Verification

Lecture 5

Bernd Finkbeiner
Peter Faymonville
Michael Gerke

UNIVERSITÄT
DES
SAARLANDES

REVIEW: Safety

- Safety properties ≈ "nothing bad should happen" [Lamport 1977]
- Typical safety property: mutual exclusion property
  - the bad thing (having $>$ 1 process in the critical section) never occurs
- Another typical safety property is deadlock freedom

⇒ These properties are in fact invariants

- An invariant is an LT property
  - that is given by a condition Φ for the states
  - and requires that Φ holds for all reachable states
  - e.g., for mutex property $\Phi \equiv \neg crit_1 \lor \neg crit_2$

REVIEW: Safety properties and closures

LT property P over AP is a safety property

if and only if $closure(P) = P$

REVIEW: Liveness properties

LT property P_{live} over AP is a liveness property whenever

$$pref(P_{live}) = \left(2^{AP}\right)^*$$

- A liveness property is an LT property
  - that does not rule out any prefix
- Liveness properties are violated in "infinite time"
  - whereas safety properties are violated in finite time
  - finite traces are of no use to decide whether P holds or not
  - any finite prefix can be extended such that the resulting infinite trace satisfies P

REVIEW: A non-safety and non-liveness property

"the machine provides infinitely often beer
after initially providing sprite three times in a row"

- This property consists of two parts:
  - it requires beer to be provided infinitely often
  - ⇒ as any finite trace fulfills this, it is a liveness property
  - the first three drinks it provides should all be sprite
  - ⇒ bad prefix = one of first three drinks is beer; this is a safety property
- Property is thus a conjunction of a safety and a liveness property

does this apply to all such properties?

REVIEW: Decomposition theorem

For any LT property P over AP there exists
a safety property P_{safe} and a liveness property P_{live}
(both over AP) such that:

$$P = P_{safe} \cap P_{live}$$

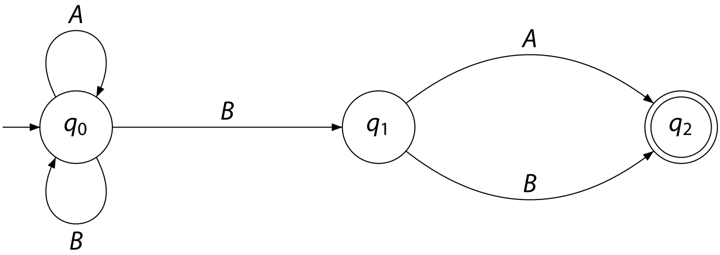
Proposal: $P = \underbrace{closure(P)}_{=P_{safe}} \cap \underbrace{\left(P \cup \left(\left(2^{AP}\right)^{\omega} \setminus closure(P)\right)\right)}_{=P_{live}}$

Regular properties

Finite automata

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

- Q is a finite set of states
- Σ is an alphabet
- $\delta : Q \times \Sigma \to 2^Q$ is a transition function
- $Q_0 \subseteq Q$ a set of initial states
- $F \subseteq Q$ is a set of accept (or: final) states



Size of an NFA

The size of $\mathcal{A}$, denoted $|\mathcal{A}|$, is the number of states and transitions in $\mathcal{A}$:

$$|\mathcal{A}| = |Q| + \sum_{q \in Q} \sum_{A \in \Sigma} |\, \delta(q, A) \,|$$

Language of an automaton

- NFA $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ and word $w = A_1 \dots A_n \in \Sigma^*$
- A *run* for w in $\mathcal{A}$ is a finite sequence $q_0\, q_1\ \dots\ q_n$ such that:
  - $q_0 \in Q_0$ and $q_i \xrightarrow{A_{i+1}} q_{i+1}$ for all $0 \leq i < n$
- Run $q_0\, q_1\ \dots\ q_n$ is accepting if $q_n \in F$
- $w \in \Sigma^*$ is *accepted* by $\mathcal{A}$ if there exists an accepting run for w
- The accepted language of $\mathcal{A}$:

$$\mathcal{L}(\mathcal{A}) = \left\{ w \in \Sigma^* \mid \text{there exists an accepting run for } w \text{ in } \mathcal{A} \right\}$$

