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 *mu*-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 peut-être soit un type de base (noté $\bullet$), soit un type flèche (noté $\tau_1^v \to \tau_2$ 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, $\to$) où Pr est un ensemble d'états, Act un ensemble d'actions et $\to$: Pr $\times$ Act $\to$ 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, $\to$) où Pr $= \{P, Q, \ldots\}$ est un ensemble d'états, Act $= \{a, b, \ldots\}$ un ensemble d'actions et $\to$ 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 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, $\to$) 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.

Exemple 1. A faire

2.2 Variances

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.

Définition 3 (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 $\overline{\varnothing}$.

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

Une propriété importante des variance est l'additivité puisqu'il s'agit de la propriété particulière que doivent avoir toutes les variance présentes dans tous les types τ présents dans les opérateurs $\mu X : \tau.\Phi$ pour que l'algorithme de model-checking du μ-calcul polyadique d'ordre 1 fonctionne avec une complexité de PSPACE au lieu de EXPTIME.

Définition 4 ($\sqcap$-additivité et $\sqcup$-additivité)**.** Une fonction $f : \mathrm{A} \rightarrow \mathrm{B}$ est $\sqcap$-additive (resp $\sqcup$-additive) si $\forall x, y \in \mathrm{A}^2$, $f(x \sqcap y) = f(x) \sqcap f(y)$ (resp $f(x \sqcup y) = f(x) \sqcup f(y)$).

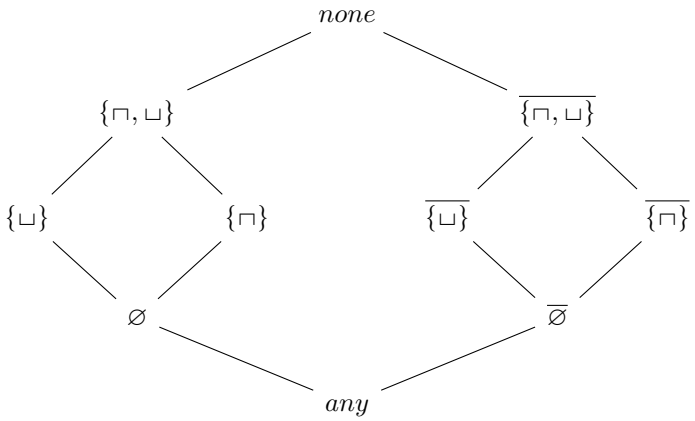

FIGURE 1 – Treillis des variances.

Notation 2 ($\preccurlyeq$)**.** On utilise le symbole $\preccurlyeq$ pour représenter la relation d'ordre partielle donnée par le treillis des variances (figure 1) dans lequel *any* est le plus petit élément et *none* le plus grand. Soit deux variances 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 3 ($v_1 \wedge v_2$)**.** 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 sa propre sémantique.

Définition 5 (Sémantique)**.** 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 6 (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 7 (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 :

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

Remarque 1. 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).

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 environnement de typage et un environnement de typage incomplet 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 la notion d'environnement de typage incomplet.

Définition 8 (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 $\tau_1^{v_1} \to \tau_2$).

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.

Notation 5 ($\emptyset$)**.** On note $\emptyset$ l'environnement de typage vide.

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 9 (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 10 (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 $(X^{v_1 \wedge v_2} : \tau_1) \in \Gamma_1 \wedge \Gamma_2$.

Maintenant que l'on a vu ce qu'est un environnement de typage classique, définissons un environnement de typage incomplet.

Définition 11 (Environnement de typage incomplet Δ)**.** Un environnement de typage incomplet, noté Δ est un ensemble d'assignements de typage incomplets de la forme $X : \tau$ où X est une variable et τ son type (qui peut être de la forme $\bullet$ ou $\tau_1^{v_1} \to \tau_2$).

Remarque 2. Un environnement de typage incomplet est un environnement de typage sans les variances des variables. Les indications de variance à l'intérieur des types $\to$ restent néanmoins toujours présentes.

