9. CTL Model Checking

Computer-Aided Verification

Dave Parker

University of Birmingham

2017/18

Recap: CTL vs. LTL

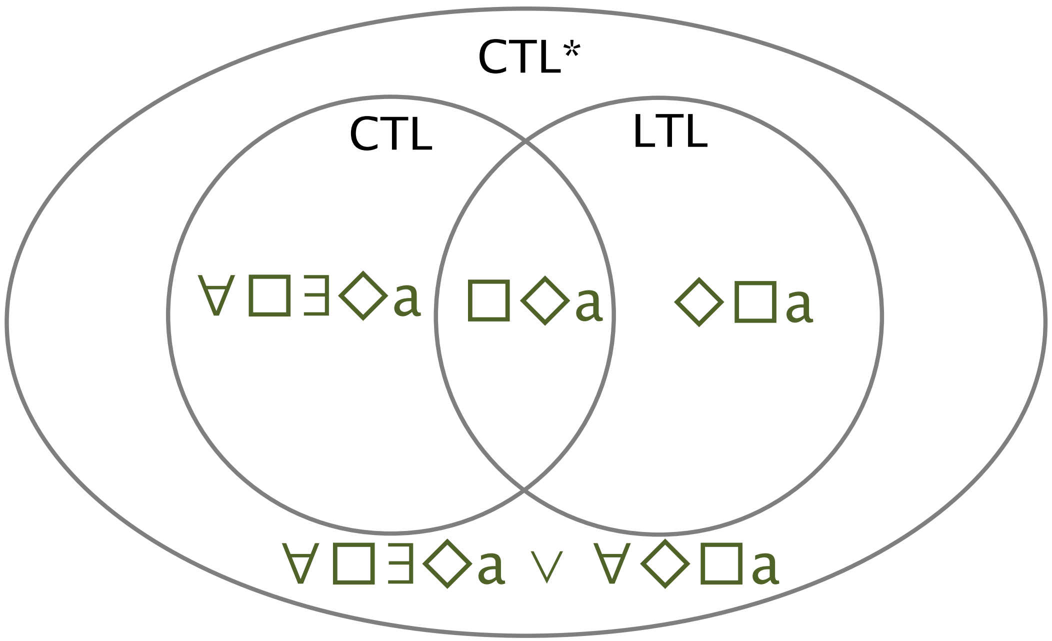
- Key differences between CTL and LTL:
  - branching-time vs. linear-time
  - state-based vs. path-based
  - expressiveness: incomparable
  - model checking algorithms differ
    - CTL simpler and lower complexity than LTL
    - (linear in size of ϕ vs. exponential in size of ψ)
  - fairness dealt with more easily in LTL

- Both CTL and LTL are a subset of the logic CTL*
  - path quantifiers ($\forall$, $\exists$) arbitrarily nested with temporal operators

