# Model-Checking Algorithms

A model checker answers the *model-checking question*:

"given an LTS $\mathcal{M} = \langle S, \longrightarrow, L \rangle$, a state $s$ and a formula $\varphi$, does $\mathcal{M}, s \models \varphi$ hold?"

With the LTL and CTL semantics, we can answer it in principle by considering every path from $s$. Yet, a program cannot do that: there are usually infinitely many paths, each of them infinite, and the LTS of a real system has a huge number of states. A program has to work on the finite graph of the LTS instead [1]. In this sheet, we will study the algorithms that model checkers use, in three steps.

1. *CTL* (Section 1). Compute, for each subformula, the set of states where it holds, from its smallest subformula outwards.
2. *Fairness* (Section 2). Some properties fail only because of behaviour that cannot happen in reality. Fairness constraints exclude those impossible situations.
3. *LTL* (Section 3). A formula is evaluated along paths, so sets of states are not enough. Translate the negated formula into a Büchi automaton, an automaton that accepts infinite words, combine it with the LTS, and search the combination for a counterexample.

## 1 CTL: computing satisfaction sets

A model checker could answer the model-checking question with "yes" or "no" for one state $s$. Yet, it is easier to compute, for a formula $\varphi$, the *satisfaction set* of *all* states where $\varphi$ holds:

$$\mathrm{SAT}(\varphi) = \{s \in S \mid \mathcal{M}, s \models_{\mathsf{CTL}} \varphi\}.$$

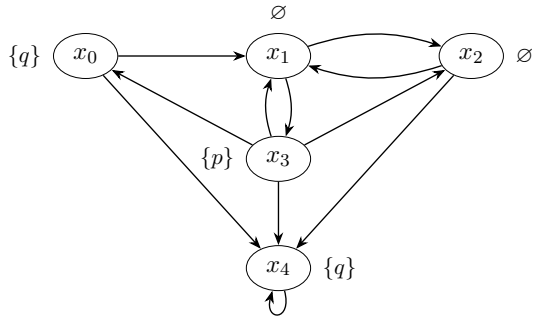
Then, the answer to the model-checking question for $s$ is whether $s \in \mathrm{SAT}(\varphi)$ [1]. The set of a formula is built from the sets of its immediate subformulas, so the algorithm starts from the atomic propositions and works outwards.

Only a few connectives need their own algorithm: $\neg$, $\wedge$, $\exists\bigcirc$, $\exists\mathrm{U}$, and either $\forall\Diamond$ or $\exists\Box$. Every other CTL formula is first rewritten into these, for example $\forall\Box\varphi \equiv \neg\exists\Diamond\neg\varphi$ and $\exists\Diamond\varphi \equiv \exists(\top\ \mathrm{U}\ \varphi)$ [1]. The subsections below explore these connectives in order.

### 1.1 Negation, conjunction and next

The connectives $\neg$ and $\wedge$ combine the sets of their subformulas state by state, and $\exists\bigcirc$ looks only one transition ahead.

**Exercise 1 (From subformulas to sets).** Compute $\mathrm{SAT}(\exists\bigcirc(p \wedge \neg q))$ in the LTS below: complete the table with the satisfaction set of each subformula, from the smallest outwards. Use set complement for negation and intersection for conjunction. For $\exists\bigcirc\psi$, include a state when *at least one* successor belongs to $\mathrm{SAT}(\psi)$. The first row is the example.




| Formula | Satisfaction set |
|---|---|
| $p$ | $\{x_3\}$ |
| $q$ | _____ |
| $\neg q$ | _____ |
| $p \wedge \neg q$ | _____ |
| $\exists\bigcirc(p \wedge \neg q)$ | _____ |

## 1.2 Exists until: growing a set backwards

**Exercise 2 (Building "until").** For this exercise, observe the equivalence [1]

$$\exists(\varphi \,\mathrm{U}\, \psi) \equiv \psi \vee \big(\varphi \wedge \exists\bigcirc \exists(\varphi \,\mathrm{U}\, \psi)\big).$$

Let us use it, and the set $\mathrm{Pre}_\exists(T) = \{s \in S \mid \exists s' \in T.\ s \longrightarrow s'\}$, to compute $\mathrm{SAT}(\exists(\top \,\mathrm{U}\, c_1))$ and $\mathrm{SAT}(\exists(\neg c_2 \,\mathrm{U}\, c_1))$ for Huth and Ryan's mutual-exclusion protocol [1] from a previous session (Figure 1).[1] The equivalence suggests starting with $T_0 = \mathrm{SAT}(\psi)$ and recursively computing:

$$T_{i+1} = \mathrm{SAT}(\psi) \cup \big(_____ \cap \mathrm{Pre}_\exists(T_i)\big)$$

(complete the missing part, notice the resemblance to the equivalence above). Each computation of $T_{i+1}$ from $T_i$ is an *iteration*. We stop when $T_{i+1} = T_i$. From Figure 1, please complete the tables below (examples are provided):

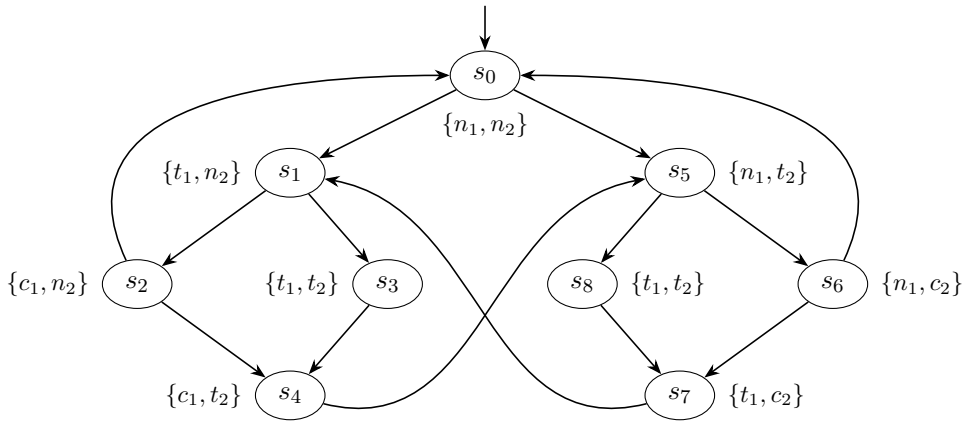



Figure 1: The mutual-exclusion protocol (MEP) in Huth and Ryan's book [1].

| $\mathrm{SAT}(c_1)$ | $\mathrm{SAT}(c_2)$ |
|---|---|
| $\{s_2, s_4\}$ | _____ |

[1] Recall that the MEP describes two processes sharing a resource. Each process $i$ is in one of 3 possible states: non-critical ($n_i$), trying to enter its critical section ($t_i$), or in its critical section ($c_i$), where it uses the resource. Only one process moves at a time. A trying process may enter when the other one is non-critical. When both are trying, the one that started trying first enters first.

| | $\exists(\top \,\mathrm{U}\, c_1)$ | | $\exists(\neg c_2 \,\mathrm{U}\, c_1)$ | |
|---|---|---|---|---|
| $i$ | $T_i$ | $\mathrm{Pre}_\exists(T_i)$ | $T_i$ | $\mathrm{Pre}_\exists(T_i)$ |
| 0 | $\{s_2, s_4\}$ | $\{s_1, s_2, s_3\}$ | _____ | _____ |
| 1 | _____ | _____ | _____ | _____ |
| 2 | _____ | _____ | _____ | _____ |
| 3 | _____ | _____ | _____ | (stable) |
| 4 | _____ | _____ | | |
| 5 | _____ | (stable) | | |

Mark with a checkmark (✓) why this procedure computes the satisfaction set:

( ) It adds every state from which a state satisfying $\psi$ is reachable.

( ) It follows every path from each state until the path ends.

