Formal Methods in Software Development

Automata Theory and Model Checking

Ivano Salvo

Computer Science Department

SAPIENZA
UNIVERSITÀ DI ROMA

Lesson **4**, October 26th, 2020

Lesson 4a:

Regular Properties and Finite Automata

Non-Determ. Finite Automata

A **nondeterministic finite automata** (**NFA**) $\mathcal{A}$ is a tuple $(\Sigma, Q, \delta, Q_0, F)$ where:

- Σ is the finite input **alphabet**
- Q is the finite set of **states**
- $\delta \subseteq Q \times \Sigma \times Q$ is the **transition relation**
- $Q_0 \subseteq Q$ is the set of **initial states**
- $F \subseteq Q$ is the set of **accepting states**

Let w be a word in Σ^* of length $|w| = n$.

A **run** over w is a finite sequence of states $q_0 q_1 \ldots q_n$ such that $q_0 \in Q_0$ is an initial state and $(q_i, w_{i+1}, q_{i+1}) \in \delta$ for all $0 \leq i < n$.

A run is **accepting** if $q_n \in F$.

The automaton $\mathcal{A}$ **accepts** w if **there exists** an accepting run over w

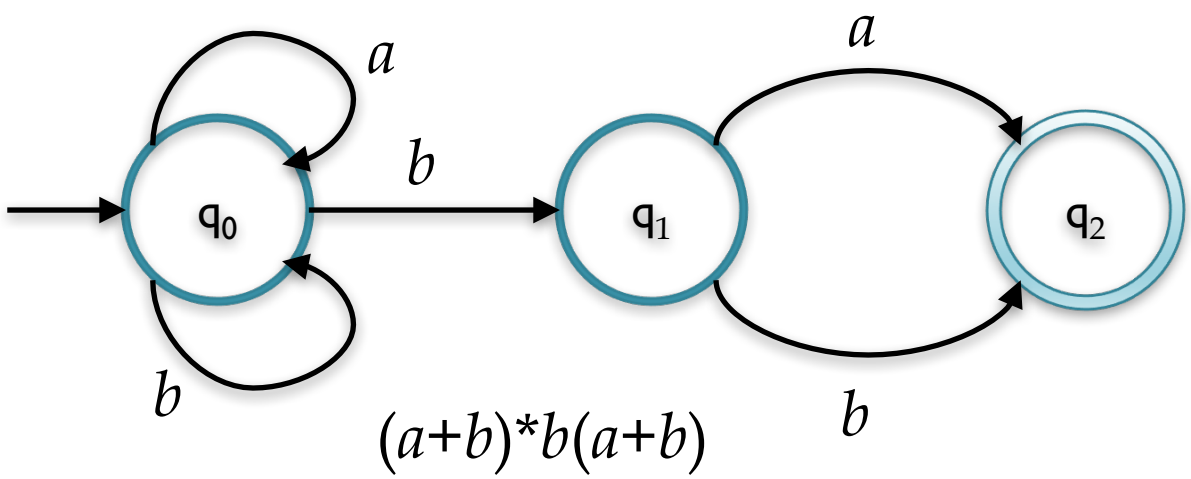
The **language** $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^*$ consists of all words accepted by $\mathcal{A}$.

(Non-Det.) Finite Automata

An automaton $\mathcal{A}$ is **deterministic** if δ is a **function** [for all states s and all symbols a there exists a **unique next state** $\delta(s, a)$] and there exists a **unique initial state** ($|\delta(s, a)| \leq 1$ and $|Q_0| = 1$)).

For each non-deterministic automaton $\mathcal{A}$ there exists an automaton $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$. However, the size of $\mathcal{A}'$ **can be exponential** w.r.t. the size of $\mathcal{A}$.

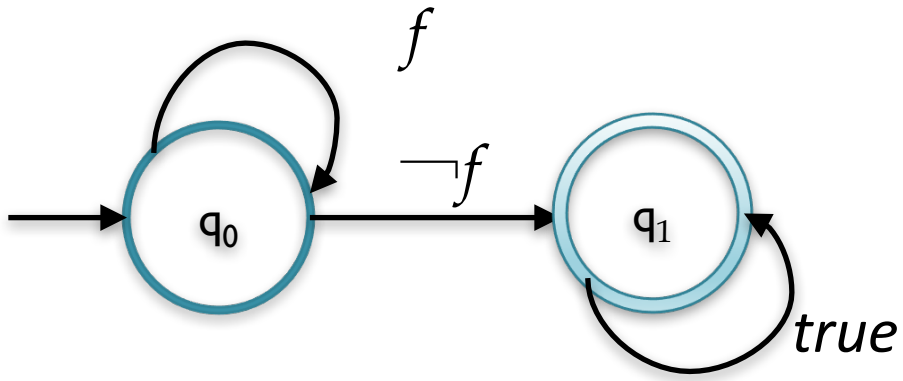
The class of languages accepted by finite automata is the class of **regular languages**, that can be characterized by **regular expressions**.



Regular Safety Properties (1)

Definition. A safety property P is **regular** if its **set of bad prefixes is a regular** language over 2^{AP}.

