10. Automata-based Model Checking

Computer-Aided Verification

Dave Parker

University of Birmingham

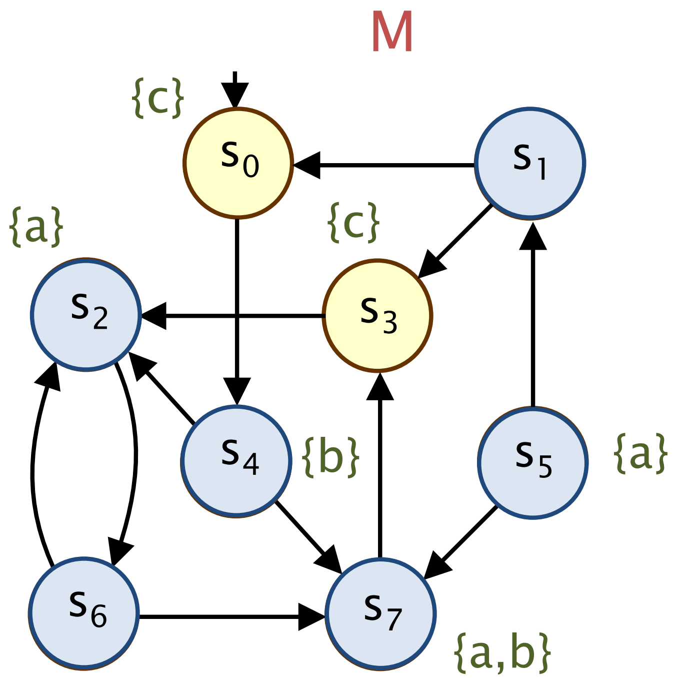
2017/18

Model checking $\exists\Box$

- Procedure to compute $Sat(\exists\Box\phi)$
  - given $Sat(\phi)$
- It again helps to consider expansion laws:
  - $\exists(\phi_1 \; U \; \phi_2) \;\equiv\; \phi_2 \vee (\phi_1 \wedge \exists\bigcirc \, \exists(\phi_1 \; U \; \phi_2))$
  - $\exists\Box\phi \;\equiv\; \phi \wedge \exists\bigcirc \, \exists\Box\phi$
- Basic idea: again, backwards search of the LTS
  - $T_0 := Sat(\phi)$
  - $T_i := T_{i-1} \cap \{ \, s \in Sat(\phi) \mid Post(s) \cap T_{i-1} \neq \emptyset \, \}$
  - until $T_i = T_{i-1}$
  - $Sat(\exists\Box\phi) = T_i$
- (i.e. keep <u>removing</u> states that are not predecessors of T_{i-1})

Example – $\exists\Box$

- Model the check CTL formula: $\phi = \forall\Diamond c$
  - convert to ENF: $\forall\Diamond c \equiv \neg\exists\Box\neg c$
  - $Sat(\neg c) = S \setminus \{s_0,s_3\} = \{s_1,s_2,s_4,s_5,s_6,s_7\}$



- Backwards search
  - $T_0 := Sat(\neg c) = \{s_1,s_2,s_4,s_5,s_6,s_7\}$
  - $T_1 := T_0 \cap \{s_2,s_4,s_5,s_6\} = \{s_2,s_4,s_5,s_6\}$
  - $T_2 := T_1 \cap \{s_2,s_4,s_6\} = \{s_2,s_4,s_6\}$
  - $T_3 := T_2 \cap \{s_2,s_4,s_6\} = \{s_2,s_4,s_6\}$
  - $T_3 = T_2$
  - $Sat(\exists\Box\neg c) = \{s_2,s_4,s_6\}$
  - $Sat(\phi) = S \setminus \{s_2,s_4,s_6\} = \{s_0,s_1,s_3,s_5,s_7\}$

- So: $M \vDash \phi$

Model checking ∃□

- More detailed algorithm:

```
CheckExistsAlways(Sat(ϕ)):

E := S \ Sat(ϕ)
T := Sat(ϕ)
for all s ∈ Sat(ϕ) do count[s] := |Post(s)| od
while (E ≠ ∅) do
    let s' ∈ E
    E := E \ {s'}
    for all s ∈ Pre(s') do
        if s ∈ T then
            count[s] := count[s] – 1
            if (count[s] = 0) then T := T \ {s}; E := E ∪ {s} fi
        fi
    od
od
return T
```

