Projet : Lustre Model Checker

Théotime Grohens

31 décembre 2016

1 Travail réalisé

Pour ce projet, j'ai implémenté un model checker Lustre basé sur la k-induction, tel que décrit dans la thèse de George Hagen. Le principe est d'essayer de prouver que la sortie d'un nœud Lustre donnée vaut toujours `true`. Pour cela, on traduit le programme Lustre en un ensemble $\Delta(n)$ de formules logiques décrivant l'état des variables du programme au temps n en fonction de leurs valeurs précédentes, et on essaye de prouver à l'aide d'un solveur SMT la propriété $P(n)$: la variable de sortie vaut `true`, pour tout n.

Le principe de la k-induction est de généraliser le principe de récurrence. Faire une récurrence pour démontrer $P(n)$ revient à montrer $\Delta(0) \models P(0)$ (l'initialisation), et $\Delta(n), P(n) \models P(n+1)$ (l'hérédité).

Cette technique est sûre : si elle parvient à prouver ces deux implications à l'aide du solveur, alors on a démontré que pour tout n, $P(n)$ est vraie. Si le cas de base est faux, elle nous fournit de plus un contre-exemple à la propriété, c'est-à-dire une trace d'exécution pour laquelle la propriété n'est pas vérifiée. Elle n'est toutefois pas complète : il existe des programmes qui vérifient la propriété, mais pour laquelle on n'arrive pas à prouver l'hérédité.

Pour améliorer la méthode, on essaye donc de faire une récurrence qui s'appuie sur les k précédentes valeurs de Δ (et donc les k derniers états du programme), au lieu d'utiliser juste le dernier. On essaye donc de prouver comme cas de base que $\Delta(0), \ldots, \Delta(k) \models P(0) \ \wedge \ldots \wedge P(k)$, et comme cas inductif que $\Delta(n), \ldots, \Delta(n+k+1), P(n), \ldots, P(n+k) \models P(n+1)$.

L'idée est que le cas inductif sera plus facile à prouver si l'on fait plus d'hypothèses (sur les états précédents du programme) ; pour être capable de faire ces hypothèses supplémentaires, il faut toutefois prouver un cas de base plus puissant.

L'algorithme du model checker est donc simple : il essaie de prouver le programme par k-induction, avec k croissant, jusqu'à soit prouver cas de base et cas inductif, soit trouver un cas de base faux, soit atteindre un k_{max} après lequel on abandonne.

2 Choix techniques

Mon programme utilise le solveur SMT Z3. Les étapes de l'algorithme sont les suivantes :

— *Inliner* les nœuds Lustre appelés par le nœud à vérifier, en prenant garde à renommer les variables pour éviter les conflits

— Créer des flux auxiliaires afin de supprimer les constructeurs `pre` imbriqués, de sorte que l'état des flux au temps n ne dépende explicitement que du temps $n-1$

— Transformer récursivement les parties droites des équations (expressions) en termes du solveur

— Construire $\Delta(n)$, la conjonction des définitions de variables `x = e` vues comme des formules, et $P(n)$

— Essayer de prouver les cas de base et cas inductif de la k-induction, pour k allant de 0 à k_{max}

3 Extension

En plus de l'algorithme de base, j'ai implanté l'extension de compression de chemins décrite dans la thèse. Celle-ci rend le k-model checker complet pour les systèmes ayant au plus k états distincts.

Cependant, cette extension n'est pas suffisante pour prouver que le programme `counter.lus` est correct, car le checker ne devine pas le fait que la variable `time` est à valeurs dans $\{0, \dots, 3\}$ (et donc que l'ensemble des états est bien fini).

4 Résultats

Le model checker arrive à prouver des exemples simples ne nécessitant que de la 0-induction (soit une récurrence simple), mais aussi des exemples plus complexes nécessitant d'utiliser des k non nuls. Par exemple, le programme `ex003.lus` nécessite une 1-induction.