Analyse statique d'un calcul d'acteur par interprétation abstraite.

Pierre-Loïc Garoche
Institut de Recherche en Informatique de Toulouse – École Normale Supérieure de Cachan
garoche@enseeiht.fr

Résumé

Le modèle des acteurs permet la définition de programmes concurrents avec un comportement non uniforme. L'analyse statique d'un tel modèle a été effectuée précédemment par typage. Cette approche était basée sur de la résolution de contraintes et n'était pas capable de traiter des propriétés suffisament précises pour la communication de comportements. Nous décrivons ici une nouvelle approche, orienté flot de contrôle, et basée sur le cadre de l'interprétation abstraite, capable de traiter le passage de comportement.

Mots Clef

Analyse statique, calculs de processus, acteurs, interprétation abstraite.

Abstract

The actor model eases the definition of concurrent programs with non uniform behaviors. Static analysis for this model was previously done in a data-flow oriented way, with type systems. This approach was based on constraint set resolution and was not able to deal with precise enough properties for communications of behaviors. We present here a new approach, control-flow oriented, based on the abstract interpretation framework, able to deal with communication of behaviors.

Keywords

Static analysis, process calculi, actors, abstract interpretation.

1 Introduction

Avec le développement des réseaux de télécommunications et du déploiement d'applications distribuées et concurrentes sur ceux-ci, l'analyse formelle des programmes est devenu un enjeu majeur. Les calculs de processus permettent de modéliser la topologie de ces systèmes et leur évolution. Le modèle des acteurs proposé par Agha et Hewitt [1] permet de représenter des systèmes asynchrones communiquant par des messages étiquetés. Ce modèle est particulièrement réaliste pour représenter des systèmes au comportement non uniforme définis dans des langages objets où l'envoi de messages se fait par l'appel à des méthodes.

La validation de ces systèmes par analyse statique permet de vérifier les propriétés de programme réels asynchrones en analysant leur code source, leur description. Notre équipe s'intéresse à l'analyse statique d'un calcul d'acteur [3] et a défini des analyses par typage sur ce calcul [2, 4], comme la vérification de linéarité ou la détection de messages orphelins. Nous nous intéressons ici a une autre approche formelle, celle de l'interprétation abstraite introduite par Cousot et Cousot [5, 6]. Nous voulons définir des analyses sur des programmes embarqués et répartis afin d'en vérifier les propriétés.

Cette communication est organisée comme suit. Nous introduirons d'abord CAP, notre calcul d'acteur, puis nous décrirons une sémantique non standard, dans lequel les informations contenues dans les termes sont explicites. La troisième partie sera consacrée à l'utilisation de l'interprétation abstraite pour approximer de façon correcte la sémantique collectrice des termes de CAP afin d'en observer les propriétés. Quelques propriétés seront ensuite introduites.

2 CAP : un Calcul d'Acteur Primitif

Le calcul CAP a été introduit dans [3] afin de pouvoir définir des analyses statiques sur un calcul formel modélisant le modèle de la programmation par acteurs. Les termes de ce calcul décrivent des systèmes *asynchrones* composés d'*acteurs* et de *messages* étiquetés. Un acteur ($a \rhd P$) peut être vu comme un lien entre une adresse et un comportement. Le comportement est la capacité à comprendre un certain nombre de messages ainsi que la réaction effectuée lors de la prise en compte de ces messages. L'opérateur ζ inclus dans les sous-terme de comportement permet de désigner dans la continuation l'adresse et le comportement total de l'acteur recevant le message. CAP permet donc de désigner des comportements par variables et d'envoyer celles-ci. C'est donc un calcul de processus de second ordre. La syntaxe et la sémantique de CAP sont décrites dans la Figure 1. Un exemple illustrant le calcul est donné dans la Figure 2.