( ) (Sound) A state $s$ added in $i$ steps has a path $s \longrightarrow \cdots \longrightarrow s'$ of at most $i$ transitions, where $s'$ satisfies $\psi$ and the states before satisfy $\varphi$, so this path satisfies $\varphi \,\mathrm{U}\, \psi$. (Complete) If $s$ satisfies $\exists(\varphi \,\mathrm{U}\, \psi)$, some path from $s$ reaches a $\psi$-state after $k$ transitions through $\varphi$-states, so $s \in T_k$. The sets only grow and $S$ is finite, so they stabilise.

### 1.3 All paths eventually

**Exercise 3 (Building all-eventually).** For this exercise, observe the equivalence [1]

$$\forall\Diamond\psi \equiv \psi \vee \forall\bigcirc\forall\Diamond\psi,$$

and the set $\mathrm{Pre}_\forall(T) = \{s \in S \mid \forall s'.\ \text{if } s \longrightarrow s' \text{ then } s' \in T\}$. Let us compute $\mathrm{SAT}(\forall\Diamond p)$ for the LTS in Figure 2, and for the same LTS with an added self-loop on $u_2$. As before, we start with $T_0 = \mathrm{SAT}(\psi)$. Complete the equation that computes $T_{i+1}$ from $T_i$ for any formula $\psi$:

$$T_{i+1} = _____ \cup \mathrm{Pre}_\forall(T_i)$$

We stop when $T_{i+1} = T_i$. From Figure 2, complete the table below (an example is provided):

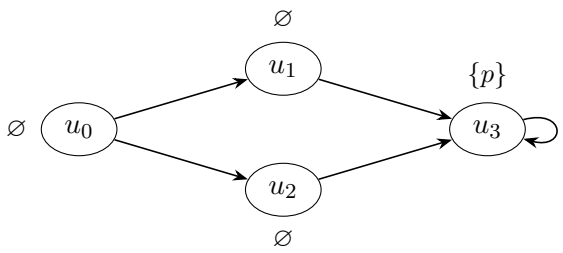



Figure 2: An LTS in which every path from $u_0$ reaches $u_3$.

| | Original LTS | | With a self-loop on $u_2$ | |
|---|---|---|---|---|
| $i$ | $T_i$ | $\mathrm{Pre}_\forall(T_i)$ | $T_i$ | $\mathrm{Pre}_\forall(T_i)$ |
| 0 | $\{u_3\}$ | $\{u_1, u_2, u_3\}$ | _____ | _____ |
| 1 | _____ | _____ | _____ | _____ |
| 2 | _____ | _____ | _____ | (stable) |
| 3 | _____ | (stable) | | |

## 1.4 The whole algorithm

Exercises 1–3 cover the connectives that need their own algorithm. Together, they give the *labelling algorithm*: a function SAT that computes the satisfaction set of any CTL formula by recursion on the formula [1].

**Exercise 4 (The labelling algorithm).** Given an LTS $\langle S, \longrightarrow, L\rangle$, complete the definition of the SAT function below by matching each formula $\varphi$ to its satisfaction set $\mathrm{SAT}(\varphi)$: write the formula's letter in the brackets. The first group of cases is computed directly, with the methods of Exercises 1–3. The second group first rewrites the formula into connectives of the first group, then calls SAT on the result. One example is provided in each group.

**First group: computed directly.**

| **If $\varphi$ is** | **then $\mathrm{SAT}(\varphi)$ is** |
|---|---|
| a) $\top$ | ( ) the stable $T_i$, where $T_0 = \mathrm{SAT}(\varphi_1)$ and $T_{i+1} = \mathrm{SAT}(\varphi_1) \cup \mathrm{Pre}_\forall(T_i)$ |
| b) $\bot$ | ( ) $\varnothing$ |
| c) an atomic proposition $p$ | ( ) $\mathrm{Pre}_\exists(\mathrm{SAT}(\varphi_1))$ |
| d) $\neg\varphi_1$ | ( a ) $S$ |
| e) $\varphi_1 \wedge \varphi_2$ | ( ) $\mathrm{SAT}(\varphi_1) \cap \mathrm{SAT}(\varphi_2)$ |
| f) $\exists\bigcirc\varphi_1$ | ( ) the stable $T_i$, where $T_0 = \mathrm{SAT}(\varphi_2)$ and $T_{i+1} = \mathrm{SAT}(\varphi_2) \cup \big(\mathrm{SAT}(\varphi_1) \cap \mathrm{Pre}_\exists(T_i)\big)$ |
| g) $\exists(\varphi_1 \,\mathrm{U}\, \varphi_2)$ | ( ) $\{s \in S \mid p \in L(s)\}$ |
| h) $\forall\Diamond\varphi_1$ | ( ) $S \setminus \mathrm{SAT}(\varphi_1)$ |

**Second group: rewritten first.**

| **If $\varphi$ is** | **then** SAT($\varphi$) **is** | |
|---|---|---|
| a) $\varphi_1 \vee \varphi_2$ | ( ) | SAT($\neg\exists\bigcirc\neg\varphi_1$) |
| b) $\varphi_1 \to \varphi_2$ | ( ) | SAT$\big(\neg(\exists(\neg\varphi_2 \,\mathrm{U}\, (\neg\varphi_1 \wedge \neg\varphi_2)) \vee \exists\Box\neg\varphi_2)\big)$ |
| c) $\forall\bigcirc\varphi_1$ | ( ) | SAT($\exists(\top \,\mathrm{U}\, \varphi_1)$) |
| d) $\exists\Diamond\varphi_1$ | ( ) | SAT($\neg\varphi_1 \vee \varphi_2$) |
| e) $\forall\Box\varphi_1$ | ( ) | SAT($\neg\forall\Diamond\neg\varphi_1$) |
| f) $\exists\Box\varphi_1$ | ( a ) | SAT($\neg(\neg\varphi_1 \wedge \neg\varphi_2)$) |
| g) $\forall(\varphi_1 \,\mathrm{U}\, \varphi_2)$ | ( ) | SAT($\neg\exists\Diamond\neg\varphi_1$) |

## 1.5 Practice: a microwave oven

**Exercise 5 (A microwave oven).** Clarke, Grumberg and Peled model a microwave oven with the LTS of Figure 3 [2]. The atomic propositions say whether the start button has been pressed (*start*), the door is closed (*close*), the oven heats (*heat*) and an error has occurred (*error*). The edge labels name the actions that cause the transitions. Let us check here whether whenever the oven is started, it eventually heats:

$$\theta = \forall\Box(start \to \forall\Diamond heat).$$

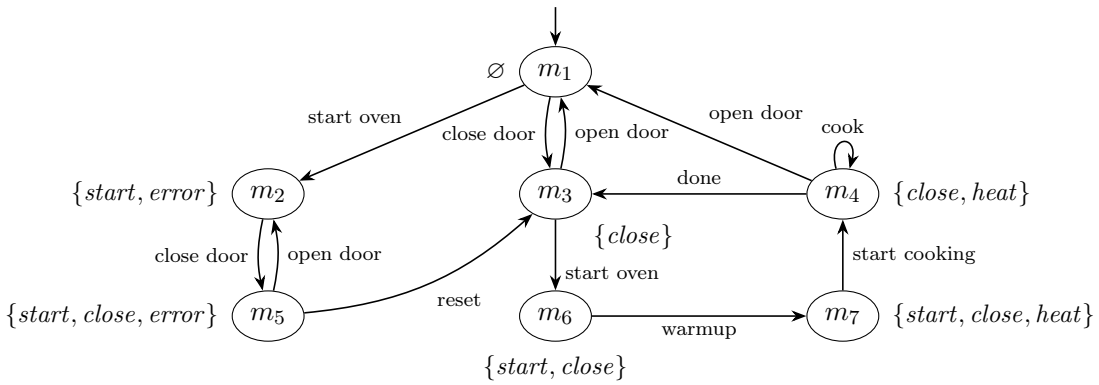



Figure 3: A microwave oven, after Clarke, Grumberg and Peled [2].

