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 (Kozen, 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 différentes entre des processus. 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 et cette méthode est 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. La méthode est ainsi plus facilement adaptable quels que soient P et Q et fonctionne 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 il en va autrement de l'interprétation), ce TER se focalise sur les formules du μ-calcul d'ordre supérieur sans l'aspect polyadique.

Dans l'article de Lange, Lozes et Guzmán (Lange *et al.*, 2014), les variables ne possèdent pas uniquement un type mais aussi une variance (décrite en section 2.2). Le problème de l'inférence de types consiste donc à trouver le type de la formule générale mais aussi la variance et le type de toutes les variables libres de cette formule. Dans ce TER, nous allons réduire le problème. Au lieu de nous intéresser à l'inférence de types, nous allons regarder l'inférence de variances. Nous connaîtrons ainsi déjà le type des variables et nous devrons uniquement trouver leur variance ainsi que le type de la formule générale.

Pour aborder ce problème, nous commencerons par donner une présentation du μ-calcul d'ordre supérieur ainsi que de la notion de variance. Ensuite nous définirons plus en détail le problème de l'inférence de variances avant de présenter l'algorithme qui permet de le résoudre et son implémentation. Nous donnerons ensuite un tableau avec les résultats des tests réalisés avant de conclure.

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 rajouté 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 (ou processus), Act un ensemble d'actions et $\rightarrow \subseteq \text{Pr} \times \text{Act} \rightarrow \text{Pr}$ une relation de transition. Un système de transition d'états labellisé décrit les passages d'un élément 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 $\text{Pr} = \{P, Q, \ldots\}$ est un ensemble d'états, $\text{Act} = \{a, b, \ldots\}$ 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 transition d'états labellisé (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 ultimement 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 notion de variable libre sera utile pour le typage des formules. 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. On pose v et v' deux variances, les formules du μ-calcul d'ordre supérieur suivantes sont syntaxiquement correctes et ont les variables libres et liées suivantes :

- $(\lambda X^v : \bullet \,.\, X) \wedge X$, dans cette formule, il y a deux variables qui portent le même nom, X, la variable X dans la sous-formule entre parenthèses est liée par l'opérateur λ tandis que l'autre variable X apparaissant à la droite de l'opérateur $\wedge$ est libre. On pourrait réécrire cette formule de cette manière : $(\lambda Y^v : \bullet \,.\, Y) \wedge X$ où Y serait liée et X libre.
- $\mu X : \bullet \,.\, X$ possède une seule variable liée (par μ), X et aucune variable libre.
- $\mu X : \bullet \,.\, \neg X$ possède une seule variable liée (par μ), X et aucune variable libre.
- $\mu X : \bullet \,.\, ((\lambda Y^v : \bullet \,.\, \neg Y)X)$ possède deux variables liées, X (par μ) et Y (par λ) et aucune variable libre.
- $\mu F : (\bullet^{v'} \rightarrow \bullet) \,.\, \lambda X^v : \bullet \,.\, F(\neg(FX))$ possède deux variables liées, F (par μ) et X (par λ) et aucune variable libre.
- $(\mu F : (\bullet^{v'} \rightarrow \bullet) \,.\, \lambda X^v : \bullet \,.\, \langle a\rangle(Y \wedge (F\neg(FX)))) \; [b]Y$ possède deux variables liées, F (liée par μ) et X (liée par λ) et une variable libre, Y.
- $\mu X : \bullet \,.\, [a]X$ possède une seule variable liée, X (par μ). Cette formule représente le fait que tout chemin composé de a-transitions est fini.

Toutes ces formules sont syntaxiquement correctes, mais nous verrons par la suite que certaines d'entre-elles ne sont pas bien formées à cause notamment des variances.

2.2 Variances

Avant de parler de variance, rappelons la définition d'un treillis.

Définition 3 (Treillis)**.** Un treillis est un triplet (E, $\sqcap$, $\sqcup$) où E est un ensemble partiellement ordonné tel que tout couple d'éléments de E possède une borne supérieure et une borne inférieure. $\sqcap$ et $\sqcup$ sont les deux lois internes de E définies de la manière suivante : soient e et e' deux éléments de E, $e \sqcap e' = inf(e,\ e')$ et $e \sqcup e' = sup(e,\ e')$.