Nous nous intéressons à la vérification par analyse statique de systèmes distribués décrit dans CAP. Les propriétés strictement liées à l'aspect concurrent et distribué de ces systèmes que nous voulons ici vérifier sont par exemple la propriété de linéarité ou la détection de messages orphelins. La première considère chaque adresse comme une ressource et consiste à vérifier qu'au plus un acteur est associé à une même adresse dans tous les termes atteignables à par-

$$C ::= 0 \mid \nu a^{\alpha}\, C \mid C \,||\, C \mid a \rhd^{l} P \mid a \lhd^{l} m(\tilde{P})$$
$$P ::= x \mid [m_i^{l_i}(\tilde{y}) = \zeta(e,s)C_i^{\,i=1\ldots n}]$$

$$\frac{T = [m_i^{l_i}(\tilde{x_i}) = \zeta(e_i,s_i)C_i^{\,i=1,\ldots,n}] \quad \begin{cases} m = m_k, \\ |\tilde{y_l}| = |\tilde{x_k}|, \\ k \in [1,\ldots,n] \end{cases}}{a \rhd T \,||\, a \lhd^{l} m(\tilde{y_l}) \xrightarrow{(l,l_i,l_k)} C_k[e_k \leftarrow a, s_k \leftarrow T, \tilde{x_k} \leftarrow \tilde{y_l}]}$$

Figure 1: Syntaxe et sémantique de CAP

$$\begin{array}{rl} \nu a^{\alpha}, b^{\beta}, & a \rhd^{1} [m^{2}() = \zeta(e,s)(a \rhd^{3} s), \\ & \quad send^{4}(x) = \zeta(e,s)(x \lhd^{5} beh(s))] \\ || & b \rhd^{6} [beh^{7}(x) = \zeta(e,s)(e \rhd^{8} x)] \\ || & a \lhd^{9} send(b) \,||\, b \lhd^{10} m() \end{array}$$

Deux acteurs et deux messages sont présents dans le système. Initialement seul le message *send* envoyé à *a* peut être compris. La transition s'effectue. Il n'y a plus qu'un acteur, sur *b* et deux messages adressés à cette adresse. Le message *m* ne peut toujours pas être compris, mais le nouveau message *beh* peut l'être. Il est alors reçu. La variable *s*, argument du message, dénote tout le comportement de l'acteur initial sur *a*. Ce comportement est utilisé pour être associé à l'adresse décrite par *e* au point de programme 7, c'est à dire *b*. L'acteur sur *b* peut maintenant comprendre en compte les messages *m* et *send*. Il peut alors recevoir le premier message *m*.

Figure 2: Exemple d'un système décrit par un terme CAP

tir du terme initial. Les orphelins sont les messages qui sont envoyés à une adresse qui ne pourra pas les traiter.

Afin de pouvoir définir une analyse sur les termes de ce calcul, nous l'exprimons dans un autre formalisme afin d'expliciter les informations contenues dans les termes.

3 Sémantique non standard

Afin de proposer un calcul commun pour définir des analyses génériques sur plusieurs calcul de processus, Feret exploite dans [7] un méta-langage qui explicite l'historique des transitions conduisant à la création des valeurs et des agents. Une telle approche permet d'éviter les ambiguïtés liés à l'α-conversion dans l'analyse des termes. Nous décrivons dans cette section comment donner à CAP une telle sémantique. Par soucis de brièveté, nous décrivons ici le système sans donner de preuves ni de définitions formelles de toutes les primitives, celles-ci peuvent être trouvés dans [8, 9].

Une configuration est maintenant un ensemble de processus. Les processus permettent de désigner les messages, les acteurs et enfin les comportements atomiques. Un processus est défini par un triplet (p, id, E) où p est le point de programme permettant d'associer le processus à un sous-terme du terme initial, *id* est le mot désignant la suite de transition ayant conduit à la création du processus. Enfin E, l'environnement, est une fonction partielle des variables du terme vers leur valeur. La valeur associée à une variable est un couple (α, id_α) où α est le point de programme de l'opérateur de restriction (ν ou ζ) ayant défini le nom et id_α le mot qui représente la série de transitions ayant conduit à la création de cette valeur.