- NFA $\mathcal{A}$ and $\mathcal{A}'$ are equivalent if $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$

Accepted language revisited

Extend the transition function δ to $\delta^* : Q \times \Sigma^* \rightarrow 2^Q$ by:

$$\delta^*(q, \varepsilon) = \{\, q \,\} \quad \text{and} \quad \delta^*(q, A) = \delta(q, A)$$

$$\delta^*(q, A_1 A_2 \ldots A_n) = \bigcup_{p \in \delta(q, A_1)} \delta^*(p, A_2 \ldots A_n)$$

$\delta^*(q, w)$ = set of states reachable from q for the word w

Then: $\mathcal{L}(\mathcal{A}) = \left\{ w \in \Sigma^* \mid \delta^*(q_0, w) \cap F \neq \varnothing \text{ for some } q_0 \in Q_0 \right\}$

The class of languages accepted by NFA (over Σ)

= the class of regular languages (over Σ)

Intersection

- Let NFA $\mathcal{A}_i = (Q_i, \Sigma, \delta_i, Q_{0,i}, F_i)$, with i=1, 2
- The product automaton

$$\mathcal{A}_1 \otimes \mathcal{A}_2 \; = \; (Q_1 \times Q_2, \Sigma, \delta, Q_{0,1} \times Q_{0,2}, F_1 \times F_2)$$

  where δ is defined by:

$$\frac{q_1 \xrightarrow{A}_1 q_1' \;\wedge\; q_2 \xrightarrow{A}_2 q_2'}{(q_1, q_2) \xrightarrow{A} (q_1', q_2')}$$

- Well-known result: $\mathcal{L}(\mathcal{A}_1 \otimes \mathcal{A}_2) \; = \; \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$

Total NFA

Automaton $\mathcal{A}$ is called deterministic if

$$|Q_0| \leq 1 \quad \text{and} \quad |\delta(q,A)| \leq 1 \quad \text{for all } q \in Q \text{ and } A \in \Sigma$$

DFA $\mathcal{A}$ is called total if

$$|Q_0| = 1 \quad \text{and} \quad |\delta(q,A)| = 1 \quad \text{for all } q \in Q \text{ and } A \in \Sigma$$

any DFA can be turned into an equivalent total DFA

total DFA provide unique successor states, and thus, unique runs for each input word

Determinization

For NFA $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ let $\mathcal{A}_{det} = (2^Q, \Sigma, \delta_{det}, Q_0, F_{det})$ with:

$$F_{det} = \{Q' \subseteq Q \mid Q' \cap F \neq \varnothing\}$$

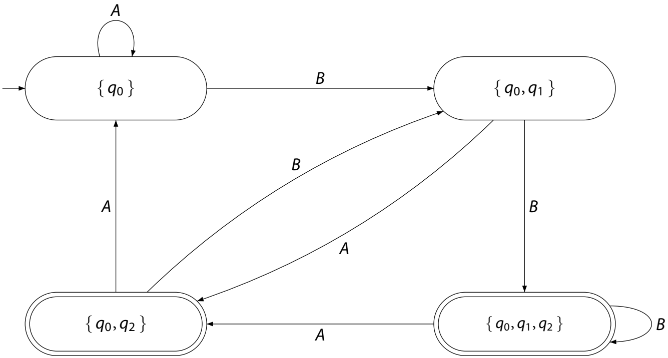
and the total transition function $\delta_{det} : 2^Q \times \Sigma \to 2^Q$ is defined by:

$$\delta_{det}(Q', A) = \bigcup_{q \in Q'} \delta(q, A)$$

$\mathcal{A}_{det}$ is a total DFA and, for all $w \in \Sigma^*$: $\delta^*_{det}(Q_0, w) = \bigcup_{q_0 \in Q_0} \delta^*(q_0, w)$

