Automata-Based LTL Model Checking

Tomáš Vojnar, Ondřej Lengál

Brno University of Technology
Faculty of Information Technology
Božetěchova 2, 612 00 Brno

SAV 2025/2026

Introduction

- We need to check whether $M \models \varphi$ holds for a Kripke structure M and an LTL formula φ.
- We are going to use an automata-theoretic approach to solve the above problem.
- The semantics of LTL formulae is defined over infinite paths—hence, when considering labelling of states as letters, we need to work with infinite words over the alphabet 2^{AP}.
- We need a suitable kind of automata to represent languages of infinite words: we are going to use the so-called Büchi automata (BA) and their variants (called, in general, ω-automata).
  - At the first glance, Büchi automata look like ordinary finite automata, but they accept infinite words by *looping* through accepting states.

Introduction

- We need to check whether $M \models \varphi$ holds for a Kripke structure M and an LTL formula φ.
- We are going to use an automata-theoretic approach to solve the above problem.
- The semantics of LTL formulae is defined over infinite paths—hence, when considering labelling of states as letters, we need to work with infinite words over the alphabet 2^{AP}.
- We need a suitable kind of automata to represent languages of infinite words: we are going to use the so-called Büchi automata (BA) and their variants (called, in general, ω-automata).
  - At the first glance, Büchi automata look like ordinary finite automata, but they accept infinite words by *looping* through accepting states.
  - We transform a Kripke structure M to a BA $\mathcal{B}_M$ accepting words that correspond to the paths in $\bigcup_{s_0 \in S_0} \Pi(M, s_0)$ when only the labelling of the states is considered.

Introduction

- We need to check whether $M \models \varphi$ holds for a Kripke structure M and an LTL formula φ.
- We are going to use an automata-theoretic approach to solve the above problem.
- The semantics of LTL formulae is defined over infinite paths—hence, when considering labelling of states as letters, we need to work with infinite words over the alphabet 2^{AP}.
- We need a suitable kind of automata to represent languages of infinite words: we are going to use the so-called Büchi automata (BA) and their variants (called, in general, ω-automata).
  - At the first glance, Büchi automata look like ordinary finite automata, but they accept infinite words by *looping* through accepting states.
  - We transform a Kripke structure M to a BA $\mathcal{B}_M$ accepting words that correspond to the paths in $\bigcup_{s_0 \in S_0} \Pi(M, s_0)$ when only the labelling of the states is considered.
  - We translate an LTL formula φ into a BA $\mathcal{B}_{\neg\varphi}$ accepting words corresponding to paths π such that $\pi \not\models \varphi$. (We do not refer to any concrete M and consider paths in all Kripke structures.)

Introduction

- We need to check whether $M \models \varphi$ holds for a Kripke structure M and an LTL formula φ.
- We are going to use an automata-theoretic approach to solve the above problem.
- The semantics of LTL formulae is defined over infinite paths—hence, when considering labelling of states as letters, we need to work with infinite words over the alphabet 2^{AP}.
- We need a suitable kind of automata to represent languages of infinite words: we are going to use the so-called Büchi automata (BA) and their variants (called, in general, ω-automata).
  - At the first glance, Büchi automata look like ordinary finite automata, but they accept infinite words by *looping* through accepting states.
  - We transform a Kripke structure M to a BA $\mathcal{B}_M$ accepting words that correspond to the paths in $\bigcup_{s_0 \in S_0} \Pi(M, s_0)$ when only the labelling of the states is considered.
  - We translate an LTL formula φ into a BA $\mathcal{B}_{\neg\varphi}$ accepting words corresponding to paths π such that $\pi \not\models \varphi$. (We do not refer to any concrete M and consider paths in all Kripke structures.)
  - We check that $L(\mathcal{B}_M) \cap L(\mathcal{B}_{\neg\varphi}) = \emptyset$.

Büchi Automata
for use in LTL Model Checking

Büchi automata

- A (non-deterministic) Büchi automaton $\mathcal{B}$ is a tuple $\mathcal{B} = (Q, \Sigma, \delta, Q_0, F)$ where
  - Q is a finite set of states,
  - Σ is a finite alphabet,
  - $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation,
  - $Q_0 \subseteq Q$ is the set of initial states,
  - $F \subseteq Q$ is the set of *accepting* states.

Büchi automata

- A (non-deterministic) Büchi automaton $\mathcal{B}$ is a tuple $\mathcal{B} = (Q, \Sigma, \delta, Q_0, F)$ where
  - Q is a finite set of states,
  - Σ is a finite alphabet,
  - $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation,
  - $Q_0 \subseteq Q$ is the set of initial states,
  - $F \subseteq Q$ is the set of *accepting* states.

- Provided $(q_1, a, q_2) \in \delta$, we often write $q_1 \xrightarrow{a} q_2$.

Büchi automata

- A (non-deterministic) Büchi automaton $\mathcal{B}$ is a tuple $\mathcal{B} = (Q, \Sigma, \delta, Q_0, F)$ where
  - Q is a finite set of states,
  - Σ is a finite alphabet,
  - $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation,
  - $Q_0 \subseteq Q$ is the set of initial states,
  - $F \subseteq Q$ is the set of *accepting* states.

- Provided $(q_1, a, q_2) \in \delta$, we often write $q_1 \xrightarrow{a} q_2$.

- The language of a BA $\mathcal{B} = (Q, \Sigma, \delta, Q_0, F)$ is defined as follows:
  - A run ϱ of $\mathcal{B}$ over an infinite word $w = a_0 a_1 a_2 \ldots \in \Sigma^\omega$ is an infinite sequence $q_0 q_1 q_2 \ldots \in Q^\omega$ of states such that $q_0 \in Q_0$ and $\forall i. q_i \xrightarrow{a_i} q_{i+1}$.

Büchi automata

- A (non-deterministic) Büchi automaton $\mathcal{B}$ is a tuple $\mathcal{B} = (Q, \Sigma, \delta, Q_0, F)$ where
  - Q is a finite set of states,
  - Σ is a finite alphabet,
  - $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation,
  - $Q_0 \subseteq Q$ is the set of initial states,
  - $F \subseteq Q$ is the set of *accepting* states.

- Provided $(q_1, a, q_2) \in \delta$, we often write $q_1 \xrightarrow{a} q_2$.

- The language of a BA $\mathcal{B} = (Q, \Sigma, \delta, Q_0, F)$ is defined as follows:
  - A run ϱ of $\mathcal{B}$ over an infinite word $w = a_0 a_1 a_2 \ldots \in \Sigma^\omega$ is an infinite sequence $q_0 q_1 q_2 \ldots \in Q^\omega$ of states such that $q_0 \in Q_0$ and $\forall i. q_i \xrightarrow{a_i} q_{i+1}$.
  - A run ϱ is accepting iff $\mathtt{inf}(\varrho) \cap F \neq \emptyset$ where $\mathtt{inf}(\varrho)$ is the set of states that appear infinitely often in ϱ.

