma)$)**.** Soit un environnement de typage $\Gamma = \{X_1^{v_1} : \tau_1, \ldots, X_n^{v_n} : \tau_n\}$, on note $vars(\Gamma) = \{X_1, \ldots, X_n\}$ l'ensemble des variables présentes dans Γ.

Remarque 1. Un environnement de typage sans variances est un environnement de typage sans les variances des variables. Les indications de variance à l'intérieur des types $\to$ restent néanmoins toujours présentes.

Dans les règles de typage, on va avoir besoin de pouvoir composer une variance avec les variances de toutes les variables présentes dans un environnement de typage et de pouvoir combiner deux environnements de typage d'où les deux définitions suivantes.

Définition 15 (Composition variance environnement de typage)**.** Soit un environnement de typage $\Gamma = \{X_1^{v_1} : \tau_1, \ldots, X_n^{v_n} : \tau_n\}$ et une variance v, on défini la composition de v avec Γ, notée $v \circ \Gamma$ de la manière suivante :

$$v \circ \Gamma := \Gamma = \{X_1^{v \circ v_1} : \tau_1, \ldots, X_n^{v \circ v_n} : \tau_n\}.$$

Définition 16 (Combinaison de deux environnements de typage)**.** Soient deux environnements de typage, Γ_1 et Γ_2, v, v_1, v_2 des variances et τ, τ_1, τ_2 des types, on défini la combinaison de Γ_1 et Γ_2, notée $\Gamma_1 \wedge \Gamma_2$ comme étant un environnement de typage construit de la manière suivante :

- si $(X^v : \tau) \in \Gamma_1$ et $X \notin vars(\Gamma_2)$, alors $(X^v : \tau) \in \Gamma_1 \wedge \Gamma_2$;
- si $X \notin vars(\Gamma_1)$ et $(X^v : \tau) \in \Gamma_2$, alors $(X^v : \tau) \in \Gamma_1 \wedge \Gamma_2$;
- si $(X^{v_1} : \tau_1) \in \Gamma_1$ et $(X^{v_2} : \tau_2) \in \Gamma_2$, alors :
  - si $\tau_1 \neq \tau_2$, alors $\Gamma_1 \wedge \Gamma_2$ n'est pas défini ;
  - sinon, si $\tau_1 = \tau_2$, alors $(X^{v_1 \wedge v_2} : \tau_1) \in \Gamma_1 \wedge \Gamma_2$.

On peut maintenant énoncer les règles de typage du du μ-calcul d'ordre supérieur.

Définition 17 (Règles de typage)**.** Les règles de typage du μ-calcul d'ordre supérieur sont les suivantes :

$$\frac{}{type(\Delta, \top) = (\emptyset, \bullet)}(\text{T-TOP}) \qquad \frac{type(\Delta, \Phi) = (\Gamma_1, \bullet) \qquad type(\Delta, \Psi) = (\Gamma_2, \bullet)}{type(\Delta, \Phi \wedge \Psi) = (\Gamma_1 \wedge \Gamma_2, \bullet)}(\text{T-ET})$$

$$\frac{type(\Delta, \Phi) = (\Gamma, \tau)}{type(\Delta, \neg\Phi) = (\overline{\{\sqcap, \sqcup\}} \circ \Gamma, \tau)}(\text{T-NEG}) \qquad \frac{type(\Delta, \Phi) = (\Gamma, \bullet)}{type(\Delta, \langle a \rangle \Phi) = (\{\sqcup\} \circ \Gamma, \bullet)}(\text{T-DIAMANT})$$

$$\frac{(X : \tau) \in \Delta}{type(\Delta, X) = (X^{\{\sqcap, \sqcup\}} : \tau, \tau)}(\text{T-VAR})$$

$$\frac{type(\Delta \cup \{X : \tau\}, \Phi) = (\Gamma, \sigma) \qquad \sigma = \tau \qquad \Gamma = \Gamma' \cup \{X^v : \sigma\} \qquad v \geq \varnothing}{type(\Delta, \mu X : \tau \,.\, \Phi) = (\Gamma', \tau)}(\text{T-MU})$$

