Solutions to TD6

1 SCC-based Büchi Emptiness test

1. By way of contradiction, suppose that some inactive SCC C in the explored graph is not an SCC of $\mathcal{B}$. Then is a state or an transition outside the explored graph that is reachable from C. However, since the DFS has concluded at every state of C, any state or transition reachable from C must have been visited by the time C becomes inactive.

2. In any active SCC, there is a state v on the search path, i.e., the DFS has not yet concluded at v. Since the root of the SCC is visited before v, the DFS should conclude at the root after it concludes at v. Thus each active SCC has its root on the search path.

   Let s be an active state, and suppose that it is in the active SCC of r_i. By the definition of a root, $r_i.num \leq s.num$. To show that $s.num < r_{i+1}.num$, suppose on the contrary that $r_{i+1}.num \leq s.num$. Since r_{i+1} is on the search path when s is visited, there is a path from r_{i+1} to s. But then we obtain a cycle as there are paths from s to r_i and from r_i to r_{i+1}. Thus $r_i.num \leq s.num < r_{i+1}.num$.

   For the converse direction, assume that $r_i.num \leq s.num < r_{i+1}.num$. Assume that s does not belong to the active SCC of r_i. Then s must belong to the active SCC of some r_j where $j \neq i$. We cannot have $j < i$, because then we would have a path in the active graph from r_i to s, then from s to r_j and then from r_j to r_i. This shows that the root of a new active SCC occurs between r_i and r_{i+1}, which is not possible.

3. Algorithm 2 maintains a stack of active SCCs of the explored subgraph. When a back-edge is found to one of the active SCCs, closing a cycle, the SCCs forming that cycle are merged.

4. - We prove this by induction on the number of active SCCs in the explored graph. The induction hypothesis is that if the active graph consists of the SCCs $C_1, \ldots, C_m$ where r_i is the root of C_i and $r_i.num$ increases with i, then W consists of $(r_1, C_1), \ldots, (r_m, C_m)$ (rightmost element at the top).

     Assume that the I.H. is true for some value of m. Continuing, DFS looks at the successor t of some vertex $s \in C_m$. For the cases where t is not visited or is visited but not active, it is easy to check that the I.H. holds. Suppose t is active and that it belongs to some C_i, where $i \leq m$. Then the new active SCCs become $C_1, \ldots, C_{i-1}, C_i \cup \cdots \cup C_m$, with r_i being the root of $C_i \cup \cdots \cup C_m$.

     To see that this is also reflected in the stack W, note that the algorithm iteratively keeps popping (r_j, C_j) till $j = i$ and obtains $D = C_i \cup \cdots \cup C_m$, and finally pushes (r_i, D) onto W.

   - A vertex v becomes inactive once the DFS concludes at the root r_i of its active SCC C_i.

     Right before the DFS concludes at r_i, (r_i, C_i) must be on the top of W. Once the DFS concludes at r_i, Algorithm 2 does not find any more new transitions out of r_i and goes to line 21, and marks all states in C_i as inactive. Thus the algorithm behaves correctly.

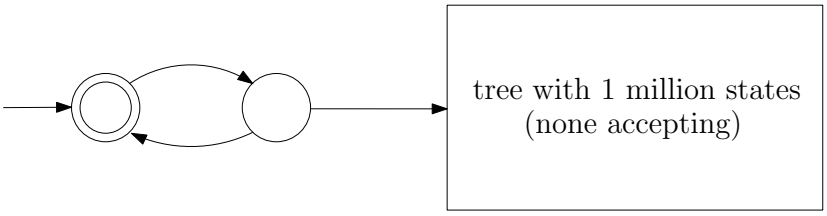

Figure 1: Example in which nested DFS takes a long time to find a counterexample

5. Consider the point at which the explored graph contains a counterexample for the first time. This must have happened because an edge (u_k, u_1) was added, completing the cycle $(u_1, u_2, \ldots, u_k, u_1)$ and for some $1 \leq f \leq k$, state u_f is an accepting state. The following sequence of observations show that the algorithm returns false immediately.

   Observation 1: State u_1 must have been an active state at this time; otherwise if u_1 was inactive, by Question 1 the cycle $(u_1, u_2, \ldots, u_k, u_1)$ would belong to the inactive SCC containing u_1, which is a contradiction since u_k is definitely active when (u_k, u_1) is added. □

   Observation 2: Stack W must have contained $(u_f, \{u_f\})$; otherwise W would have contained some element (r, C) where $u_f \in C$ and $|C| \geq 2$, but then a counterexample would already have been formed before (u_k, u_1) was added. □

   Thus Algorithm 2 starts popping elements from W, merging the second components until the first component becomes less than $u_1.num$. We need to show that it encounters $(u_f, \{u_f\})$.

   Observation 3: $u_f.num > u_1.num$. Suppose on the contrary that $u_f.num < u_1.num$. Then then there is a path from u_f to u_1 in the explored graph that existed before (u_k, u_1) was added. But then joining this path with the path $(u_1, \ldots, u_f)$ implies that there was a counterexample in the explored graph before (u_k, u_1) was added. □

   Thus the algorithm encounters $(u_f, \{u_f\})$ and returns false.

6. Instead of keeping track only of the roots of the SCCs and said SCCs, we also keep track of the final sets that have a representative in the SCC (that is, we keep track of the (r, C, R) where $R \subseteq \{1, \ldots, n\}$ is such that $i \in R$ if and only if there exists v in C such that $v \in F_i$). We then return false when we find an SCC containing a representative of each final set and a non-trivial path.

7. The nested DFS algorithm uses only two bits per state. Algorithm 2 assigns a number to each state, thus requires more space.