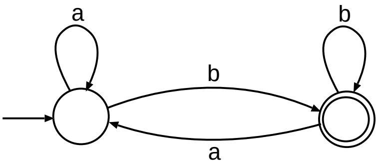
Büchi automata

- A (non-deterministic) Büchi automaton $\mathcal{B}$ is a tuple $\mathcal{B} = (Q, \Sigma, \delta, Q_0, F)$ where
  - Q is a finite set of states,
  - Σ is a finite alphabet,
  - $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation,
  - $Q_0 \subseteq Q$ is the set of initial states,
  - $F \subseteq Q$ is the set of *accepting* states.

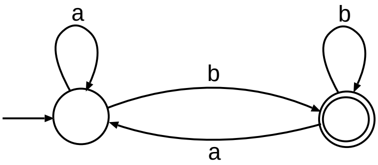
- Provided $(q_1, a, q_2) \in \delta$, we often write $q_1 \xrightarrow{a} q_2$.

- The language of a BA $\mathcal{B} = (Q, \Sigma, \delta, Q_0, F)$ is defined as follows:
  - A run ϱ of $\mathcal{B}$ over an infinite word $w = a_0 a_1 a_2 \ldots \in \Sigma^\omega$ is an infinite sequence $q_0 q_1 q_2 \ldots \in Q^\omega$ of states such that $q_0 \in Q_0$ and $\forall i. q_i \xrightarrow{a_i} q_{i+1}$.
  - A run ϱ is accepting iff $\mathtt{inf}(\varrho) \cap F \neq \emptyset$ where $\mathtt{inf}(\varrho)$ is the set of states that appear infinitely often in ϱ.
  - The language of $\mathcal{B}$ is defined as $L(\mathcal{B}) = \{w \in \Sigma^\omega \mid \text{ there is an accepting run of } \mathcal{B} \text{ over } w\}$.

Two Examples of BA

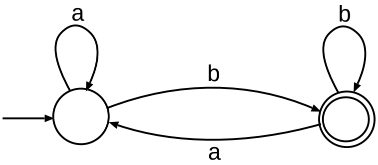


Two Examples of BA

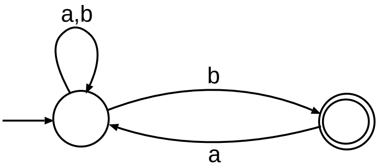


A BA accepting the language of infinite words over $\Sigma = \{a, b\}$
in which b appears infinitely often.

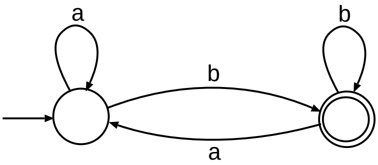
Two Examples of BA



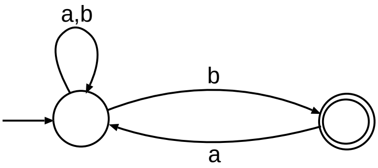
A BA accepting the language of infinite words over $\Sigma = \{a, b\}$
in which b appears infinitely often.



Two Examples of BA



A BA accepting the language of infinite words over $\Sigma = \{a, b\}$ in which *b* appears infinitely often.



A BA accepting the language of infinite words over $\Sigma = \{a, b\}$ in which *ba* appears infinitely often.

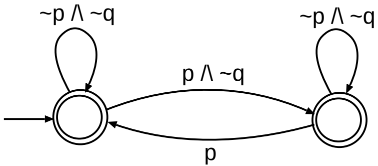
ω-Regular Languages

- When dealing with BA with $\Sigma = 2^{AP}$, we often describe (sets of) transitions using propositional formulae:
  - e.g., for $AP = \{p, q, r\}$, we may write a transition labelled with $p \wedge \neg q$ instead of two transitions labelled with $\{p\}$ and $\{p, r\}$.

ω-Regular Languages

- When dealing with BA with $\Sigma = 2^{AP}$, we often describe (sets of) transitions using propositional formulae:
  - e.g., for $AP = \{p, q, r\}$, we may write a transition labelled with $p \wedge \neg q$ instead of two transitions labelled with $\{p\}$ and $\{p, r\}$.

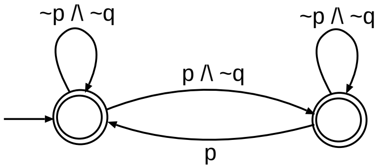
- An example:



ω-Regular Languages

- When dealing with BA with $\Sigma = 2^{AP}$, we often describe (sets of) transitions using propositional formulae:
  - e.g., for $AP = \{p, q, r\}$, we may write a transition labelled with $p \wedge \neg q$ instead of two transitions labelled with $\{p\}$ and $\{p, r\}$.

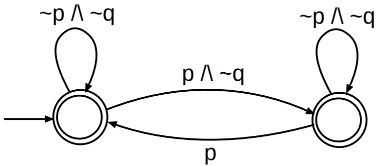
- An example:



the BA describes the set of words over 2^{AP}, $AP = \{p, q, r\}$, such that q may appear (if at all) at *even occurrences* of p only

ω-Regular Languages

- When dealing with BA with $\Sigma = 2^{AP}$, we often describe (sets of) transitions using propositional formulae:
  - e.g., for $AP = \{p, q, r\}$, we may write a transition labelled with $p \wedge \neg q$ instead of two transitions labelled with $\{p\}$ and $\{p, r\}$.
- An example:



the BA describes the set of words over 2^{AP}, $AP = \{p, q, r\}$, such that q may appear (if at all) at *even occurrences* of p only

- The above language is not expressible using LTL.
  - BA have a strictly higher expressive power than LTL.
  - The languages that are accepted by some BA are called ω-regular.

Alternative Accepting Conditions

- Several other forms of accepting conditions replacing the simple set of accepting states F are in use:
  - generalised Büchi: $\mathcal{F} \subseteq 2^Q$—a run ϱ is accepting iff $\forall F \in \mathcal{F}.\ \mathsf{inf}(\varrho) \cap F \neq \emptyset$.

Alternative Accepting Conditions

- Several other forms of accepting conditions replacing the simple set of accepting states F are in use:
  - generalised Büchi: $\mathcal{F} \subseteq 2^Q$—a run ϱ is accepting iff $\forall F \in \mathcal{F}.\ \mathsf{inf}(\varrho) \cap F \neq \emptyset$.
  - Muller: $\mathcal{F} \subseteq 2^Q$—a run ϱ is accepting iff $\exists F \in \mathcal{F}.\ \mathsf{inf}(\varrho) = F$.