Alternative algorithm for $\exists\Box$

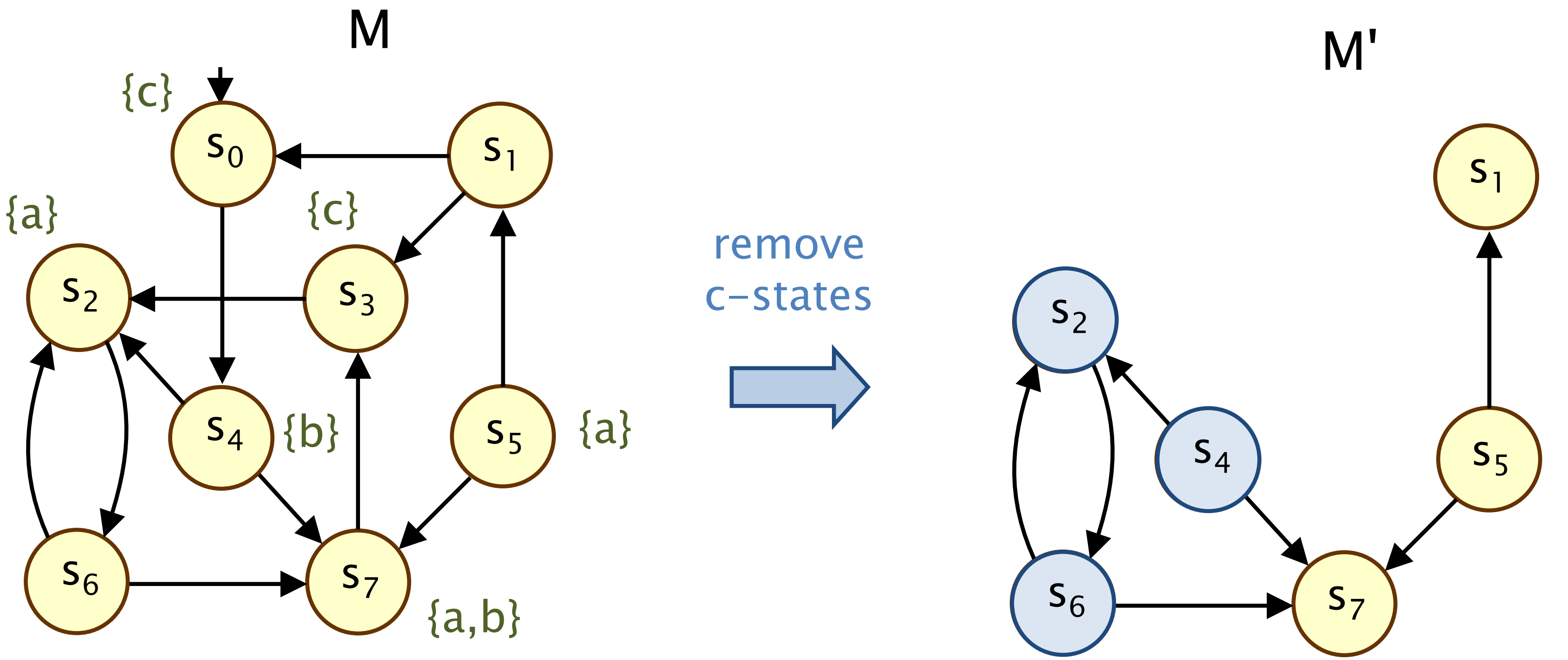
- An alternative algorithm to model check $\exists\Box\phi$ on LTS M
  - based on strongly connected components

- Strongly connected components (SCCs)
  - SCC = maximal, connected sub-graph
  - non-trivial SCC = SCC with at least one transition

- Model checking $\exists\Box$
  1. construct a modified LTS M' by
     - removing all states not satisfying ϕ, i.e. those in $S \setminus Sat(\phi)$
     - and removing all transitions to/from those states
  2. find the non-trivial strongly connected components (SCCs) in M'
  3. $Sat(\phi)$ is the set of states that can reach an SCC in M'