$$\frac{type(\Delta \cup \{X : \sigma\}, \Phi) = (\Gamma, \tau) \qquad \Gamma = \Gamma' \cup \{X^v : \sigma'\}}{type(\Delta, \lambda X : \sigma \,.\, \Phi) = (\Gamma', \sigma'^v \to \tau)}(\text{T-LAMDDA})$$

$$\frac{type(\Delta, \Phi) = (\Gamma_1, \sigma^v \to \tau) \qquad type(\Delta, \Psi) = (\Gamma_2, \sigma)}{type(\Delta, \Phi\Psi) = (\Gamma_1 \wedge v \circ \Gamma_2, \tau)}(\text{T-APP})$$

Avant d'expliquer plus en détail ces règles de typage, on peut remarquer que l'on doit connaître à l'avance le type des variables libres de la formule avec leurs informations de variances. Connaître le type de ces variables est possible avec une version annotée du μ-calcul d'ordre supérieur mais connaître les informations de variance de ces types est une hypothèse plus compliquée à mettre en pratique.

Donnons maintenant quelques explications sur ces règles de typage .

- (T-TOP) : axiome.
- (T-ET) : Φ et Γ doivent être des types $\bullet$ car la conjonction d'une fonction avec un type de base ou de deux fonctions n'a aucun sens. On obtient un environnement de typage qui combine les environnements de typage de Φ et Γ et donc qui interdit de se retrouver avec une même variable possédant deux types différents et qui adapte si besoin la variance d'une variable à la plus grande variance commune.
- (T-NEG) : Le type de Φ ne change pas avec le passage à la négation, mais les variables de l'environnement de typage de Φ voient toutes leur variance passer de v à $\overline{v}$ après le passage à la négation. Que le type ne change pas semble naturel, pour la variance, c'est en accord avec la 5^e propriété de la définition 7 de la section 2.2.
- (T-DIAMANT) : Le type de Φ ne change pas avec l'intervention du $\diamond$, mais les variables de l'environnement de typage de Φ voient toutes leur variance passer de v à $\sqcup \circ v$ après l'intervention du $\diamond$. Ce dernier point met à jour les propriétés d'additivité des variances des variables de Φ en supprimant la $\sqcap$-additivité éventuelle de ces dernière. On peut voir cette règle comme un cas de (T-APP) où $\langle a \rangle$ serait une fonction appliquée à Φ avec une variance de $\{\sqcup\}$.
- (T-VAR) : On conserve le type de X donné par Δ et on assigne à X la variance $\{\sqcap, \sqcup\}$ quelque soit ce type car il s'agit de la plus grande variance que X est autorisée à prendre selon les règles de (Lange *et al.*, 2014), par récursivité, les autres règles permettront de réduire cette variance si besoin, on évite ainsi les erreurs dues au fait d'avoir choisi une variance plus petite que ce qui était possible. On rappelle que le but est de trouver la plus grande variance possible pour chaque variable libre.
- (T-MU) : On impose le fait que la variance v de X doit être respecter $v \geqslant \varnothing$. En effet, si la fonction décrite par X est croissante (ou constante), on est sûr de bien obtenir un point fixe et c'est ce que vérifie le fait d'avoir une variance $v \geqslant \varnothing$. Que le type du résultat soit le type de X est une application de la définition du point fixe ($x = f(x)$).
- (T-LAMBDA) : On rajoute temporairement X dans Δ avec le type indiqué dans la formule pour trouver sa variance pour pouvoir l'indiquer dans le type du résultat. On en a aussi besoin de la rajouter pour typer Phi. Une fois cela fait, on retire X de l'environnement de typage du résultat.
- (T-APP) : On vérifie que le type de Ψ est bien celui de l'antécédent de Φ, et on donne au résultat de l'application le type de l'image de Φ. Au niveau des variances, on compose la variance de l'antécédent de Ψ avec les variances de l'environnement de typage de Φ pour être en accord avec les informations de typage dans le type de l'antécédent de Φ avant de combiner les environnements de typage de Φ et Ψ pour obtenir celui du résultat.

