1 Automata and logic on infinite words

As a motivating example, we describe briefly one method in the area of model checking, which is a part of the broader area of (automated) verification.

Peterson’s mutual exclusion protocol

For concreteness, recall Peterson’s protocol solving the mutual exclusion problem. We imagine two processes $A, B$ running in parallel. (An abstraction of) process $A$ performs the infinite loop

\[
\text{while true do (noncrit}_A; flag_A := true; turn := B; \text{waitfor (flag}_B = false) \lor (\text{turn} = A); \text{crit}_A; \text{flag}_A := false)
\]

and process $B$ performs

\[
\text{while true do (noncrit}_B; flag_B := true; turn := A; \text{waitfor (flag}_A = false) \lor (\text{turn} = B); \text{crit}_B; \text{flag}_B := false).
\]

A global state of our system $S = A \parallel B$ (processes $A, B$ running in parallel; they communicate by shared variables) can be described by the values of a few (boolean) variables, i.e., by a (column) vector from $\{0, 1\}^k$ for some fixed $k$. Our system $S$ thus determines (can be equated with) a (finite) transition system $(G, \rightarrow)$ where $G$ is the set of all global states and $\rightarrow \subseteq G \times G$ is the generated (unlabelled) transition relation, describing how a global state can change in one step; a concrete $g_0 \in G$ is the initial state. (E.g., the vector from $\{0, 1\}^k$ which corresponds to the global state with $\text{noncrit}_A = true, \text{noncrit}_B = true, \text{flag}_A = false, \text{flag}_B = false, \text{turn} = A, \ldots$ can be the initial state.)

**Exercise.** You can draw at least a part of the graph of $S = (G, \rightarrow)$.

Each run of our system can be seen as a (potentially infinite) sequence $g_0, g_1, g_2, \ldots$ such that $g_i \rightarrow g_{i+1}$ for all $i$. The system thus determines the set of its infinite runs, which is a **language of infinite words**, i.e. a subset of $\Sigma^\omega$.
As expected, a run of $B$ whenever process $A$ (i.e., $(q_\omega B$ It is easy to construct $B$ Büchi automata $\phi$. One general method for checking if a given property $B$ Büchi automata $\phi$ is satisfied: (starting from the initial state,) the processes can never be in the critical section simultaneously, i.e., the global state $g$ satisfying $(\text{crit}_A \land \text{crit}_B)$ is not reachable; in other words, the states on each run satisfy $\neg \text{crit}_A \lor \neg \text{crit}_B$. Another desirable property, an example of a so-called liveness property, is that whenever process $A$ wants to enter the critical section, i.e., sets $\text{flag}_A := \text{true}$, it will eventually enter that section, i.e., a global state in future will satisfy $\text{crit}_A$.

One general method for checking if a given property $\phi$ is satisfied in (the initial state of) system $S$ is to construct an automaton accepting all runs that violate $\phi$, let us denote this automaton $B_{\neg \phi}$; this automaton will be combined with the (transition) system $S$, yielding an automaton $B(S,\phi)$, and we will ask if $B(S,\phi)$ accepts an (infinite) word. If yes then $S$ allows a run which violates $\phi$, if not, i.e., if $L_\omega(B(S,\phi)) = \emptyset$ [the language of all infinite words accepted by $B(S,\phi)$ is empty], then $S$ satisfies $\phi$ (i.e., all runs of $S$ satisfy $\phi$).

Büchi automata

What kind of automata do we have in mind when speaking about $B_w$, $B_{(S,\phi)}$? We mean Büchi automata; a Büchi automaton is, in fact, the usual nondeterministic finite automaton, only the acceptance condition is different (dealing with infinite words, not the finite ones). Given a (nondeterministic finite) automaton $B = (Q, \Sigma, \delta, q_0, F)$ (where $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation and $F \subseteq Q$ is the set of accepting states) we denote

$$L_\omega(B) = \{ w \in \Sigma^\omega \mid \text{there is a run of } B \text{ on } w \text{ which goes through } F \text{ infinitely often} \}.$$  

