Progetto 1 - Safety

Simone Ballarin, Paolo Eccher, Alessio Gobbo

Team Pirati Ciber-Fisici

1 Codice

L'implementazione della funzione `check_explain_inv_spec(spec)` corrisponde pressoché con l' implementazione del Symbolic Breadth-First-Search Algorithm.
Quest'ultimo si appoggia alla subroutine ricorsiva `path_to(fsm, reach, state_bdd)` così da restituire anche una traccia d'esecuzione che funge da controesempio per la specifica in questione.

1.1 check_explain_inv_spec(spec)

La funzione verifica se, in ogni stato raggiungibile dalla macchina FSM, la proprietà **spec** vale. L'algoritmo, partendo dagli stati iniziali verifica la completa conformità alla specifica in maniera iterativa, esplorando tutti gli stati raggiungibili. Più precisamente, all'iterazione i-esima la variabile $reach_i$ contiene gli stati il cui percorso più breve contiene al più i transizioni.
Ad ogni iterazione i si calcola quindi l'insieme new_i contenente gli stati raggiungibili con minimo i transizioni. Questo si ottiene computando la post-immagine degli stati new_{i-1} e quindi sottraendogli $reach_{i-1}$ (riga 17).
La verifica dell'invarianza avviene quindi verificando che ogni regione di stati *new* abbia intersezione vuota con la negazione di `spec` (righe 11-12).

Se questo risultasse non vero implicherebbe l'esistenza di uno stato in *new* in cui la negazione è soddisfatta. Tali stati sono denominati `unhappy_states`. L'avere almeno uno stato infelice e raggiungibile comporta la non validità dell'invariante `spec`, quindi si setta la variabile di guardia `found_unhappy_state` a `True` (riga 13).

Se non si ritorna mai `False`, ad un certo punto, presupponendo di avere cardinalità di stati finita, la variabile *new* non conterrà nessuno stato. Questo significa che la nostra ricerca è stata esaustiva e quindi, non avendo trovato stati infelici, possiamo affermare l'invarianza di `spec` e ritornare la negazione di `found_unhappy_state`, che assumera' valore booleano `False` (riga 19) .

Le righe 2 e 3 sono, invece, di carattere piu' implementativo e dovute al funzionamento della libreria; infatti `PyNuSMV` mantiene le informazioni relative al programma da analizzare tramite l'uso di variabili globali. Attraverso la chiamata `pynusmv.glob.prop_database().master.bddFsm` viene quindi ottenuto un diagramma di decisione binaria corrispondente alla macchina a stati finiti codificata nel file. Questo bdd viene quindi puntato dalla variabile `fsm`.

```
1  def check_explain_inv_spec(spec):
2      fsm = pynusmv.glob.prop_database().master.bddFsm
3      spec = spec_to_bdd(fsm,spec)
4
5      reach = fsm.init
6      new = fsm.init
7      n = fsm.count_states(reach)
8      found_unhappy_state = False
9      counterexample = None
10     while fsm.count_states(new)>0 & found_unhappy_state==False:
11         unhappy_states = new*(~spec)
12         if fsm.count_states(unhappy_states)>0 :
13             found_unhappy_state = True
14             unhappy_state = fsm.pick_one_state(unhappy_states)
15             counterexample = path_to(fsm,reach,unhappy_state)
16
17         new = fsm.post(new) - reach
18         reach = reach + new
19     return (not found_unhappy_state,counterexample)
```

Listing 1: Funzione check_explain_inv_spec

1.2 path_to

La funzione ricorsiva `path_to` datogli in input uno stato infelice cerca procedendo a ritroso una possibile sequenza di stati che parta da uno stato iniziale e arrivi allo stato infelice.
La funzione ritorna una sequenza di stati solo se lo stato passato è effettivamente raggiungibile. In caso contrario lancerà un'eccezione.
Più dettagliatamente, se `state_bdd` (lo stato che voglio ricorsivamente raggiungere) è vuoto vuol dire che lo stato è non raggiungibile, quindi viene lanciata un'eccezione (riga 2).
Se lo stato attuale è uno stato iniziale allora la ricerca è finita (riga 5). In caso contrario, dalla sua pre-immagine intersecata con gli stati raggiungibili si estrae uno stato qualsiasi (riga 6) e quindi si effettua la chiamata ricorsiva su questo stato. Questa chiamata, per ipotesi induttiva, ritornerà una sequenza di stati che porta allo stato selezionato. Sapendo che lo stato selezionato e' un predecessore

dello stato che vogliamo raggiungere possiamo quindi ritornare come soluzione del problema la concatenazione della lista ottenuta dalla chiamata ricorsiva con lo stato corrente (riga 7). Questa lista quindi rappresenta una della possibili tracce $[init \rightarrow^* unhappy\_state]$.

```
1 def path_to(fsm, reach, state_bdd):
2     if fsm.count_states(state_bdd) == 0 : raise Exception('no reachable state')
3     i = fsm.count_states(state_bdd * fsm.init)
4     if i>0 :
5         return [state_bdd.get_str_values()]
6     pred = fsm.pick_one_state(fsm.pre(state_bdd)*reach)
7     return path_to(fsm,reach,pred)+[state_bdd.get_str_values()]
```

