1 CTL

We consider Kripke structures without deadlocks, i.e. where every state has at least one outgoing transition, and with one atomic proposition $p$.

1. Prove or disprove:
   
   (i) $\text{EF AG} p \Rightarrow \text{EF} p$
   
   (ii) $\text{AG} p \Rightarrow \text{AG EF AG} p$

   Both properties are true. Let $s$ be a state satisfying $\text{AG} p$. Since there are no deadlocks, there exists at least one infinite path from $s$, in which every state, including $s$, satisfies $p$, so we have $\text{AG} p \Rightarrow p$, which implies the first property. Moreover, $\text{AG} p \equiv \text{AG AG} p$, so every state reachable from $s$ satisfies $\text{AG} p$, and hence trivially $\text{EF AG} p$, which implies the second property.

2. Consider the following CTL formulae:
   
   $\bullet \phi_1 = \text{AF AG} p$
   
   $\bullet \phi_2 = \text{AG AF} p$
   
   $\bullet \phi_3 = \text{EG EF} p$
   
   $\bullet \phi_4 = \text{EF EG} p$

   For each pair of formulae $\phi_i, \phi_j$, prove whether $\phi_i$ implies $\phi_j$ (if not, give a Kripke structure with a state in which $\phi_i$ is satisfied but not $\phi_j$).

   (i) Since there are no deadlocks, there is at least one infinite path from every state. Therefore $\text{AG} \phi \Rightarrow \text{EG} \phi$ and $\text{AF} \phi \Rightarrow \text{EF} \phi$ for any property $\phi$.

   (ii) Let $s$ be a state satisfying $\phi_1 = \text{EF EG} p$. Then there exists a path $s = s_0s_1\cdots j$ such that for all $k \geq j$, $s_k \models \text{EG} p$ (and $\text{EG} p \Rightarrow p \Rightarrow \text{EF} p$). Because of $s_j \models p$, we have $s_{k'} \models \text{EF} p$ for all $k' < j$. So in fact $\text{EF} p$ holds in all states along the path, thus $s \models \text{EG EF} p$. A very similar argument (with all-quantifiers) shows $\text{AF AG} p \Rightarrow \text{AG AF} p$.

   (iii) Consider the following structure, where a double border means that the state satisfies $p$. Here, both states satisfy $\text{AG AF} p$ and $\text{EG EF} p$ but not $\text{EF EG} p$ nor $\text{AF AG} p$.

   (iv) Consider the following structure, where $s_0$ satisfies $\text{EF EG} p$ but not $\text{AG AF} p$: 

---

Exam solution – Introduction to Verification

November 12, 2021
To summarize, we get the following results. Each cell indicates whether $\phi_i \Rightarrow \phi_j$, where + indicates results due to transitivity.

<table>
<thead>
<tr>
<th>$\phi_i \ \setminus \ \phi_j$</th>
<th>$\text{AF AG } p$</th>
<th>$\text{AG AF } p$</th>
<th>$\text{EG EF } p$</th>
<th>$\text{EF EG } p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{AF AG } p$</td>
<td>no (iii)</td>
<td>yes (ii)+(i)</td>
<td>yes (i)</td>
<td></td>
</tr>
<tr>
<td>$\text{AG AF } p$</td>
<td></td>
<td>yes (i)</td>
<td>no (iii)</td>
<td></td>
</tr>
<tr>
<td>$\text{EG EF } p$</td>
<td>no (iii)</td>
<td>no (iv)+(i)</td>
<td></td>
<td>no (iii)</td>
</tr>
<tr>
<td>$\text{EF EG } p$</td>
<td>no (iv)+(ii)</td>
<td>no (iv)</td>
<td>yes (ii)</td>
<td></td>
</tr>
</tbody>
</table>

3. Prove the following properties:

- $\text{AG EF AG EF } p \equiv \text{AG EF } p$
- $\text{EF AG EF AG } p \equiv \text{EF AG } p$

This follows from the properties in the first part of the question, together with the idempotences $\text{EF EF } p \equiv \text{EF } p$ and $\text{AG AG } p \equiv \text{AG } p$.

