Transformation et vérification de logiciel C embarqué
Proposition de collaboration CEA-INRIA

Version préliminaire du 6 juillet 2005

Résumé

Dans le cadre d'une collaboration entre le laboratoire LSL du CEA et le projet EVEREST de l'INRIA Sophia-Antipolis, sur le thème de la vérification de programmes pour logiciels embarqués, ce document présente un programme de recherche qui définit les principaux objectifs de ce projet, et un certain nombre de travaux que l'INRIA propose de réaliser pour le CEA.

1 Contexte

1.1 Le projet EVEREST de l'INRIA

Le projet EVEREST vise à promouvoir l'utilisation des méthodes formelles dans le cadre de la sûreté et de la sécurité des systèmes. Les domaines d'application privilégiés sont les petits objets portables sécurisés, et les applications mobiles et embarquées. Son programme de recherche se focalise sur les trois thèmes suivants : plateformes sécurisées, vérification de programmes, et formalismes de spécification et vérification.

Le projet est impliqué dans de nombreuses collaborations académiques et industrielles. En particulier, le projet EVEREST est impliqué dans plusieurs projets européens, et coordonne notamment le projet intégré FET «MOBIUS : Mobility, Ubiquity, and Security».

Pour plus d'information visiter `http ://www-sop.inria.fr/everest/`

1.2 Le laboratoire LSL du CEA

L'activité principale du laboratoire LSL est la production de prototypes de d'outils de vérification de programme dans le but de les transférer à l'industrie. Sa double compétence en analyse statique (interprétation abstraite, preuve, analyse de flots) et en analyse dynamique (génération de test, simulation) lui a permis d'acquérir de bonnes connaissances dans le domaine de la sûreté du logiciel.

Le projet Caveat, par exemple, a conduit à développer un outil de preuve de programmes en C combinant des techniques élémentaires d'interprétation abstraite et un calcul de précondition à la Hoare afin de fournir à l'utilisateur un outil de vérification interactif.

Pour plus d'information visiter
`http ://www-list.cea.fr/fr/programmes/sys_embarques/labo_lsl/caveat/index.html`

2 Programme de recherches

L'objectif de cette collaboration est de concevoir et développer des outils pour l'analyse de programmes embarqués C, en s'appuyant sur des techniques de vérification de programmes comme la logique de Hoare et les calculs de préconditions, et les techniques d'analyse de programmes, comme l'interprétation abstraite et les analyses de flots de contrôle et de données.

2.1 Transformation de programmes

La complexité des programmes embarqués est souvent un facteur limitant pour la vérification formelle de propriétés de sûreté et sécurité. L'objectif de cet axe est d'utiliser des techniques de transformation de programmes permettant de réduire la complexité des programmes avant leur vérification. Plus précisément, il s'agit de construire, à partir d'un programme P et d'une propriété ϕ que P doit vérifier, un programme P' et une propriété ϕ' qui seront vérifiés par des outils automatiques ou à défaut interactifs. L'intérêt de cette approche est de réduire considérablement la complexité du travail de vérification, car le programme obtenu (ici P') par une transformation de programmes dirigée par l'objectif de vérification (ici la propriété ϕ) est généralement beaucoup plus simple que le programme original (ici P).

Nous proposons donc d'intégrer cette phase de transformation de programmes dans le processus de vérification de programmes C : nous nous intéresserons tout particulièrement aux transformations de type «slicing», qui permettent de restreindre le programme aux fragments de code dont dépend la propriété à vérifier. Ce travail est décrit plus précisément en 3.1.

Il s'agira également d'établir la correction de la méthode, c.à.d. de démontrer qu'il est légitime de vérifier la propriété transformée sur le programme transformé. Formellement, il s'agira d'établir que P satisfait ϕ dès lors que P' satisfait ϕ'. Ce point s'intègre dans l'étude plus générale sur les liens entre propriétés et le programme que nous nous proposons de réaliser, et dont les objectifs sont précisés en 3.3.

2.2 Combinaison de techniques d'analyse et de vérification de programmes

La vérification statique d'applications peut être effectuée par le biais de méthodes d'analyse de programme, qui sont automatiques, mais qui ne couvrent qu'un certain nombre de propriétés et peuvent être peu précises, ou par le biais de méthodes de vérification de programmes (logique de Hoare et calcul de préconditions), qui sont précises et peuvent être utilisées pour un grand nombre de propriétés, mais qui ne sont pas automatique et demandent une intervention manuelle parfois trop importante (en termes de temps et/ou niveau d'expertise).

Par ailleurs, le calcul de précondition dépend beaucoup du modèle choisi pour représenter la mémoire car celui-ci peut être plus ou moins abstrait, et il a donc une influence importante sur la précision des résultats.

L'objectif de cet axe est de combiner les techniques de vérification de programmes avec les techniques d'analyses de programmes, afin de fournir des méthodes de vérification qui sont à la fois précises et suffisamment automatiques.