As expected, a run of $B$ on $w = a_0a_1a_2 \ldots$ is a sequence $\sigma = q_0, q_1, q_2, \ldots$ such that $q_i \overset{a_i}{\rightarrow} q_{i+1}$ (i.e., $(q_i, a_i, q_{i+1}) \in \delta$) for all $i = 0, 1, 2, \ldots$. By $\text{Infin}(\sigma)$ we denote the set of those states $q \in Q$ which appear infinitely often (i.e., infinitely many times) in $\sigma$. Hence $L_\omega(B) = \{ w \in \Sigma^\omega \mid \text{there is a run } \sigma \text{ of } B \text{ on } w \text{ such that } \text{Infin}(\sigma) \cap F \neq \emptyset \}.$

It is easy to construct $B_{\neg \phi_1}$ where $\phi_1$ is the above safety property. We can take $\langle \{ q_0, q_1 \}, \Sigma, \delta, q_0, \{ q_1 \} \rangle$ where $\Sigma = \{ 0, 1 \}^k$ as discussed above, and $\delta$ contains the triples (the transitions) $q_0 \overset{x}{\rightarrow} q_0$ for all $x \in \Sigma$ which represent the global states satisfying $\neg \text{crit}_A \lor \neg \text{crit}_B$, $q_0 \overset{y}{\rightarrow} q_1$ for all $y \in \Sigma$ which represent the global states satisfying $\text{crit}_A \land \text{crit}_B$, and $q_1 \overset{z}{\rightarrow} q_1$ for all $z \in \Sigma$.

Model checking (safety and liveness properties)

In our example system $S = A\|B$ we can be naturally interested in checking if the following property, an example of a so-called safety property, is satisfied: (starting from the initial state,) the processes can never be in the critical section simultaneously, i.e., the global state $g$ satisfying $(\text{crit}_A \land \text{crit}_B)$ is not reachable; in other words, the states on each run satisfy $\neg \text{crit}_A \lor \neg \text{crit}_B$. Another desirable property, an example of a so-called liveness property, is that whenever process $A$ wants to enter the critical section, i.e., sets $\text{flag}_A := \text{true}$, it will eventually enter that section, i.e., a global state in future will satisfy $\text{crit}_A$.

Remark. If there is a global state $g$ with no successor (there is no $g'$ such that $g \rightarrow g'$) then we can assume that there is a loop $g \rightarrow g$, by which we make all maximal runs infinite. This is usually the case at so called Kripke structures, which are, in fact, nothing else than the transition systems as described above. (We have a set of atomic propositions, like “turn = $B$”, “crit$_A$”, etc., which are either true or false in each given [global] state.)
Remark. Our automaton is, in fact, deterministic. A certain disadvantage of Büchi automata, discussed also later, is that the nondeterministic version is more powerful than the deterministic one (unlike the case for finite automata accepting languages of finite words). This is demonstrated in the next exercise.

Exercise. Construct a (nondeterministic) Büchi automaton \( B \) such that \( L_\omega(B) = \{ w \in \{0,1\}^\omega \mid w \text{ contains only finitely many 1's} \} \). There is no deterministic Büchi automaton accepting this language; can you prove this?

Exercise. Construct \( B_{\neg \phi_2} \) where \( \phi_2 \) is the above liveness property.

We now look how to create the above mentioned automaton \( B(S,\phi) \), which is a combination of the transition system \( S \) (like the one generated by our “Peterson’s system”) and the Büchi automaton \( B = B_{\neg \phi} \). In principle, we will do the usual product construction (which synchronizes the runs of \( S \) and \( B \)); we just have to overcome the technical problem that the states of \( S \) are, in fact, the letters for \( B \). But this problem is easy: we can introduce a special “starting state” \( q_{\text{start}} \) and add a special “exit state” \( e_g \) to each \( g \in S \); now the original states \( g \in S \) are handled as letters (edge-labels), and we put \( q_{\text{start}} \overset{q}{\rightarrow} e_{g_0} \), and \( e_g \overset{g'}{\rightarrow} e_{g'} \) for all \( g \overset{g'}{\rightarrow} \). Then we can define \( B(S,\phi) = A_S \times B_{\neg \phi} \), where \( A_S \) is the (nondeterministic finite) automaton arising from \( S \) as discussed above.