Thus: $\mathcal{L}(\mathcal{A}_{det}) = \mathcal{L}(\mathcal{A})$

Determinization



a deterministic finite automaton accepting $\mathcal{L}((A+B)^*B(A+B))$

Facts about finite automata

- They are as expressive as regular languages
- They are closed under $\cap$ and complementation
  - NFA $\mathcal{A} \otimes B$ (= cross product) accepts $\mathcal{L}(A) \cap \mathcal{L}(B)$
  - Total DFA $\overline{\mathcal{A}}$ (= swap all accept and normal states) accepts $\overline{\mathcal{L}(A)} = \Sigma^* \setminus \mathcal{L}(\mathcal{A})$
- They are closed under determinization (= removal of choice)
  - although at an exponential cost.....
- $\mathcal{L}(\mathcal{A}) = \emptyset$? = check for reachable accept state in $\mathcal{A}$
  - this can be done using a simple depth-first search
- For regular language $\mathcal{L}$ there is a unique minimal DFA accepting $\mathcal{L}$

Peterson's banking system

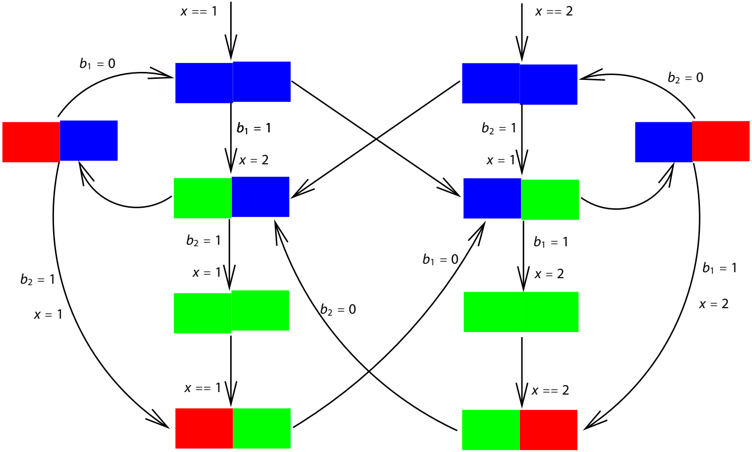
Person Left behaves as follows:

	while true {
	......
rq :	$b_1, x = \text{true}, 2;$
wt :	**wait until**$(x == 1 \,\|\|\, \neg b_2)$ {
cs :	$\ldots$@account$_L$ $\ldots$}
	$b_1 = \text{false};$
	......
	}

Person Right behaves as follows:

	while true {
	......
rq :	$b_2, x = \text{true}, 1;$
wt :	**wait until**$(x == 2 \,\|\|\, \neg b_1)$ {
cs :	$\ldots$@account$_R$ $\ldots$}
	$b_2 = \text{false};$
	......
	}

Is the banking system safe?



Can we guarantee that only one person at a time has access to the bank account?

"always $\neg($@account$_L$ $\wedge$ @account$_R)$"

Is the banking system safe?

- Safe = at most one person may have access to the account
- Unsafe: two have access to the account simultaneously
  - unsafe behaviour can be characterized by bad prefix
  - alternatively (in this case) by the finite automaton:

- Checking safety: *Traces*(*System*) ∩ *BadPref*(P_{safe}) = ∅?
  - intersection, complementation and emptiness of languages . . .

Regular safety properties

Safety property P_{safe} over AP is regular

if its set of bad prefixes is a regular language over 2^{AP}

every invariant is regular

Problem statement

Let

- P_{safe} be a regular safety property over *AP*
- $\mathcal{A}$ an NFA recognizing the bad prefixes of P_{safe}
  - assume that $\varepsilon \notin \mathcal{L}(\mathcal{A})$

  $\Rightarrow$ otherwise all finite words over 2^{AP} are bad prefixes
- *TS* a finite transition system (over *AP*) without terminal states

How to establish whether $TS \models P_{safe}$?