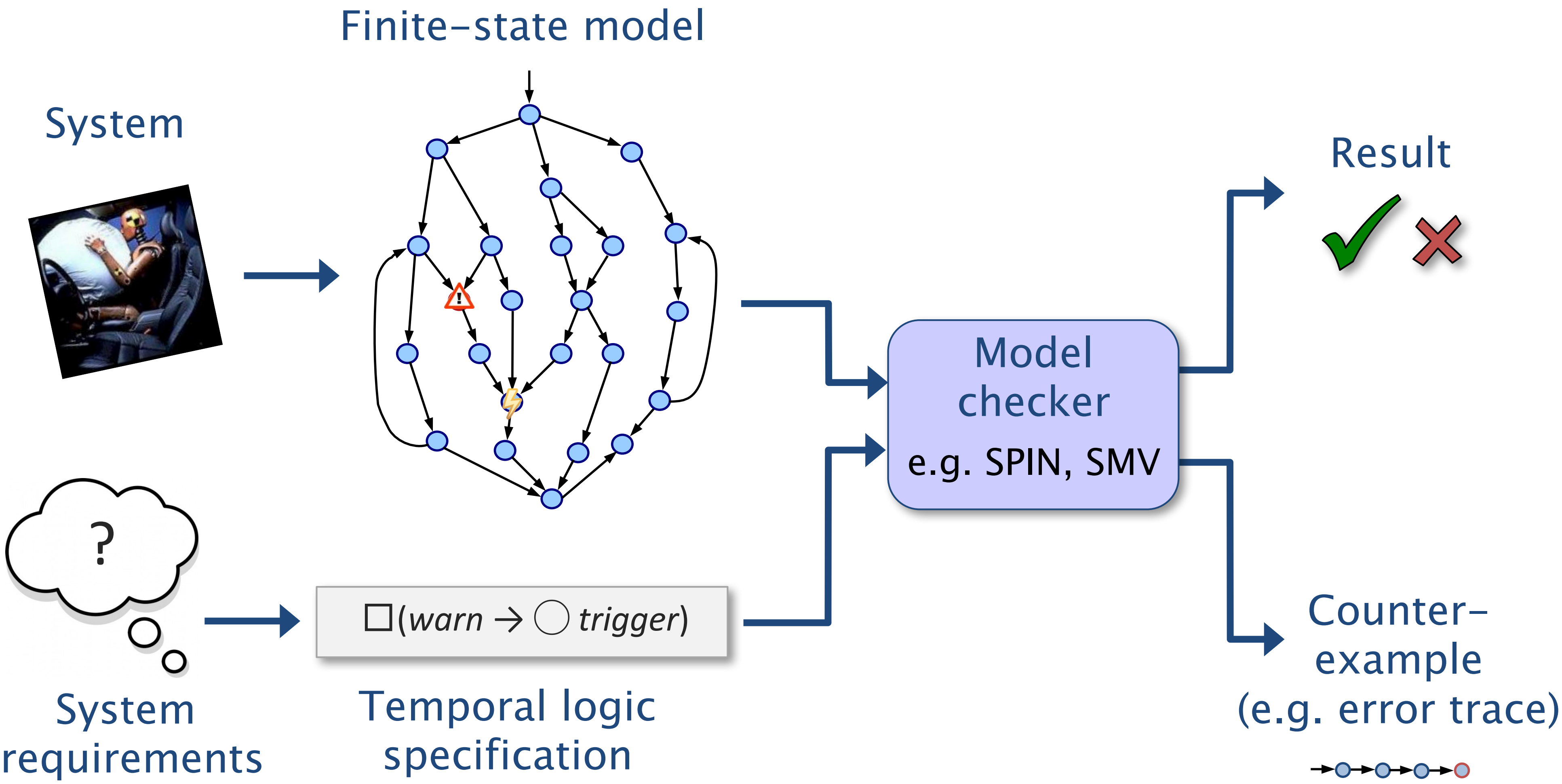
CTL*

- CTL* syntax
  - ϕ ::= true | a | $\phi \wedge \phi$ | $\neg\phi$ | $\forall\ \psi$ | $\exists\ \psi$
  - ψ ::= ϕ | $\psi \wedge \psi$ | $\neg\psi$ | $\bigcirc\ \psi$ | $\psi\ \mathsf{U}\ \psi$ | $\Diamond\ \psi$ | $\Box\ \psi$

- Example
  - $\forall\bigcirc\Box a \wedge \exists\Diamond\Box b$



Verification via model checking



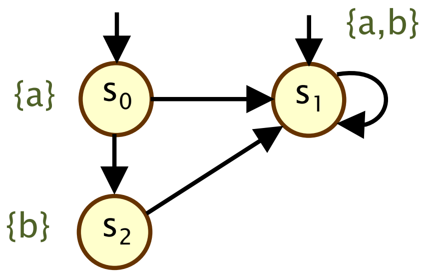
Overview

- CTL model checking
  - the model checking problem
  - basic algorithm
  - (existential normal form)
  - model checking ∃U
  - model checking ∃□

- See [BK08] Section 6.4

CTL model checking

- The CTL model checking problem is:
  - given an LTS $M = (S, Act, \to, I, AP, L)$ and a CTL formula ϕ,
  - check whether $M \vDash \phi$
  - i.e. whether $s \vDash \phi$ for all initial states $s \in I$