Exercise. Give a precise definition of \( A \times B \).

There are more variants how to define the accepting states of the product automaton. In our discussed case, \( A_S \) has no accepting states, and the accepting states of \( A_S \times B_{\neg \phi} \) will be the pairs \( (q_1,q_2) \) with \( q_2 \in F \), where \( F \) is the set of accepting states of \( B_{\neg \phi} \). (Why?)

Recall that our general (model checking) method finally checks whether \( L_\omega(B(S,\phi)) = \emptyset \).

Exercise. Suggest a method how to decide whether \( L_\omega(B) = \emptyset \) for a given (description of a) Büchi automaton \( B \).

Note that if \( L_\omega(B(S,\phi)) \neq \emptyset \) then you can provide a counterexample, i.e., (a description of) a run of the (original) system \( S \) (like the “Peterson’s system”) which violates the property \( \phi \).

Monadic second order logic S1S

Now we are interested in characterizing what kind of properties \( \phi \) allow to construct the appropriate Büchi automata \( B_\phi \) (or \( B_{\neg \phi} \)). It turns out that they are precisely those expressible in the monadic second-order logic of one successor, briefly denoted as S1S.

Recall what is the first order logic: the language contains (symbols for) variables \( x, y, z, \ldots \), logical connectives like \( \neg, \lor, \land, \rightarrow, \leftrightarrow \), quantifiers \( \exists, \forall \), usually also the special predicate = (obligatorily interpreted as the equality), and “nonlogical” symbols, i.e., some function symbols \( f, g, \ldots \) (with their arities) and/or some predicate symbols \( P, Q, \ldots \) (with their arities).

Exercise. Recall the syntactic rules for creating terms, atomic formulas, formulas (including the “syntactic sugar” like parantheses), free and bound occurrences of variables, etc., and the interpretation of terms and formulas in concrete (mathematical) structures.

Remark. Also recall that we can “narrow” the logical symbols to \( \neg, \lor, \exists \), and handle the other connectives and \( \forall \) as derived. For expressing statements it is useful to have the “broad” logic, for proving things about the logic it is useful to keep it “narrow”.

3
Logic S1S has all ingredients of the first order logics, with only one nonlogical symbol: the unary function symbol $s$ (“successor”). Moreover, it is a second-order logic, so it also has variables $X, Y, Z, \ldots$ for predicates but it is monadic, i.e., these variables only range over unary predicates, i.e., sets. Finally, we have a special predicate $\in$; as expected, its type only allows to use it in atomic formulas of the form $t \in X$ where $t$ is a (1st order) term (so $t = ss \ldots sx$ for some (maybe zero) number of occurrences of $s$).

We have a concrete structure in mind, in which we interpret logic S1S, namely the structure $(\mathbb{N}, s, \in)$, with the usual interpretation $(s(n) = n + 1, n \in C$ iff number $n$ is an element of set $C$).

**Remark.** Note that we follow the usual practice and use the same typographical symbols $s, \in$ for both the symbols in the logic and for denoting one concrete function and one concrete predicate in the structure $\mathbb{N}$. It is also useful to note that here $\mathbb{N} = \{0, 1, 2, \ldots\}$ serves primarily for modelling (discrete) time (rather than for arithmetics).

We will naturally need to express that a time point $x$ is initial, i.e., $x = 0$. The constant 0 is not in the logic S1S (for keeping the logic as “narrow” as possible) but $x = 0$ can be expressed by the formula $\forall y.(x = sy)$. In fact, even “$=$” is not included in S1S since $t_1 = t_2$ can be expressed by $\forall X(t_1 \in X \leftrightarrow t_2 \in X)$, and $X = Y$ can be expressed by $\forall x(x \in X \leftrightarrow x \in Y)$. It is certainly also useful to express that a time point $x$ is earlier than $y$, i.e., $x < y$.