Example revisited – ∃☐

- Model the check CTL formula: $\phi' = \exists\Box\neg c$
  - convert M to produce M'
  - identify non-trivial SCCs in M': $\{s_2,s_6\}$
  - identify states than can reach the SCCs: Sat(ϕ') = $\{s_2,s_4,s_6\}$



Complexity

- The time complexity of CTL model checking
  - for LTS M and CTL formula ϕ
- is: $O(|M| \cdot |\phi|)$
  - i.e. linear in both model and formula size
  - where $|M|$ = number of states + number of transitions in M
  - and $|\phi|$ = number of operators in ϕ
- Worst-case execution:
  - all operators are temporal operators
  - each one performs single traversal of whole model

CTL model checking: Wrapping up

- CTL model checking
  - global model checking algorithm
  - recursive computation of Sat(ϕ)
  - based on parse tree of ϕ

- Conversion to existential normal form (ENF)
  - $\exists\bigcirc$, $\exists$U, $\exists\Box$ only
  - i.e. reduces to looking for <u>existence</u> of paths

- Graph-based algorithms on LTS
  - backwards graph traversal or SCCs

9. Automata-based Model Checking

Computer-Aided Verification

Dave Parker

University of Birmingham

2017/18

Branching-time vs. linear-time

- So far:
  - model checking for branching-time properties (CTL)
  - e.g. $\phi = (\forall\Box\ \exists\Diamond\ a) \wedge (\exists\Box\ b)$
- Now: linear-time properties, e.g. as specified in LTL
  - e.g. $\Diamond\Box c \wedge \Box(d \rightarrow \bigcirc\neg c)$
- Next lectures: automata-based properties
  - connections between automata and logic
  - first: finite automata and safety properties

Overview

- Recap
  - linear-time properties
  - safety properties
- Nondeterministic finite automata (NFAs)
  - regular languages
  - regular expressions
- Regular safety properties
  - LTS-NFA products
  - model checking
- See [BK08] Sections 4–4.2

Reminder: Notation

- A (finite or infinite) word over a finite alphabet Σ is
  - a finite sequence $w = A_0A_1 \ldots A_n$ where $A_i \in \Sigma$ for all $0 \leq i < n$
  - an infinite sequence $\sigma = A_0A_1 \ldots$ where $A_i \in \Sigma$ for all $i \geq 0$

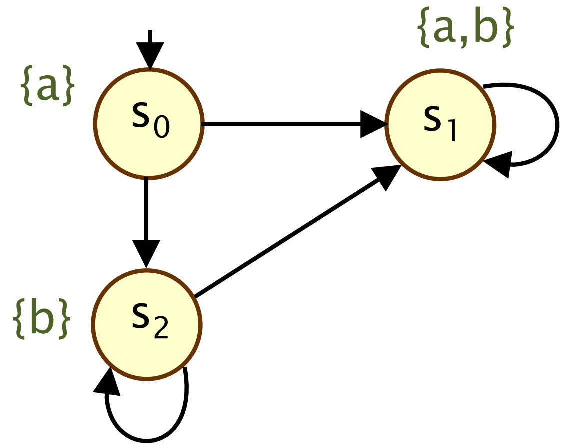
- A prefix w of word $\sigma = A_0A_1 \ldots$ is
  - a finite word $B_0B_1 \ldots B_n$ with $B_i = A_i$ for all $0 \leq i \leq n$
- A suffix σ' of word $\sigma = A_0A_1 \ldots$ is
  - an infinite word $B_0B_1 \ldots$ with $B_i = A_{i+j}$ for some $j \geq 0$ and all $0 \leq i \leq n$

- Σ^* denotes the set of finite words over Σ

- Σ^ω denotes the set of infinite words over Σ

Recap – Linear-time properties

- Paths: sequences of connected states
  - e.g. $\pi = s_0 s_2 s_2 s_1 s_1 s_1 \ldots$
- Traces: infinite words over 2^{AP}
  - $trace(\pi) = \{a\}\{b\}\{b\}\{a,b\}\{a,b\}\{a,b\}\ldots$
  - $Traces(M)$ = traces of all paths



