Nicolas Vallée (nicolas.vallee@ensta.fr)
sous la co-direction de Bruno Monsuez (ENSTA – UEI) et de Michel Mauny (ENSTA – UMA)
Laboratoire d'Electronique et d'Informatique de l'Ecole Nationale Supérieure de Techniques Avancées

Intitulé de la thèse :
Conception d'un outil de débogage formel pour systèmes logiciels et matériels selon l'approche « Debug as Design »

Résumé :
Le but final de ma thèse était de modéliser et concevoir un outil de débogage formel pour systèmes logiciels ou matériels selon l'approche « debug as design ». Nous avons choisi d'analyser des descriptions de systèmes. Pour un logiciel nous pouvons étudier son code source, alors que pour un circuit électronique nous nous intéressons à sa description dans un langage adapté (HDL, SystemC dans mon cas). Un débogueur formel est un outil capable de détecter les erreurs et de signaler leur emplacement sans avoir besoin de faire fonctionner réellement le système étudié. Cette technique a l'avantage de ne pas dépendre des 'conditions expérimentales', et donc de pouvoir certifier le résultat dans tous les cas possibles. Enfin, l'approche « debug as design » consiste à utiliser des outils de vérification très tôt dans le cycle de conception du système. Ainsi il signalerait rapidement les petits problèmes locaux, et fusionnerait ensuite tous les sous-résultats pour analyser un plus gros morceau, jusqu'à finir par avoir vérifié l'intégralité du système.

Principaux résultats :

Dans un premier temps, j'ai travaillé sur la confrontation de deux grands modèles de représentation hiérarchisée, d'une part les « graphes hiérarchiques » d'Hoffmann & al, et d'autre part les « hypergraphes fractals » de Bruno Monsuez. J'ai achevé les démonstrations nécessaires pour mettre en évidence que les graphes hiérarchiques ne pouvaient répondre aux contraintes imposées par les caractéristiques de « haut niveau » des langages de programmation modernes, tels que l'inlining ou des appels mutuellement récursifs de fonctions. (*résultats soumis deux fois sans succès*)

En collaboration avec Franck Védrine (CEA), j'ai ensuite travaillé sur l'analyse de composants SystemC. J'ai contribué à la création d'un modèle formel basé sur les hypergraphes fractals pour les composants SystemC et la gestion des événements du scheduler (*résultat publié à Datics Imecs 2010*). En partant de ce modèle, j'ai mis en place une méthode d'extraction de comportement abstrait d'un composant SystemC sous la forme d'un hypergraphe fractal représentant une formule logique d'ordre supérieur. Cette méthode utilise principalement l'exécution symbolique pour les parties linéaires du code SystemC et des phases de généralisation ou d'abstraction pour passer les branchements, boucles et appels de fonction avec un coût en précision raisonnable (*résultat soumis sans succès*). En outre, j'ai également mis en place une sémantique sous-jacente aux hypergraphes fractals, présentant l'avantage d'être 'multi-niveau', et donc de pouvoir naviguer dans la hiérarchie des sémantiques définie par Cousot & Cousot aussi bien en 'abstraction' qu'en 'concrétisation'. Ce changement de sémantique minimise la perte d'information, tant que des contraintes externes fortes n'interviennent pas : il s'agit principalement la limite de l'espace mémoire sur nos serveurs de calcul. (*résultat soumis en attente*)

Perspectives envisagées :
L'utilisation des formules logiques d'ordre supérieur générées n'étant pas humainement facile à réaliser, il conviendrait de les employer dans un outil pour vérifier si le comportement abstrait extrait de la description SystemC du composant colle avec ce qu'on attend de lui. Il existe actuellement deux grandes familles d'outils qui sembleraient adaptés.

D'une part, on pourrait créer à partir du comportement abstrait du système un automate dont les transitions représentent les évolutions possibles du système. Les hypergraphes fractals pouvant être vus comme des automates hiérarchisés, cette transformation semble a priori assez naturelle. Il suffit ensuite de convertir les spécifications souhaitées pour le composant sous la forme d'une formule logique temporelle (LTL par exemple), et de faire tourner le **model-checker**, qui peut renvoyer un contre-exemple montrant en quoi le comportement abstrait ne respecte pas cette spécification. Si aucun contre-exemple n'est renvoyé, le comportement abstrait du composant SystemC satisfait donc la spécification. Etant donné que les abstractions employées dans la création de cette formule respecte la propriété de « sûreté », on pourra alors en déduire que le composant respecte également cette spécification.

D'autre part, étant donné que le comportement abstrait est exprimée via une formule logique d'ordre supérieur, il semble concevable d'utiliser des **assistants de preuves** (Coq ou Isabelle) pour essayer, en prenant ce comportement abstrait comme 'axiome', de démontrer la spécification souhaitée.