Exemple 3. Dans cet exemple, on va noter $\tau = \bullet^{\varnothing} \to \bullet$.

On va chercher à typer la formule $(\mu F : \tau \,.\, \lambda X^{\overline{\varnothing}} : \bullet \,.\, \langle a \rangle (Y \wedge (F \neg (FX))))) \; [b]Y$ à partir de $\Delta = \{\{Y : \bullet\}\}$.

On va aussi noter les environnements de typage sans variables que nous serons amenés à utiliser de la manière suivante :
$\Delta_{F,X} = \{\{F : \tau\}, \{X : \bullet\}\}$;
$\Delta_{Y,F,X} = \{\{Y : \bullet\}, \{F : \tau\}, \{X : \bullet\}\}$;
$\Delta_{Y,F} = \{\{Y : \bullet\}, \{F : \tau\}\}$.

On note (1) l'arbre suivant :

$$\dfrac{\dfrac{}{type(\Delta_{F,X}, F) = (\{\{F^{\{\sqcap,\sqcup\}} : \tau\}\}, \tau)}(\text{T-VAR}) \quad \dfrac{}{type(\Delta_{F,X}, X) = (\{\{X^{\{\sqcap,\sqcup\}} : \bullet\}\}, \bullet)}(\text{T-VAR})}{type(\Delta_{F,X}, FX) = (\{\{F^{\varnothing} : \tau\}, \{X^{\overline{\varnothing}} : \bullet\}\}, \bullet)}(\text{T-APP})$$
$$\dfrac{}{type(\Delta_{F,X}, \neg(FX)) = (\{\{F^{\overline{\varnothing}} : \tau\}, \{X^{\varnothing} : \bullet\}\}, \bullet)}(\text{T-NEG})$$

On note (2) l'arbre suivant :

$$\dfrac{\dfrac{}{type(\Delta, Y) = (\{\{Y^{\{\sqcap,\sqcup\}} : \bullet\}\}, \bullet)}(\text{T-VAR}) \quad \dfrac{\dfrac{}{type(\Delta_{F,X}, F) = (\{\{F^{\{\sqcap,\sqcup\}} : \tau\}\}, \tau)}(\text{T-VAR}) \quad (1)}{type(\Delta_{F,X}, F\neg(FX)) = (\{\{F^{\varnothing} : \tau\}, \{X^{\overline{\varnothing}} : \bullet\}\}, \bullet)}(\text{T-APP})}{type(\Delta_{Y,F,X}, Y \wedge (F\neg(FX))) = (\{\{Y^{\{\sqcap,\sqcup\}} : \bullet\}, \{F^{\varnothing} : \tau\}, \{X^{\overline{\varnothing}} : \bullet\}\}, \bullet)}(\text{T-ET})$$
$$\dfrac{type(\Delta_{Y,F,X}, \langle a\rangle(Y \wedge (F\neg(FX)))) = (\{\{Y^{\{\sqcup\}} : \bullet\}, \{F^{\varnothing} : \tau\}, \{X^{\overline{\varnothing}} : \bullet\}\}, \bullet)}{type(\Delta_{Y,F}, \lambda X^{\overline{\varnothing}} : \bullet \,.\, \langle a\rangle(Y \wedge (F\neg(FX)))) = (\{\{Y^{\{\sqcup\}} : \bullet\}, \{F^{\varnothing} : \tau\}\}, \tau)}(\text{T-LAMBDA})$$
$$\dfrac{}{type(\Delta, \mu F : \tau \,.\, \lambda X^{\overline{\varnothing}} : \bullet \,.\, \langle a\rangle(Y \wedge (F\neg(FX)))) = (\{\{Y^{\{\sqcup\}} : \bullet\}\}, \tau)}(\text{T-DIAMANT})(\text{T-MU})$$