Nous décrivons maintenant les étapes nécessaires au calcul d'une transition. Il faut tout d'abord choisir un *ensemble de processus* participant à la transition dans l'ensemble des processus qui constituent la configuration courante.

Il faut ensuite vérifier des *conditions de synchronisations*. En particulier, nous vérifions que le processus représentant le message est bien adressé à l'adresse associée au processus qui dénote l'acteur. La vérification des ces conditions dépend du domaine utilisé pour représenter le flot de contrôle.

Si le n-uplet de processus sélectionné satisfait aux conditions, la transition peut s'effectuer, il faut *retirer l'ensemble des processus qui a interagit* et calculer les *nouveaux processus à insérer* dans la configuration. Pour cela, la continuation associé au comportement qui a été activé par le message est mise à jour à la fois par les valeurs contenues dans le message mais aussi par l'opérateur ζ, c'est à dire, par les valeurs décrivant l'adresse de l'acteur interagissant et son comportement.

Afin d'exprimer CAP dans le cadre générique proposé par Feret [7], nous avons définit une certain nombre de primitives :

- un ensemble d'interaction partielles décrivant les différentes sortes de processus, les messages, les acteurs et les comportements ;
- une fonction d'extraction β construit la configuration non standard associée à un terme CAP et est utilisé notamment lors du calcul de continuations pour déterminer les processus nouveaux à insérer dans le système ;
- enfin deux *règles formelles* ou règles de réductions permettant de modéliser dans la sémantique non standard les transitions. La première relation de transition permet de représenter une transition faisant intervenir un acteur défini syntaxiquement ($a \rhd [\ldots]$) tandis que la seconde nécessite trois processus et représente un acteur dont le comportement est défini par une variable ($a \rhd b$).

La sémantique non standard est décrite dans la Figure 3.

Théorème 1 (correspondance) *La sémantique standard et non standard de CAP sont en bisimulation forte.*

La preuve est presentée dans [10].

4 Sémantique abstraite

L'expression d'un terme CAP dans cette syntaxe et cette sémantique non standard a permis d'expliciter certaines ambiguïtés dues, entre autre, à la règle d'α-conversion de la relation de congruence définissant la sémantique du

Soit C une configuration. Soit $\mathscr{R} = (n, components, compatibility, v\_passing)$ une règle de réduction.
Nous supposons qu'il existe à la fois un n-uplet $(t^k)_{1\leqslant k\leqslant n} = (p^k, id^k, E^k)_{1\leqslant k\leqslant n} \in C^n$ de processus distincts et un n-uplet $(pi^k)_{1\leqslant k\leqslant n} = (s^k, (parameter_l)^k, (bd_l)^k, constraints^k, continuation^k)_{1\leqslant k\leqslant n}$ d'interactions partielles, tels que :

1. $\forall k \in [\![1;n]\!], exhibits(t^k, pi^k)$;
2. $\forall k \in [\![1;n]\!], components(k) = s^k$;
3. $sync((t^k)_k, ((parameter_l)^k)_k, compatibility)$ ne soit pas l'élément $\perp$.

Alors
$C \xrightarrow{(\alpha^i)^n} (C \setminus removed\_threads) \cup news\_threads$
avec :
– $removed\_threads = remove\left((t^k)_{1\leqslant k\leqslant n}\right)$;
– $new\_threads = \bigcup_{1\leqslant k\leqslant n} launch\left(Ct^k, \overline{id}, \overline{E}^k\right)$,
où $\forall k \in [\![1;3]\!]$:
– $Ct_k \in continuation^k$;
– $\overline{id} = marker\left((p^{k'}, id^{k'}, E^{k'})_{1\leqslant k'\leqslant n}\right)$;
– $\overline{E}^k = vpassing\left(k, (t^{k'})_{1\leqslant k'\leqslant n}, ((bd_l)^k)_k, ((parameter_l)^k)_k, communications\right)$.
– $\forall k \in [\![1;n]\!], \alpha^k = (t^k, pi^k, E^k)$.