**Part 1 — *Rewrite.*** Following your function SAT, rewrite $\theta$ into $\neg$, $\wedge$, $\exists$U and $\forall\Diamond$:

$\theta \equiv$ _____

**Part 2 — *Compute.*** Complete the satisfaction sets of the subformulas of $\theta$, from the smallest outwards. Compute the sets for $\forall\Diamond$ and $\exists$U by iteration, with the equations of Exercise 4. The first row is the example.

| Formula | Satisfaction set |
|---|---|
| $heat$ | $\{m_4, m_7\}$ |
| $\forall\Diamond heat$ | _____ |
| $\neg\forall\Diamond heat$ | _____ |
| $start$ | _____ |
| $start \wedge \neg\forall\Diamond heat$ | _____ |
| $\exists(\top \text{ U } (start \wedge \neg\forall\Diamond heat))$ | _____ |
| $\theta$ | _____ |

**Part 3 — *Interpret.*** What would be a path from the initial state on which the oven is started but never heats? Mark what it describes:

( ) The oven heats before the start button is pressed.

( ) The start button is pressed with the door open, and the user then keeps closing and opening the door.

( ) The door is never closed.

On such a path, the user never operates the oven correctly. Exercise 17 excludes such paths with fairness constraints.

### 1.6 A faster algorithm: some path always, and cycles

Huth and Ryan improve the efficiency of the labelling algorithm by handling $\exists\Box$ in a cleverer way: the computed connectives become $\exists\bigcirc$, $\exists$U and $\exists\Box$, instead of $\exists\bigcirc$, $\exists$U and $\forall\Diamond$. Also, $\forall\Diamond\varphi$ is rewritten as $\neg\exists\Box\neg\varphi$ [1]. Let us explore the benefits of those changes.

**Exercise 6 (Some path always).** Observe the equivalence [1]

$$\exists\Box\psi \equiv \psi \wedge \exists\bigcirc\exists\Box\psi.$$

As in previous exercises, using the set $\text{Pre}_\exists(V) = \{s \in S \mid \exists s' \in V.\ s \longrightarrow s'\}$, write the equation that computes $V_{i+1}$ from $V_i$ for any formula $\psi$:

$$V_{i+1} = _____$$

Start from $V_0 = \text{SAT}(\psi)$ (the sets will now shrink). We stop when $V_{i+1} = V_i$. Compute $\text{SAT}(\exists\Box p)$ for the LTS in Figure 4. The first row is the example.

| $i$ | $V_i$ | $\text{Pre}_\exists(V_i)$ |
|---|---|---|
| 0 | $\{v_0, v_1, v_3, v_4\}$ | $\{v_0, v_3, v_4\}$ |
| 1 | _____ | _____ |
| 2 | _____ | _____ |
| 3 | _____ | (stable) |

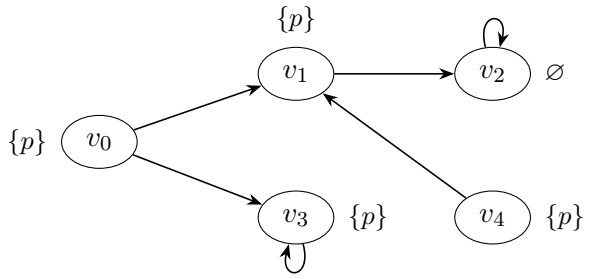



Figure 4: An LTS with a branch that leaves $p$.

**Exercise 7 (Strongly connected components).** A *strongly connected component* (SCC) of a graph is a maximal set of states that can all reach each other.[2] An SCC is *nontrivial* if it contains a transition between two of its states, possibly the same one [3].[3] List the SCCs of each graph below and mark the nontrivial ones. The first row is the example.

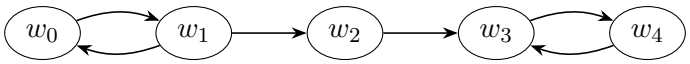



Figure 5: A graph with five states.

| Graph | SCCs | Nontrivial SCCs |
|---|---|---|
| Figure 4, restricted to the $p$-states and the transitions between them | $\{v_0\}, \{v_1\}, \{v_3\}, \{v_4\}$ | $\{v_3\}$ |
| Figure 5 | _____ | _____ |
| Figure 3, restricted to the $\neg heat$-states | _____ | _____ |
| Figure 1, restricted to the $\neg c_1$-states | _____ | _____ |

**Exercise 8 (Some path always, by cycles).** To compute SAT($\exists\Box\psi$) with SCCs, restrict the LTS to the $\psi$-states, find the nontrivial SCCs, then search backwards, within the restricted graph, for the states that reach them [1]. Use your SCCs from Exercise 7.

| Formula | Union of nontrivial SCCs | States that reach them |
|---|---|---|
| $\exists\Box p$ in Figure 4 | _____ | _____ |
| $\exists\Box\neg heat$ in Figure 3 | _____ | _____ |

Compare with your results: the first row should equal the last $V_i$ of Exercise 6, and the second, the complement of SAT($\forall\Diamond heat$) in Exercise 5. Mark with a checkmark (✓) why this procedure computes SAT($\exists\Box\psi$):

(  ) It keeps only the states that lie in a nontrivial SCC.

[2] Formally, $C \subseteq S$ is an SCC if for all $s, s' \in C$ there is a path of zero or more transitions from $s$ to $s'$, and no set $C' \supsetneq C$ has this property.

[3] Formally, an SCC $C$ is nontrivial if $s \longrightarrow s'$ for some $s, s' \in C$.

( ) An infinite path on which $\psi$ always holds should stay in the restricted graph. Since $S$ is finite, it eventually returns to a state it has visited, so it enters a nontrivial SCC. Conversely, from a state that reaches a nontrivial SCC in the restricted graph, a path can enter that SCC and then stay in it for ever, because each of its states has a successor in it, so $\psi$ always holds on it.

( ) It keeps every $\psi$-state from which another $\psi$-state is reachable.

**Exercise 9 (Counting the visits of the two methods).** Compute SAT($\exists\Box p$) for the LTS in Figure 6 with both methods, by iteration (Exercise 6) and with SCCs (Exercise 8), and count the states each method *visits*. Both methods start from SAT($p$) = $\{y_1, y_2, y_3, y_4, y_5\}$.

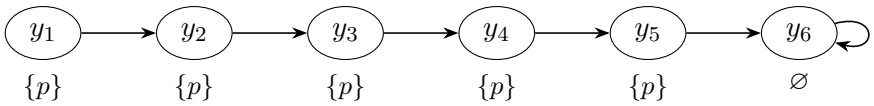



Figure 6: A chain of $p$-states.

**Part 1 — *Count the visits of iteration.*** Computing $V_{i+1}$ visits every state of $V_i$, to check whether it has a successor in $V_i$. Complete the table below.

| $i$ | $V_i$ | States visited |
|---|---|---|
| 0 | $\{y_1, y_2, y_3, y_4, y_5\}$ | 5 |
| 1 | _____ | _____ |
| 2 | _____ | _____ |
| 3 | _____ | _____ |
| 4 | _____ | _____ |
| 5 | _____ | _____ |
| Total | | _____ |

**Part 2 — *Count the visits of the SCC method.*** Finding the SCCs of a graph visits each of its states once, and the backward search visits only the states that reach a nontrivial SCC [1]. Complete the table for the graph restricted to the $p$-states.

| Step | Result | Therefore, states visited |
|---|---|---|
| Find the SCCs | Number of nontrivial SCCs: _____ | _____ |
| Search backwards | Number of backward visits required: _____ | _____ |
| Thus | SAT($\exists\Box p$) = _____ | _____ |

**Part 3 — *Count the visits on longer chains.*** Repeat the counts for chains of $n$ $p$-states ending, as in Figure 6, in a state without $p$. The first row is Figure 6.

*Hint.* The sum $1 + 2 + \cdots + n$ equals $n(n+1)/2$.

| Number of $p$-states | Visits by iteration | Visits by the SCC method |
|---|---|---|
| 5 | $5+4+3+2+1=15$ | 5 |
| 10 | _____ | _____ |
| 100 | _____ | _____ |
| $n$ | _____ | _____ |

Mark the conclusion:

( ) Both methods take time linear in the size of the LTS.

( ) Iteration can take time quadratic in the number of states, because each iteration may remove a single state, while the SCC method is linear in the size of the LTS [1].