Basic idea of the algorithm

$TS \models P_{safe}$ if and only if $Traces_{fin}(TS) \cap BadPref(P_{safe}) = \varnothing$

if and only if $Traces_{fin}(TS) \cap \mathcal{L}(\mathcal{A}) = \varnothing$

if and only if $TS \otimes \mathcal{A} \models$ "always" Φ to be proven

But . . . . . . this amounts to invariant checking on $TS \otimes \mathcal{A}$

$\Rightarrow$ checking regular safety properties can be done by depth-first search!

Synchronous product (revisited)

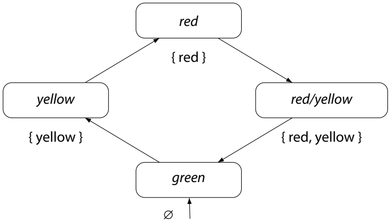
For transition system $TS = (S, Act, \rightarrow, I, AP, L)$ without terminal states and $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ an NFA with $\Sigma = 2^{AP}$ and $Q_0 \cap F = \varnothing$, let:

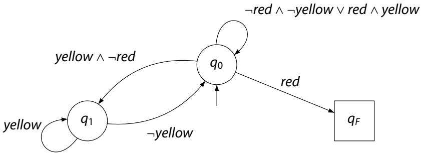
$$TS \otimes \mathcal{A} \;=\; (S', Act, \rightarrow', I', AP', L') \qquad \text{where}$$

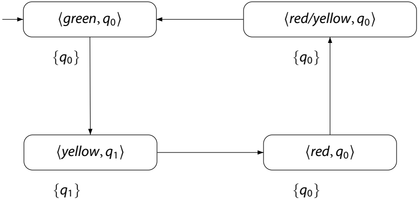
- $S' = S \times Q$, $AP' = Q$ and $L'(\langle s, q \rangle) \;=\; \{ \, q \, \}$
- $\rightarrow'$ is the smallest relation defined by: $\dfrac{s \xrightarrow{\alpha} t \;\wedge\; q \xrightarrow{L(t)} p}{\langle s, q \rangle \xrightarrow{\alpha}' \langle t, p \rangle}$
- $I' = \{ \, \langle s_0, q \rangle \mid s_0 \in I \;\wedge\; \exists q_0 \in Q_0.\, q_0 \xrightarrow{L(s_0)} q \, \}$

without loss of generality it may be assumed that $TS \otimes \mathcal{A}$ has no terminal states

Example product







Verification of regular safety properties

Let TS over AP and NFA $\mathcal{A}$ with alphabet 2^{AP} as before, regular safety property P_{safe} over AP such that $\mathcal{L}(\mathcal{A})$ is the set of bad prefixes of P_{safe}.

The following statements are equivalent:

(a) $TS \models P_{safe}$

(b) $Traces_{fin}(TS) \cap \mathcal{L}(\mathcal{A}) = \varnothing$

(c) $TS \otimes \mathcal{A} \models P_{inv(A)}$

where $P_{inv(A)} = \bigwedge_{q \in F} \neg q$

Counterexamples

For each initial path fragment $\langle s_0, q_1 \rangle \ldots \langle s_n, q_{n+1} \rangle$ of $TS \otimes \mathcal{A}$:

$$q_1, \ldots, q_n \notin F \text{ and } q_{n+1} \in F \quad \Rightarrow \quad \underbrace{trace(s_0 \, s_1 \ldots s_n)}_{\text{bad prefix for } P_{safe}} \in \mathcal{L}(\mathcal{A})$$

Verification algorithm

Require: finite transition system TS and regular safety property P_{safe}
Ensure: true if $TS \models P_{safe}$. Otherwise false plus a counterexample for P_{safe}.

Let NFA $\mathcal{A}$ (with accept states F) be such that $\mathcal{L}(\mathcal{A}) = BadPref(P_{safe})$;
Construct the product transition system $TS \otimes \mathcal{A}$;
Check the invariant $P_{inv(\mathcal{A})}$ with proposition $\neg F = \bigwedge_{q \in F} \neg q$ on $TS \otimes \mathcal{A}$

if $TS \otimes \mathcal{A} \models P_{inv(\mathcal{A})}$ **then**
  return true
else
  Determine initial path fragment $\langle s_0, q_1 \rangle \ldots \langle s_n, q_{n+1} \rangle$ of $TS \otimes \mathcal{A}$ with $q_{n+1} \in F$
  return $(\text{false}, s_0\, s_1 \ldots s_n)$
end if

Time complexity

The time and space complexity of checking a regular safety property P_{safe}

against transition system TS is in:

$$\mathcal{O}(|TS| \cdot |\mathcal{A}|)$$

where $\mathcal{A}$ is an NFA recognizing the bad prefixes of P_{safe}

Can time complexity be improved?