**Exercise.** Find an S1S-formula expressing $x < y$ (in our intended structure $(\mathbb{N}, s, \in)$).

Also find S1S-formulas expressing “$X$ is finite” and “$X$ is infinite”.

**Remark.** As mentioned above, it is usual (and convenient) to use the symbols $0, 1 (= s0), 2, \ldots, =, <;, \leq, \subseteq$ (for $X \subseteq Y$) as if they belonged to S1S, while we are aware that they are, in fact, just “abbreviations”.

### Expressing properties of runs in S1S

Let us look how we can express properties of runs, like our safety property $\phi_1$ and our liveness property $\phi_2$, in S1S. Imagine a run $\rho$ of (e.g., “Peterson’s”) system $S$, and recall that $\rho$ can be viewed as a sequence of column vectors $a_{0}a_{1}a_{2} \ldots (a_i \in \{0, 1\}^{k})$; we can say that in time $i \in \mathbb{N}$ the run goes through the (global) state $a_i$. Run $\rho$ thus determines a $k$-track tape with cells $0, 1, 2, \ldots$. Each track $j$ is (filled with) an infinite sequence of 0’s and 1’s (an element of $\{0, 1\}^{\omega}$), and thus can be viewed as the characteristic sequence of a set $T_j^\rho \subseteq \mathbb{N}$: $i \in T_j^\rho$ iff the value in Track $j$ in the cell (time point) $i$ is 1. In our example, one of the tracks, say Track 1, determines the set $T_1^\rho$ of time points where crit$_A$ holds; another track, say Track 2, determines the set $T_2^\rho$, the set of time points where crit$_B$ holds.

So we have seen how a sequence $\rho \in (\{0, 1\}^{k})^{\omega}$ determines the $k$-tuple of sets $T_1^\rho, T_2^\rho, \ldots, T_k^\rho$. On the other hand, each $k$-tuple $T_1, T_2, \ldots, T_k$ of subsets of $\mathbb{N}$ determines one $\rho \in (\{0, 1\}^{k})^{\omega}$ such that $T_j^\rho = T_j$ (for $j = 1, 2, \ldots, k$).

It is thus clear that the S1S formula $\Phi_1(X_1, X_2, \ldots, X_k)$ defined as $\forall i : i \in X_1 \rightarrow i \not\in X_2$ can be naturally viewed as an expression of the safety property $\phi_1$, since $\Phi_1(T_1, T_2, \ldots, T_k)$ is true precisely for those $k$-tuples of sets $T_1, T_2, \ldots, T_k \subseteq \mathbb{N}$ which correspond to sequences of (potential) global states of $S$ which do not contain any “forbidden” state, i.e., the state with crit$_A \land$ crit$_B$.

**Remark.** $\Phi_1(X_1, X_2, \ldots, X_k)$ thus determines all “safe sequences” from $(\{0, 1\}^{k})^{\omega}$, where not
all of them necessarily correspond to real runs of system \( S \). But we have that system \( S \) has (i.e., all runs of system \( S \) have) the property \( \phi_1 \) iff \( \Phi_1(T_1^\rho, T_2^\rho, \ldots, T_k^\rho) \) holds for all runs \( \rho \) of \( S \); in other words, iff there is no run \( \rho \) of \( S \) for which \( \neg \Phi_1(T_1^\rho, T_2^\rho, \ldots, T_k^\rho) \).

**Exercise.** Suggest a formula \( \Phi_2(X_1, X_2, \ldots, X_k) \) so that \( S \) has the above defined liveness property \( \phi_2 \) iff for all runs \( \rho \) of \( S \) we have \( \Phi_2(T_1^\rho, T_2^\rho, \ldots, T_k^\rho) \).

### Equivalence of S1S and Büchi automata

Now we want to show the equivalence (wrt the expressive power) between S1S and Büchi automata. The following exercise deals with the easier direction.