Notation 6 ($type(\Delta, \Phi)$)**.** Soit un environnement de typage incomplet Δ et une formule du μ-calcul d'ordre supérieur Φ, on note $type(\Delta, \Phi)$ le couple (Γ, τ) avec Γ l'environnement de typage de Φ et τ le type de Φ.

Notation 7 ($vars(\Delta)$)**.** Soit un environnement de typage incomplet $\Delta = \{X_1 : \tau_1, \dots, X_n : \tau_n\}$, on note $vars(\Delta) = \{X_1, \dots, X_n\}$ l'ensemble des variables présentes dans Δ.

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

Définition 12 (Règles de typage)**.** Les règles de typage du μ-calcul d'ordre supérieur sont les suivantes :

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

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

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

$$\frac{type(\Delta \cup \{X : \tau\}, \Phi) = (\Gamma, \sigma) \qquad \sigma = \tau \qquad \Gamma = \Gamma' \cup \{X^v : \sigma\} \qquad v \succeq \varnothing}{type(\Delta, \mu X : \tau \,.\, \Phi) = (\Gamma', \tau)}$$

$$\frac{type(\Delta \cup \{X : \sigma\}, \Phi) = (\Gamma, \tau) \qquad \Gamma = \Gamma' \cup \{X^v : \sigma'\}}{type(\Delta, \lambda X : \sigma \,.\, \Phi) = (\Gamma', \sigma'^v \to \tau)}$$

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

Remarque 3. On peut faire trois remarques sur ces règles de typage.

- On doit connaître à l'avance le type des variables libres de la formule ce qui est possible avec une version annotée du μ-calcul d'ordre supérieur. Le fait de connaître les informations de variance de ces types est plus compliqué.
- Dans la règle de typage des variables (5^e règle), on assigne à X la variance $\{\sqcap, \sqcup\}$ quelque soit son 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 (non introduite par un μ ou un λ).
- Dans la règle de typage du μ (6^e règle), on impose le fait que la variance v de X doit être respecter $v \succcurlyeq \varnothing$. 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 $v \succcurlyeq \varnothing$.

Exemple 3. Donner des exemples (reprendre les précédents + prendre des exemples des formules pour les résultats), faire les arbres + commenter.

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`). Ce TER étant la continuité du stage réalisé par Thomas Portet durant l'été, certaines parties du code étaient déjà écrites. Le travail réalisé par Thomas portait sur l'implémentation des règles présentes dans l'article de Lange, Lozes et Guzmán (Lange *et al.*, 2014) (sans la partie polyadique) et celles exposées en section 2.3. Ces règles n'étant pas récursives, l'algorithme de Thomas testait toutes les variances pour toutes les variables libres de la formule dans un certain ordre et s'arrêtait lorsqu'une combinaison possible était trouvée.

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 $\bullet$) a été récupérée ainsi que la fonction d'affichage des formule. Les fonctions permettant les calculs entre variances (*dual*, $\circ$, $\wedge$) et la représentation des types ont également été récupérées. Le reste a été complètement réécrit. Les algorithmes de ces fonctions 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 incomplets) par des listes, 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.

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);
```

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.

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 attribué une variance à toutes les variables libres.

4 Résultats

Faire un tableau avec les résultats de tests.

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

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 = ∅) :
    retourner Résultat ∪ Γ2;
sinon :
    (X^v : τ) ← Γ1[1];
    si (X ∉ vars(Γ2)) :
        Util-VarRondEnvTypage(enleverIndice(1, Γ1), Γ2, {X^v : τ} ∪ Résultat);
    sinon :
        (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";
```

5 Conclusion

Bref résumé + avis personnel + piste(s) d'approfondissement.

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.

Algorithme 3 Retourne le typage de la formule f.

```
TYPAGE(formule f, environnement de typage incomplet 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}, σ_2^v → σ);

       ΦΨ :  (Γ1, τ1) ← TYPAGE(Φ, Δ);
             (Γ2, τ2) ← TYPAGE(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(l1[2 ::], 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));
```