## 2 Fairness: excluding unrealistic paths

Models can have paths that real systems never take (like the user infinitely opening and closing the microwave), and properties could fail only because of those infinite paths. In this section, we check a property of the mutual-exclusion protocol that fails in such fashion. We thus make the model more realistic and exclude the unrealistic paths with fairness constraints [1].

**Exercise 10 (Writing liveness in CTL).** A requirement for the mutual-exclusion protocol is *liveness*: whenever a process is trying to enter its critical section, it eventually enters it [1]. Complete the CTL formula $\theta$ for the liveness of process 1, then rewrite it with the equivalences of Exercise 4 and $\neg(\chi_1 \to \chi_2) \equiv \chi_1 \wedge \neg\chi_2$ into a formula with $\exists\Diamond$ and $\exists\Box$.

$$\begin{aligned}\theta &= \forall\Box(t_1 \to _____)\\ &\equiv \neg\exists\Diamond\neg(t_1 \to \forall\Diamond c_1)\\ &\equiv \neg\exists\Diamond(t_1 \wedge _____)\\ &\equiv \neg\exists\Diamond(t_1 \wedge _____)\end{aligned}$$

**Exercise 11 (Checking liveness).** Check liveness of process 1 in the LTS of Figure 1.

**Part 1 — *Liveness holds.*** In the table below, compute the satisfaction sets of the subformulas of $\theta$. Use your nontrivial SCCs from Exercise 7 for $\exists\Box\neg c_1$.

| Formula | Satisfaction set in Figure 1 |
|---|---|
| $\neg c_1$ | $\{s_0, s_1, s_3, s_5, s_6, s_7, s_8\}$ |
| $\exists\Box\neg c_1$ | _____ |
| $t_1 \wedge \exists\Box\neg c_1$ | _____ |
| $\exists\Diamond(t_1 \wedge \exists\Box\neg c_1)$ | _____ |
| $\theta$ | _____ |

Mark the conclusion with a checkmark (✓):

( ) Liveness of process 1 holds in every state: on every path, whenever process 1 is trying, it later enters its critical section.

( ) Liveness of process 1 fails in $s_7$, where process 1 is trying and process 2 is critical.

( ) Liveness of process 1 holds only in the states where process 1 is not trying.

**Part 2 — *A problem.*** The model in Figure 1 is unrealistic. Why? What is the problem? Mark the conclusion with a checkmark (✓):

( ) Nothing: liveness of a process never depends on another in the real world.

( ) Liveness holds in the model only because the model forces process 2 out of its critical section, which a real process does not necessarily do.

( ) Both processes would be in their critical sections at once.

**Exercise 12 (Checking liveness when a process can stay).** To let a process stay in its critical section as long as it needs while the other one is trying, add the transitions $s_4 \longrightarrow s_4$ and $s_7 \longrightarrow s_7$. This gives the LTS in Figure 7. Check liveness of process 1 again.

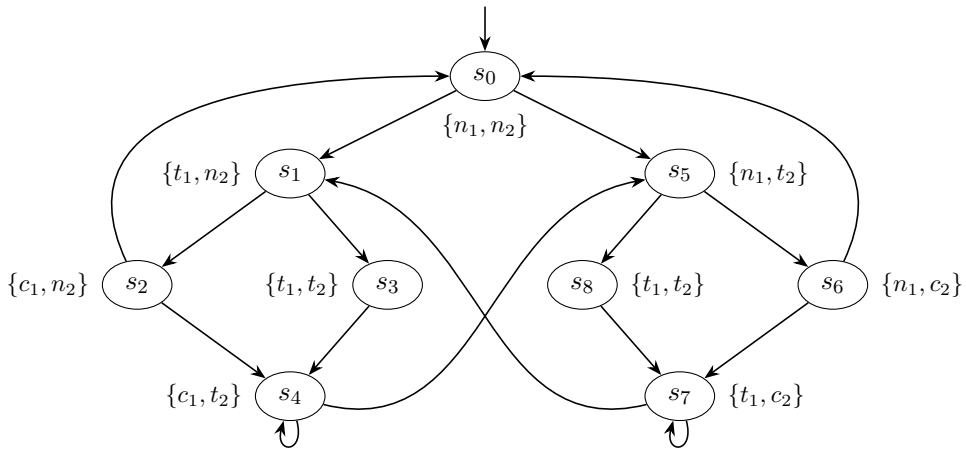



Figure 7: The mutual-exclusion protocol with self-loops at $s_4$ and $s_7$.

**Part 1 — *Find the SCCs.*** To find SAT($\exists\Box\neg c_1$), identify the SCCs in Figure 7 restricted to the $\neg c_1$-states. Complete the table with the SCC of each state. An example is provided.

| State | SCC |
|---|---|
| $s_0$ | $\{s_0, s_5, s_6\}$ |
| $s_1$ | _____ |
| $s_3$ | _____ |
| $s_5$ | _____ |
| $s_6$ | _____ |
| $s_7$ | _____ |
| $s_8$ | _____ |

Complete the sentences:

- the nontrivial SCCs are _____ and _____.
- the union of the nontrivial SCCs is _____.
- therefore, SAT($\exists\Box\neg c_1$) =_____ (states that reach the nontrivial SCCs).

**Part 2 — *Compute the satisfaction sets.*** Complete the table.

| Formula | Satisfaction set in Figure 7 |
|---|---|
| $\neg c_1$ | $\{s_0, s_1, s_3, s_5, s_6, s_7, s_8\}$ |
| $\exists\Box\neg c_1$ | your result of Part 1 |
| $t_1 \wedge \exists\Box\neg c_1$ | _____ |
| $\exists\Diamond(t_1 \wedge \exists\Box\neg c_1)$ | _____ |
| $\theta$ | _____ |

Now liveness ($\theta$) does not hold. Mark the conclusion with a checkmark (✓):

( ) The self-loops let both processes be in their critical sections at once, so the protocol must be changed.

( ) Liveness fails because of paths on which process 2 stays in its critical section for ever (state $s_7$).

( ) The loop at $s_4$ lets process 1 stay critical, so process 1 never enters.

## 2.1 Fair paths

As seen above, the self-loops let a process stay in its critical section for ever. If we are confident our system does not behave that way, then we are only interested in checking liveness on the paths on which each process is repeatedly outside its critical section.

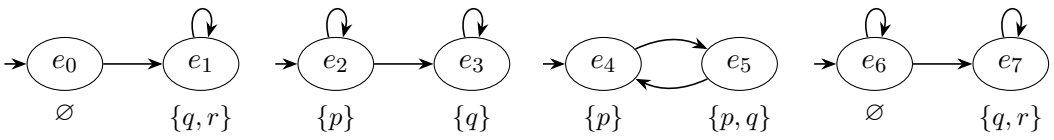
**Exercise 13 (Trying to select the fair paths in CTL).** A first attempt for process 2 is the CTL formula $\vartheta = (\forall\Box\forall\Diamond\neg c_2) \rightarrow \theta$: "if process 2 always eventually leaves its critical section, then liveness holds". Its premise rewrites, as in Exercise 10, to $\forall\Box\forall\Diamond\neg c_2 \equiv \neg\exists\Diamond\exists\Box c_2$. Complete the table for the LTS of Figure 7.

| Formula | Satisfaction set in Figure 7 |
|---|---|
| $\exists\Box c_2$ | _____ |
| $\exists\Diamond\exists\Box c_2$ | _____ |
| $\forall\Box\forall\Diamond\neg c_2$ | _____ |

Knowing the truth table of an implication, and your previous result, what is the satisfaction set for $\vartheta$? Have we managed to encode correctly "being repeatedly outside the critical section"?

The problem is even greater than for our specific case. No CTL formula selects these kinds of "fair" paths [1]. Model checkers therefore take them as a separate assumption. A *fairness constraint* is a CTL formula. A path $s_0 \longrightarrow s_1 \longrightarrow \cdots$ is *fair* with respect to a set $C = \{\psi_1, \ldots, \psi_k\}$ of fairness constraints if, for each $\psi_i$, infinitely many states $s_j$ of the path satisfy $\psi_i$. The *fair quantifiers* $\exists_C$ and $\forall_C$ are $\exists$ and $\forall$ restricted to fair paths: for example, $s$ satisfies $\exists_C \Box \varphi$ if some fair path from $s$ satisfies $\Box\varphi$ [1].