**Exercise.** Assume a Büchi automaton \( B = (Q, \{0, 1\}, \delta, q_0, F) \) and suggest an S1S formula \( \Phi_B(X) \) such that \( \Phi_B(T) \) for \( T \subseteq \mathbb{N} \) iff \( \text{charseq}(T) \in \mathcal{L}_\omega(B) \). (Here \( \text{charseq}(T) \in \{0, 1\}^\omega \) is the characteristic sequence of \( T \).)

Sketch how you would generalize the result for alphabet \( \{0, 1\}^k \).

Now assume a S1S formula \( \Phi(x_1, x_2, \ldots, x_m, X_1, X_2, \ldots, X_k) \). We want to construct a Büchi automaton \( B_{\Phi} \), over the alphabet \( \{0, 1\}^{m+k} \) such that \( \Phi(t_1, t_2, \ldots, t_m, T_1, T_2, \ldots, T_k) \) (for \( t_i \in \mathbb{N}, T_i \subseteq \mathbb{N} \)) is true iff \( B_{\Phi} \) accepts the infinite word \( \rho \in \{0, 1\}^{m+k} \) where \( T_1 = \{t_1\}, \ldots, T_m = \{t_m\}, T_{m+1} = T_1, \ldots, T_{m+k} = T_k \).

We proceed by induction on the structure of \( \Phi \).

**Exercise.** Show how to construct an equivalent automaton for atomic formula \( ss \ldots sx \in X \).

**Exercise.** Show how to construct an equivalent automaton for the formula \( \Phi_1(\overline{x}, \overline{X}) \lor \Phi_2(\overline{x}, \overline{X}) \) when we already have the respective automata \( B_1, B_2 \) for \( \Phi_1(\overline{x}, \overline{X}), \Phi_2(\overline{x}, \overline{X}) \). (\( \overline{\cdot} \) is a shorthand for \( x_1, \ldots, x_m \) and \( \overline{\cdot} \) is a shorthand for \( X_1, \ldots, X_k \).)

**Exercise.** Show how to construct an equivalent automaton for the formula \( \exists x_1 \Phi(x_1, \ldots, x_m, \overline{X}) \) [with free variables \( x_2, \ldots, x_m, \overline{X} \)] when we already have an automaton \( B \) equivalent with \( \Phi(x_1, \ldots, x_m, \overline{X}) \). (Hint. Use nondeterministic guessing of a [nonpresent] track content.) Generalize for the case \( \exists X_1 \Phi(\overline{x}, X_1, \ldots, X_k) \).

### Complementation; Muller automata

We can see that it only remains to handle the case \( \neg \Phi(\overline{x}, \overline{X}) \), when having \( B \) for \( \Phi(\overline{x}, \overline{X}) \). (Other logical connectives and the universal quantifier are expressible by those handled.)

We note that even in the case when \( B = (Q, \Sigma, \delta, q_0, F) \) is deterministic, the construction (of \( B' \) accepting the complement of \( \mathcal{L}_\omega(B) \)) is not immediate. (We could not just replace \( F \) with \( Q - F \); why?) This problem could be solved by using (deterministic) Muller automata. Such an automaton \( M = (Q, \Sigma, \delta, q_0, F) \) differs in that \( F \subseteq 2^Q \), i.e., there is a set of accepting sets of states (instead of a set of accepting states); the language is then defined

\[
\mathcal{L}_\omega(M) = \{ w \in \Sigma^\omega \mid \text{there is a run } \sigma \text{ of } M \text{ on } w \text{ such that } Infinit(\sigma) \in F \}.
\]

**Exercise.** Show how, given a Büchi automaton \( B \), we can construct a Muller automaton \( M \) so that \( \mathcal{L}_\omega(M) = \mathcal{L}_\omega(B) \); moreover, if \( B \) is deterministic then \( M \) is deterministic.

**Exercise.** Show how to construct an equivalent Muller automaton for the formula \( \neg \Phi(\overline{x}, \overline{X}) \) when we have a deterministic Muller automaton \( M \) equivalent with \( \Phi(\overline{x}, \overline{X}) \).
Unfortunately, Büchi automata cannot be generally determinized, as we already noted. Muller automata can be determinized, as we shall show. The next exercise thus shows that nondeterministic Muller automata, deterministic Muller automata, and nondeterministic Büchi automata are equally expressive.

