Prof. Dr. Andreas Podelski
Dr. Matthias Heizmann
Christian Schilling

Delivery: November 28th, 2016
16:15 via the post boxes
Discussion: October 30th, 2016

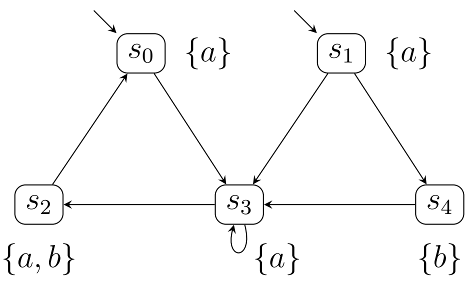
Tutorial for Cyber-Physical Systems - Discrete Models
Exercise Sheet 6

Exercise 1: Invariant checking I 2 Points

Apply the "DFS-based invariant checking" algorithm which was presented in the lecture 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 procedure `DFS`.



Exercise 2: Invariant checking II 1 Point

The "DFS-based invariant checking" algorithm that was presented in the lecture always computes a minimal bad prefix. However, the algorithm does not necessarily compute a bad prefix of minimal total length (there might be two minimal bad prefixes of different length). What is an example that shows that the prefix that is returned does not always have minimal total 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) bad prefix that is computed by the algorithm that uses the following convention for iterating over a set of states. Always take state s_i before state s_j if i is smaller than j.
- A minimal bad prefix.

Exercise 3: Invariant checking III 2 Points

Give an algorithm (in pseudocode, analogously to the algorithm in the lecture) for invariant checking such that, in case the invariant is refuted, a bad prefix of minimal total length is provided as an error indication.
The algorithm should terminate for all finite transition systems.

Hint: You may modify the algorithm presented in the lecture appropriately. You may also want to use two data structures: A *queue* and a *map*.
A queue is a list with two operations:

- `void add(Element)` adds a new element at the end.
- `Element remove()` removes the element at the front (FIFO principle).

A map behaves like a partial function. That is, it stores a value for a given key. It has the following operations:

- `void add(Key, Value)` adds a new mapping from a key to a value.
- `Value get(Key)` returns the value for the given key.
- `boolean has(Key)` returns `true` iff the map stores a value for the given key.

You can use the map to store a predecessor state for a given state. This can be helpful for constructing the bad prefix in the end.

Exercise 4: Terminal states revisited 1 Point

In Exercise 4 on Sheet 5 we considered a transformation of a transition system with terminal states to a transition system without terminal states. Recall the following condition.

> Note that the transformation preserves trace-equivalence, i.e., if TS_1, TS_2 are transition systems (possibly with terminal states) such that $\underline{\mathit{Traces}_{\text{fin}}}(TS_1) = \underline{\mathit{Traces}_{\text{fin}}}(TS_2)$, and TS'_1, TS'_2 are the transformations of TS_1, TS_2, respectively, then $\mathit{Traces}(TS'_1) = \mathit{Traces}(TS'_2)$.

In the original exercise there was a typo: the two (underlined) occurrences of "$\mathit{Traces}_{\text{fin}}$" were instead written as "*Traces*".

Consider the following transformation:

> Given a transition system $TS = (S, \mathit{Act}, \rightarrow, I, AP, L)$, we construct the transition system $TS' = (S', \mathit{Act}, \rightarrow', I, AP, L')$ as follows. We introduce a new sink state $\bot$, i.e., $S' = S \uplus \{\bot\}$. We add the missing transitions to the sink state and a self-loop, i.e.,
>
> $$\rightarrow' = \rightarrow \uplus \{s \xrightarrow{\alpha} \bot \mid s \in S, \alpha \in \mathit{Act}, \nexists s' \in S.\ s \xrightarrow{\alpha} s'\} \uplus \{\bot \xrightarrow{\alpha} \bot \mid \alpha \in \mathit{Act}\}.$$
>
> The labels of the old states are the same, and we label the sink state with the empty set, i.e., L' is the function mapping from S' to 2^{AP} given by
>
> $$s \mapsto \begin{cases} L(s) & s \in S \\ \emptyset & s = \bot. \end{cases}$$

Construct two transition systems TS_1 and TS_2 such that $\mathit{Traces}(TS_1) = \mathit{Traces}(TS_2)$ but after applying the transformation we have $\mathit{Traces}(TS'_1) \neq \mathit{Traces}(TS'_2)$.

Exercise 5: Properties of the closure 2 Points

Prove that for the closure operator cl the following inclusion (resp. equality) holds for every linear time property P.

(a) $P \subseteq cl(P)$

(b) $cl(P) = cl(cl(P))$ (the closure operator cl is idempotent)