Figure 3: Règle de transition non standard

langage. Nous décrivons dans cette partie, comment sur-approximer la sémantique collectrice des termes exprimés dans cette sémantique non standard. Nous utilisons pour cela le cadre de l'interprétation abstraite, afin de définir une approximation correcte et décidable, c'est à dire ici, calculable en un temps fini, de la sémantique collectrice du terme analysé.

4.1 Interprétation abstraite

L'interprétation abstraite [5, 6] est une théorie de l'approximation discrète de sémantique. La description des propriétés d'une sémantique dans un treillis complet munis d'opérateurs monotones, permet de définir une correspondance de Galois (c'est à dire, un couple de morphisme (α, γ)) entre deux descriptions de sémantiques. La première étant la sémantique concrète $(C, \subseteq)$ que nous voulons analyser et la seconde une sémantique abstraite $(C^\#, \sqsubseteq)$ dans laquelle nous observons les propriétés. La correspondance de Galois (α, γ) exhibée entre ces deux sémantiques permet de garantir la correction de l'abstraction : c'est une sur-approximation de la sémantique concrète. Ainsi, $\forall x \in C, \forall y \in C^\#, \alpha(x) \sqsubseteq y \implies x \subseteq \gamma(y)$. L'utilisation d'opérateurs monotones sur ces treillis permet de conserver cette propriété. Enfin, les points fixes pour ces opérateurs monotones existent (Tarski) et peuvent être obtenus de façon constructive en itérant à partir d'un élément du treillis (Kleene, Cousot). Les propriétés étant approchées, représentées dans des domaines abstraits, nous ne pouvons répondre de façon automatique à toutes les questions, mais si la propriété étudiée peut être observée dans l'abstrait, elle est valide dans le concret. Le cadre de l'interprétation abstraite permet donc de sur-approximer de façon correcte, mais généralement pas complète, les propriétés d'une sémantique.
Nous voulons ici vérifier des propriétés sur l'ensemble des termes atteignables, c'est à dire constructibles à partir du terme initial. C'est la sémantique collectrice du terme. Elle peut être définie plus formellement comme le plus petit point fixe du morphisme complet pour l'union $\mathbb{F}$:

$$\mathbb{F}(X) = \left\{ \begin{array}{l} (\{\varepsilon\} \times \mathscr{C}_0) \bigcup \\ \left\{ \; (u.\lambda, C') \;\middle|\; \exists C \in \mathscr{S}, (u, C) \in X \text{ et } C \xrightarrow{\lambda} C' \; \right\} \end{array} \right.$$

où $\mathscr{C}_0$ désigne la configuration initiale et $\mathscr{S}$ l'ensemble des configurations.
Nous voulons vérifier des propriétés sur l'ensemble des ces configurations atteignables. Mais cet ensemble n'est, en général, pas calculable. Nous devons donc recourir à des méthodes permettant de vérifier ces propriétés, tout en assurant d'obtenir une réponse au bout d'un temps fini raisonnable.
L'interprétation abstraite de systèmes mobiles, définie par Feret [7], consiste à définir une sémantique opérationnelle abstraite des termes. L'isomorphisme $\mathbb{F}^\#$ sur $C^\#$ permettant d'approximer $\mathbb{F}$ est défini par :

$$\mathbb{F}^\#(c^\#) = \bigsqcup^\# \left(\begin{array}{l} \{C_0^\#; c^\#\} \sqcup \\ \left\{ c'^\# \mid \exists \lambda \in \Sigma, c^\# \rightsquigarrow^\lambda c'^\# \right\} \end{array} \right)$$