**Exercise.** Show how, given a (nondeterministic) Muller automaton $\mathcal{M}$, we can construct a (nondeterministic) Büchi automaton $\mathcal{B}$ so that $L_\omega(\mathcal{B}) = L_\omega(\mathcal{M})$.

**Rabin automata**

Hence we will be done when we show how a nondeterministic Büchi automaton can be transformed to an equivalent deterministic Muller automaton. As a convenient intermediate step, we introduce Rabin automata:

A Rabin automaton $\mathcal{R}$ is defined by $(Q, \Sigma, \delta, q_0)$ and some pairs $(G_1, R_1), \ldots, (G_m, R_m)$; where $G_i, R_i \subseteq Q$ (think of $G$ as “green light” and $R$ as “red light”); the accepted language is now defined as follows:

$$L_\omega(\mathcal{R}) = \{ w \in \Sigma^\omega \mid \text{there is a run } \sigma \text{ of } \mathcal{R} \text{ on } w \text{ such that for some } i \text{ we have } \text{Inf}_\sigma(\sigma) \cap G_i \neq \emptyset \text{ and } \text{Inf}_\sigma(\sigma) \cap R_i = \emptyset \}.$$  

(A green light is on infinitely often while the respective red light is on only finitely often.)

**Exercise.** Show that a Rabin automaton $\mathcal{R}$ can be transformed to an equivalent Muller automaton $\mathcal{M}$; moreover, if $\mathcal{R}$ is deterministic then $\mathcal{M}$ is deterministic.

**Safra’s construction (nondet-Büchi $\rightarrow$ det-Rabin)**

The crucial point is a construction due to Safra; we now explain its basic ideas.

Assume a given nondeterministic Büchi automaton $\mathcal{B} = (Q, \Sigma, \delta, q_0, F)$ over alphabet $\Sigma$, with $n$ states; we will construct an equivalent deterministic Rabin automaton $\mathcal{R}$ (thus $L_\omega(\mathcal{B}) = L_\omega(\mathcal{R})$), with $2^{O(n \log n)}$ states.

Consider a word $w = a_1a_2a_3 \cdots \in \Sigma^\omega$, and imagine the respective “determinized token-run” on the graph of $\mathcal{B}$: we start with a token on the initial state $q_0$, and move (and duplicate and remove) the tokens so that after processing $a_1a_2 \ldots a_t$ the tokens are precisely on the states which are reachable from $q_0$ by $a_1a_2 \ldots a_t$.

**Exercise.** Recall the construction of a deterministic finite automaton equivalent to a given nondeterministic finite automaton; this is, in fact, the above idea.

It is clear that if there is an infinite run of $\mathcal{B}$ on $w$ then the tokens can never completely disappear from the “board” in our “token-game” when processing $w$. (Why?) The other direction needs a moment of thought:

**Exercise.** Show that if the tokens never completely disappear from the board in our token-game when processing $w$ then there is an infinite run of $\mathcal{B}$ on $w$. (Hint. Note that each state $q$ which has a token after processing the prefix $a_1a_2 \ldots a_t$ ($t \geq 1$) can be reached by $a_t$ from a state $q'$ (a “predecessor”) which had a token after processing $a_1a_2 \ldots a_{t-1}$; we can imagine an appropriate edge from “vertex” $(q', t-1)$ to vertex $(q, t)$. This idea leads to an infinite tree which is finitely branching (i.e., each vertex has finitely many
outgoing edges). We can thus apply the well-known König’s Lemma which tells us that every infinite tree which is finitely branching has an infinite branch. [Can you prove this lemma?] But the above determinization is not sufficient for us.

Exercise. Show that when we just declare the sets (the token distributions) \( Q' \subseteq Q \) such that \( Q' \cap F \neq \emptyset \) as the accepting states of the constructed deterministic Büchi automaton \( \mathcal{B}' \), it can not be guaranteed that \( \omega(B') = \omega(B) \).

