Paper Code: COMP711

Theory of Computation

Lecturer: Ji Ruan

Assignment 2

Due 11:59pm, Monday 23 October 2017

Name ..... **ID number.....**

Task	Marks Possible	Marks Given
1	70	
2	30	
Total	100	
Bonus	10	

Instructions:

Please attach this sheet to the front of your assignment.

The assignment must be submitted before 11:59 p.m. on **Monday 23 October 2017**, via Blackboard.
It is your responsibility to keep a copy of your assignment.

Plagiarism has occurred where a person effectively and without acknowledgement presents as their own work the work of others. That may include published material, such as books, newspapers, lecture notes or handouts, material from the www or other students' written work. That work also includes computer output. The School of Engineering, Computer and Mathematical Sciences regards any act of cheating including plagiarism, unauthorised collaboration and theft of another student's work most seriously. Any such act will result in a mark of zero being given for this part of the assessment and may lead to disciplinary action.

Please sign to signify that you understand what this means, and that the assignment is your own work.

Signature: .....

Model Checking Invariants

This assignment aims to provide you a practical exercise of the model checking process by checking invariant properties.

Instructions

- You may collaborate with (at most) one other student to complete this assignment.
- Each team shall put all files in a folder with name COMP711-2017A2-XY where XY refers to your ID(s). Submit the .zip format of this folder via Blackboard.
- Inside each file, please clearly indicate your names and student IDs
- You must clearly describe the main methods of your program (i.e., methods that implement the main algorithms, etc)

Marking Guide

The total marks of this assignment is 100. Marks will be allocated based on the correctness, clarity and originality of your code.

Task 1 Invariant Checking

(a) Implement the invariant checking algorithm by forward DFS

Input: finite transition system TS and propositional formula ϕ

Output: "yes" if $TS \models$ "$always\ \phi$", otherwise "no" plus a counterexample.

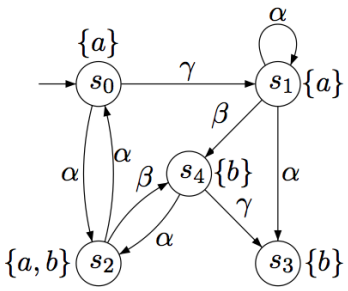
The detail of the algorithm is given bellow.

The propositional formula ϕ is given by the following BNF:

$\phi ::= true \mid P \mid \mathsf{NOT}\ \phi \mid \phi\ \mathsf{AND}\ \phi \mid \phi\ \mathsf{OR}\ \phi \mid \phi\ \mathsf{IMP}\ \phi$ where $P \in AT$, IMP refers to implication.

(b) Give an example (including TS and ϕ) where your implementation outputs "yes" and another example where your implementation outputs "no" with a counterexample. An example should have at least 15 states.

Here is a small example TS:



You program shall be able to read the following encoding of the TS: (note that α can be represented as a string $alpha$).

$S = \{s_0, s_1, s_2, s_3, s_4\}$

$Act = \{\alpha, \beta, \gamma\}$

$- >= \{< s_0, \alpha, s_2 >, < s_0, \gamma, s_1 >, < s_1, \alpha, s_1 >, < s_1, \alpha, s_3 >, < s_1, \beta, s_4 >, < s_2, \alpha, s_0 >, < s_2, \beta, s_4 >, < s_4, \alpha, s_2 >, < s_4, \gamma, s_3 >\}$

$I = \{s_0\}$

$AP = \{a, b\}$

$L(s_0) = \{a\}, L(s_1) = \{a\}, L(s_2) = \{a, b\}, L(s_3) = \{b\}, L(s_4) = \{b\}$

And following encoding of propositional formulas

a OR b

a IMP (NOT b)

Algorithm 4 Invariant checking by forward depth-first search

Input: finite transition system TS and propositional formula Φ
Output: "yes" if $TS \models$ "always Φ", otherwise "no" plus a counterexample

```
set of states R := ∅;                          (* the set of reachable states *)
stack of states U := ε;                        (* the empty stack *)
bool b := true;                                (* all states in R satisfy Φ *)
while (I \ R ≠ ∅ ∧ b) do
  let s ∈ I \ R;                               (* choose an arbitrary initial state not in R *)
  visit(s);                                    (* perform a DFS for each unvisited initial state *)
od
if b then
  return("yes")                                (* TS ⊨ "always Φ" *)
else
  return("no", reverse(U))                     (* counterexample arises from the stack content *)
fi
```

```
procedure visit (state s)
  push(s, U);                                  (* push s on the stack *)
  R := R ∪ { s };                              (* mark s as reachable *)
  repeat
    s' := top(U);
    if Post(s') ⊆ R then
      pop(U);
      b := b ∧ (s' ⊨ Φ);                       (* check validity of Φ in s' *)
    else
      let s'' ∈ Post(s') \ R
      push(s'', U);
      R := R ∪ { s'' };                        (* state s'' is a new reachable state *)
    fi
  until ((U = ε) ∨ ¬b)
endproc
```

Task 2 Experiments

Consider the following generalization of Peterson's mutual exclusion algorithm that is aimed at an arbitrary number $n (n \geq 2)$ processes. The basic concept of the algorithm is that each process passes through n "levels" before acquiring access to the critical section. The concurrent processes share the bounded integer arrays $y[0..n-1]$ and $p[1..n]$ with $y[i] \in \{1, ..., n\}$ and $p[i] \in \{0, ..., n-1\}$. $y[j] = i$ means that process i has the lowest priority at level j, and $p[i] = j$ expresses that process i is currently at level j. Process i starts at level 0. On requesting access to the critical section, the process passes through levels 1 through $n-1$. Process i waits at level j until either all other processes are at a lower level (i.e., $p[k] < j$ for all $k \neq i$) or another process grants process i access to its critical section (i.e., $y[j] \neq i$). The behavior of process i is in pseudocode:

```
while true do
  ... noncritical section ...
  forall j = 1, ..., n−1 do
      p[i] := j;
      y[j] := i;
      wait until (y[j] ≠ i) ∨ (⋀_{0<k≤n, k≠i} p[k] < j)
  od
  ... critical section ...
  p[i] := 0;
od
```

Do the following subtasks:

(a) Give the program graph for process i.

(b) Write a program to produce the transition system of the parallel composition of n processes.

Input: n - the number of processes

Output: the transition system $TS_{Pet(n)} = (S, Act, \rightarrow, I, AP, L)$ the parallel composition of n processes.

(Act is of your choice; we assume that the initial values of the arrays $y[0..n-1]$ and $p[1..n]$ are minimal ones in their domains. Let $AP = \{n_i, w_i, c_i \mid 0 < i \leq n\}$)

(c) Feed the transition system $TS_{Pet(n)}$ to the program developed in Task 1 and write an experimental report including the following:

- Formulate the invariant property "no more than 1 process is in the critical section" as a propositional formula.
- Estimate the size of $TS_{Pet(n)}$ (number of states, and number of transitions) in terms of process number n.
- Measure the time and space used by the invariant checking algorithm by variating the number of process n.

Bonus Task

Implement an algorithm for invariant checking such that in case the invariant is refuted, a minimal counterexample, i.e., a counterexample of minimal length, is provided as an error indication.

This task has 10 marks.