Example: Every **invariant is a regular property**. Let f be the invariant property. The language of bad prefixes is $f^*(\neg f)\text{true}^*$ (**we use a propositional formulas to identify subsets of** AP).



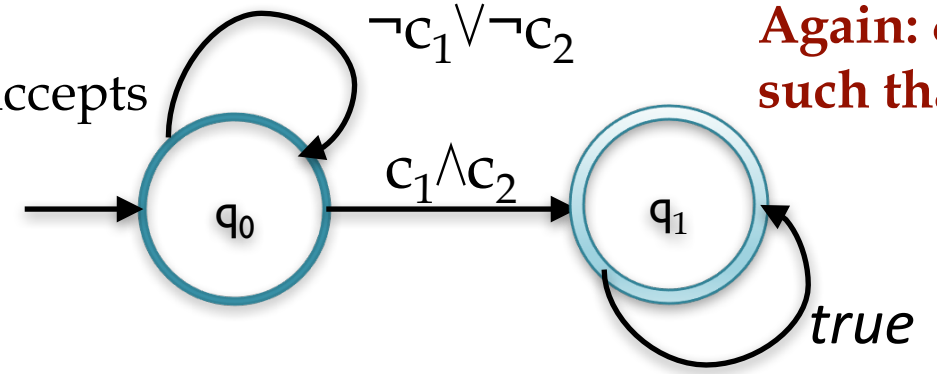
This automaton accepts words that **violates** the invariant f.

Remark: Here we assume f be a shorthand for $L(s) \models f$

Regular Safety Properties (2)

Example. The mutual exclusion property can be easily modeled by a NFA as follows:

this automata accepts **bad prefixes**



Again: φ means $L(s)$ such that $L(s) \models \varphi$

If we remove the transition labeled *true*, this automaton accepts **minimal** bad prefixes of the mutual exclusion property (words that end at the first violation).

This is not incidental!

Theorem. A safety property P is regular iff the set of **minimal** bad prefixes for P is regular.

A non-regular safety property

Example: Let us consider again the Beverage vending machine and the property: "***The number of inserted coins is always at least the number of dispensed drinks***"

The language of **minimal bad prefixes** over the alphabet $\sum = \{pay, drink\}$ is (first violation of the property):

$$\{ w \cdot drink \mid \#_{drink}(w) = \#_{pay}(w) \}$$

which **is not regular**, but context-free (usually **counting properties are never regular**, because they require memory to count occurrences and this is not possible with a NFA).

Verifying Regular Safety Prop.

Idea: Run **in parallel** the system model $\mathcal{M}$ and the automaton for $\neg f$. $\mathcal{M} \models f$ **iff** $\text{traces}_{\text{fin}}(\mathcal{M}) \cap \text{badPrefixes}(P_f) = \varnothing$
iff $\text{traces}_{\text{fin}}(\mathcal{M}) \cap \mathcal{L}(\neg f) = \varnothing$

Ingredients:

- build an automata for the **intersection of two languages**
- checking **language emptiness**

Definition: [**Product of a Transition System $\mathcal{M}$ and a NFA**]
Let $\mathcal{M} = (S, A, I, \rightarrow, AP, L)$ and $\mathcal{A} = (\Sigma, Q, \delta, Q_0, F)$, such that $\Sigma = 2^{AP}$ and $Q_0 \cap F = \varnothing$. Then $\mathcal{M} \otimes \mathcal{A} = (S', A, I', \rightarrow', AP', L')$ where:

- $S' = S \times Q$
- $(s, q) \rightarrow'_a (s', q')$ whenever $s \rightarrow_a s'$ and $\delta(q, L(s'), q')$
- $I' = \{(s, q) \mid s \in I \wedge \exists q_0 \in Q_0 . (q_0, L(s), q)\}$
- $AP' = Q$
- $L' : S \times Q \rightarrow 2^Q$ is given by $L'(s, q) = \{q\}$

This construction works also for **Kripke structures**.

Verifying Regular Safety Prop.

Let us define: $\neg F = P_{\text{inv}} = \bigwedge_{q_i \in F} \neg q_i$

Theorem. Let $\mathcal{A}$ be a NFA such that $\mathcal{L}(\mathcal{A}) = \mathsf{badPrefixes}(P)$ of some safety property P and let $\mathcal{M}$ be a transition system. Then the following statements are equivalent:

- $\mathcal{M} \vDash P$
- $\mathsf{traces}_{\text{fin}}(\mathcal{M}) \cap \mathcal{L}(\mathcal{A}) = \varnothing$
- $\mathcal{M} \otimes \mathcal{A} \vDash P_{\text{inv}}$

Checking a **regular safety property** has been reduced to a **invariant checking**, that in turn it can be solved by a **reachability**.

Equivalently, **emptiness** of a regular language is a **reachability** problem (check whether **accepting states are reachable** from **initial states**)

The accepted words are **counterexamples**

Lesson 4b:

Finite Automata over Infinite Words

ω-regular Languages

ω-regular languages are subsets of infinite words Σ^ω over a finite alphabet Σ generated by ω-regular expressions.

Example: $(ab)^\omega = ababababab\ldots$ Observe that $(ab)^*$ is **an infinite set of finite** words, but $(ab)^\omega$ is **one infinite word**.