Listing 2: Funzione path_to

2 PyNuSMV

E' stato fatto uso dei seguenti moduli:

- `pynusmv.glob` per il recupero del BDD rappresentante il modello corrente;
- `pynusmv.fsm` per i metodi della classe BddFsm (e.g. il calcolo della pre/post immagine di un BDD);
- `pynusmv.dd` per le operazioni fra BDD definite come operator overloading nella classe BDD.

3 Railroad

Il codice implementato e' stato testato tramite l'uso dell'esempio visto in classe del `Railroad system`. In particolare abbiamo utilizzato `railroad_wrong.smv` e `railroad.smv` (una versione corretta del controllore).
`railroad_wrong.smv` e' la prima versione del controllore in cui i due treni possono ritrovarsi entrambi sul ponte.
Questo e' dovuto al fatto che il controllore in caso di arrivi contemporanei fa' passare un treno e poi dimentichi che l'altro e' in attesa. Quindi appena il ponte viene liberato, il treno in attesa entra senza che il controllore lo sappia (e quindi senza che venga messo il semaforo rosso, dando l'opportunita' di passare anche al secondo treno). La versione corretta invece, rispetto a `railroad_wrong.smv`, utilizza una variabile booleana in aggiunta al controllore (`near_e` e `near_w`) per evitare di dimenticarsi dei treni in attesa di entrare nel ponte.
Abbiamo verificato l'invarianza della seguente specifica: $\neg(train_w.mode = bridge \wedge train_e.mode = bridge)$. A causa del problema precedentemente citato ci aspettatiamo, quindi, che la verifica di invarianza rispetto a `railroad_wrong.smv` dia esito negativo e che ritorni un possibile controesempio. Infatti nel Listing 3 si puo' osservare l'output del programma con controesempio.

```
1 >>python3 inv_mc.py "railroad_wrong.smv"
2 >>Property !(train_w.mode = bridge & train_e.mode = bridge) is an INVARSPEC.
3 >>Invariant is not respected
```

```
>>[{'train_w.mode': 'away', 'train_w.out': 'arrive', 'train_e.mode': 'away', 'train_e
    .out': 'arrive', 'contr.signal_e': 'green', 'contr.signal_w': 'green', 'contr.
    west': 'green', 'contr.east': 'green'}, {'train_w.mode': 'wait', 'train_w.out': '
    none', 'train_e.mode': 'wait', 'train_e.out': 'none', 'contr.signal_e': 'green',
    'contr.signal_w': 'red', 'contr.west': 'red', 'contr.east': 'green'}, {'train_w.
    mode': 'wait', 'train_w.out': 'none', 'train_e.mode': 'bridge', 'train_e.out': '
    leave', 'contr.signal_e': 'green', 'contr.signal_w': 'red', 'contr.west': 'red',
    'contr.east': 'green'}, {'train_w.mode': 'wait', 'train_w.out': 'none', 'train_e.
    mode': 'away', 'train_e.out': 'arrive', 'contr.signal_e': 'green', 'contr.
    signal_w': 'green', 'contr.west': 'green', 'contr.east': 'green'}, {'train_w.mode
    ': 'bridge', 'train_w.out': 'none', 'train_e.mode': 'wait', 'train_e.out': 'none'
    , 'contr.signal_e': 'green', 'contr.signal_w': 'red', 'contr.west': 'red', 'contr
    .east': 'green'}, {'train_w.mode': 'bridge', 'train_w.out': 'none', 'train_e.mode
    ': 'bridge', 'train_e.out': 'none', 'contr.signal_e': 'green', 'contr.signal_w':
    'red', 'contr.west': 'red', 'contr.east': 'green'}]
```

Listing 3: Esempio di esecuzione con output del terminale. Questa esecuzione riguarda il modello "railroad_wrong.smv"

Invece, la verifica sul file corretto da esisto positivo.

```
>>python3 inv_mc.py "rail_road.smv"
>>Property !(train_w.mode = bridge & train_e.mode = bridge) is an INVARSPEC.
>>Invariant is respected
```

Listing 4: Esempio di esecuzione con output del terminale. Questa esecuzione riguarda il modello "railroad.smv" in cui - giustamente - non viene trovato nessun controesempio all'invariante.