The safety property P_{safe} is regular

if and only if

the set of minimal bad prefixes for P_{safe} is regular

$BadPref(P_{safe})$ is regular if and only if $MinBadPref(P_{safe})$ is regular

$\Rightarrow$ use automaton for minimal bad prefixes in product construction

Büchi Automata

Peterson's banking system

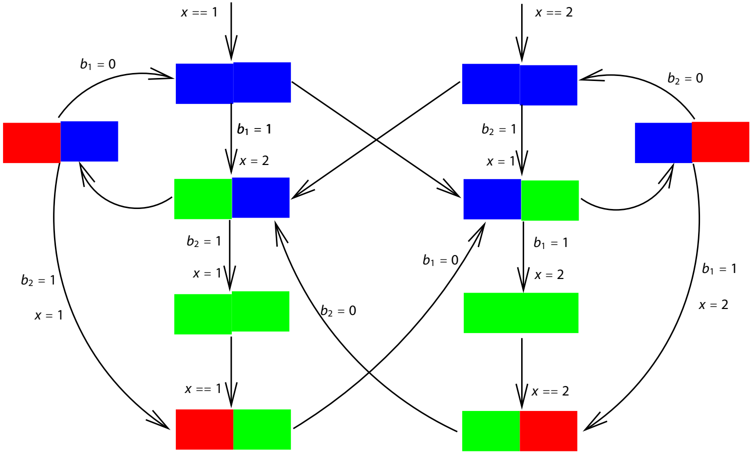
Person Left behaves as follows:

	while true {
	......
rq :	b_1, x = true, 2;
wt :	**wait until**$(x == 1 \,\|\|\, \neg b_2)$ {
cs :	...@account$_L$ ...}
	b_1 = false;
	......
	}

Person Right behaves as follows:

	while true {
	......
rq :	b_2, x = true, 1;
wt :	**wait until**$(x == 2 \,\|\|\, \neg b_1)$ {
cs :	...@account$_R$ ...}
	b_2 = false;
	......
	}

Is the banking system live?

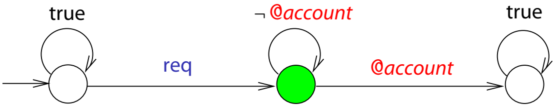


If someone wants to update the account, does (s)he ever get the opportunity to do so?

"always (req_L $\Rightarrow$ eventually $@account_L$) $\wedge$ always (req_R $\Rightarrow$ eventually $@account_R$)"

Is the banking system live (revisited)?

- Live = when you want access to account, you eventually get it
- Unlive: once you want access to the account, you never get it
  - unlive behaviour can be characterized as a (set of) infinite traces
  - or, equivalently, by a Büchi-automaton:



- Checking liveness: $Traces(System) \cap L_\omega(\overline{Live}) = \emptyset$?
  - (explicit) complementation, intersection and emptiness of Büchi automata!

ω-regular expressions

1. $\underline{\varnothing}$ and $\underline{\varepsilon}$ are regular expressions over Σ
2. if $A \in \Sigma$ then $\underline{A}$ is a regular expression over Σ
3. if E, E_1 and E_2 are regular expressions over Σ
   then so are $E_1 + E_2$, $E_1.E_2$ and E^*

E^+ is an abbreviation for the regular expression $E.E^*$

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

$$G = E_1.F_1^\omega + \ldots + E_n.F_n^\omega \quad \text{for } n > 0$$

where E_i, F_i are regular expressions over Σ such that $\varepsilon \notin \mathcal{L}(F_i)$, for all $0 < i \leq n$