The operator $^\omega$ lifts to languages. $\mathcal{L}^\omega = \{ w_1 w_2 w_3 \ldots \mid w_i \in \mathcal{L} \}$

Definition: An **ω-regular expression** over Σ has the form:

$$G = E_1 \cdot F_1^\omega + \ldots + E_n \cdot F_n^\omega$$

where $n \geq 1$ and $E_1, F_1, \ldots, E_n\ F_n$ are regular expressions.

$$\mathcal{L}(G) = \mathcal{L}(E_1) \cdot \mathcal{L}(F_1)^\omega \cup \ldots \cup \mathcal{L}(E_n) \cdot \mathcal{L}(F_n)^\omega$$

$\mathcal{L}$ is ω-regular if $\mathcal{L} = \mathcal{L}(G)$ for some ω-regular expression G.

ω-regular languages are closed under **union**, **intersection** and **complementation**.

Examples: $(a+b)^* \cdot b^\omega$ is the language of words with finitely many a's. $(b^* a)^\omega$ is the language of words with infinitely many a's.

(Non-Det.) Büchi Automata

A **non-determistic Büchi automata** $\mathcal{A}$ is a 5-tuple $(\Sigma, Q, \delta, Q_0, F)$ where:

- Σ is the finite input **alphabet**
- Q is the finite set of **states**
- $\delta \subseteq Q \times \Sigma \times Q$ is the **transition relation**
- $Q_0 \subseteq Q$ is the set of **initial states**
- $F \subseteq Q$ is the set of **accepting states**

This definition is exactly the same of NFA, but the semantics of accepted words change!

Let w be an **infinite** word in Σ^ω. A **run** ρ over w is an **infinite sequence** of states $q_0 q_1 \ldots q_n \ldots$ such that $q_0 \in Q_0$ is an initial state and $(q_i, w_{i+1}, q_{i+1}) \in \delta$ for all $i \in \mathbb{N}$. **inf(ρ)** is the set of states that occur infinitely often in ρ.

A run is **accepting** if $q_j \in F$ for **infinitely many** i.

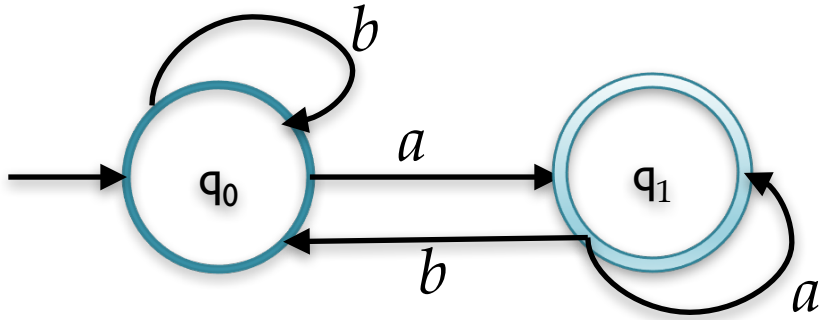
The automaton $\mathcal{A}$ **accepts** w if **there exists** an accepting run ρ over w such that $\inf(\rho) \cap F \neq \varnothing$

The **language** $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^\omega$ consists of all the words accepted by $\mathcal{A}$.

Büchi Autom. and ω-regular lang.

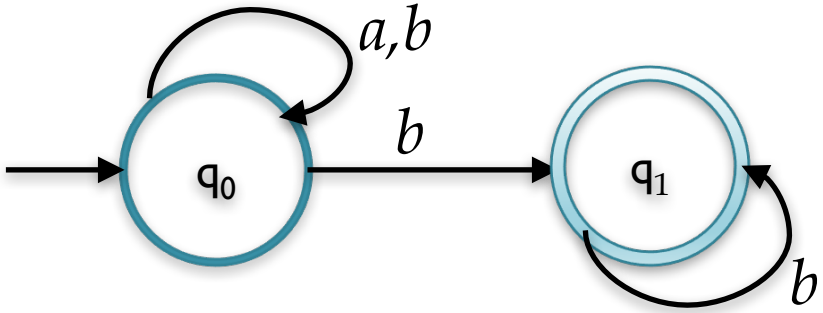
Theorem. The class of languages accepted by NBA's is exactly the class of ω-regular languages.

Examples: Infinitely many a's: $(b^*a)^\omega$



deterministic

Finitely many a's: $(a+b)^*\cdot b^\omega$



non-deterministic
b in q_0: two trans.

The automaton "**knows**" when the sequence of finitely many a's stops

NBA for $\mathcal{L}_1 + \mathcal{L}_2$ with $\mathcal{L}_1$, $\mathcal{L}_2$ ω-reg

Theorem. If $\mathcal{L}_1$ and $\mathcal{L}_2$ are ω-regular, then **$\mathcal{L}_1 \cup \mathcal{L}_2$ is ω-regular**.

