Model checker pour un sous-ensemble de Lustre

Projet systèmes synchrones

Victor Nicolet

December 18, 2015

Plan

Transformation : lustre $\rightarrow$ formule logique

Model-checking

Extensions

Démo

Mini lustre

↓

Mini-lustre normalisé

↓

Formule dans Alt-Ergo-Zero

↓

Model checking avec k-induction

Mini lustre → Mini lustre normalisé

Eliminer :

- les appels de noeuds en les inlinant
- les tuples

```
(a, b, c) = (true, false, false) -> pre (x, y, z);
```

deviennent

```
a = true -> pre x;
b = false -> pre y;
c = false-> pre z;
```

Plan

Transformation : lustre → formule logique

Model-checking

Implémentation

Extensions

Démo

Mini-lustre normalisé → Formules d' ALT-ERGO-ZERO

$$\langle ident \rangle \rightarrow (int \rightarrow term)$$

$$\langle expr \rangle \rightarrow \begin{cases} term \\ formula \end{cases}$$

$$\langle eqn \rangle := (\langle ident \rangle = \langle expr \rangle) \rightarrow \begin{cases} term1 = term2 \\ formula1 \Leftrightarrow formula2 \end{cases}$$

$$\langle node \rangle \rightarrow \bigwedge(formula)$$

Invariant à vérifier P_n, modèle du noeud Δ_n.

k-induction

k-induction : P est invariant pour le noeud si on a $k \geq 0$ tel que :

$$\Delta_0 \wedge \Delta_1 \wedge ... \wedge \Delta_k \vDash P_0 \wedge P_1 \wedge ... \wedge P_k$$

$$\begin{array}{l} \Delta_n \wedge \Delta_{n+1} \wedge ... \wedge \Delta_{n+k} \wedge \Delta_{n+k+1} \wedge \\ P_n \wedge P_{n+1} \wedge ... \wedge P_{n+k} \end{array} \vDash P_{n+k+1}$$

- Correct
- Incomplet : on peut chercher k longtemps ...

Cas de base

```
let ind_case n delta_incr ok k =
  let kt = tofi k in
  let kt_p_k = tplus n kt in
  IND_solver.assume ~id:0 (delta_incr (tplus kt_p_k one))
    ;
  IND_solver.assume ~id:0 (ok kt_p_k);
  IND_solver.check ();
  IND_solver.entails ~id:0 (ok (tplus kt_p_k one))
```

Cas inductif

```
let init_ind_case n delta_incr =
  IND_solver.assume ~id:0
     (Formula.make_lit Formula.Le [zero; n]);
  IND_solver.assume ~id:0 (delta_incr n);
  IND_solver.check ()

let ind_case n delta_incr ok k =
  let kt = tofi k in
  let kt_plus_n = tplus n kt in
  IND_solver.assume ~id:0
      delta_incr (tplus kt_plus_n one));
  IND_solver.assume ~id:0 (ok kt_plus_n);
  IND_solver.check ();
  IND_solver.entails ~id:0 (ok (tplus kt_plus_n one))
```

Plan

Transformation : lustre → formule logique

Model-checking

Extensions

Démo

Boucles

Dans le cas de base si :

$$\Delta_0 \wedge \Delta_1 \wedge ... \wedge \Delta_k \models \neg C_{0,k+1}$$

(le chemin de 0 à k+1 est compressible)
on peut affirmer que P est un invariant du noeud.

```
let check_noloop prev_states states delta_incr kt =
  /* */
  LOOP_solver.assume ~id:0 (delta_incr kt);
  (LOOP_solver.entails ~id:0 loop_condition), new_states
```

⇒ Model checking complet !

Dans le cas inductif :

$$\begin{array}{l} \Delta_n \wedge \Delta_{n+1} \wedge ... \wedge \Delta_{n+(k+1)} \wedge \\ P_n \wedge P_{n+1} \wedge ... \wedge P_{n+k} \wedge C_{n,k} \end{array} \models P_{n+(k+1)}$$

on vérifie qu'il y a un chemin non-compressible à chaque étape.

Plan

Transformation : lustre $\rightarrow$ formule logique

Model-checking

Extensions

Démo

Démo