La variance d'une fonction $f : \mathrm{A} \to \mathrm{B}$, où A et B sont deux treillis, généralise et étend la notion de monotonie. Par exemple, une fonction de variance *none* est une fonction constante (à la fois croissante et décroissante) et une fonction avec une variance *any* est une fonction quelconque. Les fonctions avec les autres variances sont des fonctions monotones, croissantes pour les variances $\{\sqcap, \sqcup\}$, $\{\sqcap\}$, $\{\sqcup\}$ et $\varnothing$, décroissantes pour 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*, $\{\sqcap, \sqcup\}$, $\{\sqcap\}$, $\{\sqcup\}$, $\varnothing$, $\overline{\{\sqcap, \sqcup\}}$, $\overline{\{\sqcap\}}$, $\overline{\{\sqcup\}}$ ou $\overline{\varnothing}$.

Une propriété importante caractérisée par les variances est l'additivité.

Définition 5 ($\sqcap$-additivité et $\sqcup$-additivité)**.** Soient (A, $\sqcap$, $\sqcup$) et (B, $\sqcap$, $\sqcup$) deux treillis. 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 $f(x \sqcup y) = f(x) \sqcup f(y)$).

Énonçons maintenant la définition de variance d'une fonction.

Définition 6 (Variance d'une fonction)**.** Soient (A, $\sqcap$, $\sqcup$) et (B, $\sqcap$, $\sqcup$) deux treillis et $f : \mathrm{A} \to \mathrm{B}$ une fonction :

- si f est monotone, $\sqcap$-additive et $\sqcup$-additive, alors elle possède la variance $\{\sqcap, \sqcup\}$;
- si f est monotone et $\sqcap$-additive, alors elle possède la variance $\{\sqcap\}$;
- si f est monotone et $\sqcup$-additive, alors elle possède la variance $\{\sqcup\}$;
- si f est monotone, alors elle possède la variance $\varnothing$;
- 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.

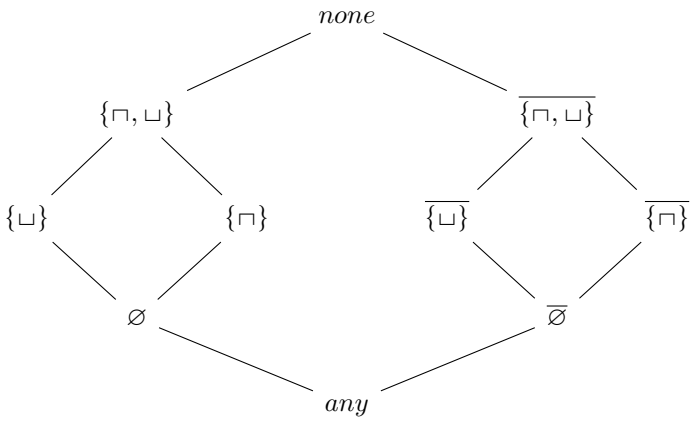

FIGURE 1 – Treillis des variances.

Définition 7 ($\preccurlyeq$)**.** 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. Soient 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 $v \preccurlyeq v_2$. Par exemple, $\{\sqcap, \sqcup\} \wedge \{\sqcap\} = \{\sqcap\}$, $\{\sqcap\} \wedge \{\sqcup\} = \varnothing$ et $\overline{\{\sqcap, \sqcup\}} \wedge \{\sqcap\} = any$.

Après avoir défini la variance d'une fonction, définissons maintenant la variance d'une variable.

Définition 8 (Variance d'une variable)**.** Soient A_i, $i \in \{1, \ldots, n\}$, et B des treillis et $f : \mathrm{A}_1 \times \ldots \times \mathrm{A}_n \to \mathrm{B}$ une fonction de n variables. Pour chaque $\mathbf{a} = (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n)$, on peut définir une fonction $f_{\mathbf{a}} : \mathrm{A}_i \to \mathrm{B}$ telle que $f_{\mathbf{a}}(x) = (a_1, \ldots, a_{i-1}, x, a_{i+1}, \ldots, a_n)$. Soit $v_{\mathbf{a}}$ la variance de $f_{\mathbf{a}}$, on dit que la variable x_i de f a la variance $v = \bigwedge\{v_{\mathbf{a}} \mid \mathbf{a} \in \mathrm{A}_1 \times \ldots \times \mathrm{A}_{i-1} \times \mathrm{A}_{i+1} \times \ldots \times \mathrm{A}_n\}$.