$\exists \bigcirc \forall \Box (a \wedge b)$

- Assumptions:
  - M is finite and has no terminal states

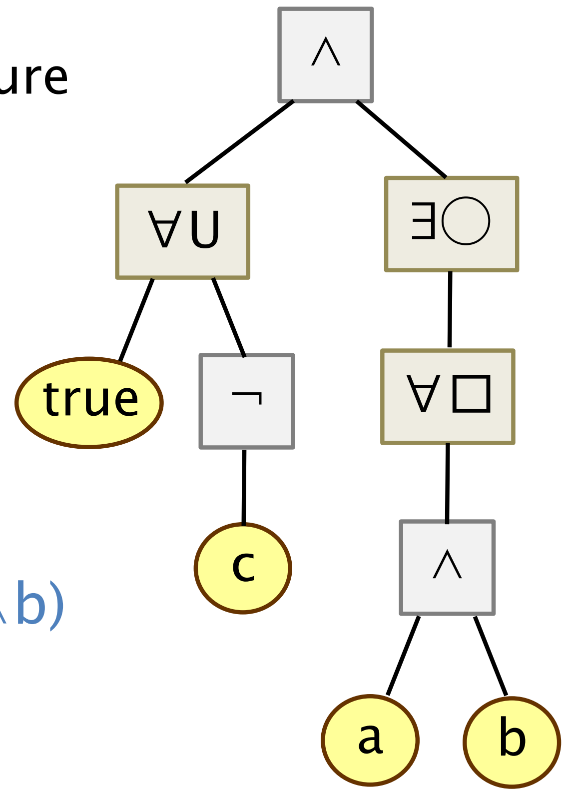
- Sat(ϕ) is the the satisfaction set for CTL formula ϕ
  - i.e. the set of all states that satisfy ϕ
  - $Sat(\phi) = \{ s \in S \mid s \vDash \phi \}$
  - so model checking is determining whether $I \subseteq Sat(\phi)$

Basic algorithm

- Compute Sat(ϕ), then check if I ⊆ Sat(ϕ)
  - so we in fact check if $s \vDash \phi$ for <u>all</u> states s
  - known as a global model checking procedure

- Sat(ϕ) is computed recursively
  - bottom-up traversal
  - of the parse tree of formula ϕ

- Example: $\phi = \forall(\text{true } U \neg c) \wedge \exists\bigcirc \forall\Box (a \wedge b)$



Computing Sat(ϕ)

- Recursive computation of Sat(ϕ):
  - $Sat(true) = S$
  - $Sat(a) = \{ s \in S \mid a \in L(s) \}$
  - $Sat(\phi_1 \wedge \phi_2) = Sat(\phi_1) \cap Sat(\phi_2)$
  - $Sat(\neg \phi) = S \setminus Sat(\phi)$
  - $Sat(\exists \bigcirc \phi) = \ldots$
  - $Sat(\forall \bigcirc \phi) = \ldots$
  - $Sat(\exists(\phi_1 \mathbin{U} \phi_2)) = \ldots$
  - $Sat(\forall(\phi_1 \mathbin{U} \phi_2)) = \ldots$
  - …

Existential Normal Form (ENF)

- We simplify model checking by first converting to ENF
  - no universal path quantifier ($\forall$) allowed, and no $\exists\Diamond$ formulae
  - $\phi ::= \text{true} \mid a \mid \phi \wedge \phi \mid \neg\phi \mid \exists\bigcirc\phi \mid \exists(\phi \text{ U } \phi) \mid \exists\Box\phi$

- Conversion to ENF:
  - $\exists\Diamond\phi \equiv \exists(\text{true U } \phi)$
  - $\forall\bigcirc\phi \equiv \neg\exists\bigcirc\neg\phi$
  - $\forall\Diamond\phi \equiv \neg\exists\Box\neg\phi$
  - $\forall\Box\phi \equiv \neg\exists\Diamond\neg\phi \equiv \neg\exists(\text{true U } \neg\phi)$
  - $\forall(\phi_1 \text{ U } \phi_2) \equiv \neg\exists((\neg\phi_2 \text{ U } (\neg\phi_1 \wedge \neg\phi_2))) \wedge \neg\exists(\Box\neg\phi_2)$

Computing Sat(ϕ)