   Algorithm 2 outputs false as soon as a counterexample occurs in the explored graph, whereas this is not the case for the nested DFS algorithm, as demonstrated by the example in Figure 1.

2 Exercise 2

1. Consider some LTL$(\mathrm{AP}, \mathsf{U})$ formula φ. We will show by induction that for any stuttering equivalent words σ and ρ that we have $\sigma \models \varphi \Leftrightarrow \rho \models \varphi$.

- The cases $\varphi = \top, p \in \mathrm{AP}, \neg\varphi, \varphi_1 \vee \varphi_2$ are trivial.
- Consider then $\varphi = \psi_1 \,\mathsf{U}\, \psi_2$ and suppose that for any two stuttering equivalent words ζ and γ we have $\zeta \models \psi_1$ iff $\gamma \models \psi_1$ and $\zeta \models \psi_2$ iff $\gamma \models \psi_2$. Take two stuttering equivalent words σ and ρ. Consider then two sequences $0 = i_0 < i_1 < \dots$ and $0 = j_0 < j_1 < \dots$ such that for all ℓ we have $\sigma_{i_\ell} = \cdots = \sigma_{i_{\ell+1}-1} = \rho_{j_\ell} = \cdots = \rho_{j_{\ell+1}-1}$. Now suppose there exists k such that $\sigma, k \models \psi_2$ and for all $k' < k$ we have $\sigma, k' \models \psi_1$. Consider ℓ such that $i_\ell \leq k < i_{\ell+1}$. We can easily show that $\sigma_{\geq k}$ and ρ_{j_ℓ} are stuttering equivalent, and hence $\rho, j_\ell \models \psi_2$. Similarly, for all n such that $n < j_\ell$ there exists $\ell' < \ell$ such that $\sigma_{i_{\geq \ell'}}$ and $\rho_{\geq n}$ are stuttering equivalent (ℓ' is such that $j_{\ell'} \leq n < j_{\ell'+1}$ and $\ell' < \ell$), and since $\sigma, i_{\ell'} \models \psi_1$ we have $\rho, n \models \psi_1$. Hence we have shown $\sigma, 0 \models \varphi$ implies $\rho, 0 \models \varphi$ and since σ and ρ play a symmetrical role the mirror implication is also true.

2. Consider the sequence $i_0, i_1, \dots$ defined as

   - $i_0 = 0$
   - for $k \geq 0$, i_{k+1} is the smallest number strictly greater than i_k such that $\sigma_{i_{k+1}} \neq \sigma_{i_k}$ if one such number exists, $i_k + 1$ else.

   Then σ' defined by $\sigma'_k = \sigma_{i_k}$ is the only stutter-free word that is stuttering equivalent to σ.

3. (a) If a stutter-free word σ is such that $\sigma, 0 \models a \wedge \mathsf{X}\, a$ then it is such that $\sigma_0 = a$ and $\sigma_1 = a$, hence by definition $\sigma = a^\omega$. Thus $\psi_{a,a} = \neg(\top \,\mathsf{U}\, \neg a) = \mathsf{G}\, a$ works.

   (b) The formula $\psi_{a,b} = a \wedge (a \,\mathsf{U}\, b)$ works.

4. - $\tau(\top) = \top$
   - $\tau(p) = p$ where $p \in \mathrm{AP}$
   - $\tau(\neg\psi) = \neg\tau(\psi)$
   - $\tau(\psi_1 \vee \psi_2) = \tau(\psi_1) \vee \tau(\psi_2)$
   - $\tau(\psi_1 \,\mathsf{U}\, \psi_2) = \tau(\psi_1) \,\mathsf{U}\, \tau(\psi_2)$
   - The case $\mathsf{X}\,\psi$ is more involved. It can be rewritten as

$$\mathsf{X}\,\psi = \bigvee_{a\in\Sigma} a \wedge \mathsf{X}\,\psi = \bigvee_{a\in\Sigma} \left[a \wedge \bigvee_{b\in\Sigma} \mathsf{X}(\psi \wedge b) \right]$$
$$= \bigvee_{a\in\Sigma} \left[(a \wedge \mathsf{X}(\psi \wedge a)) \vee \bigvee_{b\neq a} a \wedge \mathsf{X}(\psi \wedge b) \right]$$

   Following the same ideas in Question 3, we see that for stutter-free words, $a \wedge \mathsf{X}(\psi \wedge a)$ and $\mathsf{G}\, a \wedge \psi$ are equivalent, and if $a \neq b$ then $a \wedge \mathsf{X}(b \wedge \psi)$ and $a \wedge (a \,\mathsf{U}\, (b \wedge \psi))$ are equivalent. Hence:

$$\tau(\mathsf{X}\,\psi) = \bigvee_{a\in\Sigma} \left((\mathsf{G}\, a \wedge \tau(\psi)) \vee \bigvee_{a\neq b} a \wedge (a \,\mathsf{U}\, (b \wedge \tau(\psi))) \right)$$

5. For any word σ, let $f(\sigma)$ be the only stutter-free word that is stuttering-equivalent to σ (as seen in Question 2). Then if $L(\varphi)$ is stutter-invariant we have $\sigma \models \varphi \Leftrightarrow f(\sigma) \models \varphi \Leftrightarrow f(\sigma) \models \tau(\varphi)$. However from Question 1 we know that $L(\tau(\varphi))$ is stutter-invariant, and hence $f(\sigma) \models \tau(\varphi) \Leftrightarrow \sigma \models \tau(\varphi)$. This means $\sigma \models \varphi \Leftrightarrow \sigma \models \tau(\varphi)$ and thus $L(\varphi) = L(\tau(\varphi))$.