Il s'agit dans un premier temps d'exprimer les propriétés à établir par les analyses de programmes sous forme d'assertions en logique de Hoare ; ce travail est facile pour certaines propriétés obtenues par des analyses de propagation de constantes (la variable x vaut 1 à un point de programme donné), mais beaucoup plus difficile pour d'autres propriétés (exécution de telle méthode utilisera telle quantité de mémoire). Le projet EVEREST dispose d'une expérience significative dans ce domaine, et a déjà abordé sous ce jour des propriétés liées au calcul de dépendances comme la non-interférence.

Il s'agira ensuite de raffiner les analyses statiques afin qu'elles prennent en compte les annotations qui seront fournies avec les programmes (ou synthétisées par des techniques automatiques), et inversement il s'agira d'intégrer les résultats de l'analyse sous forme d'assertions qui pourront simplifier les obligations de preuve engendrées par le calcul de préconditions.

Afin de garantir le passage à l'échelle de cette méthode, il faudra également envisager d'engendrer des certificats de correction de l'analyse, qui établissent la validité des assertions engendrées par l'analyse (et insérées dans le programme), et ce dans un format qui soit compréhensible par les outils utilisés pour décharger les obligations de preuve.

Ces objectifs seront atteints grâce aux travaux présentés en 3.2.

3 Description technique des tâches

3.1 Outil de réduction de logiciels

3.1.1 Spécifications de l'outil

Le CEA met actuellement en place un environement d'analyse de programme C permettant de lever des limitations de l'outil CAVEAT liées à certaines opérations complexes du langage comme les *cast* de pointeurs par exemple. Cette plateforme doit permettre de développer des outils d'analyses basés sur différentes techniques en favorisant une collaboration étroite entre les méthodes.

C'est dans ce cadre que nous proposons de réaliser un module de réduction de programmes en nous inspirant du prototype déjà développé dans l'environnement CAVEAT. Il s'agit principalement d'utiliser un modèle plus complet de la représentation des données, tout en en profitant pour intégrer de nouveaux aspects tant pratiques (optimisations) que théoriques (formalisation, traitement des points difficiles du langage).

Le travail de spécification consistera tout d'abord à définir l'algorithme de calcul du flot de données. Il faudra ensuite définir comment l'utiliser pour effectuer des sélections d'instructions en fonction de leurs dépendances, mais aussi comment transformer des instructions pour introduire des assertions lors de la coupure de branche ou faire des appels à des fonctions spécialisées par exemple.

Les spécifications obtenues feront l'objet d'un rapport R1 livré à M+6.

3.1.2 Développement du module

La plateforme développée au CEA est basée sur la bibliothèque CIL (*C Intermediate Langage*) librement mise à disposition par l'université de Berkley. L'ensemble est développé en OCAML.
Une première étape consistera donc à s'initier à cette bibliothèque et à la plateforme proposée.

Une première version V1 du module sera ensuite développée pour calculer le flot de données.

La version V2 devra permettre des réductions basées sur les dépendances, c'est-à-dire que les informations exploitées seront dans un premier temps peu liées à la sémantique du programme, mais plutôt à la structure du flot de données. Il faut néanmoins noter que des informations sur les alias sont indispensables à un tel calcul. Ces informations sont le résultats d'autres analyses implémentées dans la plateforme.

Le portage du prototype dans ce nouvel environnement est l'occasion de choisir de meilleures structures de donnée que celles utilisées dans le prototype, et en particulier l'introduire du partage pour ne pas consommer trop de mémoire lors de l'étude d'applications de taille industrielle.

La version V1 de l'outil sera livré à M+12, et la version V2 à M+24.

3.1.3 Utilisation des résultats d'autres analyses

L'utilisation des résultats de l'incontournable analyse d'alias a déjà été évoquée dans la partie précédente. D'autres analyses, comme la propagation de constante, peuvent être exploitée avec profit. Les différentes analyses à notre disposition devront être étudiées afin de voir si elles peuvent enrichir l'outil de réduction, et elles devront être utilisée le cas échéant.

Le rapport sera mis à jour en conséquence et les analyses seront connectées à l'outil version V3 livré à M+36.

3.2 Module de calcul de WP

Pour améliorer les résultats obtenus par preuve de programme, il faut considérer un modèle qui représente le plus fidèlement possible le comportement des lectures et écritures dans la mémoire. Le problème est que plus le modèle est précis, c'est-à-dire moins il est abstrait, et plus les calculs et les preuves deviennent complexes. Le compromis entre les deux semblant difficile à obtenir, nous proposons la réalisation d'un module de calcul de WP générique, paramétrable en fonction du modèle mémoire choisi.

Par ailleurs, le calcul peut être simplifié par l'utilisation des résultats d'autres analyses comme l'interprétation abstraite qui peut donner des indications relatifs à des invariants de boucles, ou encore la réduction de code qui peut indiquer les instructions qui peuvent être ignorées.

A l'inverse, les résultats obtenus à l'aide d'un calcul de WP peuvent être utilisés pour améliorer d'autres analyses. La coupure d'une branche lors de la réduction de programme peut par exemple permettre de déterminer des contraintes sur les conditions de chemin, qui pourront être propagées par WP afin de déterminer des préconditions de fonctions.