Alternative Accepting Conditions

- Several other forms of accepting conditions replacing the simple set of accepting states F are in use:
  - generalised Büchi: $\mathcal{F} \subseteq 2^Q$—a run ϱ is accepting iff $\forall F \in \mathcal{F}.\ \mathtt{inf}(\varrho) \cap F \neq \emptyset$.
  - Muller: $\mathcal{F} \subseteq 2^Q$—a run ϱ is accepting iff $\exists F \in \mathcal{F}.\ \mathtt{inf}(\varrho) = F$.
  - Streett: $\mathcal{F} \subseteq 2^Q \times 2^Q$—a run ϱ is accepting iff $\forall (E, F) \in \mathcal{F}.\ \mathtt{inf}(\varrho) \cap E \neq \emptyset \implies \mathtt{inf}(\varrho) \cap F \neq \emptyset$.

Alternative Accepting Conditions

- Several other forms of accepting conditions replacing the simple set of accepting states F are in use:
  - generalised Büchi: $\mathcal{F} \subseteq 2^Q$—a run ϱ is accepting iff $\forall F \in \mathcal{F}.\ \mathrm{inf}(\varrho) \cap F \neq \emptyset$.
  - Muller: $\mathcal{F} \subseteq 2^Q$—a run ϱ is accepting iff $\exists F \in \mathcal{F}.\ \mathrm{inf}(\varrho) = F$.
  - Streett: $\mathcal{F} \subseteq 2^Q \times 2^Q$—a run ϱ is accepting iff $\forall (E, F) \in \mathcal{F}.\ \mathrm{inf}(\varrho) \cap E \neq \emptyset \implies \mathrm{inf}(\varrho) \cap F \neq \emptyset$.
  - Rabin: $\mathcal{F} \subseteq 2^Q \times 2^Q$—a run ϱ is accepting iff $\exists (E, F) \in \mathcal{F}.\ \mathrm{inf}(\varrho) \cap E = \emptyset \ \wedge\ \mathrm{inf}(\varrho) \cap F \neq \emptyset$.

Alternative Accepting Conditions

- Several other forms of accepting conditions replacing the simple set of accepting states F are in use:
  - generalised Büchi: $\mathcal{F} \subseteq 2^Q$—a run ϱ is accepting iff $\forall F \in \mathcal{F}.\ \mathsf{inf}(\varrho) \cap F \neq \emptyset$.
  - Muller: $\mathcal{F} \subseteq 2^Q$—a run ϱ is accepting iff $\exists F \in \mathcal{F}.\ \mathsf{inf}(\varrho) = F$.
  - Streett: $\mathcal{F} \subseteq 2^Q \times 2^Q$—a run ϱ is accepting iff $\forall (E, F) \in \mathcal{F}.\ \mathsf{inf}(\varrho) \cap E \neq \emptyset \implies \mathsf{inf}(\varrho) \cap F \neq \emptyset$.
  - Rabin: $\mathcal{F} \subseteq 2^Q \times 2^Q$—a run ϱ is accepting iff $\exists (E, F) \in \mathcal{F}.\ \mathsf{inf}(\varrho) \cap E = \emptyset \ \wedge\ \mathsf{inf}(\varrho) \cap F \neq \emptyset$.
  - parity: states of $\mathcal{B}$ are labelled with colours from the set $C = \{0, \ldots, k\}$ by a function $c\colon Q \to C$. A run ρ is accepting iff $\mathsf{min}\{c(q) \mid q \in \mathsf{inf}(\varrho)\}$ is even (alternative definitions for max/odd).

Alternative Accepting Conditions

- Several other forms of accepting conditions replacing the simple set of accepting states F are in use:
  - generalised Büchi: $\mathcal{F} \subseteq 2^Q$—a run ϱ is accepting iff $\forall F \in \mathcal{F}.\ \mathsf{inf}(\varrho) \cap F \neq \emptyset$.
  - Muller: $\mathcal{F} \subseteq 2^Q$—a run ϱ is accepting iff $\exists F \in \mathcal{F}.\ \mathsf{inf}(\varrho) = F$.
  - Streett: $\mathcal{F} \subseteq 2^Q \times 2^Q$—a run ϱ is accepting iff $\forall (E, F) \in \mathcal{F}.\ \mathsf{inf}(\varrho) \cap E \neq \emptyset \implies \mathsf{inf}(\varrho) \cap F \neq \emptyset$.
  - Rabin: $\mathcal{F} \subseteq 2^Q \times 2^Q$—a run ϱ is accepting iff $\exists (E, F) \in \mathcal{F}.\ \mathsf{inf}(\varrho) \cap E = \emptyset \ \wedge\ \mathsf{inf}(\varrho) \cap F \neq \emptyset$.
  - parity: states of $\mathcal{B}$ are labelled with colours from the set $C = \{0, \ldots, k\}$ by a function $c\colon Q \to C$. A run ρ is accepting iff $\mathsf{min}\{c(q) \mid q \in \mathsf{inf}(\varrho)\}$ is even (alternative definitions for max/odd).
  - Emerson-Lei: states of $\mathcal{B}$ are labelled with sets of colours from C and the acceptance condition is given by an arbitrary Boolean formula φ over atoms of the form $\mathsf{inf}(c_i)$ for $c_i \in C$. A run ρ is accepting iff $\mathsf{inf}(\varrho)$ is a model of φ.

Alternative Accepting Conditions

- Several other forms of accepting conditions replacing the simple set of accepting states F are in use:
  - generalised Büchi: $\mathcal{F} \subseteq 2^Q$—a run ϱ is accepting iff $\forall F \in \mathcal{F}.\ \mathsf{inf}(\varrho) \cap F \neq \emptyset$.
  - Muller: $\mathcal{F} \subseteq 2^Q$—a run ϱ is accepting iff $\exists F \in \mathcal{F}.\ \mathsf{inf}(\varrho) = F$.
  - Streett: $\mathcal{F} \subseteq 2^Q \times 2^Q$—a run ϱ is accepting iff $\forall (E, F) \in \mathcal{F}.\ \mathsf{inf}(\varrho) \cap E \neq \emptyset \implies \mathsf{inf}(\varrho) \cap F \neq \emptyset$.
  - Rabin: $\mathcal{F} \subseteq 2^Q \times 2^Q$—a run ϱ is accepting iff $\exists (E, F) \in \mathcal{F}.\ \mathsf{inf}(\varrho) \cap E = \emptyset \ \wedge\ \mathsf{inf}(\varrho) \cap F \neq \emptyset$.
  - parity: states of $\mathcal{B}$ are labelled with colours from the set $C = \{0, \ldots, k\}$ by a function $c\colon Q \to C$. A run ρ is accepting iff $\mathsf{min}\{c(q) \mid q \in \mathsf{inf}(\varrho)\}$ is even (alternative definitions for max/odd).
  - Emerson-Lei: states of $\mathcal{B}$ are labelled with sets of colours from C and the acceptance condition is given by an arbitrary Boolean formula φ over atoms of the form $\mathsf{inf}(c_i)$ for $c_i \in C$. A run ρ is accepting iff $\mathsf{inf}(\varrho)$ is a model of φ.
  - transition-based acceptance: as above, but states are substituted with transitions.