If we want, e.g., that the configuration \( C_t \), i.e. the distribution of tokens after processing \( a_1a_2\ldots a_t \), provides for each state \( q \) not only information if \( q \) is reachable (from \( q_0 \) by \( a_1a_2\ldots a_t \)) but also if it is reachable via \( F \), we can enhance our token-game as follows:

We imagine that the basic tokens mentioned sofar are white, and we also have a source of blue tokens. If a white token is placed on some (accepting state) \( q \in F \), we put a blue token on top of it; thus a “stack” \((white, blue)\) with height 2 arises (its bottom being on the left in our presentation), and we handle this stack as a unit in the following moving (and duplicating and removing); moreover, when there is a conflict, i.e., when in performing step \( C_t \xrightarrow{a_{t+1}} C_{t+1} \) some state \( q \) can get both the stack \((white)\) and the stack \((white, blue)\), the conflict is resolved in favour of \((white, blue)\).

Exercise. Verify that this modified token game serves to the announced aim (\( C_t \) provides information about the reachable states and also about those reachable via \( F \)). But show that even if the \((white, blue)\)-stacks are present in all \( C_j, C_{j+1}, C_{j+2}, \ldots \) (for some \( j \)) when processing \( w \), this does not necessarily mean that there is an accepting run of \( B \) on \( w \).

We have to go more deeply. Note that to each \( q \) which has a token in \( C_t \) we can attach (maybe several) vectors of the type \((i_1, i_2, \ldots, i_r)\) with \( 0 \leq i_1 < i_2 < \cdots < i_r \leq t \) presenting information that \( q \) is reachable from \( q_0 \) by \( a_1a_2\ldots a_t \) by a path visiting \( F \) at “time points” \( i_1, i_2, \ldots, i_r \). Given two such vectors \((i_1, i_2, \ldots, i_r), (j_1, j_2, \ldots, j_s)\) we can compare them wrt the following (modified lexicographic) order: \((i_1, i_2, \ldots, i_r) < (j_1, j_2, \ldots, j_s)\) (where \( x < y \) can be read as “\( x \) is better than \( y \)”) iff \( i_1 = j_1, i_2 = j_2, \ldots, i_m = j_m \), and \((i_{m+1} < j_{m+1} \text{ or } (m = s \text{ and } r > s))\), for some \( m \leq \min(r, s) \). The quality of \( q \) in \( C_t \) can be defined as the best appropriate vector \((i_1, i_2, \ldots, i_r)\) (the least wrt \(<\)): if \( q \) is reachable by \( a_1a_2\ldots a_t \) but not via \( F \) then its quality is the empty vector \((\emptyset)\) (which is worse than any nonempty vector); if \( q \) is not reachable by \( a_1a_2\ldots a_t \) then we can define its quality (in \( C_t \)) as \( \bot \), which is viewed as worse than any vector.

You can now contemplate how to implement a program which should process (reading from left to right) an input word \( a_1a_2a_3\ldots \) while being able to compare all the states of \( B \) wrt their qualities – after each prefix \( a_1a_2\ldots a_t \). For this task it is not necessary that your program always stores the current values of the qualities; finite memory should be sufficient. Hopefully you would finally come up with something like the construction described below; it uses more colours and bigger stacks of coloured tokens. We also add some “(green and red) light effects”, to make explicitly clear that the resulting program is, in fact, a deterministic Rabin automaton. (So it would be now really useful if you think a while before reading further.)

Our program starts just with a white token, i.e. a \((white)\)-stack, on \( q_0 \); this is configuration \( C_0 \).

Generally, configuration \( C_t \) will consist of stacks of coloured tokens on states of \( B \), and also contain information about which colour has its green and/or red light on, and information
about the current “age-order” of colours (some tokens of) which are present on the board; $c < c'$ is read as “$c$ is older than $c'$” (i.e., last introducing of $c$ on the board happened earlier than last introducing of $c'$).