Proof: Given an automaton $\mathcal{A}_1 = (\Sigma, Q_1, \delta_1, I_1, F_1)$ accepting $\mathcal{L}_1$ and an automaton $\mathcal{A}_2 = (\Sigma, Q_2, \delta_2, I_2, F_2)$ accepting $\mathcal{L}_2$ we build the automata

$$\mathcal{A} = \mathcal{A}_1 + \mathcal{A}_2 = (\Sigma, Q_1 \cup Q_2, \delta, I_1 \cup I_2, F_1 \cup F_2)$$

where $(q, a, q') \in \delta$ if $(q, a, q') \in \delta_1$ **or** $(q, a, q') \in \delta_2$.

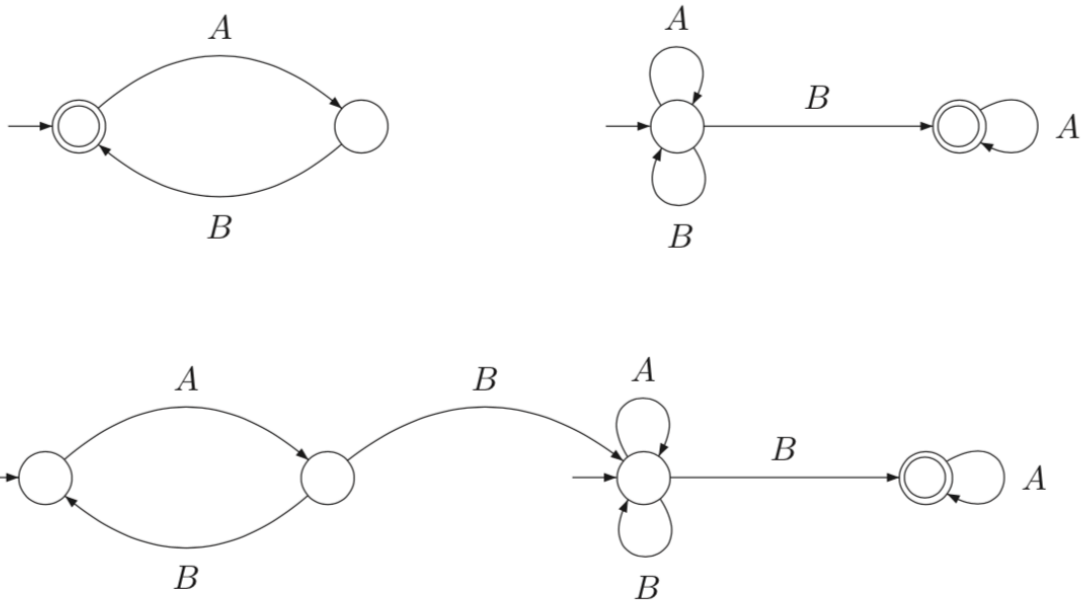
It is easy to see that $\mathcal{A}_1 + \mathcal{A}_{2\,a}$ accepts $\mathcal{L}_1 \cup \mathcal{L}_2$. (**Exercise** ☺). □

NBA for $\mathcal{L}_1 \cdot \mathcal{L}_2$, $\mathcal{L}_1$ reg. $\mathcal{L}_2$ ω-reg

Idea: Taking the NFA $\mathcal{A}_1$ accepting $\mathcal{L}_1$, the basic trick is **adding a transition** to an **initial state** of the NBA $\mathcal{A}_2$ accepting $\mathcal{L}_2$, whenever there is a transition **to a final state of $\mathcal{A}_1$**.

Final states are those of the NBA $\mathcal{A}_2$. Observe that possible infinite runs inside $\mathcal{A}_1$ are not accepting.

Example: Let us consider $\mathcal{L}_1=(ab)^*$, $\mathcal{L}_2=(a+b)^*ba^\omega$ and $\mathcal{L}_1 \cdot \mathcal{L}_2=(ab)^*(a+b)^*ba^\omega$

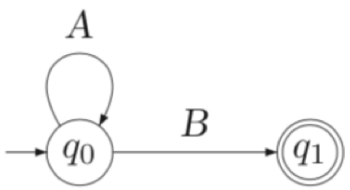


NBA accepting $\mathcal{L}^\omega$, $\mathcal{L}$ regular

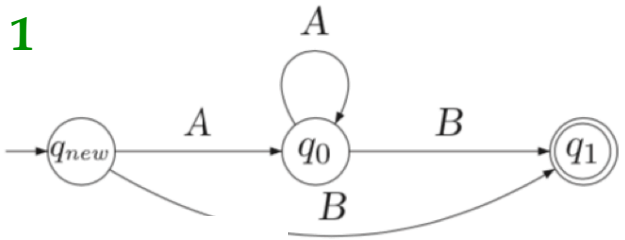
Idea: Insert a **new initial** (and **accepting** state) q_{new} and:

1. put a transition from q_{new} to any successor of initial states;
2. 2. put a transition to q_{new} from any accepting state.
   [q_{new} is not necessary if initial states are without ingoing transitions and they are not accepting]

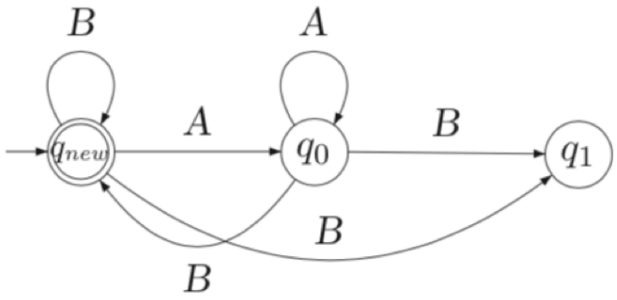
Example: $\mathcal{L}=a^*b$.