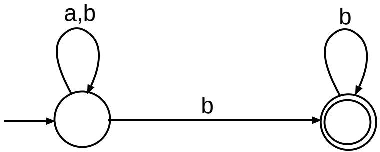
Alternative Accepting Conditions

- Several other forms of accepting conditions replacing the simple set of accepting states F are in use:
  - generalised Büchi: $\mathcal{F} \subseteq 2^Q$—a run ϱ is accepting iff $\forall F \in \mathcal{F}.\ \mathtt{inf}(\varrho) \cap F \neq \emptyset$.
  - Muller: $\mathcal{F} \subseteq 2^Q$—a run ϱ is accepting iff $\exists F \in \mathcal{F}.\ \mathtt{inf}(\varrho) = F$.
  - Streett: $\mathcal{F} \subseteq 2^Q \times 2^Q$—a run ϱ is accepting iff $\forall (E, F) \in \mathcal{F}.\ \mathtt{inf}(\varrho) \cap E \neq \emptyset \implies \mathtt{inf}(\varrho) \cap F \neq \emptyset$.
  - Rabin: $\mathcal{F} \subseteq 2^Q \times 2^Q$—a run ϱ is accepting iff $\exists (E, F) \in \mathcal{F}.\ \mathtt{inf}(\varrho) \cap E = \emptyset \ \wedge\ \mathtt{inf}(\varrho) \cap F \neq \emptyset$.
  - parity: states of $\mathcal{B}$ are labelled with colours from the set $C = \{0, \ldots, k\}$ by a function $c\colon Q \to C$. A run ρ is accepting iff $\mathtt{min}\{c(q) \mid q \in \mathtt{inf}(\varrho)\}$ is even (alternative definitions for $\mathtt{max}$/odd).
  - Emerson-Lei: states of $\mathcal{B}$ are labelled with sets of colours from C and the acceptance condition is given by an arbitrary Boolean formula φ over atoms of the form $\mathtt{inf}(c_i)$ for $c_i \in C$. A run ρ is accepting iff $\mathtt{inf}(\varrho)$ is a model of φ.
  - transition-based acceptance: as above, but states are substituted with transitions.

- All the above conditions yield automata of equal expressive power.

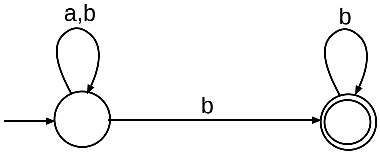
Deterministic BA

- When considering the basic Büchi acceptance condition, deterministic BA are strictly less powerful than ordinary (non-deterministic) BA.



Deterministic BA

- When considering the basic Büchi acceptance condition, deterministic BA are strictly less powerful than ordinary (non-deterministic) BA.



- The above BA expressing the language of words over $\Sigma = \{a, b\}$ in which eventually only b appears (i.e., $(a + b)^* b^\omega$) does not have a deterministic variant:

- Deterministic and non-deterministic Muller, Streett, Rabin, parity, and Emerson-Lei automata have the same expressive power.

Complementation of BA

- The automata-theoretic approach to LTL model checking could be formulated as checking whether $L(\mathcal{B}_M) \subseteq L(\mathcal{B}_\varphi)$, which would naturally reduce to using complementation to check $L(\mathcal{B}_M) \subseteq L(\mathcal{B}_\varphi)$ as $L(\mathcal{B}_M) \cap \overline{L(\mathcal{B}_\varphi)} = \emptyset$.

Complementation of BA

- The automata-theoretic approach to LTL model checking could be formulated as checking whether $L(\mathcal{B}_M) \subseteq L(\mathcal{B}_\varphi)$, which would naturally reduce to using complementation to check $L(\mathcal{B}_M) \subseteq L(\mathcal{B}_\varphi)$ as $L(\mathcal{B}_M) \cap \overline{L(\mathcal{B}_\varphi)} = \emptyset$.

- Due to the non-equivalent power of deterministic and non-deterministic BA, complementation is much more complicated than for finite-word finite automata.

- However, BA are still closed wrt complementation:
  - One can complement BA, e.g., using the so-called Safra construction going through deterministic Rabin automata.
    - The complement of a BA with n states using this way has $2^{\mathcal{O}(n\log(n))}$ states.
      (more precisely, $12^n n^{2n}$ for Safra (Rabin) and $2n^n n!$ for Piterman (parity))

Complementation of BA

- The automata-theoretic approach to LTL model checking could be formulated as checking whether $L(\mathcal{B}_M) \subseteq L(\mathcal{B}_\varphi)$, which would naturally reduce to using complementation to check $L(\mathcal{B}_M) \subseteq L(\mathcal{B}_\varphi)$ as $L(\mathcal{B}_M) \cap \overline{L(\mathcal{B}_\varphi)} = \emptyset$.

- Due to the non-equivalent power of deterministic and non-deterministic BA, complementation is much more complicated than for finite-word finite automata.

- However, BA are still closed wrt complementation:
  - One can complement BA, e.g., using the so-called Safra construction going through deterministic Rabin automata.
    - The complement of a BA with n states using this way has $2^{\mathcal{O}(n \log(n))}$ states. (more precisely, $12^n n^{2n}$ for Safra (Rabin) and $2n^n n!$ for Piterman (parity))
  - There are other procedures for complementation (the lower bound is $\Omega(\frac{1}{n}(0.76n)^n)$)
    - Ramsey-based, determinization-based, rank-based (tight: $\mathcal{O}(n(0.76n)^n)$), slice-based, learning-based, subset-tuple construction, semideterm.-based, decomposition-based (+ specialized procedures for subclasses)

Complementation of BA

- The automata-theoretic approach to LTL model checking could be formulated as checking whether $L(\mathcal{B}_M) \subseteq L(\mathcal{B}_\varphi)$, which would naturally reduce to using complementation to check $L(\mathcal{B}_M) \subseteq L(\mathcal{B}_\varphi)$ as $L(\mathcal{B}_M) \cap \overline{L(\mathcal{B}_\varphi)} = \emptyset$.

- Due to the non-equivalent power of deterministic and non-deterministic BA, complementation is much more complicated than for finite-word finite automata.

- However, BA are still closed wrt complementation:
  - One can complement BA, e.g., using the so-called Safra construction going through deterministic Rabin automata.
    - The complement of a BA with n states using this way has $2^{\mathcal{O}(n\log(n))}$ states. (more precisely, $12^n n^{2n}$ for Safra (Rabin) and $2n^n n!$ for Piterman (parity))
  - There are other procedures for complementation (the lower bound is $\Omega(\frac{1}{n}(0.76n)^n)$)
    - Ramsey-based, determinization-based, rank-based (tight: $\mathcal{O}(n(0.76n)^n)$), slice-based, learning-based, subset-tuple construction, semideterm.-based, decomposition-based (+ specialized procedures for subclasses)

- To avoid the complex complementation of BA, complementation is usually done on the level of formulae, and the model checking checks that $L(\mathcal{B}_M) \cap L(\mathcal{B}_{\neg\varphi}) = \emptyset$.