- $\text{AG EF AG EF } p \Rightarrow (i) \quad \text{AG EF } p \equiv \text{AG EF } p$;
- $\text{AG EF } p \Rightarrow (ii) \quad \text{AG EF AG } p$;
- $\text{EF AG EF AG } p \Rightarrow (i) \quad \text{EF EF AG } p \equiv \text{EF AG } p$;
- $\text{EF AG } p \Rightarrow (ii) \quad \text{EF AG AG } p$.

2 $\omega$-automata

An $\omega$-automaton is a tuple $\langle \Sigma, S, s_0, \Delta, F \rangle$, where $\Sigma$ is a finite alphabet, $S$ is a finite set of states, $s_0$ the initial states, and $\Delta \subseteq S \times \Sigma \times S$ the transitions, with the usual notions. $F$ is an acceptance component, to be clarified below.

For a run $\rho \in S^\omega$, we note $\text{Occ}(\rho) = \{ s \mid \exists i : \rho(i) = s \}$ the set of states that occur at least once in $\rho$ and $\text{Inf}(\rho) = \{ s \mid \forall i \exists j \geq i : \rho(i) = s \}$ the set of states occurring infinitely often in $\rho$.

The following types of $\omega$-automata were shown to be equivalent in the course and exercises:

- Büchi automata (BA) with $F \subseteq S$, where a run $\rho$ is accepted if $\text{Inf}(\rho) \cap F \neq \emptyset$;
- generalized BA (GBA) with $F \subseteq 2^S$, where $\rho$ is accepted if $\forall F \in F : \text{Inf}(\rho) \cap F \neq \emptyset$;
- Muller automata (MA) with $F \subseteq 2^S$, where $\rho$ is accepted if $\text{Inf}(\rho) \in F$.

We consider the following additional models:

- Occurrence automata (OA) with $F \subseteq 2^S$, where $\rho$ is accepted if $\text{Occ}(\rho) \in F$;
- Streett automata (SA) with $F \subseteq 2^S \times 2^S$, where $\rho$ is accepted if for all $\langle E, F \rangle \in F$, $\text{Inf}(\rho) \cap E \neq \emptyset$ implies $\text{Inf}(\rho) \cap F \neq \emptyset$.

1. Given an OA resp. SA, can one construct a BA accepting the same language?

2. Given a BA, can one construct an OA resp. SA accepting the same language?

- OA $\rightarrow$ BA: We construct a BA that, in addition to its current state, remembers a set $T$ of all states that have occurred in the past. At some point, every state that occurs at all will have occurred for the first time, and from that point $T$ no longer changes. We make those states accepting where $T$ is an accepting occurrence set of the OA. Formally, for an OA $\langle \Sigma, S, s_0, \Delta, F \rangle$, we construct the BA $\langle \Sigma, S \times 2^S, \langle s_0, \emptyset \rangle, \Delta', S \times F \rangle$, where $\Delta' = \{ \langle \langle q, T \rangle, a, \langle q', T \cup \{ q \} \rangle \rangle \mid \langle q, a, q' \rangle \in \Delta \}$.
SA → BA: It suffices to create an MA whose accepting sets are those that satisfy the accepting conditions of the SA, i.e. the accepting condition of the MA is:
\[ \mathcal{F}' = \{ S' \subseteq S | \forall (E,F) \in \mathcal{F} : S' \cap E \neq \emptyset \Rightarrow S' \cap F \neq \emptyset \} \]

BA → SA: A Büchi automaton is a special case of a Streett automaton with one accepting pair, the condition of the SA being \( \mathcal{F}' = \{ (S,F) \} \).