On appelle (3) l'arbre suivant :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{type(\Delta, Y) = (\{\{Y^{\{\sqcap,\sqcup\}} : \bullet\}\}, \bullet)}(\text{T-VAR})}{type(\Delta, \neg Y) = (\{\{Y^{\overline{\{\sqcap,\sqcup\}}} : \bullet\}\}, \bullet)}(\text{T-NEG})}{type(\Delta, \langle b\rangle(\neg Y)) = (\{\{Y^{\overline{\{\sqcap\}}} : \bullet\}\}, \bullet)}(\text{T-DIAMANT})}{type(\Delta, [b]Y) = (\{\{Y^{\{\sqcap\}} : \bullet\}\}, \bullet)}(\text{T-NEG})$$

On peut maintenant construire l'arbre de la formule entière :

$$\dfrac{(2) \qquad (3)}{type(\Delta, (\mu F : \tau \,.\, \lambda X^{v} : \bullet \,.\, \langle a\rangle(Y \wedge (F\neg(FX))))) \; [b]Y = (\{\{Y^{any} : \bullet\}\}, \bullet)}(\text{T-APP})$$

On rappelle pour la dernière ligne de l'arbre (3) que $[b]Y ::= \neg\langle b\rangle\neg Y$. On remarque aussi que l'on rajoute F et X à Δ pour les étages du dessus lorsque l'on les rencontre dans les opérateurs μ et λ de la formule bien qu'elles n'étaient pas présentes en-dessous.

Dans la dernière ligne de l'arbre, on se retrouve dans les hypothèses (arbres (2) et (3)) avec Y qui a deux variances différentes, $\{\sqcup\}$ et $\{\sqcap\}$, pour trouver la variance finale de Y, on commence par calculer $\overline{\varnothing} \circ \{\sqcap\} = \overline{\varnothing}$ pour suivre la règle T-APP avant de chercher la plus grande variance v telle que $v \preccurlyeq \{\sqcup\}$ et $v \preccurlyeq \overline{\varnothing}$, selon le treillis des variances (figure 1), cette variable est *any* d'où le résultat.

3 Implémentation des règles de typage

L'implémentation de ces règles de typage a été réalisée en OCaml (on peut la trouver avec ce lien https://github.com/CamilleBonnin/TER-S3-Typing-the-higher-order-polyadic-mu-calculus.git).

Cette implémentation reprend les parties suivantes du code écrites par Thomas Portet (https://github.com/ThomasPortet/higher-order-mu-calculus) :

- le parseur (modifié au niveau du λ pour autoriser la variable X introduite dans cette règle à avoir un type quelconque et non plus uniquement $\bullet$) ;
- la fonction d'affichage des formules ;
- les fonctions permettant les calculs entre les variances ($dual$, $\circ$, $\wedge$) ;
- la représentation des types et leur affichage.

Les algorithmes des fonctions de calcul des variances étant des applications directes des formules de calcul (la seule difficulté étant pour le calcul de $\wedge$ de tester les variances du sommet vers le bas du treillis), leurs algorithmes ne seront pas donnés ici.

On a choisi ici de représenter les environnements de typage (classiques comme sans variances) par des listes associatives, les fonctions de recherche, d'ajout ou de suppression d'un élément dans un environnement de typage sont donc réalisées par des fonctions classiques voir souvent déjà inclues dans le langage et ne serons pas décrites.

Les opérations $v \circ \Gamma$ et $\Gamma_1 \wedge \Gamma_2$ sont réalisées par les algorithmes 1 et 2. A chaque fois, tout le traitement est fait par la fonction utilitaire, la fonction principale servant uniquement à initialiser les paramètres qui jouent le rôle d'accumulateurs dans la fonction utilitaire afin de faciliter l'emploi de la fonction de calcul par la suite. Dans les deux cas, on parcourt un environnement de typage en entier (Γ et Γ_1) et on traite chaque élément l'un après l'autre. On utilise ici la récurrence car OCaml est un langage qui s'y prête bien mais on aurait aussi pu utiliser une boucle `for`.