Emptiness of BA

- Emptiness of a given BA $\mathcal{B}$ can be checked in the following way:
  - compute the SCCs of $\mathcal{B}$, which can be done using the algorithm of Tarjan in time linear in the size of $\mathcal{B}$,
  - check whether there is a non-trivial SCC that contains an accepting state and is reachable from some initial state.
- The above procedure can be done in time $\mathcal{O}(|Q| + |\delta|)$.

Emptiness of BA

- Emptiness of a given BA $\mathcal{B}$ can be checked in the following way:
  - compute the SCCs of $\mathcal{B}$, which can be done using the algorithm of Tarjan in time linear in the size of $\mathcal{B}$,
  - check whether there is a non-trivial SCC that contains an accepting state and is reachable from some initial state.

- The above procedure can be done in time $\mathcal{O}(|Q| + |\delta|)$.

- Nested depth-first search—two interleaved depth-first searches:
  - The outer DFS searches for accepting states and the inner DFS tries to find a loop on the encountered, fully-expanded by the outer DFS, accepting states (while going through states not visited by the inner DFS).

Emptiness of BA

- Emptiness of a given BA $\mathcal{B}$ can be checked in the following way:
  - compute the SCCs of $\mathcal{B}$, which can be done using the algorithm of Tarjan in time linear in the size of $\mathcal{B}$,
  - check whether there is a non-trivial SCC that contains an accepting state and is reachable from some initial state.

- The above procedure can be done in time $\mathcal{O}(|Q| + |\delta|)$.

- Nested depth-first search—two interleaved depth-first searches:
  - The outer DFS searches for accepting states and the inner DFS tries to find a loop on the encountered, fully-expanded by the outer DFS, accepting states (while going through states not visited by the inner DFS).
  - *Note*: A naive two-phase DFS (first find accepting states, then search from each of them for a loop) gives time complexity $\mathcal{O}(|Q| \cdot (|Q| + |\delta|))$.

Emptiness of BA

- Emptiness of a given BA $\mathcal{B}$ can be checked in the following way:
  - compute the SCCs of $\mathcal{B}$, which can be done using the algorithm of Tarjan in time linear in the size of $\mathcal{B}$,
  - check whether there is a non-trivial SCC that contains an accepting state and is reachable from some initial state.

- The above procedure can be done in time $\mathcal{O}(|Q| + |\delta|)$.

- Nested depth-first search—two interleaved depth-first searches:
  - The outer DFS searches for accepting states and the inner DFS tries to find a loop on the encountered, fully-expanded by the outer DFS, accepting states (while going through states not visited by the inner DFS).
  - *Note*: A naive two-phase DFS (first find accepting states, then search from each of them for a loop) gives time complexity $\mathcal{O}(|Q| \cdot (|Q| + |\delta|))$.

- *In the literature, various improved versions of both the SCC-based as well as the nested DFS have been proposed: these are beyond the scope of this lecture.*

Product of BA

- Given two BA $\mathcal{B}_1, \mathcal{B}_2$, constructing a BA accepting the language $L(\mathcal{B}_1) \cap L(\mathcal{B}_2)$ is easy.
- However, one has to be careful of the fact that accepting states may be reached in $\mathcal{B}_1$ and $\mathcal{B}_2$ at different times.
  - Have two copies of the cross product of the transition graphs of $\mathcal{B}_1$ and $\mathcal{B}_2$.
  - For $q_1^1 \in F_1$, redirect each transition going from a state (q_1^1, q_1^2) to (q_2^1, q_2^2) in the first copy of the cross product to go from (q_1^1, q_1^2) in the first copy to (q_2^1, q_2^2) in the second copy.
  - Redirect in a similar fashion transitions from the second copy back to the first one.
  - Consider as accepting the states (q_1, q_2) of the second copy where $q_2 \in F_2$.

Product of BA

- Given two BA $\mathcal{B}_1, \mathcal{B}_2$, constructing a BA accepting the language $L(\mathcal{B}_1) \cap L(\mathcal{B}_2)$ is easy.
- However, one has to be careful of the fact that accepting states may be reached in $\mathcal{B}_1$ and $\mathcal{B}_2$ at different times.
  - Have two copies of the cross product of the transition graphs of $\mathcal{B}_1$ and $\mathcal{B}_2$.
  - For $q_1^1 \in F_1$, redirect each transition going from a state (q_1^1, q_1^2) to (q_2^1, q_2^2) in the first copy of the cross product to go from (q_1^1, q_1^2) in the first copy to (q_2^1, q_2^2) in the second copy.
  - Redirect in a similar fashion transitions from the second copy back to the first one.
  - Consider as accepting the states (q_1, q_2) of the second copy where $q_2 \in F_2$.
- In the LTL model checking procedure, the construction of the product may be simplified since $\mathcal{B}_M$ for a Kripke structure M will have all states accepting:
  - Hence, no need to create two copies of the cross product.
  - One can consider as accepting the states of the cross product in which the $\mathcal{B}_{\neg\varphi}$ component reaches an accepting state.

From Kripke Structures to Büchi Automata

From KS to BA

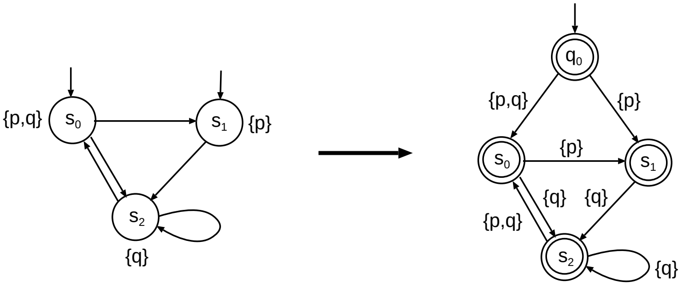
- We transform a given Kripke structure $M = (S, S_0, R, \ell)$ over atomic propositions from AP to the Büchi automaton $\mathcal{B}_M = (S \cup \{q_0\}, 2^{AP}, \delta, \{q_0\}, S \cup \{q_0\})$ where
  - $q_0 \notin S$ and
  - δ is the smallest relation such that
    - if $(s_1, s_2) \in R$, then $(s_1, \ell(s_2), s_2) \in \delta$ and
    - if $s_0 \in S_0$, then $(q_0, \ell(s_0), s_0) \in \delta$.
- We have that $L(\mathcal{B}_M) = \{\ell(s_0)\ell(s_1)\ell(s_2)\ldots \mid s_0 \in S_0 \ \wedge \ s_0 s_1 s_2 \ldots \in \Pi(M, s_0)\}$.