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

Définition 9 (Dual d'une variance)**.** Le dual d'une variance v, noté $dual(v)$ est défini de la façon 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 10 (Composition de deux variances)**.** La composition de deux variances v_1 et v_2, notée $v_1 \circ v_2$ est définie de la façon 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 v_1.
- 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 $v_1 \circ v_2$.

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 Système de typage

Maintenant que l'on a vu la syntaxe du μ-calcul d'ordre supérieur et présenté les variances, il nous reste encore à décrire le système de typage du μ-calcul d'ordre supérieur.

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 (Type)**.** Le type d'une variable ou d'une formule peut prendre deux formes. Il peut être soit un type de base (ici il s'agit du type des prédicats, 2^{Pr}), on le note alors $\bullet$, soit un type $\rightarrow$ représentant une fonction, on le note alors $\tau_1^v \rightarrow \tau_2$ avec v, une variance, et τ_1 et τ_2, deux types quelconques.

Définition 12 (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 $\tau_1^{v_1} \rightarrow \tau_2$).
On note $\emptyset$ l'environnement de typage vide.

Notation 3 ($vars(\Gamma)$)**.** 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.

Définition 13 (Jugement de typage)**.** Un jugement de typage, noté $\Gamma : \Phi \vdash \tau$ est un triplet où Γ est un environnement de typage, Φ une formule du μ-calcul d'ordre supérieur et τ un type. Cela signifie que lorsque toutes les variables libres de Φ ont le type et la variance qui leur sont attribués dans Γ, alors Φ a le type τ. $\Gamma : \Phi \vdash \tau$ est dérivable à partir de règles de typage.

Les règles de typage qui permettent de dériver $\Gamma : \Phi \vdash \tau$ sont présentées dans l'article de Lange, Lozes et Guzmán (Lange *et al.*, 2014). Ces règles permettent de dériver des jugements de typage avec des formules du μ-calcul polyadique d'ordre supérieur mais on peut les appliquer sur des jugements de typage avec des formules du μ-calcul d'ordre supérieur en ignorant les opérateurs polyadiques.

Ces règles de typage permettent également de dire que certaines formules ne sont pas correctes à cause des variances, notamment la règle du μ qui indique que la variance v de la variable liée par l'opérateur μ doit respecter $v \geqslant \varnothing$.

Exemple 2. En appliquant les règles de typage de l'article de Lange, Lozes et Guzmán (Lange *et al.*, 2014) sur les formules de l'exemple 1, on obtient ce qui suit en terme de validité :

- $(\lambda X^v : \bullet \,.\, X) \wedge X$: non valide.
- $\mu X : \bullet \,.\, X$: valide.
- $\mu X : \bullet \,.\, \neg X$: non valide.
- $\mu X : \bullet \,.\, ((\lambda Y^v : \bullet \,.\, \neg Y)X)$: non valide.
- $\mu F : (\bullet^{v'} \rightarrow \bullet) \,.\, \lambda X^v : \bullet \,.\, F(\neg(FX))$: valide.
- $(\mu F : (\bullet^{v'} \rightarrow \bullet) \,.\, \lambda X^v : \bullet \,.\, \langle a \rangle (Y \wedge (F \neg (FX)))))\ [b]Y$: valide.
- $\mu X : \bullet \,.\, [a]X$: valide.

3 Règles d'inférence de variances

Maintenant que l'on a fini d'exposer le contexte, on peut se concentrer sur le problème traité par ce TER, l'inférence de variances. L'inférence de variances est le problème qui consiste, étant donné un environnement de typage sans variance et une formule du μ-calcul d'ordre supérieur, à compléter cet environnement de typage avec les variances des variables et à donner le type de la formule. Le jugement de typage ainsi obtenu doit alors être dérivable à partir des règles de typage de l'article de Lange, Lozes et Guzmán (Lange *et al.*, 2014).

Pour résoudre ce problème, nous allons utiliser un algorithme d'inférence décrit en section 4. Nous allons dans cette section définir une fonction récursive $type(.)$ qui représente cet algorithme et est décrite par des règles d'inférence.

Définition 14 (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 4 ($vars(\Delta)$)**.** 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.

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.

