


La session poster a réuni une quinzaine de posters et démos, installés le long de la "rue des posters" (merci à [Jean-Pierre Jacquot](#) pour cette photo).



Le prix du meilleur poster a été attribué à Pierre Neron pour son poster intitulé ["Elimination des racines et divisions pour du code embarqué"](#).



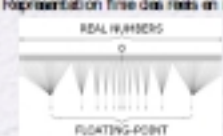
Inria

Elimination des racines et divisions pour du code embarqué

Pierre NERON
Inria/ École polytechnique

1. Contexte

- Code embarqué critique :
 - système ACCORD pour l'aéronautique (NASA Langley)
 - opérateur conditionnel (if then else)
 - programmes sans boucles
 - pas d'allocation dynamique de mémoire
- Arithmétique réelle $+$, $-$, \times , $/$, $\sqrt{\quad}$
- Représentation finie des réels en informatique :



$$\Rightarrow \sqrt{2} \times \sqrt{2} > 2$$
- $\sqrt{\quad}$ et / orientés des suites infinies :
 - $\sqrt{2} = 1.41421356237\dots$
 - $1/7 = 0.14285714285\dots$
- Arithmétique exacte avec $+$, $-$, \times :
 - entiers dynamiques
 - taille max par analyse statique

2. Spécification

Étant donné qu'il est possible de calculer exactement avec $+$, $-$, \times , le but est de définir une **transformation de programmes** qui :

- élimine les racines et les divisions
- préserve la sémantique lorsqu'il n'y a pas d'écarts

Si on ne peut pas toujours éliminer ces opérations (le programme $\sqrt{2}$ retournera toujours une valeur approchée), on peut en revanche les éliminer de valeurs booléennes et ainsi protéger le graphe de contrôle du programme des erreurs d'arronds.

3. Langage

Prog ::= Constant let Prog sup Prog (Prog, Prog) if Prog then Prog else Prog	Var and Prog Prog op Prog let Var = Prog in Prog
--	---

avec : Constant $\in \mathbb{R} \cup \{\text{True}, \text{False}\}$
 $op \in \{+, \times, /, =, \neq, >, \geq, <, \leq, \wedge, \vee\}$
 $sup \in \{\sqrt{\quad}, -, \neg\}$

4. Expressions booléennes

Soit $E_1 \mathcal{R} E_2$ une comparaison, on élimine racines et divisions en appliquant la transformation suivante qui définit la fonction el i rbool :

- Mettre les divisions en tête :

$$E_1 \mathcal{R} E_2 \rightarrow \frac{A}{B} \mathcal{R} \frac{C}{D}$$
- Éliminer les divisions de tête :

$$\frac{A}{B} \mathcal{R} \frac{C}{D} \rightarrow A.B.D^2 \mathcal{R} C.D.B^2$$
- Choisir une racine et factoriser :

$$A.B.D^2 \mathcal{R} C.D.B^2 \rightarrow P.\sqrt{Q} + R \mathcal{R} 0$$
- Éliminer la racine choisie : $P.\sqrt{Q} + R \mathcal{R} 0 \rightarrow$
 $(P \mathcal{R} 0 \wedge R \mathcal{R} 0) \vee (P \geq 0 \wedge P^2.Q - R^2 \mathcal{R} 0) \vee (R \geq 0 \wedge 0 \mathcal{R} P^2.Q - R^2)$
- Tant qu'il y a des racines, recommencer

5. Définitions de variables

Afin d'éviter que ces expressions booléennes ne dépendent de racines ou de divisions indirectement, on élimine également ces opérations des définitions de variable en utilisant un **let**ing partiel :

- let $x = a.b + \sqrt{(c+d)/7}$ in $P \rightarrow$
 let $(x_1, x_2, x_3) = (a.b, c+d, 7)$ in $P[x := x_1 + \sqrt{x_2/x_3}]$
- Nommer les sous-expressions qui ne contiennent ni racine ni division
- Éliminer le contexte qui les contient

6. Définitions avec conditionnelles

Cette notion d'**let**ing partiel peut s'étendre à des définitions de variables qui contiennent des tests :

- let $x = \text{if } F \text{ then } a/b \text{ else } c + \sqrt{2}$ in P

Le but est alors de trouver une représentation commune à toutes les expressions qui correspondent aux différents cas et d'éliminer cette expression :

- let $(x_1, x_2, x_3) = \text{if } F \text{ then } (a, b, 0) \text{ else } (c, 1, d)$ in $P[x := \frac{x_1 + x_2}{x_3}]$

Cette représentation commune provient d'une **anti-unification** avec contraintes des expressions correspondant aux différents cas des tests. Soient e_1, \dots, e_n un anti-unificateur de ces termes est un terme f tel que :

$$\forall i \in \{1, \dots, n\} \exists \alpha_i \in \text{Params}(Var), f.\alpha_i = e_i$$

Avec la contrainte que α_i ne contient ni racine ni division.

Cette anti-unification nous permet donc de définir une fonction el i rbool et $(x, p1, p2)$ qui renvoie x' , $p1'$, $p2'$ tels que :

$$\text{let } x = p1 \text{ in } p2 \xrightarrow{el} \text{let } x' = p1' \text{ in } p2'$$

où $p1'$ ne contient pas de racine.

7. Transformation complète

La transformation est donnée par la fonction récursive E i rbool :

- si p est une expression booléenne, retourner el i rbool (p)
- si p est une expression arithmétique, retourner p
- si $p = \text{let } x = p1 \text{ in } p2$:
 - $p1' := E$ i rbool ($p1$)
 - $x', p1', p2' := el$ i rbool et $(x, p1', p2)$
 - retourner $\text{let } x' = p1' \text{ in } E$ i rbool ($p2$)
- si $p = \text{if } F \text{ then } p1 \text{ else } p2$:
 - retourner $\text{if } E$ i rbool (F) then E i rbool ($p1$) else E i rbool ($p2$)

8. Conclusion

Nous avons donc conçu une transformation de programme qui permet d'éliminer les racines et les divisions de tous les booléens d'un programme. Cette transformation est **implémentable en OCaml**.

De plus nous avons :

- une spécification et la **preuve de correction** en PVS
- transposé cette spécification en une **tactique réflexive** qui permet de transformer automatiquement des buts dans PVS

9. Référence

P. Neron. A formal proof of square root and division elimination in embedded programs. In C. Hawblitzel and D. Miller, editors, CPP, volume 7679 of Lecture Notes in Computer Science, pages 259–272, 2012.

© 1997-2013 Inria. Tous droits réservés. <http://www.inria.fr>

Le programme a été présenté et clôturé les journées GPL 2013 au terme d'une semaine bien