La fonction de typage est donnée par l'algorithme 3. Les objectifs de cette fonction sont : vérifier que la formule est bien conforme à la syntaxe et aux règles de typage du μ-calcul d'ordre supérieur et de trouver la plus grande variance possible pour chaque variable libre de la formule f. Cette fonction applique directement les règles de typage de la section 2.3.

Une autre fonction qui ne sert pas directement au typage mais qui permet d'identifier les variables libres d'une formule f et compte leur nombre d'apparitions dans la formule est donnée par l'algorithme 4. C'est utile d'avoir la liste des variables libres pour rédiger le Δ nécessaire à la fonction de typage et pour vérifier que cette dernière a bien attribuer une variance à toutes les variables libres.

Algorithme 1 Réalise l'opération $v \circ \Gamma$.

```
VarRondEnvTypage(variable v, environnement de typage Γ) :
      (environnement de typage) :
            Util-VarRondEnvTypage(v, Γ, ∅);

Util-VarRondEnvTypage(variable v, environnement de typage Γ, environnement de
    typage Résultat) :
        (environnement de typage) :
si (Γ = ∅) :
    retourner Résultat;
sinon :
    (X^{v'} : τ) ← Γ[1];
    Util-VarRondEnvTypage(v, enleverIndice(1, Γ), {X^{v∘v'} : τ} ∪ Résultat);
```

Algorithme 2 Réalise l'opération $\Gamma_1 \wedge \Gamma_2$.

```
EnvTypageInterEnvTypage(environnement de typage Γ1, environnement de typage Γ2) :
      (environnement de typage) :
            Util-EnvTypageInterEnvTypage(Γ1, Γ2, ∅);

Util-EnvTypageInterEnvTypage(environnement de typage Γ1, environnement de typage Γ2,
     environnement de typage Résultat) :
        (environnement de typage) :
si (Γ1 = ∅) :      //Cas où X est dans Γ2 mais pas dans Γ1.
    retourner Résultat ∪ Γ2;
sinon :
    (X^v : τ) ← Γ1[1];
    si (X ∉ vars(Γ2)) :      //Cas où X est dans Γ1 mais pas dans Γ2.
        Util-VarRondEnvTypage(enleverIndice(1, Γ1), Γ2, {X^v : τ} ∪ Résultat);
    sinon :      //Cas où X est à la fois dans Γ2 et dans Γ1.
        (v', τ') ← (variance de X dans Γ2, type de X dans Γ2);
        si (τ = τ') :
                Util-VarRondEnvTypage(enleverIndice(1, Γ1), Γ2\{X^v' : τ'}, {X^v∧v' : τ} ∪ Résultat)
                    ;
        sinon :
                afficher "Erreur : types incompatibles pour X";
```