**Exercise 14 (Comparing fair and plain quantifiers).** Use the fairness constraints $C = \{q\}$. For the initial state of each LTS below, tick the formulas that hold.




(A) (B) (C) (D)

| LTS | $\exists\Box p$ | $\exists_C\Box p$ | $\forall\Diamond r$ | $\forall_C\Diamond r$ |
|---|---|---|---|---|
| (A) | ( ) | ( ) | ( ) | ( ) |
| (B) | ( ) | ( ) | ( ) | ( ) |
| (C) | ( ) | ( ) | ( ) | ( ) |
| (D) | ( ) | ( ) | ( ) | ( ) |

**Exercise 15 (Reducing the fair connectives to some fair path always).** Whether a path is fair depends only on its tail: a path is fair if and only if its suffix from any of its states is fair. In particular, $\mathrm{SAT}(\exists_C \Box \top)$ is the set of states from which some fair path starts. Complete the equivalences (the first one is given as example).

$$\exists_C \bigcirc \varphi \equiv \exists \bigcirc (\varphi \wedge \exists_C \Box \top)$$
$$\exists_C (\varphi \,\mathrm{U}\, \psi) \equiv \exists \big(\varphi \,\mathrm{U}\, (\psi \wedge _____)\big)$$
$$\exists_C \Diamond \varphi \equiv \exists \Diamond (\varphi \wedge _____)$$

Mark the conclusion with a checkmark (✓):

( ) Only $\exists_C \Box$ needs a new algorithm: the other fair connectives are computed with the algorithms of Section 1, once $\mathrm{SAT}(\exists_C \Box \top)$ is known.

( ) Every fair connective needs its own algorithm.

( ) The fair connectives are computed exactly like the plain ones, because $\exists_C \Box \top$ holds in every state.

**Exercise 16 (Checking liveness on fair paths).** Use the constraints $C = \{\neg c_1, \neg c_2\}$ on the LTS of Figure 7: each process is outside its critical section infinitely often. On fair paths, liveness is $\theta_C = \forall_C \Box (t_1 \to \forall_C \Diamond c_1) \equiv \neg \exists_C \Diamond (t_1 \wedge \exists_C \Box \neg c_1)$, rewritten as in Exercise 10.

**Part 1 — *Find the fair SCCs.*** A path on which $\neg c_1$ always holds stays, from some point on, in one nontrivial SCC of the restricted graph (Exercise 8). For each nontrivial SCC from Exercise 12 and listed in the table below, tick whether it contains a $\neg c_1$-state, whether it contains a $\neg c_2$-state, and whether some fair path stays in it from some point on.

| SCC | Has a $\neg c_1$-state | Has a $\neg c_2$-state | A fair path stays in it |
|---|---|---|---|
| $\{s_0, s_5, s_6\}$ | ( ) | ( ) | ( ) |
| $\{s_7\}$ | ( ) | ( ) | ( ) |

An SCC is *fair* if some fair path stays in it from some point on. Mark the condition for an SCC to be fair:

( ) Every state of the SCC satisfies every $\psi_i$.

( ) For each $\psi_i$, some state of the SCC satisfies $\psi_i$.

( ) Some state of the SCC satisfies some $\psi_i$.

**Part 2 — *Compute the satisfaction sets.*** Complete the table with the satisfaction sets of the graph in Figure 7. Compute $\mathrm{SAT}(\exists_C \Box \neg c_1)$ with the fair SCCs you found above, and $\mathrm{SAT}(\exists_C \Diamond (t_1 \wedge \exists_C \Box \neg c_1))$ with the equivalences of Exercise 15. The first row is the example: the whole LTS is one SCC, and it is fair.

| Formula | Satisfaction set in Figure 7 |
|---|---|
| $\exists_C \Box \top$ | $S$ |
| $\exists_C \Box \neg c_1$ | _____ |
| $t_1 \wedge \exists_C \Box \neg c_1$ | _____ |
| $\exists_C \Diamond (t_1 \wedge \exists_C \Box \neg c_1)$ | _____ |
| $\theta_C$ | _____ |

Does (fair) liveness of process 1 hold now? Compare with your results in Exercises 11 and 12, and mark the conclusion with a checkmark (✓):

( ) Yes, because fairness deletes the self-loop at $s_7$, so the LTS is again the one of Exercise 11.

( ) Yes, on every fair path, as in the model without self-loops (Exercise 11). The paths that violated liveness in Exercise 12 keep process 2 in $s_7$ for ever. They are still paths of the LTS, but they are not fair, because $c_2$ holds for ever on them.

( ) No: as in Exercise 12, process 1 can wait for ever while process 2 stays in $s_7$.

## 2.2 Practice: the microwave oven on fair paths

**Exercise 17 (Checking that a started oven eventually heats, on fair paths).** In Exercise 5, the property $\theta = \forall \Box (\mathit{start} \rightarrow \forall \Diamond \mathit{heat})$ failed in the microwave oven of Figure 3 due to paths on which the user never operates it correctly. Check it again on fair paths:

**Part 1 — *Choose the fairness constraint.*** For fairness consider only the paths on which the user operates the oven correctly: the start button is pressed with the door closed and no error [2]. Complete the constraint and its satisfaction set:

$$C = \{start \wedge _____\}, \qquad \mathrm{SAT}(start \wedge close \wedge \neg error) = _____.$$

**Part 2 — *Rewrite.*** Rewrite the fair version of $\theta$ as in Exercise 10 with the equivalences of Exercise 15:

$$\theta_C = \forall_C \Box(start \rightarrow \forall_C \Diamond heat) \equiv \neg\exists_C \Diamond(start \wedge _____).$$

**Part 3 — *Find the fair SCCs.*** Answer the two questions.

a) From Exercise 7, what are the nontrivial SCCs of Figure 3 restricted to the $\neg heat$-states?

_____

b) Are they fair? Mark the answer with a checkmark (✓):

( ) Yes: they contain states that satisfy *start*.

( ) No: none of their states satisfies $start \wedge close \wedge \neg error$.

( ) Yes: a path can stay in them for ever.

**Part 4 — *Compute the satisfaction sets.*** Complete the table with the satisfaction sets in Figure 3. Compute $\mathrm{SAT}(\exists_C \Box \neg heat)$ with the fair SCCs you found above.

| Formula | Satisfaction set in Figure 3 |
|---|---|
| $\exists_C \Box \neg heat$ | _____ |
| $start \wedge \exists_C \Box \neg heat$ | _____ |
| $\exists_C \Diamond(start \wedge \exists_C \Box \neg heat)$ | _____ |
| $\theta_C$ | _____ |

Does a started oven now eventually heat? Compare with your result in Exercise 5, and mark the conclusion with a checkmark (✓):

( ) Yes, because fairness removes the transitions between $m_2$ and $m_5$.

( ) Yes, on every fair path.

( ) No: after pressing start, the user can still open and close the door for ever.

# 3 LTL: model checking with automata

As we have seen, CTL model checkers take advantage of the fact that subformulas hold or fail at a state, so a state can be labelled with them. LTL subformulas, however, hold or fail along a path. No labelling of states would work. Thus, LTL model checkers follow a different strategy, in four steps [1]:

1. Build a Büchi automaton $\mathcal{B}_{\neg\varphi}$ that accepts exactly the traces satisfying $\neg\varphi$, the traces that *violate* $\varphi$ (Sections 3.1–3.3).
2. Turn the LTS into a Büchi automaton $\mathcal{B}_{\mathcal{M}}$ whose words are its traces (Section 3.4).
3. Combine the two automata (Section 3.5).
4. Check whether the combination accepts any word (Section 3.6). If it accepts none, $\varphi$ holds. If it accepts one, its run is a counterexample.

The negation turns "every path satisfies $\varphi$" into a search for *one* path, which is what the algorithms of the previous sections do.