1



2



Non blocking NBA

As usual, we want automata with a total transition relation (**non-blocking**). If a computation **gets stuck**, it's not a problem for theory, it is just a **non-accepting computation** (the same for non-deterministic NFAs).

Proposition: For each NBA $\mathcal{A}$ there exists a non-blocking equivalnet NBA $\mathcal{A}'$ equivalent to $\mathcal{A}$.

Proof: Just add a sink (or trap) state q_{trap} and transitions to whenever a transition is not defined in some state. □

More or less, the same trick works for Kripke structures and NFAs.

Remark: $\mathcal{A}$ **equivalent** to $\mathcal{A}'$ means that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$

The need for non-determinism

Deterministic Büchi automata are **less expressive**.

Theorem. There is **no deterministic Büchi automata** that accepts the language $(a+b)^*b^\omega$.

Proof: Assume that there exists such automaton. The word b^ω belongs to the language. There exists an accepting state q_1 such that $\delta^*(q_0, b^{n_1}) = q_1$ (δ is a function!).

The word $b^{n_1}ab^\omega$ belongs to the language. There exists an accepting state q_2 such that $\delta^*(q_0, b^{n_1}ab^{n_2}) = q_2$.

The word $b^{n_1}ab^{n_2}ab^\omega$ belongs to the language. There exists an accepting state q_2 such that $\delta^*(q_0, b^{n_1}ab^{n_2}ab^{n_3}) = q_3$ **and so on**.

But **there are finitely many states**. Therefore there must be that some $q_i = q_j$ and hence $\delta^*(q_0, b^{n_1}ab^{n_2} \dots ab^{n_i}) = \delta^*(q_0, b^{n_1}ab^{n_2} \dots ab^{n_j})$, but this implies that there is an accepting run for the word $b^{n_1}ab^{n_2} \dots ab^{n_i}(ab^{n_{i+1}} \dots ab^{n_j})^\omega$ that contains infinitely many a's. Contradiction. □

The need for non-determinism

Properties of the form **"eventually forever"** has exactly the shape of the ω-regular language $(a+b)^* b^\omega$.

Definition: A **persistence property** is a linear time property $P \subseteq 2^{AP}$ such that for some propositional formula φ:

$$P = \{A_0 A_1 A_2 \ldots \in (2^{AP})^\omega \mid \exists i \geq 0 \, \forall j \geq i. \, A_j \vDash \varphi \}$$

A persistence property can be modeled in LTL as **F G** φ.

Alternatively, they can be formalised as "**$\neg\varphi$ holds finitely many times**".

Remark: $\exists i \geq 0 \, \forall j \geq i$ is sometimes written $\forall^\infty$ and can be read **"almost always"**

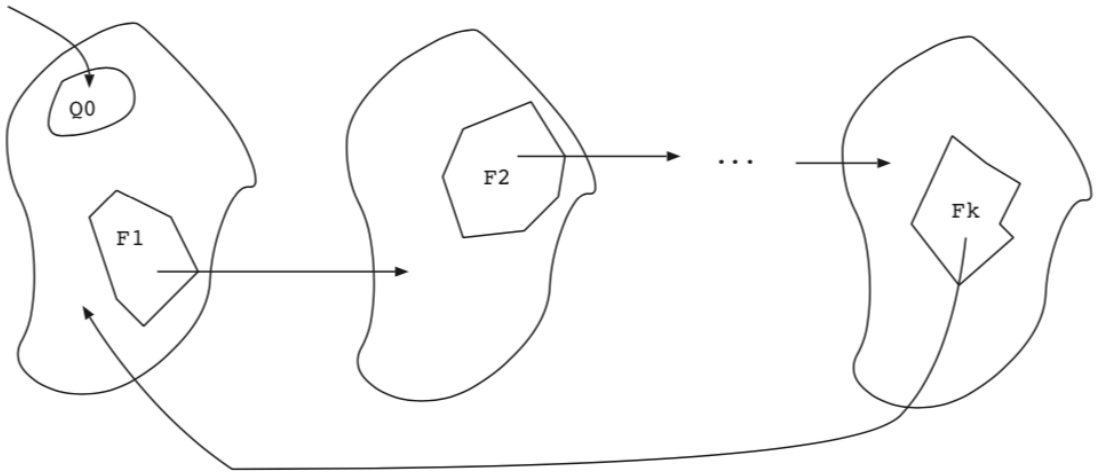
Generalised Büchi Automata

A **generelised Büchi automata** $\mathcal{A}$ is a 5-tuple $(\Sigma, Q, \delta, Q_0, \mathcal{F})$ where Σ, Q, δ, Q_0 are as for NBA, and $\mathcal{F} = \{F_1, \ldots, F_n\}$ is a possibly empty subset of 2^Q. $F_1, \ldots, F_n$ are called *accepting sets*.