From KS to BA

- We transform a given Kripke structure $M = (S, S_0, R, \ell)$ over atomic propositions from AP to the Büchi automaton $\mathcal{B}_M = (S \cup \{q_0\}, 2^{AP}, \delta, \{q_0\}, S \cup \{q_0\})$ where
  - $q_0 \notin S$ and
  - δ is the smallest relation such that
    - if $(s_1, s_2) \in R$, then $(s_1, \ell(s_2), s_2) \in \delta$ and
    - if $s_0 \in S_0$, then $(q_0, \ell(s_0), s_0) \in \delta$.

- We have that $L(\mathcal{B}_M) = \{\ell(s_0)\ell(s_1)\ell(s_2)\ldots \mid s_0 \in S_0 \ \wedge \ s_0 s_1 s_2 \ldots \in \Pi(M, s_0)\}$.

- An example:



From LTL Formulae to Büchi Automata

The Idea of Going from LTL to BA

- We consider the basic connectives ($\neg$, $\vee$, X, U) only and we skip the use of the implicit A path quantifier at the beginning of the formulae.
- We introduce a state q for each consistent subset of the set of subformulae of the given formula and their negations: these are assumed to hold in q.
- We add transitions according to the observed changes in the validity of atomic propositions (the sets of the new valid atomic propositions will label the transitions) and according to the temporal operators that appear in the formulae present in the states.
- We use generalised BA: one accepting condition for each **until**.
  - The generalised BA may be converted to plain BA in a similar way as in the product construction (just using as many copies as the number of accepting conditions is).
- Various alternative, more optimised constructions have been studied (and are available in tools such as `ltl2ba`, `ltl2tgba` (Spot), `ltl3tela`, Rabinizer, . . .).

The FL Closure of a Formula

- Let φ be an LTL formula built over atomic propositions from AP using the connectives $\neg$, $\vee$, X, and U. The Fischer-Ladner (FL) closure $cl(\varphi)$ of φ is defined inductively on the structure of φ (assuming that $\neg\neg\varphi \equiv \varphi$):
  - $cl(p) = \{p, \neg p\}$ for $p \in AP$,
  - $cl(\neg\varphi) = cl(\varphi) \cup \{\neg\varphi\}$,
  - $cl(\varphi_1 \vee \varphi_2) = cl(\varphi_1) \cup cl(\varphi_2) \cup \{\varphi_1 \vee \varphi_2, \neg(\varphi_1 \vee \varphi_2)\}$,
  - $cl(X\,\varphi) = cl(\varphi) \cup \{X\,\varphi, \neg X\,\varphi\}$,
  - $cl(\varphi_1\, U\, \varphi_2) = cl(\varphi_1) \cup cl(\varphi_2) \cup \{\varphi_1\, U\, \varphi_2, \neg(\varphi_1\, U\, \varphi_2)\}$,

The FL Closure of a Formula

- Let φ be an LTL formula built over atomic propositions from AP using the connectives $\neg$, $\vee$, X, and U. The Fischer-Ladner (FL) closure $cl(\varphi)$ of φ is defined inductively on the structure of φ (assuming that $\neg\neg\varphi \equiv \varphi$):
  - $cl(p) = \{p, \neg p\}$ for $p \in AP$,
  - $cl(\neg\varphi) = cl(\varphi) \cup \{\neg\varphi\}$,
  - $cl(\varphi_1 \vee \varphi_2) = cl(\varphi_1) \cup cl(\varphi_2) \cup \{\varphi_1 \vee \varphi_2, \neg(\varphi_1 \vee \varphi_2)\}$,
  - $cl(X\,\varphi) = cl(\varphi) \cup \{X\,\varphi, \neg X\,\varphi\}$,
  - $cl(\varphi_1\, U\, \varphi_2) = cl(\varphi_1) \cup cl(\varphi_2) \cup \{\varphi_1\, U\, \varphi_2, \neg(\varphi_1\, U\, \varphi_2)\}$,

- Example: $cl((pUq) \vee (\neg pUq))$

The FL Closure of a Formula

- Let φ be an LTL formula built over atomic propositions from AP using the connectives $\neg$, $\vee$, X, and U. The Fischer-Ladner (FL) closure $cl(\varphi)$ of φ is defined inductively on the structure of φ (assuming that $\neg\neg\varphi \equiv \varphi$):
  - $cl(p) = \{p, \neg p\}$ for $p \in AP$,
  - $cl(\neg\varphi) = cl(\varphi) \cup \{\neg\varphi\}$,
  - $cl(\varphi_1 \vee \varphi_2) = cl(\varphi_1) \cup cl(\varphi_2) \cup \{\varphi_1 \vee \varphi_2, \neg(\varphi_1 \vee \varphi_2)\}$,
  - $cl(X\,\varphi) = cl(\varphi) \cup \{X\,\varphi, \neg X\,\varphi\}$,
  - $cl(\varphi_1\, U\, \varphi_2) = cl(\varphi_1) \cup cl(\varphi_2) \cup \{\varphi_1\, U\, \varphi_2, \neg(\varphi_1\, U\, \varphi_2)\}$,

- Example: $cl((pUq) \vee (\neg pUq))$

$$cl((pUq) \vee (\neg pUq)) = \left\{ \begin{array}{rl} (pUq) \vee (\neg pUq), & \neg((pUq) \vee (\neg pUq)), \\ (pUq), & \neg(pUq), \\ (\neg pUq), & \neg(\neg pUq), \\ p, \neg p, & q, \neg q \end{array} \right\}$$

Consistent Sets of Formulae

- We want to restrict the construction to sets of formulae that do not contain contradictory formulae (i.e., formulae that can never hold together).
- Given an LTL formula φ with the chosen basic connectives, we call a set $q \subseteq cl(\varphi)$ consistent iff the following conditions hold:

1. $\forall \psi \in cl(\varphi).\ \psi \in q \Longleftrightarrow \neg\psi \notin q$.
2. $\forall (\psi_1 \vee \psi_2) \in cl(\varphi).\ (\psi_1 \vee \psi_2) \in q \Longleftrightarrow \psi_1 \in q \vee \psi_2 \in q$.
3. $\forall (\psi_1\ U\ \psi_2) \in cl(\varphi).\ \psi_2 \in q \Longrightarrow (\psi_1\ U\ \psi_2) \in q$.
4. $\forall (\psi_1\ U\ \psi_2) \in cl(\varphi).\ (\psi_1\ U\ \psi_2) \in q \wedge \psi_2 \notin q \Longrightarrow \psi_1 \in q$.

Constructing $\mathcal{B}_\varphi$

Given an LTL formula φ built over atomic propositions from AP using the basic connectives $\neg$, $\vee$, X, U, the generalised BA $\mathcal{B}_\varphi = (Q, 2^{AP}, \delta, Q_0, \mathcal{F})$ is defined as follows:

- $Q = \{q_0\} \cup \{q \subseteq cl(\varphi) \mid q \text{ is consistent}\}$, $q_0 \notin 2^{cl(\varphi)}$, and $Q_0 = \{q_0\}$.