Définition 15 (Fonction d'inférence de variances)**.** Soient Δ un environnement de typage sans variances, Φ une formule du μ-calcul d'ordre supérieur, Γ un environnement de typage et τ un type. La fonction d'inférence $type(\Delta, \Phi) = (\Gamma, \tau)$ est une fonction qui prend en entrée Δ et Φ et retourne Γ qui est en fait Δ complété avec les variances des variables et τ, le type de Φ sous les hypothèses de Γ.

$type(.)$ est une fonction récursive définie par des règles d'inférence de variances. Dans ces règles, 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 16 (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 définit 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 17 (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 définit 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, c'est-à-dire 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 d'inférence de variances du μ-calcul d'ordre supérieur qui composent la fonction $type(.)$.

Définition 18 (Règles de typage)**.** Les règles d'inférence de variances du μ-calcul d'ordre supérieur sont les suivantes :

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

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

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

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

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

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

Avant d'expliquer plus en détail ces règles d'inférence de variances, 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 à propos de 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 $\overline{v}$. Que le type ne change pas semble naturel, pour la variance, c'est en accord avec la 5^e propriété de la définition 6 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 $\sqcup \circ v$. Ce dernier point met à jour les propriétés d'additivité des variables de Φ en supprimant la $\sqcap$-additivité éventuelle de ces dernières. 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 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 afin de vérifier qu'elle est compatible avec les informations indiquées par la syntaxe pour X. On a aussi besoin de la rajouter pouvoir typer Φ. Une fois cela fait, on retire X de l'environnement de typage du résultat. La variance indiquée par le type $\rightarrow$ du résultat est celle indiquée par la syntaxe pour éviter d'imposer plus de conditions sur X que nécessaire.

- (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, nous allons appliquer les règles ci-dessus aux formules de l'exemple 1 pour vérifier que l'on retrouve bien les validités de l'exemple 2.

∗ Commençons par la formule $(\lambda X^{\varnothing} : \bullet \ . \ X) \wedge X$ avec $\Delta = \{\{X : \bullet\}\}$. On va noter $\Delta' = \{\{X : \bullet\}\}$ l'environnement de typage sans variable créé par la règle (T-LAMBDA).

$$\dfrac{\dfrac{\dfrac{}{type(\Delta', X) = (\{\{X^{\{\sqcap,\sqcup\}} : \bullet\}\}, \bullet)}(\text{T-VAR})}{type(\emptyset, \lambda X^{\varnothing} : \bullet \ . \ X) = (\emptyset, \bullet^{\varnothing} \rightarrow \bullet)}(\text{T-LAMBDA}) \quad \dfrac{}{type(\Delta, X) = (\{\{X^{\{\sqcap,\sqcup\}} : \bullet\}\}, \bullet)}(\text{T-VAR})}{type(\Delta, (\lambda X^{\varnothing} : \bullet \ . \ X) \wedge X) = \text{"Erreur : } \lambda X^{\varnothing} : \bullet \ . \ X \text{ n'a pas le type } \bullet \text{ !"}}(\text{T-ET})$$

On retrouve bien le résultat de l'exemple 2, la formule $(\lambda X^{\varnothing} : \bullet \ . \ X) \wedge X$ n'est pas valide.

∗ Continuons avec la formule $\mu X : \bullet \ . \ X$. On prend $\Delta = \emptyset$ puisqu'il n'y a pas de variable libre. On va noter $\Delta' = \{\{X : \bullet\}\}$ l'environnement de typage sans variable créé par la règle (T-MU).

$$\dfrac{\dfrac{}{type(\Delta', X) = (\{\{X^{\{\sqcap,\sqcup\}} : \bullet\}\}, \bullet)}(\text{T-VAR})}{type(\Delta, \mu X : \bullet \ . \ X) = (\emptyset, \bullet)}(\text{T-MU})$$

On retrouve bien le résultat de l'exemple 2, la formule $\mu X : \bullet \ . \ X$ est valide.