The automaton $\mathcal{A}$ **accepts** w if **there exists** an accepting run ρ over w such that for all sets $F_i \in \mathcal{F}$ we have $\inf(\rho) \cap F_i \neq \varnothing$.

Theorem. For each GNBA $\mathcal{A}$ there exists a NBA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

Proof: make n copy of $\mathcal{A}$ and jump to i+1$^{\text{th}}$ copy whenever we go trough a state in F_i.



Intersection of ω-regular lang.

Theorem. If $\mathcal{L}_1$ and $\mathcal{L}_2$ are ω-regular, then $\mathcal{L}_1 \cap \mathcal{L}_2$ is ω-regular.

Proof: Given an automaton $\mathcal{A}_1 = (\Sigma, Q_1, \delta_1, I_1, F_1)$ accepting $\mathcal{L}_1$ and an automaton $\mathcal{A}_2 = (\Sigma, Q_2, \delta_2, I_2, F_2)$ accepting $\mathcal{L}_2$ we build a **generalised automata** $\mathcal{A} = (\Sigma, Q, \delta, I, \mathcal{F})$ accepting $\mathcal{L}$. We define $\mathcal{A} = \mathcal{A}_1 \otimes \mathcal{A}_1 = (\Sigma, Q_1 \times Q_2, \delta, I_1 \times I_2, \{F_1 \times Q_2, Q_1 \times F_2\})$, where $((q_1, q_2), a, (q'_1, q'_2)) \in \delta$ iff $(q_1, a, q'_1) \in \delta_1$ and $(q_2, a, q'_2) \in \delta_2$. □

We will use this trick in verification, building an **automata for the model, one for specifications** (or better, for **bad behaviours**) and **we will check if their intersection is empty**.

This strategy will also lead to an **alternative algorithm for LTL model checking**. There is an algorithm (based again on atoms) that allow to build an automata from an LTL formula.

Lesson 4c:

Automata Theory and Model Checking

ω-Regular Properties

Definition: A linear time property P over AP is **ω-regular** if P is an ω-regular language over the alphabet 2^{AP}.

Examples:

❖ **Invariants** are ω-regular. If φ is a property over AP defining the invariant, φ^ω is a ω-regular language.

❖ **Regular safety properties** are ω-regular.

$$(2^{AP})^\omega \setminus P_{\text{safe}} = \mathsf{badPrefixes}(P_{\text{safe}}) \cdot (2^{AP})^\omega$$

[Remember that ω-regular are **closed under complementation**]

❖ Many **liveness** properties are typical examples of ω-regular (**not regular**) properties.

Example: $((\neg\text{crit})^*\text{crit})^\omega$ ="a process enters critical section infinitely often"

$((\neg\text{wait})^*\text{wait} \cdot \text{true}^* \cdot \text{crit})^\omega + ((\neg\text{wait})^*\text{wait} \cdot \text{true}^* \cdot \text{crit})^*(\neg\text{wait})^\omega$ = "whenever a process is waiting, it will enter its critical section eventually later" (**starvation freedom**)

Checking ω-regular properties

Similar to regular safety properties. However, here we have **to check language emptiness** for a (generalised) **non deterministic Büchi automata**.

Again, the idea is related to **strongly connected components** of a directed graph.

Definition: [**Product of a Transition System $\mathcal{M}$ and a NBA**] Let $\mathcal{M} = (S, A, I, \rightarrow, AP, L)$ and $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ be a non-blocking NBA, such that $\Sigma = 2^{AP}$. Then $\mathcal{M} \otimes \mathcal{A} = (S', A, I', \rightarrow', AP', L')$ where:

- $S' = S \times Q$
- $(s, q) \rightarrow'_a (s', q')$ whenever $s \rightarrow_a s'$ and $\delta(q, L(s'), q')$
- $I' = \{(s, q) \mid s \in I \wedge \exists q_0 \in Q_0 . (q_0, L(s), q)\}$
- $AP' = Q$
- $L' : S \times Q \rightarrow 2^Q$ is given by $L'(s, q) = \{q\}$

Verifying ω-regular Properties

Let us define: $\neg\varphi = \bigwedge_{q_i \in Q} \neg q_i$ and $P_{\text{persistence}} = \mathbf{F}\,\mathbf{G}\,\neg\varphi$

Theorem. Let $\mathcal{M}$ be a finite transition system and let P an ω-regular property over AP and let $\mathcal{A}$ be a nonblocking NBA such that $\mathcal{L}_\omega(\mathcal{A}) = (2^{AP})^\omega \setminus P$. Then the following are equivalent:

- $\mathcal{M} \models P$
- $\text{traces}(\mathcal{M}) \cap \mathcal{L}_\omega(\mathcal{A}) = \varnothing$
- $\mathcal{M} \otimes \mathcal{A} \models P_{\text{persistence}}(\mathcal{A})$