- Recursive computation of Sat(ϕ):
  - $Sat(true) = S$
  - $Sat(a) = \{ s \in S \mid a \in L(s) \}$
  - $Sat(\phi_1 \wedge \phi_2) = Sat(\phi_1) \cap Sat(\phi_2)$
  - $Sat(\neg\phi) = S \setminus Sat(\phi)$
  - $Sat(\exists \bigcirc \phi) = \{ s \in S \mid Post(s) \cap Sat(\phi) \neq \emptyset \}$
  - $Sat(\exists(\phi_1\ U\ \phi_2)) = CheckExistsUntil(Sat(\phi_1), Sat(\phi_2))$
  - $Sat(\exists \Box \phi) = CheckExistsAlways(Sat(\phi))$

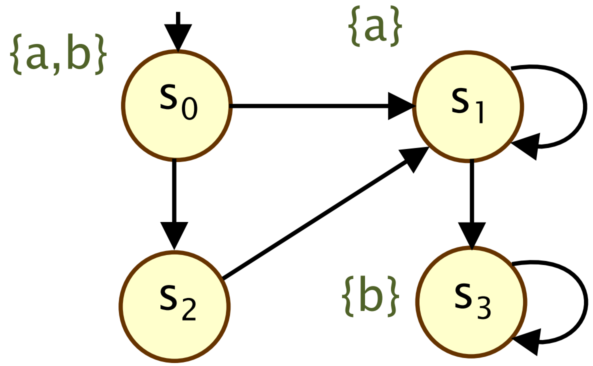
Propositional logic

Immediate predecessors

Graph algorithms

Example

- Model the check CTL formula: $\phi = \forall \bigcirc a \wedge \exists \bigcirc \neg b$



- Convert to ENF:
  - $\phi \equiv \neg \exists \bigcirc \neg a \wedge \exists \bigcirc \neg b$
- Evaluate Sat(ϕ) recursively
  - $Sat(a) = \{s_0, s_1\}$
  - $Sat(\neg a) = S \setminus Sat(a) = S \setminus \{s_0, s_1\} = \{s_2, s_3\}$
  - $Sat(\exists \bigcirc \neg a) = \{s_0, s_1, s_3\}$
  - $Sat(\neg \exists \bigcirc \neg a) = S \setminus \{s_0, s_1, s_3\} = \{s_2\}$
  - $Sat(b) = \{s_0, s_3\}$
  - $Sat(\neg b) = S \setminus \{s_0, s_3\} = \{s_1, s_2\}$
  - $Sat(\exists \bigcirc \neg b) = \{s_0, s_1, s_2\}$
  - $Sat(\phi) = \{s_2\} \cap \{s_0, s_1, s_2\} = \{s_2\} \quad \Rightarrow \quad M \not\models \phi$

Model checking $\exists U$

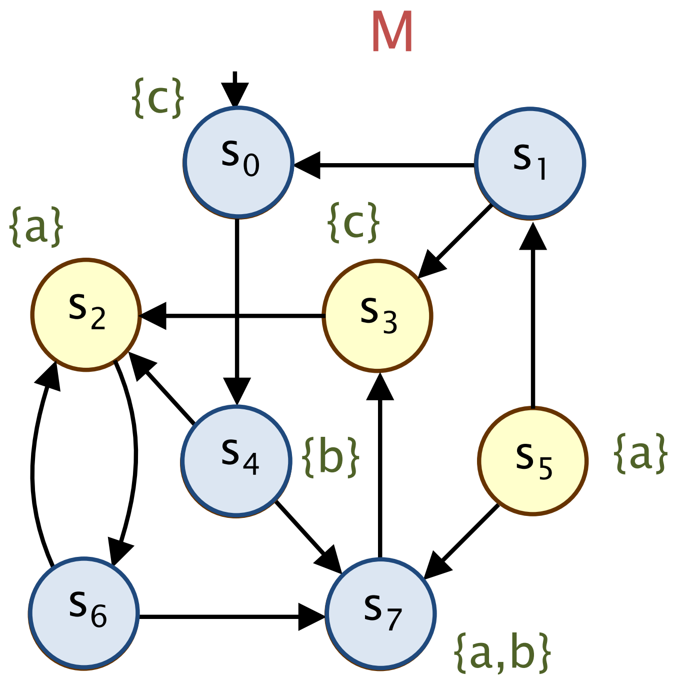
- Procedure to compute $Sat(\exists(\phi_1 \; U \; \phi_2))$
  - given $Sat(\phi_1)$ and $Sat(\phi_2)$