Algorithme 3 Effectue le typage de la formule f.

```
TYPAGE(formule f, environnement de typage sans variances Δ) :
      (environnement de typage, type) :
cas f avec :
        ⊤ :  retourner (∅, •);

    Φ ∧ Ψ :  (Γ1, τ1) ← TYPAGE(Φ, Δ);
             si (τ1 = •) :
                 (Γ2, τ2) ← TYPAGE(Ψ, Δ);
                  si (τ2 = •) :
                      retourner (Γ1 ∧ Γ2, •);
                 sinon :
                      afficher "Erreur : Ψ n'a pas le type • ";
             sinon :
                 afficher "Erreur : Φ n'a pas le type • ";

       ¬Φ :  (Γ, τ) ← TYPAGE(Φ, Δ);
             retourner ({⊓, ⊔} ∘ Γ, τ);

     ⟨a⟩Φ :  (Γ, τ) ← TYPAGE(Φ, Δ);
             si (τ = •) :
                 retourner ({⊔} ∘ Γ, •);
             sinon :
                 afficher "Erreur : Φ n'a pas le type • ";

        X :  si (X ∈ vars(Δ)) :
                 τ ← type de X dans Δ;
                 retourner (X^{⊓,⊔} : τ, τ);
             sinon :
                 afficher "Erreur : X n'est pas dans Δ";

  μX : τ.Φ :  (Γ, σ) ← TYPAGE(Φ, Δ ∪ {X : τ});
             si (τ ≠ σ) :
                 afficher "Erreur : τ et σ incompatibles";
             sinon :
                 si (X ∉ vars(Γ)) :
                     afficher "Erreur : X n'est pas dans Γ";
                 sinon :
                     (v, σ2) ← (variance de X dans Γ, type de X dans Γ);
                     si (τ ≠ σ2) :
                         afficher "Erreur : τ et σ2 incompatibles";
                     sinon :
                         si (v ⋡ ∅) :
                             afficher "Erreur : v n'est pas une des variances suivantes : none, {⊓, ⊔}, {⊓}, {⊔} ou ∅";
                         sinon :
                             retourner (Γ \ {X^v : τ}, τ);

λX^v : τ.Φ :  (Γ, σ) ← TYPAGE(Φ, Δ ∪ {X : τ});
             si (X ∉ vars(Γ)) :
                 afficher "Erreur : X n'est pas dans Γ";
             sinon :
                 (v, σ2) ← (variance de X dans Γ, type de X dans Γ);
                 retourner (Γ \ {X^v : σ2}, σ1^v → σ);

       ΦΨ :  (Γ1, τ1) ← TYPAGE(Φ, Δ);
             (Γ2, τ2) ← TYPAGE(Ψ, Δ);
             cas τ1 avec :
                      • :  afficher "Erreur : Φ n'est pas une fonction";
                 σ1^v → σ2 :  si (σ1 ≠ τ2) :
                                  afficher"Erreur : incompatibilité entre les types de Φ et de Ψ";
                              sinon :
                                  retourner (Γ1 ∧ v ∘ Γ2, σ2);
             fin cas

fin cas
```

Algorithme 4 Liste les variables libres de f et compte leur nombre d'occurances dans f.

```
VarLibres(formule f) :
        liste de (variable, entier) :
                Util-VarLibres(f, ∅, ∅);

Util-VarLibres(formule f, liste de (variable, entier) Résultat, liste de variables
    AEnlever) :
        liste de (variable, entier) :
cas f avec :
            ⊤ :  retourner Résultat;

        Φ ∧ Ψ :  ConcatListes(Util-VarLibres(Φ, Résultat, AEnlever), Util-VarLibres(Ψ, Résultat, AEnlever));

           ¬Φ :  Util-VarLibres(Φ, Résultat, AEnlever);

         ⟨a⟩Φ :  Util-VarLibres(Φ, Résultat, AEnlever);

            X :  retourner AjouterListe(X, Résultat);

      μX : τ.Φ :  EnleverListe(Util-VarLibres(Φ, Résultat, {X} ∪ AEnlever), {X} ∪ AEnlever));

     λX^v : τ.Φ :  EnleverListe(Util-VarLibres(Φ, Résultat, {X} ∪ AEnlever), {X} ∪ AEnlever));

           ΦΨ :  ConcatListes(Util-VarLibres(Φ, Résultat, AEnlever), Util-VarLibres(Ψ, Résultat, AEnlever));

fin cas

ConcatListes(liste de (variable, entier) l1, liste de (variable, entier) l2) :
        liste de (variable, entier) :
si (l1 = []) :
    retourner l2;
sinon :
    (X, n) ← l1[1];
    si (X ∉ vars(l2)) :      //où vars(l) est l'ensemble des variables présentes dans l
        cons((X, n), ConcatListes(enleverIndice(1, l1), l2,));
    sinon :
        i ← indice de X dans l2;
        m ← l2[i];
        cons((X, n + m), ConcatListes(enleverIndice(1, l1), enleverIndice(i, l2)));

AjouterListe(variable X, liste de (variable, entier) l) :
        liste de (variable, entier) :
si (X ∉ vars(l)) :          //où vars(l) est l'ensemble des variables présentes dans l
    retourner cons((X, 1), l);
sinon :
    i ← indice de X dans l;
    n ← l[i];
    cons((X, n + 1), enleverIndice(i, l));

EnleverListe(liste de (variable, entier) l, liste de variables AEnlever) :
        liste de (variable, entier) :
si (AEnlever = []) :
    retourner l;
sinon :
    i ← indice de X dans l;
    EnleverListe(enleverIndice(i, l), enleverIndice(1, AEnlever));
```