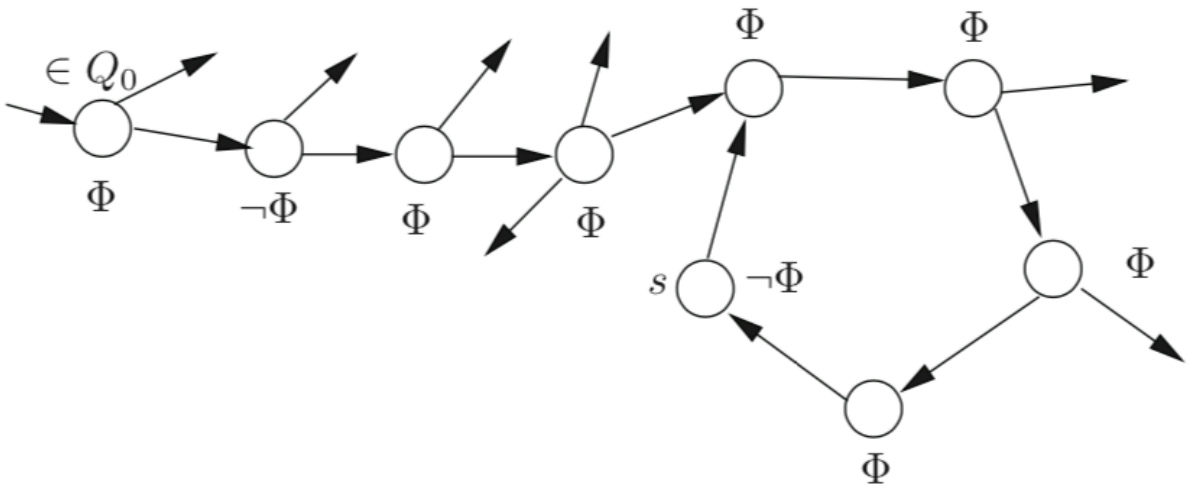
Checking a **ω-regular property** has been reduced to a **checking** a **persistence property**.

Equivalently, **emptiness** of a ω-regular language is a problem of detecting **cycles:** checking whether accepting states belong to a cycle reachable from some initial state.

In this case, **counterexamples** have the form $u \cdot v^\omega$

Counterexamples

In this case, **counterexamples** have the form $u \cdot v^{\omega}$, where for some q in v, $L(q) \models \neg\varphi$.



Checking a persistence property

Once again, a SCC decomposition of the graph $\mathcal{M} \otimes \mathcal{A}$ would solve the problem. $\text{traces}(\mathcal{M}) \cap \mathcal{L}_\omega(\mathcal{A}) = \varnothing$ if and only if **there is a SCC *C* that contains a state not satisfying φ** and ***C* is reachable** from an initial state.

This algorithm is optimal, in the sense that it is **linear** with the size of $\mathcal{M} \otimes \mathcal{A}$.

However, **in practice** cycle checking can be performed more efficiently without decomposing the whole system $\mathcal{M} \otimes \mathcal{A}$ into strongly connected components.

Many model checkers implement a **nested double DFS search**. This approach has several advantages:

- When a counterexample is found, **only a small part** of $\mathcal{M} \otimes \mathcal{A}$ is visited.
- $\mathcal{M}$ is described by a program, and **states can be generated** during the nested DFS (**on-the-fly model checking**).

Double Nested DFS

```
def dfs1(q):
    mark(q);
    forall p ∈ succ(q) do
        if p is not marked then dfs1(p);
    if accept(q) then dfs2(q)

def dfs2(q):
    flag(q);
    forall p ∈ succ(q) do
        if p is in dfs2 stack then return TRUE;
            else if p is not flagged then dfs2(p);
```

dfs1(q) is used to **finds states where property holds** (*accept*(*q*))

dfs2(q) starts when **all successors** of a **final state *q* have been explored**

State **on the stack of *dfs1*** up to *q* **are the finite prefix *u***, whereas states on the **stack of *dfs2* are the cycle *v***

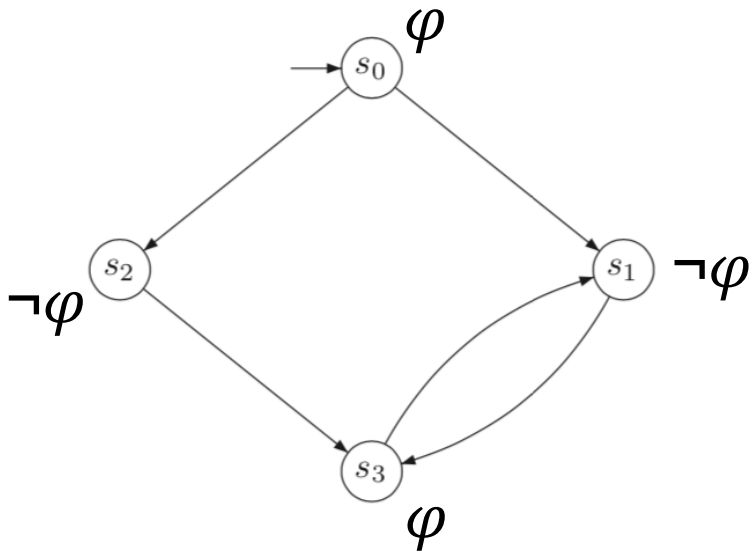
Observe that visited states by *dfs1* and *dfs2* are ***global information***. This is essential to keep **complexity *linear*** and to avoid to visit **several times the same state**.