Having $C_t$, the program (automaton) reads $a_{t+1}$ and constructs $C_{t+1}$ as follows:

- It moves (and duplicates and/or removes) all token-stacks along the arrows labelled with $a_{t+1}$; the conflicts are resolved in favour of stacks which are the least wrt the modified lexicographic order $\prec$; the stack $(c_1, c_2, \ldots, c_r)$ on $q$ in $C_t$ satisfies $c_1 < c_2 < \cdots < c_r$ and can be viewed as a substitute of the quality of $q$ in $C_t$.

Each colour which thus disappears (i.e., all tokens of this colour disappear) from the board, has its red light on in $C_{t+1}$.

- Each stack which now lies on (a state from) $F$ gets a new token on top; for all such stacks of the same type we use the tokens of the same colour which is currently not present on the board, and is thus (newly) introduced on the board. The order of introducing these (youngest) colours, for all types of stacks lying on $F$, is arbitrary.

- Any colour $c$ (there can be more of them) which is present on the board but invisible from top (since all the tokens of that colour are covered by other tokens) will have its green light on in $C_{t+1}$; moreover, all tokens above the $c$-tokens are removed – the colours which disappear in this way will also have their red light on in $C_{t+1}$.

Remark. Note that some colour can have both its green light and its red light on in $C_{t+1}$.

Exercise. Check that it is sufficient to use $n$ colours, and thus stacks of height $\leq n$ (where $n$ is the number of states of $B$).

Also verify the following properties: 1/ any two tokens of the same colour (in $C_t$) have the same (sub)stacks below; 2/ the tokens removed because of invisibility of some colour(s) all belong to the colours which completely disappear in this way.

Derive an upper bound on the number of states of the Rabin automaton, i.e. on the size of the program finite memory, showing that it is in $2^O(n \log n)$ (as announced).

Try to define the pairs $(G_t, R_t)$ to finish the definition of the Rabin automaton $R$. (Compare with the following correctness proof.)

We still need to show the correctness, i.e. that $L_{\omega}(R) = L_{\omega}(B)$.

Suppose first that $w = a_1a_2a_3 \ldots$ is accepted by $B$, i.e., there is a run $\rho = q_0 a_1 q_1 a_2 q_2 a_3 \ldots$ of $B$ on $w$ which goes through $F$ infinitely often; let $\sigma(t)$ denotes the stack which lies on $q_t$ when $R$ has processed $a_1a_2 \ldots a_t$. Let now $m \in \mathbb{N}$ be such that for some $t_0$ we have $\forall t \geq t_0 : \text{height}(\sigma(t)) \geq m$ and $\text{height}(\sigma(t)) = m$ infinitely often. (Why must such $m$ exist?)

The bottom part $(c_1, c_2, \ldots, c_m)$ of $\sigma(t)$ can change for $t = t_0, t_0 + 1, t_0 + 2, \ldots$ because of possible replacing with a stack which is lesser wrt $\prec$; but this cannot happen infinitely often. (Why?) So for some $t_0' \geq t_0$ we have that the bottom parts of height $m$ of all $\sigma(t)$, $t \geq t_0'$, are the same $(c_1, c_2, \ldots, c_m)$. This must mean that the green light of $c_m$ is on infinitely often but its red light is on only finitely often. (Why?)

Now suppose that when $R$ processes $w = a_1a_2a_3 \ldots$, some colour $c$ has its green light on infinitely often but its red light is on only finitely often. This also means that from some $t_0$ on colour $c$ is present on the board and never removed. Let $t_1 < t_2 < t_3 < \cdots$ be the time points (bigger than $t_0$) when $c$ has its green light on; this also means that in each $C_{t_j}$ there are stacks with a $c$-token on top.
**Exercise.** Show that each state $q$ which has a stack with a $c$-token on top in $C_{t+1}$ can be reached by $a_{t+1}a_{t+2} \ldots a_{t+1}$ via $F$ from some $q'$ which has a stack with a $c$-token on top in $C_{t_j}$. Then use König’s Lemma to deduce that there is an accepting run of $B$ on $w$. 