BA → OA: This translation is not possible in general. E.g., for \( \Sigma = \{a,b\} \), there is a BA accepting the language of all words with infinitely many \( b \), i.e. \( L := (a^*b)^\omega \), but no OA. Indeed, suppose there was an OA accepting \( L \) with \( n \) states. Consider an accepting run \( \rho \) for \((a^n b)^\omega\). After some finite prefix \( w := (a^n b)^k \), \( \rho \) must have visited all occurring states at least once. Now in between the prefixes \( w \) and \( wa^n \), \( \rho \) must have traversed a loop that can be repeated and contains only \( a \). Thus, there is a run of \((a^n b)^k a^\omega \notin L\) that has the same occurrence set as \( \rho \) (and is therefore accepting), which violates our assumption.

3 Mutex algorithm

Consider the following mutual-exclusion algorithm for three processes numbered 0, 1, and 2. Each process \( i \) uses a private variable \( \text{state}[i] \) that can take values \text{waiting}, \text{critical}, \text{active}, and \text{idle} (we say that a process is waiting, critical, etc if its state variable has the corresponding value); initially every process is \text{idle}. In addition, there are two shared variables \text{token} and \text{request} that can take values 0, 1, 2; initially both are 0.

A global state is a tuple consisting of the values of \text{state}[0..2], \text{token}, and \text{request}, in that order. A run is a finite or infinite sequence of such states.

Each process \( i \) executes the following algorithm:

\[
\text{while (true) do }
\text{state}[i] := \text{waiting}; \\
\text{while (token != i) request := i;} \\
\text{state}[i] := \text{critical}; \\
// Critical section \\
\text{state}[i] := \text{active}; \\
\text{while (request == i) skip;} \\
\text{token := request;} \\
\text{state}[i] := \text{idle} \\
\text{od}
\]

1. Does the algorithm contain a deadlock? Give a run leading to a deadlock or a reason why none exists.

No deadlock exists. There is one \text{while} loop in the waiting state, and another in the active state. For a deadlock to exist, all three processes would have to get stuck simultaneously in one of these loops.

It is easy to verify that \text{token} is 0 initially and then is assigned only to \text{request}, which is itself between 0 and 2. Thus, \text{token} is always either 0, 1, or 2. Therefore it is impossible for all three processes to get stuck in the “waiting” loop. It is also impossible for them to all get stuck in the “active” loop, since the waiting conditions are mutually exclusive. Moreover, if one process is in the “waiting” loop and another in the “active” loop, then the action of the first will unblock the second.

2. Formulate the following properties in LTL for this algorithm, using atomic properties of the form \text{var} = \text{value}:

- No two processes can be critical at the same time.

\[ \bigwedge_{0 \leq i < j \leq 2} G \neg(\text{state}[i] = \text{critical} \land \text{state}[j] = \text{critical}) \]
• If a process waits to enter the critical section, it can eventually do so.

\[
\bigwedge_{0 \leq i \leq 2} G (\text{state}[i] = \text{waiting} \rightarrow F \text{state}[i] = \text{critical})
\]

You may use \(\lor, \land\) with indices as abbreviation.

3. Do these two properties hold in all *fair* runs, i.e. those in which every process infinitely often gets to execute some step? Again, provide either a counterexample run, or a reason why none exists.

The first property holds. Let us show that “at most one process is critical” is an invariant of any run of the algorithm. Certainly, initially no process is critical. After that, only the process \(i\) such that \(\text{token} = i\) can become critical. Only this process can then change \(\text{token}\), but only after having left its critical state. Therefore at any time, only at most one process is critical.

The second property does not hold. Imagine that all three processes want are waiting, and only process 0 succeeds. The other two will take turns at changing the value of \(\text{request}\). It is therefore possible that process 0 sets \(\text{token} := 1\). Suppose that while process 1 is critical, process 0 becomes waiting again. By analogy to the previous situation, process 1 might give the token back to 0. This situation may repeat, and process 2 never gets to enter its critical section.