∗ Poursuivons avec la formule $\mu X : \bullet \ . \ \neg X$. On prend $\Delta = \emptyset$ puisqu'il n'y a pas de variable libre. On va noter $\Delta' = \{\{X : \bullet\}\}$ l'environnement de typage sans variable créé par la règle (T-MU).

$$\dfrac{\dfrac{\dfrac{}{type(\Delta', X) = (\{\{X^{\{\sqcap,\sqcup\}} : \bullet\}\}, \bullet)}(\text{T-VAR})}{type(\Delta', \neg X) = (\{\{X^{\overline{\{\sqcap\}}} : \bullet\}\}, \bullet)}(\text{T-NEG})}{type(\Delta, \mu X : \bullet \,.\, \neg X) = \text{"Erreur : la variance de } X \text{ est } \overline{\{\sqcap, \sqcup\}} \text{ qui n'est pas } \succcurlyeq \varnothing \text{ !"}}(\text{T-MU})$$

On retrouve bien le résultat de l'exemple 2, la formule $\mu X : \bullet \,.\, \neg X$ n'est pas valide.

∗ Regardons maintenant la formule $\mu X : \bullet \,.\, ((\lambda Y^{\overline{\varnothing}} : \bullet \,.\, \neg Y)X)$. On prend $\Delta = \emptyset$ puisqu'il n'y a pas de variable libre. On va noter $\Delta_X = \{\{X : \bullet\}\}$ l'environnement de typage sans variable créé par la règle (T-MU) et $\Delta_{X,Y} = \{\{X : \bullet\}, \{Y : \bullet\}\}$ l'environnement de typage sans variable créé par la règle (T-LAMBDA).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{type(\Delta_{X,Y}, Y) = (\{\{Y^{\{\sqcap,\sqcup\}} : \bullet\}\}, \bullet)}(\text{T-VAR})}{type(\Delta_{X,Y},\ \neg Y) = (\{\{Y^{\overline{\{\sqcap,\sqcup\}}} : \bullet\}\}, \bullet)}(\text{T-NEG})}{type(\Delta_X, \lambda Y^{\overline{\varnothing}} : \bullet \,.\, \neg Y) = (\emptyset, \bullet^{\overline{\varnothing}} \to \bullet)}(\text{T-LAMBDA}) \quad \dfrac{}{type(\Delta, X) = (\{\{X^{\{\sqcap,\sqcup\}} : \bullet\}\}, \bullet)}(\text{T-VAR})}{type(\Delta_X, ((\lambda Y^{\overline{\varnothing}} : \bullet \,.\, \neg Y)X) = (\{\{X^{\overline{\varnothing}} : \bullet\}\}, \bullet)}(\text{T-APP})}{type(\Delta, \mu X : \bullet \,.\, ((\lambda Y^{\overline{\varnothing}} : \bullet \,.\, \neg Y)X))) = \text{"Erreur : la variance de } X \text{ est } \overline{\varnothing} \text{ qui n'est pas } \succcurlyeq \varnothing \text{ !"}}(\text{T-MU})$$

On retrouve bien le résultat de l'exemple 2, la formule $\mu X : \bullet \,.\, ((\lambda Y^{\overline{\varnothing}} : \bullet \,.\, \neg Y)X))$ n'est pas valide.

∗ Étudions ensuite la formule $\mu F : (\bullet^{\overline{\varnothing}} \to \bullet) \,.\, \lambda X^{\overline{\varnothing}} : \bullet \,.\, F(\neg(FX))$. On prend $\Delta = \emptyset$ puisqu'il n'y a pas de variable libre. On va noter $\Delta_F = \{\{F : \bullet^{\overline{\varnothing}} \to \bullet\}\}$ l'environnement de typage sans variable créé par la règle (T-MU) et $\Delta_{F,X} = \{\{F : \bullet^{\overline{\varnothing}} \to \bullet\}, \{X : \bullet\}\}$ l'environnement de typage sans variable créé par la règle (T-LAMBDA).
On note (1) l'arbre suivant :

$$\dfrac{}{type(\Delta_{F,X}, F) = (\{\{F^{\{\sqcap,\sqcup\}} : \bullet^{\overline{\varnothing}} \to \bullet\}\}, \bullet^{\overline{\varnothing}} \to \bullet)}(\text{T-VAR})$$

On note (2) l'arbre suivant :