Constructing $\mathcal{B}_\varphi$

Given an LTL formula φ built over atomic propositions from AP using the basic connectives $\neg$, $\vee$, X, U, the generalised BA $\mathcal{B}_\varphi = (Q, 2^{AP}, \delta, Q_0, \mathcal{F})$ is defined as follows:

- $Q = \{q_0\} \cup \{q \subseteq cl(\varphi) \mid q \text{ is consistent}\}$, $q_0 \notin 2^{cl(\varphi)}$, and $Q_0 = \{q_0\}$.
- $\delta \subseteq Q \times 2^{AP} \times Q$ satisfies the following conditions:
  - $(q_0, a, q) \in \delta$ iff
    1. $q \neq q_0$,
    2. $\varphi \in q$, and
    3. $a = q \cap AP$.

Constructing $\mathcal{B}_\varphi$

Given an LTL formula φ built over atomic propositions from AP using the basic connectives $\neg$, $\vee$, X, U, the generalised BA $\mathcal{B}_\varphi = (Q, 2^{AP}, \delta, Q_0, \mathcal{F})$ is defined as follows:

- $Q = \{q_0\} \cup \{q \subseteq cl(\varphi) \mid q \text{ is consistent}\}$, $q_0 \notin 2^{cl(\varphi)}$, and $Q_0 = \{q_0\}$.
- $\delta \subseteq Q \times 2^{AP} \times Q$ satisfies the following conditions:
  - $(q_0, a, q) \in \delta$ iff
    1. $q \neq q_0$,
    2. $\varphi \in q$, and
    3. $a = q \cap AP$.
  - $(q_1, a, q_2) \in \delta$ for $q_1 \neq q_0$ iff
    1. $q_2 \neq q_0$,
    2. $a = q_2 \cap AP$,
    3. $\forall (X\,\psi) \in cl(\varphi).\ (X\,\psi) \in q_1 \iff \psi \in q_2$.
    4. $\forall (\psi_1\ U\ \psi_2) \in cl(\varphi).\ (\psi_1\ U\ \psi_2) \in q_1\ \wedge\ \psi_2 \notin q_1 \implies (\psi_1\ U\ \psi_2) \in q_2$.
    5. $\forall (\psi_1\ U\ \psi_2) \in cl(\varphi).\ (\psi_1\ U\ \psi_2) \notin q_1\ \wedge\ \psi_1 \in q_1 \implies (\psi_1\ U\ \psi_2) \notin q_2$.

Constructing $\mathcal{B}_\varphi$

Given an LTL formula φ built over atomic propositions from AP using the basic connectives $\neg$, $\vee$, X, U, the generalised BA $\mathcal{B}_\varphi = (Q, 2^{AP}, \delta, Q_0, \mathcal{F})$ is defined as follows:

- $Q = \{q_0\} \cup \{q \subseteq cl(\varphi) \mid q \text{ is consistent}\}$, $q_0 \notin 2^{cl(\varphi)}$, and $Q_0 = \{q_0\}$.
- $\delta \subseteq Q \times 2^{AP} \times Q$ satisfies the following conditions:
  - $(q_0, a, q) \in \delta$ iff
    1. $q \neq q_0$,
    2. $\varphi \in q$, and
    3. $a = q \cap AP$.
  - $(q_1, a, q_2) \in \delta$ for $q_1 \neq q_0$ iff
    1. $q_2 \neq q_0$,
    2. $a = q_2 \cap AP$,
    3. $\forall (X\ \psi) \in cl(\varphi).\ (X\ \psi) \in q_1 \iff \psi \in q_2$.
    4. $\forall (\psi_1\ U\ \psi_2) \in cl(\varphi).\ (\psi_1\ U\ \psi_2) \in q_1\ \wedge\ \psi_2 \notin q_1 \implies (\psi_1\ U\ \psi_2) \in q_2$.
    5. $\forall (\psi_1\ U\ \psi_2) \in cl(\varphi).\ (\psi_1\ U\ \psi_2) \notin q_1\ \wedge\ \psi_1 \in q_1 \implies (\psi_1\ U\ \psi_2) \notin q_2$.
- $\mathcal{F} = \{\{q \in Q \setminus \{q_0\} \mid \psi_2 \in q\ \vee\ (\psi_1\ U\ \psi_2) \notin q\} \mid (\psi_1\ U\ \psi_2) \in cl(\varphi)\}$.
  - Guarantees that each until (once encountered) will reach its end (i.e., a state where its right operand holds).

Constructing $\mathcal{B}_\varphi$

Given an LTL formula φ built over atomic propositions from AP using the basic connectives $\neg$, $\vee$, X, U, the generalised BA $\mathcal{B}_\varphi = (Q, 2^{AP}, \delta, Q_0, \mathcal{F})$ is defined as follows:

- $Q = \{q_0\} \cup \{q \subseteq cl(\varphi) \mid q \text{ is consistent}\}$, $q_0 \notin 2^{cl(\varphi)}$, and $Q_0 = \{q_0\}$.
- $\delta \subseteq Q \times 2^{AP} \times Q$ satisfies the following conditions:
  - $(q_0, a, q) \in \delta$ iff
    1. $q \neq q_0$,
    2. $\varphi \in q$, and
    3. $a = q \cap AP$.
  - $(q_1, a, q_2) \in \delta$ for $q_1 \neq q_0$ iff
    1. $q_2 \neq q_0$,
    2. $a = q_2 \cap AP$,
    3. $\forall (X\ \psi) \in cl(\varphi).\ (X\ \psi) \in q_1 \iff \psi \in q_2$.
    4. $\forall (\psi_1\ U\ \psi_2) \in cl(\varphi).\ (\psi_1\ U\ \psi_2) \in q_1 \ \wedge\ \psi_2 \notin q_1 \implies (\psi_1\ U\ \psi_2) \in q_2$.
    5. $\forall (\psi_1\ U\ \psi_2) \in cl(\varphi).\ (\psi_1\ U\ \psi_2) \notin q_1 \ \wedge\ \psi_1 \in q_1 \implies (\psi_1\ U\ \psi_2) \notin q_2$.
- $\mathcal{F} = \{\{q \in Q \setminus \{q_0\} \mid \psi_2 \in q \ \vee\ (\psi_1\ U\ \psi_2) \notin q\} \mid (\psi_1\ U\ \psi_2) \in cl(\varphi)\}$.
  - Guarantees that each until (once encountered) will reach its end (i.e., a state where its right operand holds).

We have that $L(\mathcal{B}_\varphi) = \{\ell(s_0)\ell(s_1)\ell(s_2)\ldots \mid$ there is a KS $M = (S, S_0, R, \ell)$ over AP such that $s_0 \in S_0$, $s_0 s_1 s_2 \ldots \in \Pi(M, s_0)$, and $M, s_0 s_1 s_2 \ldots \models \varphi\}$.