Running the Double DFS Search

Start ***dfs1*** with s_0. Consider the order of visit $s_0\ s_2\ s_3\ s_1$.

The cycle $s_1 \rightarrow s_3 \rightarrow s_1$ is found when analysing s_1. Here ***dfs2*** starts, because $s_1 \not\models \varphi$ and all its successors have been already analysed. The **counterexample** is $s_0\ s_2\ s_3\ (s_1\ s_3\ s_1)^\omega$

The order is essential. **If we start *dfs2* in s_2,** we fail to find a cycle with a state already onto the stack, but we mark as visited s_3 and s_1 and therefore we later fail to find $(s_1\ s_3\ s_1)^\omega$.



Correctness of Double DFS (1)

Lemma. Let q be a node that does not appear in any cycle. Then a DFS backtrack from q after all nodes reachable from q have been visited.

Theorem. The Double Nested DFS search returns a counterexample if and only if $\mathsf{traces}(\mathcal{M}) \cap \mathcal{L}_\omega(\mathcal{A}) \neq \varnothing$.

Proof: It is almost trivial to show that if the double DFS returns TRUE, a cycle is found (and hence an **accepted word**).

It is less obvious to show that if a cycle exists, the double DFS finds it. Or equivalently, if it returns FALSE, no cycle exists.

Let us suppose that there exists a cycle from q to a state on the stack of ***dfs1*** that goes trough a state r **already flagged** by ***dfs2***. Let q and r the first states for which this happens and let q' be the root of ***dfs2*** that flagged r (***dfs2***(q') **started before** ***dfs2***(q)).

There are two cases.

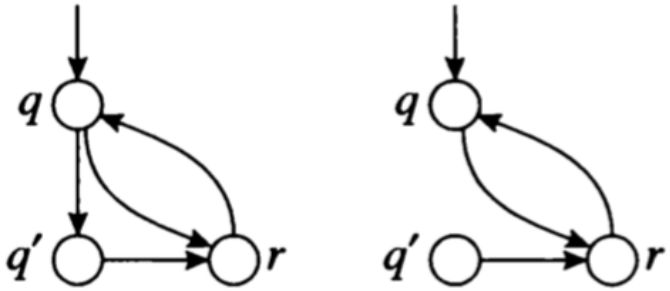
Correctness of Double DFS (2)

If q' is reachable from q, then there exists a cycle that would have been found examining q': $q' \rightarrow r \rightarrow q \rightarrow q'$ (see picture, **left**)

If q' is not reachable from q , then if q' appears in a cycle, this was missed in a previous iteration, before starting the second DFS from q, contrary to the fact that q is the first state (contraddiction).

Therefore q' does not occur in a cycle, **but q is reachable from q' (via r)**. By the Lemma, we have discovered and backtracked from q, **before** starting the DFS from q', against our assumptions (r **flagged from q'**, see picture, **right**).

□



Lesson 4d:

On the Fly LTL Model Checking

LTL model checking via NBA

Theorem. For any LTL formula φ over AP, there exists a NBA $\mathcal{A}_\varphi$ wiht Words(φ)=$\mathcal{L}_\omega(\mathcal{A}_\varphi)$ which can be constructed in time and space $2^{\mathcal{O}(|\varphi|)}$.

The proof is rather technical and tedious, but the ingredients are exactly the same of the algorithm based on tableaux, see lesson **3**. In particular:

- automata states represent **maximal consistent sets of Cl(φ),**
- **transition relation** is related to presence of subformula of the form **X** ψ in Cl(φ), and
- **accepting states** are related to the presence of some ψ_1 **U** ψ_2 in Cl(φ). (Remember that **U** (and its negation) need to consider infinite paths).

Once one has $\mathcal{M} \otimes \mathcal{A}_{\neg\varphi}$ we just need **to check language emptiness**. **Remark**: even though NBAs are closed under complementation, it is convenient to build $\mathcal{A}_{\neg\varphi}$ rather than complementing $\mathcal{A}_\varphi$.

On-the-fly LTL model checking

Usually, the model $\mathcal{M}$ is described by a **high-level language**.

The generation of reachable states of $\mathcal{M}$ can proceed in parallel with the construction of the automaton $\mathcal{A}_{\neg\varphi}$ (remember that states of $\mathcal{M} \otimes \mathcal{A}_{\neg\varphi}$ are pairs).

The product automaton $\mathcal{M} \otimes \mathcal{A}_{\neg\varphi}$ is constructed **on demand**.

A new vertex is only considered if no accepting cycle has been found in the fragment of $\mathcal{M} \otimes \mathcal{A}_{\neg\varphi}$ already explored.

When generating the successor states in $\mathcal{A}_{\neg\varphi}$ we only need to consider those **successors matching the current state in $\mathcal{M}$**.

On-the-fly technique **is particurlarly effective** when a **refutation is early found**: in this case a counterexample is returned and large parts of $\mathcal{M} \otimes \mathcal{A}_{\neg\varphi}$ are not generated.

That's all Folks!

Thanks for your attention…

…Questions?