- Linear-time properties
  - set of allowable traces $P \subseteq (2^{AP})^\omega$
  - e.g. $\Box(a \rightarrow \Diamond b)$ – "a is always eventually followed by b"
  - $M \models P \;\Leftrightarrow\; Traces(M) \subseteq P \;\Leftrightarrow\; trace(\pi) \in P$ for all paths π of M
- Classes of (linear-time) property:
  - invariant, safety property, liveness property…
  - independent of any particular model…

Recap – Safety properties

- Informally:
  - "nothing bad happens", e.g. "a failure does not occur"
  - defined in terms of the "bad" events, which happen in finite time

- More precisely
  - P_{safe} is a safety property if any (infinite) word where P_{safe} does not hold has a bad prefix
  - a bad prefix is a finite prefix σ' containing the bad event, such that no infinite path beginning with σ' satisfies P_{safe}
  - the bad prefixes define the safety property

- Formally:
  - $P_{safe} = (2^{AP})^{\omega} \setminus \{ w.\sigma' \in (2^{AP})^{*} \mid$ for some bad prefix w, suffix $\sigma' \}$

Example safety properties

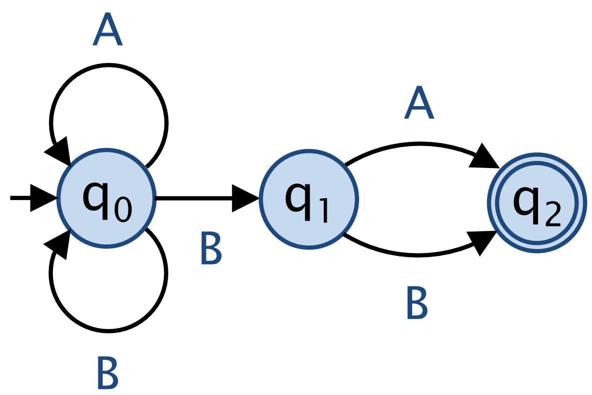
- Example safety properties:
  - over AP = {red_1,$green_1$,red_2,$green_2$}

- "the traffic lights never both show green simultaneously":
  - what are the bad prefixes?
  - <u>any</u> finite word ending in {$green_1$,$green_2$}

- "$green_1$ is always preceded (immediately) by red_1"
  - what are the bad prefixes?
  - <u>any</u> finite word where $green_1$ appears and red_1 did not appear immediately before it

Nondeterministic finite automata

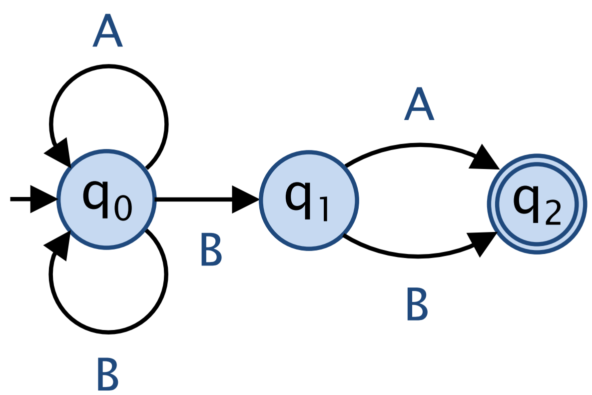
- A nondeterministic finite automaton (NFA) is:
  - a tuple $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$
- where:
  - Q is a finite set of states
  - Σ is an alphabet
  - $\delta : Q \times \Sigma \rightarrow 2^Q$ is a transition function
  - $Q_0 \subseteq Q$ is a set of initial states
  - $F \subseteq Q$ is a set of "accept" states



- Example
  - $Q = \{q_0,q_1,q_2\}$, $Q_0 = \{q_0\}$, $F = \{q_2\}$
  - $\Sigma = \{A,B\}$, $\delta(q_0,A) = \{q_0\}$, $\delta(q_0,B) = \{q_0,q_1\}$, …