La tâches que nous proposons de réaliser sont les suivantes :
- spécification des opérations génériques d'un calcul de WP,
- spécification de la collaboration entre le WP et les autres analyses,
- développement du module générique de WP,
- application au modèle mémoire défini par le CEA.

Elles seront en partie réalisé en parallèle du développement de l'outil de réduction.

Ce travail donnera lieu à la livraison :
- d'un rapport R2 correspondant aux deux premier points à M+18,
- d'un outil WP à M+24.

3.3 Liens entre les propriétés et les transformations de programme

La question des liens entre les propriétés et le programme peut être vu selon deux points de vue :

D'une part, des propriétés initiales traduisent les questions que se posent l'utilisateur, alors que si l'on s'intéresse à la réduction de programmes, les critères généralement mis en oeuvre sont principalement :
- le calcul d'une donnée à un point de programme,
- le passage par une branche,
- ou une combinaison de ces critères.

Or l'objectif des utilisateurs s'expriment rarement dans ces termes, et il semble donc intéressant d'étudier comment traduire des requêtes de haut niveau en propriétés élémentaires sur le code source.

D'autre part, lorsque l'on travaille à l'analyse d'un programme, on calcule un grand nombre de propriétés, plus ou moins élémentaires, qui doivent participer à l'obtention du résultat. Or, lorsque l'on transforme le programme, que deviennent ces propriétés ? Comment assurer que les données étudiées sont cohérentes ?

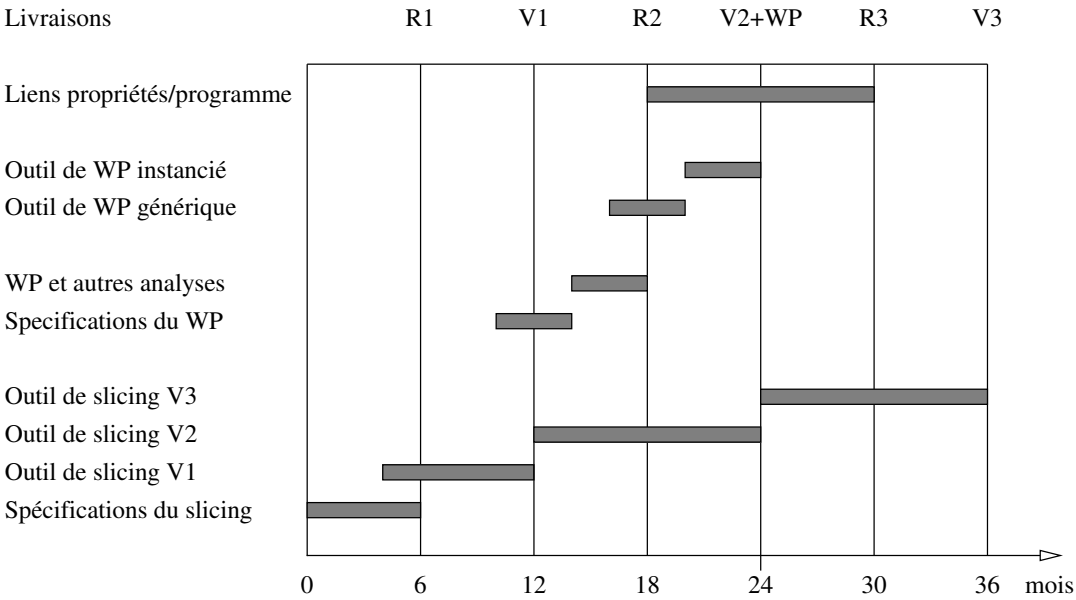
Une tâche consistera donc à étudier ces relations entre propriétés et programme transformé. Ce travail devra également permettre de justifier que les transformations effectuées lors d'une réduction sont valides par rapport aux critères définis.

Les résultats obtenus seront particulièrement utiles à la réalisation de la tâche 3.1.3.

Un rapport R3, livré à M+30, présentera les résultats de cette étude.

4 Organisation et planning

Les tâches seront réparties sur 3 ans à partir du XXX et réalisées suivant le calendrier suivant :



Les livraisons d'outil consistent à fournir un CD contenant le code source et la documentation correspondante. Les rapports seront réalisés au format LaTeX.

Des réunions régulières seront organisées, au moins à l'occasion de chaque livraison, avec éventuellement une réunion intermédiaire si nécessaire. Elles pourront, si besoin, faire l'objet d'une redéfinition des tâches ou du planning d'un commun accord. Elles donneront lieu à un compte-rendu de réunion qui résumera les décisions prises.

5 Coût et facturation

Le coût total de ces travaux est de xxx €HT, incluant les frais de mission correspondant aux réunions.

Chaque livraison fera l'objet d'une facturation correspondant à 1/6 de cette somme (soit xxx €) payable à xxx...

6 Propriété

Les modules développés dans le cadre de la présente proposition seront sous licence CeCILL (`http ://www.cecill.info/`), la licence française de logiciel libre élaborée par le CEA, le CNRS et l'INRIA.