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

L'arbre de la formule est :

$$\dfrac{\dfrac{\dfrac{(1) \qquad (2)}{type(\Delta_{F,X}, F(\neg(FX))) = (\{\{F^{\varnothing} : \bullet^{\overline{\varnothing}} \to \bullet\}, \{X^{\overline{\varnothing}} : \bullet\}\}, \bullet)}(\text{T-APP})}{type(\Delta_F, \lambda X^{\overline{\varnothing}} : \bullet \,.\, F(\neg(FX))) = (\{\{F^{\varnothing} : \bullet^{\overline{\varnothing}} \to \bullet\}\}, \bullet^{\overline{\varnothing}} \to \bullet)}(\text{T-LAMBDA})}{type(\Delta, \mu F : (\bullet^{\overline{\varnothing}} \to \bullet) \,.\, \lambda X^{\overline{\varnothing}} : \bullet \,.\, F(\neg(FX))) = (\emptyset, \bullet^{\overline{\varnothing}} \to \bullet)}(\text{T-MU})$$

On retrouve bien le résultat de l'exemple 2, la formule $\mu F : (\bullet^{\overline{\varnothing}} \to \bullet)\ .\ \lambda X^{\overline{\varnothing}} : \bullet\ .\ F(\neg(FX))$ est valide.

$*$ On note $\tau = \bullet^{\overline{\varnothing}} \to \bullet$. On cherche ici à inférer les variances de 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 :

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

On note (2) l'arbre suivant (ici $\star$ remplace (T-ET)) :

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

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 :

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

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 lorsqu'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. On retrouve bien le fait que la formule $(\mu F : \tau \,.\, \lambda X^{\overline{\varnothing}} : \bullet \,.\, \langle a \rangle (Y \wedge (F \neg (F X))))\ [b] Y$ est valide.

$*$ Pour terminer, regardons la formule $\mu X : \bullet \,.\, [a] X$. On prend $\Delta = \emptyset$ puisqu'il n'y a pas de variable libre. On va noter $\Delta' = \{\{X : \bullet\}\}$ l'environnement de typage sans variable créé par la règle (T-MU).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{type(\Delta', X) = (\{\{X^{\{\sqcap,\sqcup\}} : \bullet\}\}, \bullet)}(\text{T-VAR})}{type(\Delta', \neg X) = (\{\{X^{\overline{\{\sqcap,\sqcup\}}} : \bullet\}\}, \bullet)}(\text{T-NEG})}{type(\Delta', \langle a \rangle (\neg X)) = (\{\{X^{\overline{\{\sqcap\}}} : \bullet\}\}, \bullet)}(\text{T-DIAMANT})}{type(\Delta', [a] X) = (\{\{X^{\{\sqcap\}} : \bullet\}\}, \bullet)}(\text{T-NEG})}{type(\Delta, \mu X : \bullet \,.\, [a] X) = (\emptyset, \bullet)}(\text{T-MU})$$

On retrouve bien le résultat de l'exemple 2, la formule $\mu X : \bullet \,.\, [a] X$ est valide.

4 Implémentation des règles de typage

L'implémentation de ces règles d'inférence de variances 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 ainsi réalisées par des fonctions classiques, souvent déjà inclues dans le langage, et ne seront pas décrites.

Les opérations $v \circ \Gamma$ et $\Gamma_1 \wedge \Gamma_2$ sont réalisées par les algorithmes 1 et 2. À 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 d'inférence de variances *type*(.) est donnée par l'algorithme 3. Les objectifs de cette fonction sont de 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 d'inférence de variances de la section 3.

Une autre fonction qui ne sert pas directement à l'inférence de variances mais qui permet d'identifier les variables libres d'une formule f et de compter 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 d'inférence de variances et pour vérifier que cette dernière a bien attribué 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 àla 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 l'inférence des variances de la formule f.

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

        Φ ∧ Ψ :  (Γ1, τ1) ← TYPE(Φ, Δ);
                 si (τ1 = •) :
                     (Γ2, τ2) ← TYPE(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) ← TYPE(Φ, Δ);
                 retourner (\overline{{⊓, ⊔}} ∘ Γ, τ);

        ⟨a⟩Φ :  (Γ, τ) ← TYPE(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) ← TYPE(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) ← TYPE(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);
                     si (v ≼ v') :
                         retourner (Γ \ {X^{v'} : σ2}, σ_2^v → σ);
                     sinon :
                         afficher "Erreur : contradiction dans la variance de X";