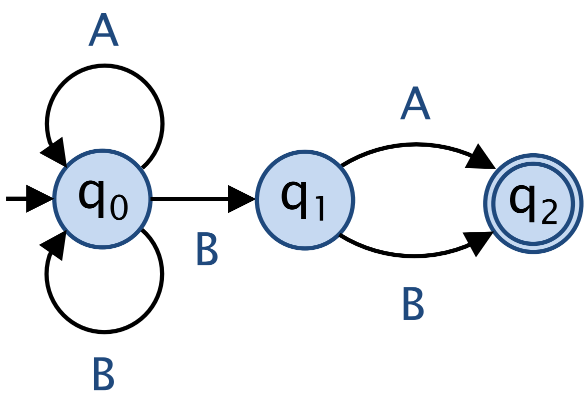
Runs of an NFA

- For an NFA $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$…
- There is an A transition from q to q' (written $q \xrightarrow{A} q'$)
  - if $q' \in \delta(q, A)$
- A run of $\mathcal{A}$ on a finite word $w = A_0A_1\ldots A_{n-1}$ is:
  - a sequence of automata states $q_0q_1\ldots q_n$ such that:
  - $q_0 \in Q_0$ and $q_i \xrightarrow{A_i} q_{i+1}$ for all $0 \leq i < n$
- Example
  - a word: BBA
  - a run: $q_0q_0q_1q_2$



Language of an NFA

- An accepting run is a run ending in an accept state
  - i.e. a run $q_0q_1\ldots q_n$ with $q_n \in F$
- Word w is accepted by $\mathcal{A}$ iff:
  - there exists an accepting run of A on w
- Example
  - BBA (accepted)
  - BAA (not accepted)
- The language of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$ is:
  - the set of all words accepted by $\mathcal{A}$
- Automata $\mathcal{A}$ and $\mathcal{A}'$ are equivalent if $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$



language: "penultimate symbol is B"

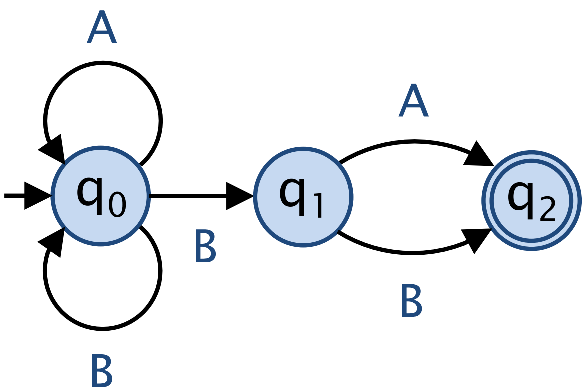
Regular expressions

- Regular expressions E over a finite alphabet Σ
  - are given by the following grammar:
  - $E ::= \varnothing \mid \varepsilon \mid A \mid E + E \mid E.E \mid E^*$
  - where $A \in \Sigma$

- Language $\mathcal{L}(E) \subseteq \Sigma^*$ of a regular expression:
  - $\mathcal{L}(\varnothing) = \varnothing$ (empty language)
  - $\mathcal{L}(\varepsilon) = \{ \varepsilon \}$ (empty word)
  - $\mathcal{L}(A) = \{ A \}$ (symbol)
  - $\mathcal{L}(E_1 + E_2) = \mathcal{L}(E_1) \cup \mathcal{L}(E_2)$ (union)
  - $\mathcal{L}(E_1.E_2) = \{ w_1.w_2 \mid w_1 \in \mathcal{L}(E_1) \text{ and } w_2 \in \mathcal{L}(E_2) \}$ (concatenation)
  - $\mathcal{L}(E^*) = \{ w^i \mid w \in \mathcal{L}(E) \text{ and } i \in \mathbb{N} \}$ (finite repetition)

Regular languages

- A set of finite words $\mathcal{L} \subseteq \Sigma^*$ is a regular language…
  - iff $\mathcal{L} = \mathcal{L}(E)$ for some regular expression E
  - iff $\mathcal{L} = \mathcal{L}(\mathcal{A})$ for some finite automaton $\mathcal{A}$



(A+B)*B(A+B)

(i.e. penultimate symbol is B)

Operations on NFAs

- Intersection of two NFAs
  - build (synchronised) product automaton
  - cross product of $\mathcal{A}_1 \otimes \mathcal{A}_2$ accepts $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$

- Language emptiness of an NFA
  - reduces to reachability
  - $L(\mathcal{A}) \neq \emptyset$ iff can reach a state in F from an initial state in Q_0

- Other important operations
  - construction of an NFA from a regular expression, inductively
  - determinisation (convert to deterministic finite automaton (DFA))
  - complementation of an NFA (via conversion to a DFA)