Prof. Dr. Andreas Podelski
Dominik Klumpp

Hand in until January 8th, 2020
15:59 via the post boxes
Discussion: January 15th, 2020

Tutorial for Cyber-Physical Systems - Discrete Models
Exercise Sheet 10

Exercise 1: Linear-Time Properties — 13 Points

The goal of this exercise is to help you better understand the representation of properties as sets of traces, as well as the notion of satisfaction by a state or a transition system.

Assume $AP = \{a, b\}$. For each of the properties P_i, complete the following tasks:

(a) Formalize P as a set of traces using set comprehension.
For example: "always a" can be formalized as $\{A_0A_1A_2\cdots \mid \forall i.\, a \in A_i\}$.

(b) Give an example of a trace that satisfies P.

(c) Give an example of a trace that does not satisfy P.

(d) Give all states of the transition system below that satisfy P.

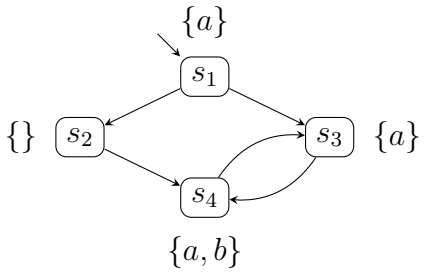
(e) State whether or not the transition system below satisfies P.

(f) Is the property a safety property? If so, give the set of all bad prefixes.
For example, the bad prefixes of "always a" can be given as

$$\mathit{BadPrefix}_{\text{always } a} = \{A_0A_1\ldots A_n \mid \exists i \in \{0,\ldots,n\}\,.\, a \notin A_i\}$$

(g) Is the property an invariant property? If so, give the invariant condition as a propositional logic formula.

(h) Is the property a liveness property? If so, show that no prefix is *"bad"*: Explain how any prefix $A_0A_1\ldots A_n$ can be continued to form a trace that satisfies the property.
For example, for the property "eventually (at some point in time) a", any prefix $A_0A_1\ldots A_n$ can be continued as $\pi = A_0A_1\ldots A_n\,\{a\}^\omega$. Then $\pi \models$ "eventually a".



(P_1) Always (at any point of time) a or b holds.

(P_2) Always (at any point of time) a and b holds.

(P_3) Never b holds before a holds.

(P_4) Every time a holds there will be eventually a point of time where b holds.

(P_5) At exactly three points of time, a holds.

(P_6) If there are infinitely many points of time where a holds, then there are infinitely many points of time where b holds.

(P_7) There are only finitely many points of time where a holds.

Exercise 2: Safety & Liveness 8 Points

Consider the categories (a)-(d) of linear-time properties below. For each of them, give two examples of a properties that fall into this category: a safety property and a liveness property (if such examples exist). If no example exists, argue why this is the case.

⚠ Careful reading required. ⚠

(a) Properties E where, for every trace $\pi = A_0A_1A_2\ldots$ with $\pi \models E$, it is sufficient to examine a finite prefix $A_0A_1\ldots A_n$ of π to determine that π satisfies property E.

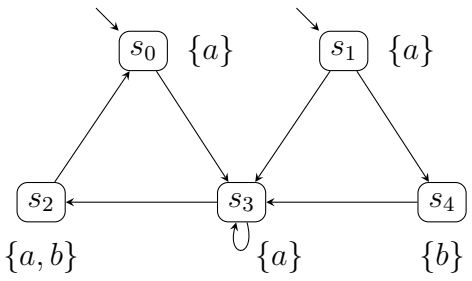
(b) Properties E where it is **not** sufficient for every trace $\pi = A_0A_1A_2\ldots$ with $\pi \models E$ to examine a finite prefix $A_0A_1\ldots A_n$ of π to determine that π satisfies property E. Instead, the whole trace must be examined.

(c) Properties E where, for every trace $\pi = A_0A_1A_2\ldots$ with $\pi \not\models E$, it is sufficient to examine a finite prefix $A_0A_1\ldots A_n$ of π to determine that π violates property E.

(d) Properties E where it is **not** sufficient for every trace $\pi = A_0A_1A_2\ldots$ with $\pi \not\models E$ to examine a finite prefix $A_0A_1\ldots A_n$ of π to determine that π violates property E. Instead, the whole trace must be examined.

Exercise 3: Invariant checking I 4 Points

In the lecture, you have seen an algorithm for invariant checking by forward depth-first search. This algorithm is displayed in algorithm 1.
Apply this algorithm to the following transition system whose set of atomic propositions is $AP = \{a, b\}$. The invariant Φ to be checked is the propositional logical formula a.



Whenever you iterate over a set of states, always take state s_i before state s_j if i is smaller than j.
Present the execution of the algorithm by writing down the contents of the set U and the stack π directly before every call to the function **DFS**.

Algorithm 1: DFS-based invariant checking

```
input  : a finite transition system 𝒯 and a propositional formula Φ
output: "yes" if 𝒯 ⊨ "always Φ", otherwise "no" and a counterexample
U := ∅;                                                  // set of states
π := ε;                                                  // stack of states
forall s ∈ I do
    if DFS(s, Φ) then
        return("no", reverse(π));                        // path from s to error state
    end
end
return("yes");                                           // 𝒯 ⊨ ''always Φ''
```

```
function DFS(s, Φ)
    push(s, π);
    if s ∉ U then
        U := U ∪ {s};                                    // mark s as reachable
        if s ⊭ Φ then
            return("true");                              // s is an error state
        else
            forall s' ∈ Post(s) do
                if DFS(s', Φ) then
                    return("true");                      // s' lies on a path to an error state
                end
            end
        end
    end
    pop(π);
    return("false");
end
```

Exercise 4⋆: Invariant checking II 2 Bonus Points

The "DFS-based invariant checking" algorithm presented above (and in the lecture) always computes a minimal counterexample (minimal in the sense that you cannot remove the last state). However, the algorithm does not necessarily compute a counterexample of minimal length (there might be two minimal counterexamples of different lengths). What is an example that shows that the counterexample that is returned does not always have minimal length?

For this purpose, provide the following.

- A transition system that has three states s_0, s_1, s_2.

- An invariant.
- The (non-minimal) counterexample that is computed by the algorithm that uses the following strategy for iterating over a set of states: always take state s_i before state s_j if i is smaller than j.
- A minimal counterexample.