

Simplification d'expressions booléennes affines

Paul Feautrier

9 décembre 2011

Une expression booléenne affine est une expression booléenne dont les atomes sont des inégalités $f(x) \geq 0$ ou $f(x) > 0$. Les variables x sont entières ou rationnelles. La fonction f est l'addition d'une expression linéaire en x a coefficients entiers et d'une terme constant également entier. L'expression est construite à l'aide des opérateurs usuels \vee (ou), \wedge (et), et \neg (négation), etc. Quant les valeurs des variables sont données l'expression retourne une valeur de vérité, **true** ou **false**.

Les expressions booléennes affines apparaissent dans de nombreux champs de l'informatique, en particulier en analyse et vérification de programmes, en compilation et en optimisation, et en parallélisation automatique. Par suite des propriétés de l'algèbre de Boole, en particulier les lois de distributivité, les expressions que l'on est conduit à manipuler ont tendance à grossir et à s'encombrer de nombreuses redondances. Un outil de simplification est donc nécessaire.

Un tel outil a été réalisé au sein de Compsys [4]. Il est basé sur la représentation de l'expression à l'aide d'un arbre de décision [1] et cherche à éliminer un maximum d'atomes affines. Il peut être amélioré sur trois points :

- La taille de l'arbre de décision dépend très fortement de l'ordre dans lequel les atomes sont énumérés [3]. On propose de développer une méthode permettant d'atteindre ou de s'approcher de la taille minimale.
- L'algorithme ne cherche pas à optimiser la structure de l'expression simplifiée, au motif que les outils de synthèse matérielle le font efficacement [2]. Si cependant l'objectif est l'écriture d'un programme, une phase d'optimisation booléenne peut être utile. On pourra chercher par exemple à minimiser le nombre d'occurrence des atomes, ou le nombre d'instances des opérateurs booléens.
- L'algorithme supprime des atomes, mais n'en crée pas de nouveaux. Il est par exemple incapable de remplacer $x = 0 \vee x > 0$ par $x \geq 0$. On propose de rechercher une heuristique pour suggérer des candidats

intéressants, suivie d'une nouvelle tentative de simplification.

Par exemple, l'expression ci-dessus peut être écrite

```
if  $x \geq 0$       then  
    (if  $x = 0$  then   true else (if  $x > 0$  then true else false))  
    else  
    (if  $x = 0$  then   true else (if  $x > 0$  then true else false))
```

On peut ensuite s'apercevoir que si le premier test $x \geq 0$ est faux, la branche correspondante conduit obligatoirement à faux. De même, si ce test est vrai, et si $x \neq 0$ est faux, alors $x > 0$ est vrai, et donc la première branche conduit toujours à vrai. La formule se simplifie donc en **if** $x \geq 0$ **then** **true** **else** **false**, soit simplement $x \geq 0$. La difficulté est de deviner que c'est l'inégalité $x \geq 0$ qui conduit à la bonne simplification.

Références

- [1] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. on Computers*, C35(8) :677–691, August 1986.
- [2] Giovanni de Micheli. *Synthesis and Optimization of Digital Circuits*. MacGraw-Hill, 1994.
- [3] Rüdiger Ebendt, Görschwin Fey, and Rolf Drechsler. *Advanced BDD Optimization*. Springer, 2006.
- [4] Paul Feautrier. Simplification of Boolean Affine Formulas. Technical Report RR-7689, INRIA, July 2011.