## 3.1 Infinite words

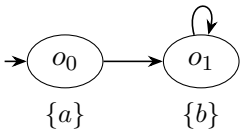
The *trace* of a path $\pi = s_0 s_1 s_2 \ldots$ is the infinite word $L(s_0)L(s_1)L(s_2)\ldots$ of its labels [3]. For a finite word or sequence $w$, we write $w^\omega$ for its infinite repetition $www\ldots$ Like a power, the superscript $^\omega$ applies to the letter or the parenthesised word just before it: $ab^\omega = abbb\ldots$

**Exercise 18 (Writing infinite words).**

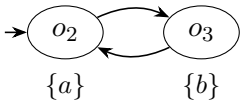
**Part 1 — *Unfolding the repetition.*** Complete the table with the word in $\omega$-notation or with its first letters. The first row is the example.

| Word | First letters |
|---|---|
| $ab^\omega$ | $abbbbbbb\ldots b\ldots$ |
| $(ab)^\omega$ | _____ |
| $(abb)^\omega$ | _____ |
| _____ | $aabababa\ldots ba\ldots$ |
| _____ | $bbbaaaaa\ldots a\ldots$ |

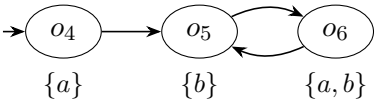
**Part 2 — *Paths and traces of small LTSs.*** In each LTS below, every state has exactly one successor, so exactly one path follows from the initial state.



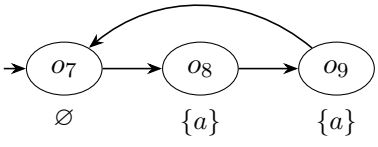

(i)




(ii)




(iii)




(iv)

Complete the table with the path of each LTS and its trace, in $\omega$-notation.

| LTS | Path | Trace |
|---|---|---|
| (i) | $o_0\, o_1^\omega$ | $\{a\}\,\{b\}^\omega$ |
| (ii) | _____ | _____ |
| (iii) | _____ | _____ |
| (iv) | _____ | _____ |

## 3.2 Büchi automata: accepting infinite words

Step 1 of LTL model-checking needs a mechanism that classifies traces into accepted and rejected ones. The devices we will use are *(nondeterministic) Büchi automata* [3]. The three diagrams in Figure 8 represent Büchi automata (over the alphabet $\Sigma = \{a, b\}$). In a drawing, an incoming arrow marks an initial state and a double circle a final state.

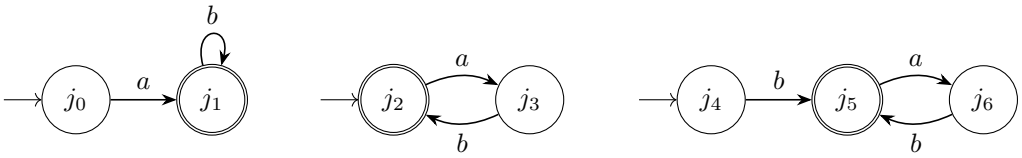



Figure 8: Three Büchi automata over the alphabet $\Sigma = \{a, b\}$.

**Exercise 19 (Definition of a Büchi automaton).**

**Part 1 — *The ingredients.*** Complete the definition by matching the columns. A *nondeterministic Büchi automaton* is a structure $\mathcal{B} = (Q, \Sigma, \delta, Q_0, F)$, where [3]:

| **Symbol** | **Meaning** |
|---|---|
| a) $Q$ | ( ) is the set of initial states |
| b) $\Sigma$ | ( ) is the transition function: $\delta(s, \ell)$ is the set of states where $s$ can move to by reading the letter $\ell$. In a drawing, an edge from $s$ to $s'$ labelled $\ell$ means $s' \in \delta(s, \ell)$. |
| c) $\delta\colon Q \times \Sigma \to \mathcal{P}(Q)$ | ( ) is the set of *accept states* |
| d) $Q_0 \subseteq Q$ | ( ) is a finite set of states |
| e) $F \subseteq Q$ | ( ) is the alphabet, a finite set of letters |

**Part 2 — *Reading a Büchi automaton.*** Complete the description of the automaton (i) in Figure 8. An example is given.

a) The set of states is $Q =$ _____.

b) The set of initial states is $Q_0 =$ _____.

c) The set of accept states is $F =$ _____.

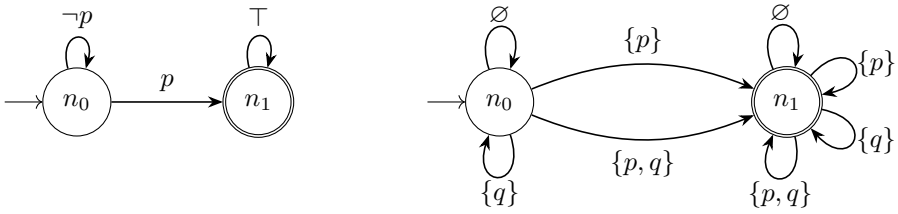
d) The transitions from $j_0$ are $\delta(j_0, a) =$ _____ and $\delta(j_0, b) = \varnothing$.

e) The transitions from $j_1$ are $\delta(j_1, a) =$ _____ and $\delta(j_1, b) =$ _____.

**Part 3 — *Runs and accepted words.*** An *infinite word* over $\Sigma$ is a sequence $\sigma : \mathbb{N} \to \Sigma$ of letters $\ell_i \in \Sigma$, that is $\sigma = \ell_0\ell_1\ell_2\ldots$. Given a Büchi automaton $\mathcal{B} = (Q, \Sigma, \delta, Q_0, F)$, a *run* of $\mathcal{B}$ for $\sigma$ is an infinite sequence of states $(s_0 s_1 s_2 \ldots)$ that starts in an initial state and reads the letters of $\sigma$ one by one: $s_0 \in Q_0$, and $s_{i+1} \in \delta(s_i, \sigma(i))$ for every $i \in \mathbb{N}$. A run is *accepting* if it visits $F$ infinitely often: $s_i \in F$ for infinitely many $i \in \mathbb{N}$. The Büchi automaton $\mathcal{B}$ *accepts* $\sigma$ if *some* run for $\sigma$ is accepting [3]. Each automaton in Figure 8 accepts exactly one word. Complete the table with its accepting run and the word. The first row is the example.

| Automaton | Accepting run | Accepted word |
|---|---|---|
| (i) | $j_0\, j_1^\omega$ | $ab^\omega$ |
| (ii) | _____ | _____ |
| (iii) | _____ | _____ |

**Exercise 20 (Automata for LTL formulas).** Step 1 needs an automaton that accepts exactly the traces satisfying a given LTL formula. A Büchi automaton *captures* an LTL formula $\varphi$ if it accepts exactly the words that satisfy $\varphi$ [3]. Consider, for instance, the automata below (shown in its condensed and expanded form) that represents $\mathrm{F}\, p$:




Notice that the condensed form uses formulas to keep drawings small: an edge labelled with a propositional formula $\psi$ represents several edges. All those edges make the formula true. Thus, if the set of atomic propositions is $P = \{p, q\}$, a $\top$-edge represents four edges as above.