où $C_0^\#$ est l'abstraction de l'état initial et $\rightsquigarrow$ représente la fonction de transition abstraite. Nous calculons donc l'ensemble des propriétés représentées dans le domaine abstrait $C^\#$ qui sont vraies dans un sur-ensemble de toutes les configurations atteignables. La précision de ce domaine abstrait permet donc de calculer à la fois un sur-ensemble le plus précis possible (le plus petit) et en même temps les propriétés d'intérêt.
La section suivante adresse ce problème et décrit quelle sémantique abstraite donner aux termes de la sémantique non standard et comment y représenter les propriétés qui nous intéressent.

4.2 Domaines abstraits

La sémantique opérationnelle non standard permet de guider les transitions et donc de décrire l'évolution du terme. Nous voulons ici représenter dans les domaines abstraits, à la fois l'information utile aux calculs des transitions ainsi que l'information nécessaire à la vérification des propriétés qui nous intéressent.
Un élément abstrait, représentant un ensemble de configurations concrètes est donc constitué de deux éléments, formant un couple (CF, P) où CF représente une approximation des valeurs des variables permettant d'effectuer les

transitions, tandis que P représente un ensemble de propriétés réalisées par l'ensemble des configurations que cet élément abstrait représente.
La précision de ces deux éléments, ou plutôt des domaines auxquels ils appartiennent, ainsi que des sémantiques associées permettent de définir d'un part, le flot de contrôle, c'est à dire l'ensemble des transitions effectuées et d'autre part la ou les propriétés à vérifier. L'utilisation de domaines simples pour représenter le flot de contrôle permet de calculer plus rapidement le point fixe de l'analyse, mais l'élément abstrait obtenu va potentiellement représenter plus de configuration concrètes que possible et certaines de ces fausses configurations concrètes représentées ne vont pas satisfaire la propriété étudiée et donc empêcher de la vérifier dans l'abstrait.
De la même façon pour l'élément abstrait qui représente les propriétés, un domaine simple va permettre de converger rapidement, mais ne permettra peut-être pas d'observer une propriété plus complexe.

Abstraction du flot de contrôle Le flot de contrôle permet de calculer les transitions dans l'abstrait. Nous utilisons ici un domaine abstrait paramétrique dont la sémantique est très proche de la sémantique non standard. Lors du calcul d'une transition dans l'abstrait, nous vérifions d'abord que les conditions de synchronisations, telles qu'elles sont représentables dans le domaine abstrait peuvent être satisfaites. Ensuite le passage de valeur est calculé afin de créer les nouveaux processus. Nous déterminons ensuite l'union de l'élément abstrait obtenu avec l'élément initial afin d'obtenir les propriétés, représentées dans le domaine, qui sont réalisées à la fois avant et après la transition. La suppression des processus interagissant n'est pas représentée ici puisqu'elle ne modifie pas la valeur des marqueurs des processus ni leurs variables. Une représentation plus formelle de cette sémantique est définie dans la Figure 4.
Le domaine utilisé pour représenter le flot de contrôle peut être ensuite choisi afin d'avoir une analyse plus ou moins précise. Un domaine très simple permet, par exemple, d'avoir une sur-approximation similaire à celle que nous obtenions en utilisant des techniques de typage : il s'agit d'abstraire complètement le marqueur des processus et des variables. Une configuration est alors un n-uplet d'environnements indicé par les points de programme du terme. Chaque environnement associe à chaque variable l'ensemble des restrictions qui ont crée la valeur. Ainsi, nous vérifions, comme dans nos système de type, que le message envoyé au nom a est bien reçu par l'acteur associé au même nom a, sans tenir compte des instances récursives du même nom.
Par contre, le domaine utilisé peut être plus complexe, par exemple le produit réduit de plusieurs autres domaine [9]. Il permet alors d'avoir une analyse précise et contenant des informations relationnelles entre les valeurs des variables. Par exemple, nous pouvons obtenir que le marqueur associé à une certaine variable est le même que celui associé