An Example of Translating from LTL to BA

- Consider $\varphi = p\ U\ q$:

An Example of Translating from LTL to BA

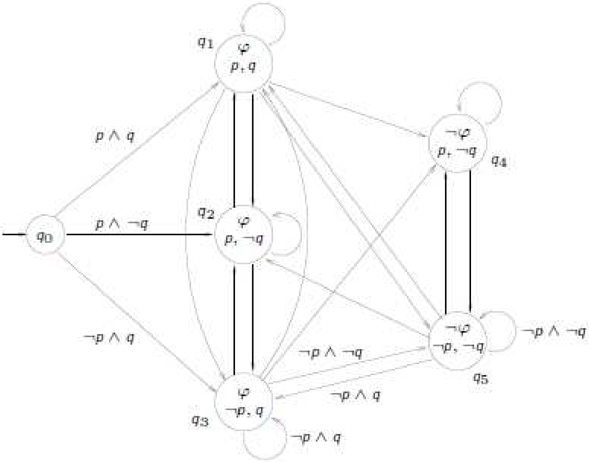
- Consider $\varphi = p\ U\ q$:
  - $cl(\varphi) = \{p, \neg p, q, \neg q, \varphi, \neg\varphi\}$.

An Example of Translating from LTL to BA

- Consider $\varphi = p\ U\ q$:
  - $cl(\varphi) = \{p, \neg p, q, \neg q, \varphi, \neg\varphi\}$.
  - Consistent subsets of $cl(\varphi)$:
    - $q_1 = \{\varphi, p, q\}$,
    - $q_2 = \{\varphi, p, \neg q\}$,
    - $q_3 = \{\varphi, \neg p, q\}$,
    - $q_4 = \{\neg\varphi, p, \neg q\}$,
    - $q_5 = \{\neg\varphi, \neg p, \neg q\}$,

An Example of Translating from LTL to BA

- Consider $\varphi = p\ U\ q$:
  - $cl(\varphi) = \{p, \neg p, q, \neg q, \varphi, \neg\varphi\}$.
  - Consistent subsets of $cl(\varphi)$:
    - $q_1 = \{\varphi, p, q\}$,
    - $q_2 = \{\varphi, p, \neg q\}$,
    - $q_3 = \{\varphi, \neg p, q\}$,
    - $q_4 = \{\neg\varphi, p, \neg q\}$,
    - $q_5 = \{\neg\varphi, \neg p, \neg q\}$,
  - $\mathcal{B}_\varphi$ is shown on the right (not all labels are shown)
  - $\mathcal{F} = \{\{q_1, q_3, q_4, q_5\}\}$.



The Top Level of the LTL MC Algorithm

A Naive LTL MC Algorithm

- A naïve procedure:

  1. generate the KS M for the given system to be verified and the atomic observations AP of interest,
  2. translate M to the BA $\mathcal{B}_M$,
  3. negate the given LTL formula φ to be checked and translate the negation into the BA $\mathcal{B}_{\neg\varphi}$,
  4. construct the product BA $\mathcal{B}_M \times \mathcal{B}_{\neg\varphi}$ representing the language $L(\mathcal{B}_M) \cap L(\mathcal{B}_{\neg\varphi})$,
  5. check language emptiness of $\mathcal{B}_M \times \mathcal{B}_{\neg\varphi}$:
     - if $L(\mathcal{B}_M \times \mathcal{B}_{\neg\varphi})$ is empty, φ holds for the given system,
     - otherwise return a path corresponding to some element from the intersection as a counterexample to the property being checked.

On-the-Fly LTL MC Algorithm

- Differences of on-the-fly model checking from the naïve procedure:
  - Do not generate the KS M and the BA $\mathcal{B}_M$ first, only then constructing the product with the negated property BA, followed by checking its emptiness.

On-the-Fly LTL MC Algorithm

- Differences of on-the-fly model checking from the naïve procedure:
  - Do not generate the KS M and the BA $\mathcal{B}_M$ first, only then constructing the product with the negated property BA, followed by checking its emptiness.
  - Instead, construct $\mathcal{B}_{\neg\varphi}$ and use it to control the construction of $\mathcal{B}_M$ and the product $\mathcal{B}_M \times \mathcal{B}_{\neg\varphi}$ while continuously checking for accepting loops:

On-the-Fly LTL MC Algorithm

- Differences of on-the-fly model checking from the naïve procedure:
  - Do not generate the KS M and the BA $\mathcal{B}_M$ first, only then constructing the product with the negated property BA, followed by checking its emptiness.
  - Instead, construct $\mathcal{B}_{\neg\varphi}$ and use it to control the construction of $\mathcal{B}_M$ and the product $\mathcal{B}_M \times \mathcal{B}_{\neg\varphi}$ while continuously checking for accepting loops:
    - if an accepting loop is detected, immediately stop and print out a counterexample without generating further states (faulty systems tend to have many strange states due to not obeying the intended invariants),

On-the-Fly LTL MC Algorithm

- Differences of on-the-fly model checking from the naïve procedure:
  - Do not generate the KS M and the BA $\mathcal{B}_M$ first, only then constructing the product with the negated property BA, followed by checking its emptiness.
  - Instead, construct $\mathcal{B}_{\neg\varphi}$ and use it to control the construction of $\mathcal{B}_M$ and the product $\mathcal{B}_M \times \mathcal{B}_{\neg\varphi}$ while continuously checking for accepting loops:
    - if an accepting loop is detected, immediately stop and print out a counterexample without generating further states (faulty systems tend to have many strange states due to not obeying the intended invariants),
    - when some transition from the state of $\mathcal{B}_M$ that is currently being explored cannot be composed with the currently executable transitions of $\mathcal{B}_{\neg\varphi}$, do not follow it (no counterexample can be reached via the transition—hence, the sub-state space reachable (exclusively) via it needs not be explored).

On-the-Fly LTL MC Algorithm

- Differences of on-the-fly model checking from the naïve procedure:
  - Do not generate the KS M and the BA $\mathcal{B}_M$ first, only then constructing the product with the negated property BA, followed by checking its emptiness.
  - Instead, construct $\mathcal{B}_{\neg\varphi}$ and use it to control the construction of $\mathcal{B}_M$ and the product $\mathcal{B}_M \times \mathcal{B}_{\neg\varphi}$ while continuously checking for accepting loops:
    - if an accepting loop is detected, immediately stop and print out a counterexample without generating further states (faulty systems tend to have many strange states due to not obeying the intended invariants),
    - when some transition from the state of $\mathcal{B}_M$ that is currently being explored cannot be composed with the currently executable transitions of $\mathcal{B}_{\neg\varphi}$, do not follow it (no counterexample can be reached via the transition—hence, the sub-state space reachable (exclusively) via it needs not be explored).
  - Combine the on-the-fly generation of states of M with suitable state space reduction techniques, e.g.,
    - partial order reduction (exploring only some interleavings of the concurrent processes running in the verified system) or
    - symmetry reduction (do not explore states that are indistinguishable from some already generated states wrt the property being checked),
    - bit-state hashing (do not distinguish states with the same hash), . . .