Semantics of ω-regular expressions

- The semantics of regular expression E is a language $\mathcal{L}(E) \subseteq \Sigma^*$:

$$\mathcal{L}(\underline{\varnothing}) = \varnothing, \;\; \mathcal{L}(\underline{\varepsilon}) = \{\, \varepsilon \,\}, \;\; \mathcal{L}(\underline{A}) = \{\, A \,\}$$

$$\mathcal{L}(E{+}E') = \mathcal{L}(E) \cup \mathcal{L}(E') \quad \mathcal{L}(E.E') = \mathcal{L}(E).\mathcal{L}(E') \quad \mathcal{L}(E^*) = \mathcal{L}(E)^*$$

- The semantics of ω-regular expression G is a language $\mathcal{L}(G) \subseteq \Sigma^\omega$:

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

- G_1 and G_2 are equivalent, denoted $G_1 \equiv G_2$, if $\mathcal{L}_\omega(G_1) = \mathcal{L}_\omega(G_2)$

ω-regular languages and properties

- $\mathcal{L} \subseteq \Sigma^\omega$ is ω-regular if $\mathcal{L} = \mathcal{L}_\omega(\mathrm{G})$ for some ω-regular expression G (over Σ)
- ω-regular languages possess several closure properties
  - they are closed under union, intersection, and complementation
  - complementation is not treated here; we use a trick to avoid it
- LT property P over AP is called ω-regular

if P is an ω-regular language over the alphabet 2^{AP}

all invariants and regular safety properties are ω-regular!

Büchi automata

- NFA (and DFA) are incapable of accepting infinite words
- Automata on infinite words
  - suited for accepting ω-regular languages
  - we consider nondeterministic Büchi automata (NBA)
- Accepting runs have to "check" the entire input word $\Rightarrow$ are infinite

  $\Rightarrow$ acceptance criteria for infinite runs are needed
- NBA are like NFA, but have a distinct acceptance criterion
  - one of the accept states must be visited infinitely often

Büchi automata

A nondeterministic Büchi automaton (NBA) $\mathcal{A}$ is a tuple $(Q, \Sigma, \delta, Q_0, F)$ where:

- Q is a finite set of states with $Q_0 \subseteq Q$ a set of initial states
- Σ is an alphabet
- $\delta : Q \times \Sigma \rightarrow 2^Q$ is a transition function
- $F \subseteq Q$ is a set of accept (or: final) states

The size of $\mathcal{A}$, denoted $|\mathcal{A}|$, is the number of states and transitions in $\mathcal{A}$:

$$|\mathcal{A}| \;=\; |Q| + \sum_{q \in Q} \sum_{A \in \Sigma} |\,\delta(q, A)\,|$$

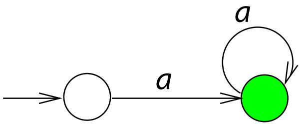
Language of an NBA

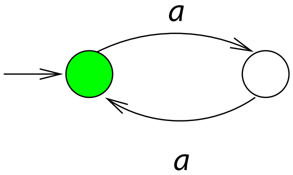
- NBA $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ and word $\sigma = A_0 A_1 A_2 \ldots \in \Sigma^\omega$
- A *run* for σ in $\mathcal{A}$ is an infinite sequence $q_0\, q_1\, q_2 \ldots$ such that:
  - $q_0 \in Q_0$ and $q_i \xrightarrow{A_{i+1}} q_{i+1}$ for all $0 \leq i$
- Run $q_0\, q_1\, q_2 \ldots$ is accepting if $q_i \in F$ for infinitely i
- $\sigma \in \Sigma^\omega$ is *accepted* by $\mathcal{A}$ if there exists an accepting run for σ
- The accepted language of $\mathcal{A}$:

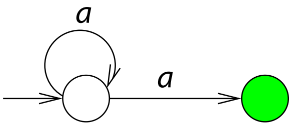
$$\mathcal{L}_\omega(\mathcal{A}) = \left\{ \sigma \in \Sigma^\omega \;\middle|\; \text{there exists an accepting run for } \sigma \text{ in } \mathcal{A} \right\}$$