Soit $C^\# \in \mathscr{C}^\#_{env}$ une configuration abstraite,
soit $(n, components, compatibility, v\_passing)$ une règle de réduction.
soit $(p_k)_{1\leqslant k\leqslant n} \in \mathscr{L}_p^n$ un n-uplet d'étiquettes de points de programme et $(pi_k)_{1\leqslant k\leqslant n} = (s_k, (parameter), (bd), constraints, static\_continuation_k)$ un n-uplet d'interactions partielles.
nous définissons :

$$mol \triangleq reagents^\#((p_k),(parameter_{k,l}),(constraints_k),C^\#).$$

si

1. $\forall k \in [\![1;n]\!], pi_k \in interaction(p_k)$;
2. $mol \neq \bot_{(I(p_k))_k}$

alors :

$$C \xrightarrow{(p_k)_k}_\# \bigsqcup \{C; new\_threads\}$$

où :
– $mol' = marker\_value(type(s_1),(p_k)_k, mol, (bd_{k,l})_{k,l}, (parameter_{k,l})_{k,l}, v\_passing)$
– $new\_threads = launch^\#((p_k, continuations_k)_k, mol')$.

Figure 4: Sémantique opérationnelle abstraite pour l'analyse de flot de contrôle.

à une autre, lorsqu'elle partage la même valeur. Plus le domaine sera précis, plus l'abstraction sera fine et la sur-approximation petite. Il y aura donc moins de fausses configurations concrètes représentées par l'élément abstrait.

Abstraction des propriétés La partie de l'élément abstrait associée aux propriétés à analyser doit être munie d'une sémantique permettant de représenter dans le domaine l'évolution de cette propriété lors de l'exécution d'une transition dans le concret. L'idée générale est que cet élément représente les propriétés réalisées par un ensemble de configurations concrètes. Le calcul, dans l'abstrait, d'une transition, doit refléter l'évolution de la propriété par la transition. Ensuite, afin de conserver les propriétés de monotonie sur les primitives du domaines abstrait, on calcule l'union des propriétés avant et après la transition, c'est à dire l'ensemble des propriétés qui sont vraies dans les deux cas.
La section suivante décrit comment instancier ce domaine afin de calculer des propriétés différentes.

5 Exemples d'instanciations des domaines abstraits

Dans cette section nous présentons deux possibilités, non exclusives, d'instanciation du domaine abstrait utilisé pour représenter les propriétés d'intérêt. Nous décrirons d'abord comment introduire la notion de dénombrement puis comment utiliser un domaine pour représenter une propriété particulière : la linéarité.

5.1 Dénombrement

Dans ce domaine, décrit dans [9], nous dénombrons à la fois l'occurrence des processus associés à chaque point de programme, mais aussi l'occurrence des points de programme dans les étiquettes des transitions. Soit $\mathscr{V}_c$ un tel ensemble de points de programme. Ici l'élément abstrait est un couple (I,K) où I est un n-uplet d'intervalles sur $\mathbb{N}$ indicé par $\mathscr{V}_c$ et K un domaine de Karr [12] construit sur l'espace vectoriel engendré par $\mathscr{V}_c$. Nous pouvons ainsi observer dans un tel élément le nombre d'occurrence de processus associé à un point de programme donné ainsi que des relations affines entre ces occurrences. Un tel domaine permet par exemple d'observer des contraintes d'exclusions mutuelles. La sémantique associée, contrairement à celle du flot de contrôle, doit représenter dans l'abstrait la suppression des processus participant à la transition.

5.2 Linéarité

La propriété de linéarité exprime le fait que dans chaque configuration, chaque adresse est associée à, au plus, un acteur. Afin de vérifier cette propriété, nous avons défini un domaine abstrait s'inspirant d'une de nos analyse par typage afin de représenter, de façon opérationnelle, le calcul de la propriété. Les détails sont présentés dans [11]. L'idée est d'associer aux variables des environnements une valeur abstraite dénotant l'utilisation de la variable. Le passage de valeur permet alors de redistribuer les modes d'utilisations parmi les variables de la continuations.