          ΦΨ :  (Γ1, τ1) ← TYPE(Φ, Δ);
                 (Γ2, τ2) ← TYPE(Ψ, Δ);
                 cas τ1 avec :
                             • :  afficher "Erreur : Φ n'est pas une fonction";
                     σ_1^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));
```

5 Résultats

Les tables suivantes, les tables 1 et 2 contiennent les résultats des tests effectués avec l'implémentation de *type*(.). Ces tests comportent des cas de base, y compris des cas d'erreur pour vérifier le bon fonctionnement de chaque règle. Il y a aussi les formules des exemples 1, 2 et 3 pour avoir quelques formules plus compliquées dont on connaît déjà le résultat.

f	Δ	Γ	τ
$\top$	$\emptyset$	$\emptyset$	$\bullet$
X	$\{\{X : \bullet\}\}$	$\{\{X^{\{\sqcap,\sqcup\}} : \bullet\}\}$	$\bullet$
X	$\{\{X : (\bullet^{\varnothing} \to \bullet)\}\}$	$\{\{X^{\{\sqcap,\sqcup\}} : (\bullet^{\varnothing} \to \bullet)\}\}$	$(\bullet^{\varnothing} \to \bullet)$
$\langle a \rangle X$	$\{\{X : \bullet\}\}$	$\{\{X^{\{\sqcup\}} : \bullet\}\}$	$\bullet$
$\langle a \rangle X$	$\{\{X : (\bullet^{\varnothing} \to \bullet)\}\}$	Erreur dans $\diamond$: X n'a pas le type $\bullet$!	
$X \wedge Y$	$\{\{X : \bullet\},$ $\{Y : \bullet\}\}$	$\{\{X^{\{\sqcap,\sqcup\}} : \bullet\},$ $\{Y^{\{\sqcap,\sqcup\}} : \bullet\}\}$	$\bullet$
$X \wedge Y$	$\{\{X : (\bullet^{\varnothing} \to \bullet)\},$ $\{Y : \bullet\}\}$	Erreur dans $\wedge$: X n'a pas le type $\bullet$!	
$X \wedge Y$	$\{\{X : \bullet\},$ $\{Y : (\bullet^{\varnothing} \to \bullet)\}\}$	Erreur dans $\wedge$: Y n'a pas le type $\bullet$!	
$\neg X$	$\{\{X : \bullet\}\}$	$\{\{X^{\overline{\{\sqcap,\sqcup\}}} : \bullet\}\}$	$\bullet$
$\neg X$	$\{\{X : (\bullet^{\varnothing} \to \bullet)\}\}$	$\{\{X^{\overline{\{\sqcap,\sqcup\}}} : (\bullet^{\varnothing} \to \bullet)\}\}$	$(\bullet^{\varnothing} \to \bullet)$
$\mu X : \bullet \,.\, X$	$\emptyset$	$\emptyset$	$\bullet$
$\mu X : (\bullet^{\overline{\varnothing}} \to \bullet) \,.\, X$	$\emptyset$	$\emptyset$	$(\bullet^{\overline{\varnothing}} \to \bullet)$
$\mu X : \bullet \,.\, \neg X$	$\emptyset$	Erreur dans μ : X a la variance $\overline{\{\sqcap,\sqcup\}}$ qui n'est pas $\geqslant \varnothing$!	
$\mu X : (\bullet^{\varnothing} \to \bullet) \,.\, \top$	$\emptyset$	Erreur dans μ : types incompatibles entre X et $\top$!	
$\mu X : \bullet \,.\, X$	$\{\{X : (\bullet^{\varnothing} \to \bullet)\}\}$	$\emptyset$	$\bullet$

TABLE 1 – Résultats des tests de la fonction $type(f,\ \Delta) = (\Gamma, \tau)$

f	Δ	Γ	τ
$\lambda X^{\varnothing} : \bullet \,.\, X$	$\emptyset$	$\emptyset$	$(\bullet^{\varnothing} \to \bullet)$
$\lambda X^{any} : (\bullet^{\varnothing} \to \bullet) \,.\, X$	$\emptyset$	$\emptyset$	$((\bullet^{\varnothing} \to \bullet)^{any} \to (\bullet^{\varnothing} \to \bullet))$
$\lambda X^{\overline{\{\sqcap\}}} : (\bullet^{\varnothing} \to \bullet) \,.\, X$	$\emptyset$	Erreur dans λ : variances incompatibles pour X !	
$\lambda X^{\{\sqcup\}} : \bullet \,.\, X$	$\{\{X : (\bullet^{\varnothing} \to \bullet)\}\}$	$\emptyset$	$(\bullet^{\{\sqcup\}} \to \bullet)$
XY	$\{\{X : (\bullet^{\varnothing} \to \bullet)\}$ $\{Y : \bullet\}\}$	$\{\{X^{\{\sqcap,\sqcup\}} : (\bullet^{\varnothing} \to \bullet)\},$ $\{Y^{\varnothing} : \bullet\}\}$	$\bullet$