4 Résultats

Faire un tableau avec les résultats de tests.
Commenter les résultats, peut-être faire des arbres de typage pour certains.

5 Conclusion

Dans ce TER nous nous somme basé sur un article, (Lange *et al.*, 2014) pour regarder une logique particulière, le μ-calcul d'ordre supérieur et découvrir la notion de variance.

Le but de ce TER était de typer le μ-calcul polyadique d'ordre supérieur, mais en étudiant les règles de typage déjà existantes, nous avons remarqué que l'aspect polyadique de cette logique n'avait que peut d'impact sur le typage et nous avons donc par la suite choisi d'ignorer cet aspect dans la suite de ce TER. Une continuité possible à ce TER serait donc de réintégrer cet aspect polyadique à fin cette fois-ci de réellement typer le μ-calcul polyadique d'ordre supérieur et non plus uniquement le μ-calcul d'ordre supérieur, néanmoins cela ne semble pas être l'amélioration la plus importante qui puisse être apportée à ce travail.

En effet, dans la fonction de typage actuelle, afin de permettre l'utilisation de la récurrence, on a du imposer à l'utilisateur de connaître certaines informations qu'il pourrait ignorer en pratique. On lui demande en effet de connaître le type des variables libres, le type des variables liées étant obligatoirement spécifié par la syntaxe. De fait, on ne réalise pas vraiment un typage du μ-calcul d'ordre supérieur mais plutôt une vérification de la compatibilité des types des variables qui permet de donner un type à la formule finale si aucune erreur n'est détectée.

Néanmoins, le plus gênant avec la version actuelle ce n'est pas le fait de devoir connaître le type des variables libres, mais de devoir connaître les annotations de variances de ces derniers. De fait, une première amélioration à apporter à ce TER serait dans un premier temps, avant de chercher à supprimer complètement les informations de type des variables libres, uniquement de chercher à supprimer les informations de variance de ces types. Pour ce faire, il semble difficile de se passer de programmation par contraintes.

Enfin, ce TER aura été l'occasion de mieux comprendre des notions vu en cours de théorie des types et de se rendre compte que l'on peut toujours affiner le typage d'une formule, par exemple ici avec des variances pour en déduire des propriétés sur cette formule comme par exemple la complexité de l'algorithme de model-checking capable de la traiter. Cela a été l'occasion aussi de se rendre compte que trouver un point fixe est une tâche qui peut se révéler compliquée en général.

Pour terminer, d'un point de vue plus pratique, ce TER aura aussi été l'occasion de découvrir un nouveau langage, OCaml et de nouvelles fonctionnalités de LaTeX .

Bibliographie

Andersen, H. R. (1994). A polyadic modal μ-calculus.

Dexter, D. K. (1983). Results on the propositional μ-calculus. *Theoretical computer science*, 27(3):333–354.

Lange, M., Martin, Lozes, E. et Vargas, M. V. G. M. (2014). Model-checking process equivalences. *Theoretical Computer Science*, 560:326–347.

Scott, D. et de Bakker, J. W. (1969). A theory of programs. *Unpublished manuscript, IBM, Vienna*.

Viswanathan, M. et Viswanathan, R. (2004). A higher order modal fixed point logic. pages 512–528. Springer.