**Part 1 — *One automaton per operator.*** Match each formula with its expansion law and with the automaton (i)–(iv) that captures it. The first row is provided. The automata (i)–(iv) read letters over the atomic propositions $P = \{p, q\}$.

| Formula | Expansion law | Automaton |
|---|---|---|
| $\mathrm{F}\, p$ | (c) | (i) |
| $\mathrm{G}\, \neg p$ | (  ) | (  ) |
| $p\, \mathrm{U}\, q$ | (  ) | (  ) |
| $\mathrm{X}\, p$ | (  ) | (  ) |

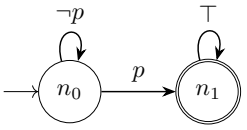
**Expansion laws**

a) $q \vee \big(p \wedge \mathrm{X}(p \,\mathrm{U}\, q)\big)$

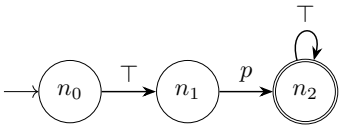
b) $\neg p \wedge \mathrm{X}\,\mathrm{G}\,\neg p$

c) $p \vee \mathrm{X}\,\mathrm{F}\,p$

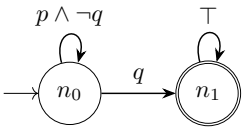
d) none: the formula only constrains the next position



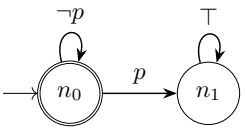

(i)




(ii)

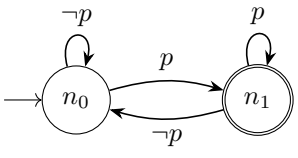



(iii)

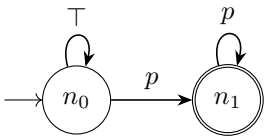



(iv)

**Part 2 — *Combining shapes.*** Automata (v) and (vi) combine the shapes of G and F.




(v)




(vi)

Tick the words that automata (i), (v) and (vi) accept. The first row is provided.

| Word | (i) | (v) | (vi) |
|---|---|---|---|
| $\{p\}^\omega$ | (✓) | (✓) | (✓) |
| $\{p\}\,\varnothing^\omega$ | ( ) | ( ) | ( ) |
| $(\varnothing\,\{p\})^\omega$ | ( ) | ( ) | ( ) |
| $\varnothing^\omega$ | ( ) | ( ) | ( ) |
| $\varnothing\,\{p\}^\omega$ | ( ) | ( ) | ( ) |

Compare your ticks with the truth of each formula on each word, and write next to each formula below the letter of the automaton that captures it:

( ) $\mathrm{G}\,\mathrm{F}\,p$ ( ) $\mathrm{F}\,\mathrm{G}\,p$

### 3.3 Step 1: an automaton for the negated property

We will model-check the property $\varphi = \neg(a \,\mathrm{U}\, b)$, for the system of Figure 9 [1]. Step 1 needs the automaton $\mathcal{B}_{\neg\varphi}$ for $\neg\varphi \equiv a \,\mathrm{U}\, b$.

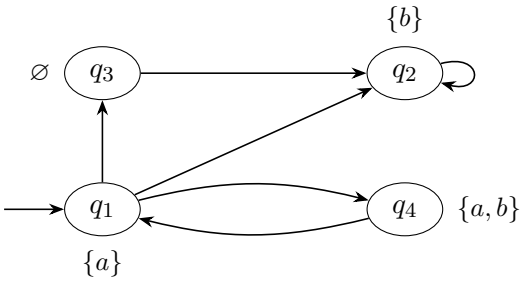



Figure 9: The LTS of a system with Boolean variables $a$ and $b$.

**Exercise 21 (The automaton for the negated property).** Figure 10 shows $\mathcal{B}_{\neg\varphi}$ without its edge labels. Complete them with propositional formulas over $a$ and $b$. Exercise 20 has an automaton of the same shape.

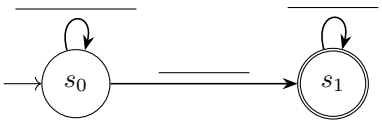


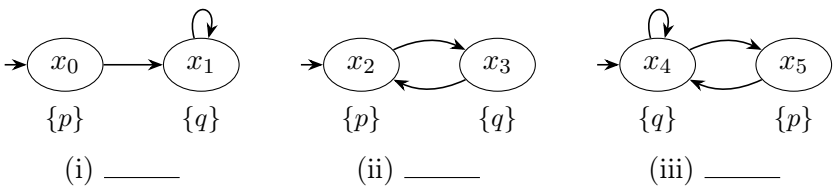

Figure 10: The automaton $\mathcal{B}_{\neg\varphi}$ for $\neg\varphi \equiv a \,\mathrm{U}\, b$.

### 3.4 Step 2: the LTS as a Büchi automaton

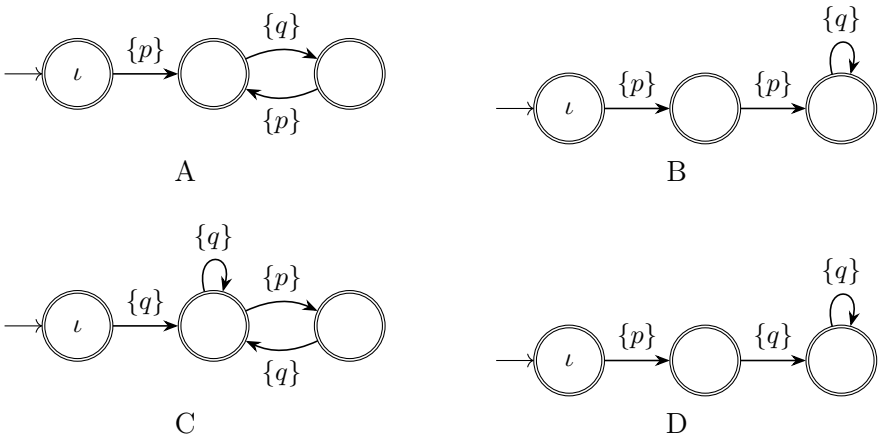
Step 2 turns the LTS into a Büchi automaton whose words are the traces of the system. For an LTS $\mathcal{M} = \langle S, \longrightarrow, L\rangle$ over the atomic propositions $P$, with initial state $q_{\text{init}}$, the automaton $\mathcal{B}_{\mathcal{M}} = (Q, \Sigma, \delta, Q_0, F)$ has [2]:

- the states $Q = S \cup \{\iota\}$, where the fresh state $\iota$ is the only initial state: $Q_0 = \{\iota\}$;
- the alphabet $\Sigma = \mathcal{P}(P)$ and the accept states $F = Q$;
- for the states $q, q' \in S$ of the LTS, $q' \in \delta(q, \ell)$ exactly when $q \longrightarrow q'$ and $\ell = L(q')$, and $q_{\text{init}} \in \delta(\iota, \ell)$ exactly when $\ell = L(q_{\text{init}})$.

**Exercise 22 (An LTS as a Büchi automaton).** The propositions are $P = \{p, q\}$. For each LTS (i)–(iii), its automaton is one of A–D, drawn without the names of the states other than $\iota$. Write the letter of its automaton under each LTS.




(i) _____ (ii) _____ (iii) _____




**Exercise 23 (The system as a Büchi automaton).** Figure 11 shows the automaton $\mathcal{B}_{\mathcal{M}}$ of the LTS of Figure 9 without its edge letters. Complete them.

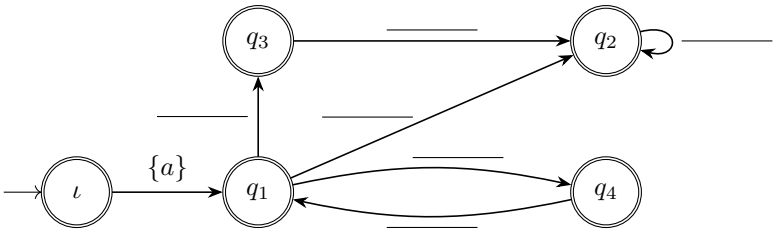



Figure 11: The automaton $\mathcal{B}_{\mathcal{M}}$ of the LTS of Figure 9.