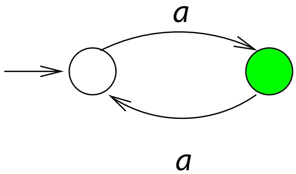
- NBA $\mathcal{A}$ and $\mathcal{A}'$ are equivalent if $\mathcal{L}_\omega(\mathcal{A}) = \mathcal{L}_\omega(\mathcal{A}')$

NBA versus NFA









finite equivalence
$\not\Rightarrow$ ω-equivalence

$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$,
but $\mathcal{L}_\omega(\mathcal{A}) \neq \mathcal{L}_\omega(\mathcal{A}')$

ω-equivalence
$\not\Rightarrow$ finite equivalence

$\mathcal{L}_\omega(\mathcal{A}) = \mathcal{L}_\omega(\mathcal{A}')$,
but $\mathcal{L}(\mathcal{A}) \neq \mathcal{L}(\mathcal{A}')$

NBA and ω-regular languages

The class of languages accepted by NBA
agrees with the class of ω-regular languages

(1) any ω-regular language is recognized by an NBA

(2) for any NBA $\mathcal{A}$, the language $\mathcal{L}_\omega(\mathcal{A})$ is ω-regular

For any ω-regular language there is an NBA

- How to construct an NBA for the ω-regular expression:

$$G = E_1.F_1^{\omega} + \ldots + E_n.F_n^{\omega} \text{ ?}$$

where E_i and F_i are regular expressions over alphabet Σ; $\varepsilon \notin F_i$

- Rely on operations for NBA that mimic operations on ω-regular expressions:
  (1) for NBA $\mathcal{A}_1$ and $\mathcal{A}_2$ there is an NBA accepting $\mathcal{L}_\omega(\mathcal{A}_1) \cup \mathcal{L}_\omega(\mathcal{A}_2)$
  (2) for any regular language $\mathcal{L}$ with $\varepsilon \notin \mathcal{L}$ there is an NBA accepting $\mathcal{L}^\omega$
  (3) for regular language $\mathcal{L}$ and NBA $\mathcal{A}'$ there is an NBA accepting $\mathcal{L}.\mathcal{L}_\omega(\mathcal{A}')$

Union of NBA

For NBA $\mathcal{A}_1$ and $\mathcal{A}_2$ (both over the alphabet Σ)

there exists an NBA $\mathcal{A}$ such that:

$$\mathcal{L}_\omega(\mathcal{A}) = \mathcal{L}_\omega(\mathcal{A}_1) \cup \mathcal{L}_\omega(\mathcal{A}_2) \quad \text{and} \quad |\mathcal{A}| = \mathcal{O}(|\mathcal{A}_1| + |\mathcal{A}_2|)$$

ω-operator for NFA

For each NFA $\mathcal{A}$ with $\varepsilon \notin \mathcal{L}(\mathcal{A})$ there exists an NBA $\mathcal{A}'$ such that:

$$\mathcal{L}_\omega(\mathcal{A}') = \mathcal{L}(\mathcal{A})^\omega \quad \text{and} \quad |\mathcal{A}'| = \mathcal{O}(|\mathcal{A}|)$$

Concatenation of an NFA and an NBA

For NFA $\mathcal{A}$ and NBA $\mathcal{A}'$ (both over the alphabet Σ

there exists an NBA $\mathcal{A}''$ with

$\mathcal{L}_\omega(\mathcal{A}'') = \mathcal{L}(\mathcal{A}).\mathcal{L}_\omega(\mathcal{A}')$ and $|\mathcal{A}''| = \mathcal{O}(|\mathcal{A}| + |\mathcal{A}'|)$

Summarizing the results so far

For any ω-regular language $\mathcal{L}$

there exists an NBA $\mathcal{A}$ with $\mathcal{L}_\omega(\mathcal{A}) = \mathcal{L}$

NBA accept ω-regular languages

For each NBA $\mathcal{A}$: $\mathcal{L}_\omega(\mathcal{A})$ is ω-regular