XY	$\{\{X : ((\bullet^{\{\sqcup\}} \to \bullet)^{\varnothing}$ $\to (\bullet^{\{\sqcap\}} \to \bullet)\},$ $\{Y : (\bullet^{\{\sqcup\}} \to \bullet)\}\}$	$\{\{X^{\{\sqcap,\sqcup\}} : ((\bullet^{\{\sqcup\}} \to \bullet)^{\varnothing}$ $\to (\bullet^{\{\sqcap\}} \to \bullet)\},$ $\{Y^{\varnothing} : (\bullet^{\{\sqcup\}} \to \bullet)\}\}$	$(\bullet^{\{\sqcap\}} \to \bullet)$
XY	$\{\{X : \bullet\},$ $\{Y : \bullet\}\}$	Erreur dans application : X n'a pas le type $\to$!	
XY	$\{\{X : (\bullet^{\varnothing} \to \bullet)\},$ $\{Y : (\bullet^{\varnothing} \to \bullet)\}\}$	Erreur dans application : types de X et Y incompatibles !	
$(\mu F : (\bullet^{\overline{\varnothing}} \to \bullet) \,.\, \lambda X^{\overline{\varnothing}} : \bullet \,.$ $\langle a \rangle (Y \wedge (F \neg (FX)))) \; [b] Y$	$\{\{Y : \bullet\}\}$	$\{\{Y^{any} : \bullet\}\}$	$\bullet$
$(\lambda X^{\varnothing} : \bullet \,.\, X) \wedge X$	$\{\{X : \bullet\}\}$	Erreur dans $\wedge$: $\lambda X^{\varnothing} : \bullet \,.\, X$ n'a pas le type $\bullet$!	
$\mu X : \bullet \,.\, ((\lambda Y^{\overline{\varnothing}} : \bullet \,.\, \neg Y) X))$	$\emptyset$	Erreur dans μ : X a la variance $\overline{\varnothing}$ qui n'est pas $\geqslant \varnothing$!	
$(\mu F : (\bullet^{\overline{\varnothing}} \to \bullet) \,.$ $\lambda X^{\overline{\varnothing}} : \bullet \,.\, F(\neg (FX))$	$\emptyset$	$\emptyset$	$(\bullet^{\overline{\varnothing}} \to \bullet)$
$\mu X : \bullet \,.\, [a] X$	$\emptyset$	$\emptyset$	$\bullet$

TABLE 2 – Suite des résultats des tests de la fonction $type(f,\ \Delta) = (\Gamma, \tau)$

6 Conclusion

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

Le but de ce TER a été de résoudre un sous-problème de l'inférence de types, l'inférence de variances. Pour ce faire, nous avons défini à l'aide de règles une fonction $type(.)$ qui prend en entrée une formule et un environnement de typage sans variances et rend le type de la formule et l'environnement de typage complété avec les variances. Nous avons aussi réalisé une implémentation de cette fonction en OCaml.

Toutefois, pour des raisons pratiques, les indications de variance présentes dans les types sont indiquées dans l'environnement de typage sans variances. Une première amélioration possible à ce TER, et sans doute la plus naturelle, serait d'enlever ces indictions de variances des types présents dans l'environnement de typage sans variances. Un outil pour le faire serait probablement la programmation par contrainte.

Dans un second temps, une autre ammélioration possible serait de s'attaquer cette fois-ci directement au problème de l'inférence de types.

Enfin, ce TER aura été l'occasion de mieux comprendre des notions vues 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 de nouvelles propriétés comme par exemple la complexité de l'algorithme de model-checking capable de traiter cette formule. Cela a été l'occasion aussi de comprendre 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 m'a permis de découvrir un nouveau langage, OCaml, et de nouvelles fonctionnalités de LaTeX .

Bibliographie

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

Kozen, D. (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.