6 Conclusion

Nous avons représenté CAP dans un cadre générique pour l'interprétation abstraite de systèmes mobiles. Nous avons ensuite utilisé et défini des domaines abstraits permettant d'une part de représenter le calcul des transitions abstraites mais aussi des propriétés sur les termes analysés. Avec ces outils, nous sommes capable de calculer un point fixe de l'analyse afin d'obtenir un ensemble correct des propriétés vérifiées par la sémantique collectrice du terme, c'est à dire dans toutes les configurations qu'il peut atteindre.

La prochaine étape majeure de notre travail va être le développement de domaines abstraits permettant la détection de messages orphelins dans un système décrit par un terme CAP. Les messages orphelins sont les messages qui sont envoyé à une adresse qui ne peut et ne pourra pas les comprendre par la suite. Une telle propriété est indispensable dans l'analyse des systèmes asynchrones.

Références

[1] Gul Agha and Carl Hewitt. Actors : a conceptual foundation for concurrent object-oriented programming. pages 49–74, 1987.

[2] Jean-Louis Colaço, Marc Pantel, Fabien Dagnat, and Patrick Sallé. Static safety analysis for non-uniform service availability in Actors . In *Proceedings of the IFIP 3rd International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'99)*, volume 139, pages 371–386. Kluwer, B.V., 1999.

[3] Jean-Louis Colaço, Marc Pantel, and Patrick Sallé. An actor dedicated process calculus. In *Proceedings of the ECOOP'96 Workshop on Proof Theory of Concurrent Object-Oriented Programming (PTCOOP'96)*, 1996.

[4] Matthias Colin, Xavier Thirioux, and Marc Pantel. Temporal logic based static analysis for non uniform behaviors. In *Proceedings of the IFIP 6th International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'03)*. Springer Verlag, 2003.

[5] Patrick Cousot and Radhia Cousot. Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM Symposium on Principles of Programming Languages (POPL'77)*, pages 238–252. ACM Press, 1977.

[6] Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *Proceedings of the 6th ACM Symposium on Principles of Programming Languages (POPL'79)*, pages 269–282. ACM Press, 1979.

[7] Jérôme Feret. *Analyse des sytèmes mobiles par interprétation abstraite*. PhD thesis, École polytechnique, Paris, France, february 2005.

[8] Pierre-Loïc Garoche, Marc Pantel, and Xavier Thirioux. Static Safety for an Actor Dedicated Process Calculus by Abstract Interpretation. In Roberto Gorrieri and Heike Wehrheim, editors, *Proceedings of 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems, Bologna, Italy (FMOODS'06)*, page xx. Springer Verlag Lecture Notes in Computer Science, 14-16 juin 2006.

[9] Pierre-Loïc Garoche. Static analysis of actors by abstract interpretation. Master's thesis, École Normale Supérieure de Cachan, 2005.

[10] Pierre-Loïc Garoche, Marc Pantel, and Xavier Thirioux. Static Safety for an Actor Dedicated Process Calculus by Abstract Interpretation . Rapport de recherche, IRIT, décembre 2005.

[11] Pierre-Loïc Garoche, Marc Pantel, and Xavier Thirioux. Static Analysis of Actors : From Type Systems to Abstract Interpretation . In Francesco Ranzato, editor, *Proceedings of 1st International Workshop on Emerging Applications of Abstract Interpretation , Vienna, Austria (EAAIT'06)*, page xx. Elsevier Electronic Notes in Theoretical Computer Science, 26 mars 2006.

[12] Michael Karr. Affine relationships among variables of a program. *Acta Informatica*, 6 :133 – 151, 1976.