### 3.5 Step 3: combining the two automata

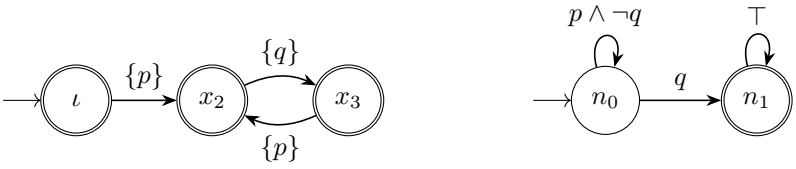
Step 3 combines $\mathcal{B}_{\mathcal{M}}$ and $\mathcal{B}_{\neg\varphi}$ into one Büchi automaton that reads a word only if both automata can read it. The combination of $\mathcal{B}_{\mathcal{M}}$ and $\mathcal{B}_{\neg\varphi}$ has [3]:

- the states $(q, s)$, pairs of a state $q$ of $\mathcal{B}_{\mathcal{M}}$ and a state $s$ of $\mathcal{B}_{\neg\varphi}$,
- the initial state $(\iota, s_0)$,
- the transition $(q, s) \longrightarrow (q', s')$, reading $\ell$, exactly when both automata read the same letter $\ell$: $q' \in \delta_{\mathcal{M}}(q, \ell)$ and $s' \in \delta_{\neg\varphi}(s, \ell)$;
- the accept states $(q, s)$ for which $s$ is an accept state of $\mathcal{B}_{\neg\varphi}$.[4]

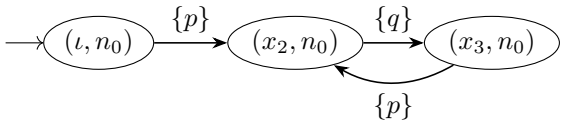
**Note:** Only the pairs reachable from the initial state are drawn below.

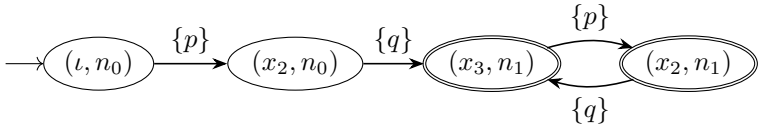
**Exercise 24 (A combination of two small automata).** Combine the two automata below over the propositions $\{p, q\}$: on the left, the automaton of LTS (ii) in Exercise 22; on the right, the automaton for $p \,\mathrm{U}\, q$ of Exercise 20.

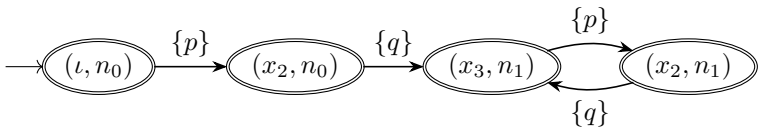
[4] This works because every state of $\mathcal{B}_{\mathcal{M}}$ is accepting [2]. In general, for Büchi automata $\mathcal{B}_k = (Q_k, \Sigma, \delta_k, Q_{0,k}, F_k)$, $k = 1, 2$, taking $F_1 \times F_2$ as accept states does not work: a run may visit $F_1$ and $F_2$ infinitely often but both at once only finitely often. The automaton accepting $\mathcal{L}(\mathcal{B}_1) \cap \mathcal{L}(\mathcal{B}_2)$ has the states $Q_1 \times Q_2 \times \{0, 1, 2\}$, the initial states $Q_{0,1} \times Q_{0,2} \times \{0\}$, the accept states $Q_1 \times Q_2 \times \{2\}$, and the transitions $(q_1, q_2, x) \longrightarrow (q_1', q_2', y)$ reading $\ell$ when $q_1' \in \delta_1(q_1, \ell)$, $q_2' \in \delta_2(q_2, \ell)$, and $y = 1$ if $x = 0$ and $q_1' \in F_1$, $y = 2$ if $x = 1$ and $q_2' \in F_2$, $y = 0$ if $x = 2$, and $y = x$ otherwise [2]. The third component goes from 0 to 1 when the run sees $F_1$, from 1 to 2 when it then sees $F_2$, and back to 0.




Which drawing is their combination? _____

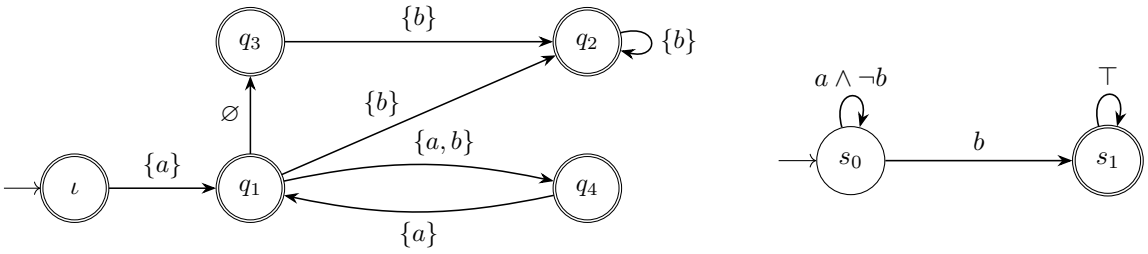
(1)



(2)



(3)



**Exercise 25 (Combining the system with the automaton).**

**Part 1** — ***The reachable pairs.*** Combine $\mathcal{B}_{\mathcal{M}}$ of Exercise 23 (left) with $\mathcal{B}_{\neg\varphi}$ of Exercise 21 (right), starting from $(\iota, s_0)$.



In a pair $(q, s)$, if the automaton $\mathcal{B}_{\mathcal{M}}$ moves from $q$ to a successor $q'$ by reading $L(q')$, then the combination with $\mathcal{B}_{\neg\varphi}$ must also read $L(q')$ from $s$. Complete $\delta_{\neg\varphi}(s, L(q'))$ and the pairs it produces. Write "none" when no pair is produced.

| Pair $(q, s)$ | Successor $q'$ | $L(q')$ | $\delta_{\neg\varphi}(s, L(q'))$ | Landing pair |
|---|---|---|---|---|
| $(\iota, s_0)$ | $q_1$ | $\{a\}$ | $\{s_0\}$ | $(q_1, s_0)$ |
| $(q_1, s_0)$ | $q_2$ | $\{b\}$ | _____ | _____ |
| $(q_1, s_0)$ | $q_4$ | $\{a, b\}$ | _____ | _____ |
| $(q_1, s_0)$ | $q_3$ | $\varnothing$ | _____ | _____ |
| $(q_2, s_1)$ | $q_2$ | $\{b\}$ | _____ | _____ |
| $(q_4, s_1)$ | $q_1$ | $\{a\}$ | _____ | _____ |
| $(q_1, s_1)$ | $q_2$ | $\{b\}$ | _____ | _____ |
| $(q_1, s_1)$ | $q_3$ | $\varnothing$ | _____ | _____ |
| $(q_1, s_1)$ | $q_4$ | $\{a, b\}$ | _____ | _____ |
| $(q_3, s_1)$ | $q_2$ | $\{b\}$ | _____ | _____ |

**Part 2 — *The combination.*** Draw the combination below.

### 3.6 Step 4: checking acceptance

The last step checks whether the combination accepts a word: whether it has a run from the initial state that visits accept states infinitely often. Label each accept state of the combination with a new atomic proposition *acc*. A run visits accept states infinitely often exactly when it is fair with respect to $C = \{acc\}$, so the combination accepts a word exactly when its initial state satisfies $\exists_C \Box \top$. The fair-SCC method of Section 2 decides this. Some model checkers implement step 4 in this way [1].

**Exercise 26 (Finding the counterexamples).**

**Part 1 — *Nontrivial SCCs.*** Write the nontrivial SCCs of the combination of Exercise 25:

$\{(q_2, s_1)\}$ and _____.

**Part 2 — *Two counterexamples.*** Both SCCs are reachable from $(\iota, s_0)$ and contain accept states, so each of them gives counterexamples. For each SCC, complete a path of the system whose run enters the SCC and stays in it for ever, and its trace. These are not the only counterexamples: for instance, $q_1\, q_4\, q_1\, q_2^{\omega}$ is another one.

| SCC | Path of the system | Trace |
|---|---|---|
| $\{(q_2, s_1)\}$ | $q_1\, q_2^{\omega}$ | $\{a\}\,\{b\}^{\omega}$ |
| $\{(q_1, s_1), (q_4, s_1)\}$ | _____ | _____ |

**Part 3 — *The verdict.*** Do both traces satisfy $a \,\mathrm{U}\, b$? _____.
Therefore, does $\varphi = \neg(a \,\mathrm{U}\, b)$ hold at $q_1$? _____.

## References

[1] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*, 2nd edition. Cambridge University Press, 2004.

[2] Edmund M. Clarke, Orna Grumberg and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[3] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.