Software Verification

2024-2025

TP2: Sémantique

Vincent Penelle

Points abordés

- Programmation de la sémantique d'un control-flow automata.

Le but de ce TP consiste à vous faire programmer le module CommandSemantics.ml de notre outil Simple Program Analyser. Plus précisément, vous aurez uniquement à fournir la traduction de chaque commande en une formule décrivant son effet sur une configuration du control-flow automaton.

Rappel:

Un *control-flow automaton* est un tuple $(Q, q_i, q_{bad}, \Delta)$ où Q est un ensemble fini d'états, $q_i \in Q$ est l'état initial, q_{bad} est l'état terminal (ou «mauvais»), et $\Delta \subseteq Q \times \text{Op} \times Q$ est l'ensemble des transitions, où Op est l'ensemble des opérations possibles de l'automate. Cet ensemble est défini ci-après. Autrement dit, c'est un automate étiqueté par des opérations (sur des variables).

On considère un ensemble de variables X. Une *expression* est une expression arithmétique sur $\mathbb{Z}$ contenant éventuellement des variables:

$$\texttt{exp} ::= n \in \mathbb{Z} | x \in X | \texttt{exp} + \texttt{exp} | \texttt{exp} - \texttt{exp} | \texttt{exp} \times \texttt{exp} | \texttt{exp} / \texttt{exp}$$

Une *garde* est une comparaison entre deux expressions:

$$\texttt{guard} ::= \texttt{exp} = \texttt{exp} | \texttt{exp} < \texttt{exp} | \texttt{exp} \leq \texttt{exp} | \texttt{exp} > \texttt{exp} | \texttt{exp} \geq \texttt{exp}$$

Une opération peut être soit `skip`, une garde ou une affectation de la forme $x := exp$, pour $x \in X$ et exp une expression arithmétique:

$$op ::= \texttt{skip} | \texttt{guard} | x := \texttt{exp}$$

Point technique:

Une *configuration* d'un control-flow automaton $\mathcal{A}$ est un couple (q, σ), où q est un état, et σ est une fonction de X dans $\mathbb{Z}$ appelée *valuation*.

La *sémantique* d'une opération op est une relation qui contient les valuations (σ_1, σ_2) telles que σ_2 peut être obtenue en appliquant op à σ_1. Formellement, on a $[\![\texttt{skip}]\!] = \text{id}$, $[\![\text{guard}]\!]$ est l'ensemble des (σ, σ) tels que σ satisfait la garde, et $[\![\texttt{x} := \texttt{exp}]\!] = \{(\sigma, \sigma[x := exp])\}$.

Plus précisément, ici cette sémantique va être décrite par une formule de la logique du premier ordre sur $\mathbb{Z}$, qui va faire intervenir deux copies de X, X_b et X_a représentant respectivement les valuations avant (before) et après (after) l'opération.

Les sémantiques des opérations sont les suivantes :

$$[\![\texttt{skip}]\!] = \bigwedge_{x \in X} x_b == x_a$$

$$[\![\texttt{guard}]\!] = \texttt{side}_b(\texttt{exp}_1) \wedge \texttt{side}_b(\texttt{exp}_2) \wedge \texttt{guard}_b \wedge \bigwedge_{x \in X} x_b == x_a$$

$$[\![x := \texttt{exp}]\!] = x_a == \texttt{exp}_b \wedge \texttt{side}_b(\texttt{exp}) \wedge \bigwedge_{y \in X, x \neq y} y_b == y_a$$

où $\texttt{exp}_b$, $\texttt{side}_b$ et $\texttt{guard}_b$ désignent ces expressions sur la copie X_b de X, $\texttt{exp}_1$ et $\texttt{exp}_2$ sont les deux membres de `guard` et `side(exp)` est une formule assurant que l'expression est bien définie, c'est-à-dire qu'aucune division par zéro n'intervient. Ceci est nécessaire, car Z3 considère que la division par zéro est définie, mais n'a pas de valeur définie (donc si vous ne mettez pas cette restriction la valeur d'une division par zéro ne sera pas contrôlable, mais existera).

Exercice 1: CommandSemantics.ml

Récupérez l'archive contenant le code de ce TP. Complétez le module CommandSemantics.ml, en vous aidant des indications ci-dessus, et des indications données dans CommandSemantics.mli qui vous indiquent la syntaxe employée dans le projet et reprécisent le rôle des fonctions.

Point important : vous devrez retourner un résultat de la forme $(f, list)$, où *list* sera la liste des variables *modifiées* par l'opération (c'est-à-dire vide pour `skip` et `guard` et égale à $[x]$ pour $x :=$ `exp`), et f sera la formule décrite plus haut, *sans la partie de droite (en rouge) indiquant les égalités entre variables non-modifiées* (cela permet d'abstraire cette partie et de vous donnez une variante plus efficace de cette partie de formule lorsque nous traiterons les exécutions).

1. Commencez par coder formula_of_skip.
2. Définissez récursivement la formule `side` (sur papier). Cette formule doit être vraie si et seulement si les valeurs des variables font que l'expression ne contiendra aucune division par zéro.
3. Codez expr_to_z3_expr. Cette fonction renvoie une formule représentant l'expression passée en argument et une liste de formules représentant les différentes parties de `side` (cela permettra de faire un seul ET à la fin).
4. Finissez en codant formula_of_guard, fwd_formula_of_assign et bwd_formula_of_assign. Notez que dans les formules, nous n'utilisons que des opérations symétriques (ceci devrait vous aider à déduire la formule en arrière simplement de la formule en avant).

À tout moment au long du TP, vous pourrez appeler `make test` pour tester votre code (il vous dira si vos formules correspondent à ce qui est attendu et affichera les éventuelles différences – notez que certaines différences ne sont pas graves si cela concerne uniquement l'ordre). Pensez toutefois à remplacer les failure avec des formules simples pour que le programme se lance (par exemple, vous pouvez mettre la même chose que pour formula_of_skip).