Rapport (structure)

Introduction

Présentation rapide du μ-calcul polyadique et de ses applications (recherches à faire regarder le model-checking). Donner l'intérêt du μ-calcul par rapport à d'autres logiques dans ces applications.

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

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

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

Le μ-calcul polyadique d'ordre supérieur est donc une logique qui mérite d'être étudiée. Mais avant d'utiliser une formule écrite avec cette logique, il faut savoir si cette formule est bien formée ou non d'où l'intérêt de savoir la typer et donc de ce TER.

1 Le μ-calcul polyadique d'ordre supérieur

1.1 Généralités

Logique modale, polyadique et d'ordre supérieur (rappeler définitions).

Le μ-calcul polyadique d'ordre supérieur est une logique modale polyadique et qui permet le passage à l'ordre supérieur.

Définition 1 (*Logique modale*)**.** Une logique modale est une logique qui spécifie la véracité d'une formule par le biais de modalités.

Remarque 1. Les logiques temporelles sont des logiques modales qui possèdent notamment les modalités suivantes : "toujours" (noté $[]$), "un jour" (noté $\langle\rangle$) et "jamais" (noté $\neg[]$).

Le μ-calcul (et par extension le μ-calcul polyadique d'ordre supérieur) est une logique temporelle dont on peut extraire d'autres logiques comme CTL* (qui contient elle-même CTL et LTL).

Définition 2 (*Logique polyadique (ou de dimension supérieure)*)**.** Une logique polyadique (ou de dimension supérieure) est une logique qui permet les opérations sur des tuples d'états.

Définition 3 (*Logique d'ordre supérieur*)**.** Une logique d'ordre supérieur est une logique qui permet les opérations des formules par opposition aux logiques du premier ordre qui permettent uniquement les opérations sur les états seuls.

1.2 Syntaxe et sémantique

1.2.1 Enoncés

Enoncer la syntaxe et la sémantique.

Pour pouvoir énoncer la syntaxe du μ-calcul polyadique d'ordre supérieur, commençons par poser quelques notations.

Supposons que nous avons un système de transition (Pr, Act, $\rightarrow$) où Pr = {P, Q, ...} est un ensemble d'états, Act = {a, b, ...} 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). i_1, i_2, ..., i_n et j_1, j_2, ..., j_n représentent les n composants d'un tuple.

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

On peut maintenant énoncer la syntaxe du μ-calcul polyadique d'ordre supérieur :
$\Phi, \Psi ::= \top \mid \Phi \wedge \Psi \mid \neg\Phi \mid \langle a \rangle_i \Phi \mid \{i_1, i_2, ..., i_n\} \leftarrow \{j_1, j_2, ..., j_n\}\Phi \mid \mathrm{X} \mid \mu \mathrm{X} : \tau.\Phi \mid \lambda \mathrm{X}^{\nu} : \tau.\Phi \mid \Phi\Psi$

Remarque 2. Dans $\langle a \rangle_i \Phi$, le i indique le i^e élément du tuple Φ qui est modifié par l'action a.

Notation 1. On peut rajouter par sucre syntaxique les cinq opérateurs suivants :
$\Phi \vee \Psi ::= \neg(\neg\Phi \wedge \neg\Psi)$;
υ X . $\Phi ::= \neg\mu \mathrm{X} . \neg\Phi[\neg \mathrm{X}/\mathrm{X}]$ (remplacement de X par $\neg$X) ;
$[a]_i \Phi ::= \neg\langle a \rangle_i \neg\Phi$;
$\Phi \Rightarrow \Psi ::= \neg\Phi \vee \Psi$;
$\Phi \Leftrightarrow \Psi ::= (\Phi \Rightarrow \Psi) \wedge (\Psi \Rightarrow \Phi)$.

On note r la taille maximale d'un tuple d'états dans la formule regardée. On peux maintenant énoncer la sémantique du μ-calcul polyadique d'ordre supérieur :
$[\![\top]\!] = \mathrm{Pr}^r$

1.2.2 Exemples

Exemples pour mieux comprendre le sens des opérateurs. Regarder dans le(s) papier(s) ou à inventer.

1.3 Variances

1.3.1 Définition et intérêt (titre à revoir)

Intérêt pour la validité de formules avec le point fixe.
Donner une intuition de ce qu'est la variance. Essayer de donner une définition

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

On peut voir la variance comme la monotonie de la fonction représentée par la variable ou la formule (rappelons que comme en λ-calcul, les variables sont des fonctions).

1.3.2 Définitions et propriétés

Treillis des variances (expliquer le sens des variances), additivité, "N-additivité", sémantiques, règles de calcul.

Faire le treillis

Pour reprendre le parallèle entre les variances et les monotonies des fonctions, on peut voir une variance *none* comme une fonction constante et une variable *any* comme une fonction quelconque. Les variables avec les autres variances sont des fonctions monotones, croissantes pour les variables avec les variances $\{\sqcap, \sqcup\}$, $\{\sqcap\}$, $\{\sqcup\}$ et $\varnothing$, décroissantes pour les variables avec les variances duales à ces dernières.

Définition ($\sqcap$-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 $f(x \sqcup y) = f(x) \sqcup f(y)$).

Notation ($\preccurlyeq$) : On utilise le symbole $\preccurlyeq$ pour représenter la relation d'ordre partielle donné par le treillis des variables 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 v' selon le treillis.

Définition (sémantique) :

1.3.3 Exemples

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

1.4 Règles de typage

1.4.1 Définitions

Gamma + Delta.

1.4.2 Enoncé

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

1.4.3 Exemples

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

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

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

2.1 Travail récupéré (titre à revoir)

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

2.2 Fonction de typage

2.2.1 Implémentation des environnements de typage

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

2.2.2 La fonction de typage

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

2.3 Fonctions annexes

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

3 (Piste(s) d')Améliorations

3.1 Les limites de l'implémentation actuelle

Le Gamma.

3.2 (Piste(s) d')Améliorations

A voir.

4 Résultats

4.1 Tableau des résultats

Faire un tableau avec les résultats de tests.

4.2 Commentaires

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

Conclusion

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

Références

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

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

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

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

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