- Basic idea: backwards search of the LTS from ϕ_2-states
  - $T_0 := Sat(\phi_2)$
  - $T_i := T_{i-1} \cup \{ s \in Sat(\phi_1) \mid Post(s) \cap T_{i-1} \neq \emptyset \}$
  - until $T_i = T_{i-1}$
  - $Sat(\exists(\phi_1 \; U \; \phi_2)) = T_i$
- (i.e. keep adding predecessors of states in T_{i-1})

- Based on expansion law
  - $\exists(\phi_1 \; U \; \phi_2) \equiv \phi_2 \vee (\phi_1 \wedge \exists\bigcirc \exists(\phi_1 \; U \; \phi_2))$
  - (can be formulated as a fixed-point equation)

Example – ∃U

- Model the check CTL formula: $\phi = \exists\, (\neg a \;\mathsf{U}\; b)$
  - $Sat(\neg a) = S \setminus \{s_2,s_5,s_7\} = \{s_0,s_1,s_3,s_4,s_6\}$
  - $Sat(b) = \{s_4,s_7\}$



- Backwards search
  - $T_0 := Sat(b) = \{s_4,s_7\}$
  - $T_1 := T_0 \cup \{s_0,s_4,s_6\} = \{s_0,s_4,s_6,s_7\}$
  - $T_2 := T_1 \cup \{s_0,s_1,s_4,s_6\} = \{s_0,s_1,s_4,s_6,s_7\}$
  - $T_3 := T_2 \cup \{s_0,s_1,s_4,s_6\} = \{s_0,s_1,s_4,s_6,s_7\}$
  - $T_3 = T_2$
  - $Sat(\phi) = \{s_0,s_1,s_4,s_6,s_7\}$

- So: $M \models \phi$

Model checking ∃U

- More detailed algorithm:

<u>CheckExistsUntil(Sat(ϕ_1), Sat(ϕ_2)):</u>

```
E := Sat(ϕ2)
T := E
while (E ≠ ∅) do
      let s' ∈ E
      E := E \ {s'}
      for all s ∈ Pre(s') do
            if s ∈ Sat(ϕ1) \ T then E := E ∪ {s} ; T := T ∪ {s} fi
      od
od
return T
```

Model checking $\exists\Box$

- Procedure to compute $\mathrm{Sat}(\exists\Box\phi)$
  - given $\mathrm{Sat}(\phi)$

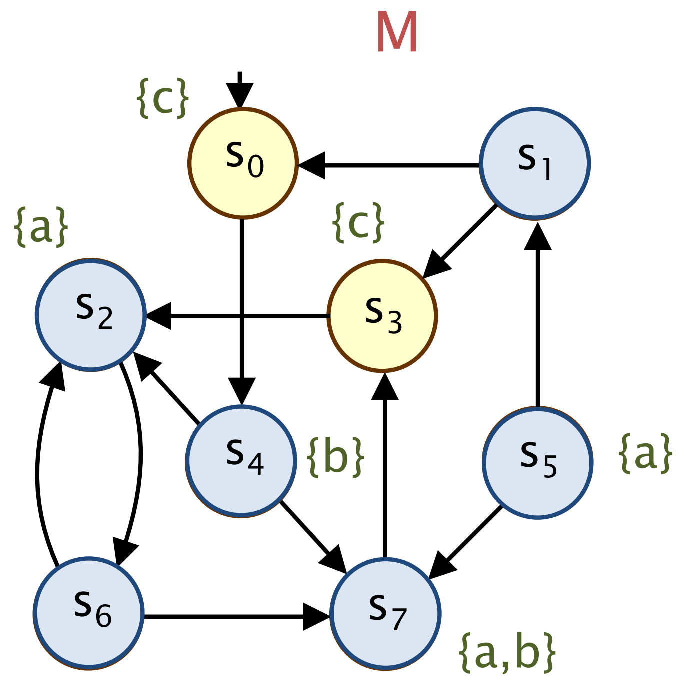
- It again helps to consider expansion laws:
  - $\exists(\phi_1 \mathsf{U} \phi_2) \equiv \phi_2 \vee (\phi_1 \wedge \exists\bigcirc \exists(\phi_1 \mathsf{U} \phi_2))$
  - $\exists\Box\phi \equiv \phi \wedge \exists\bigcirc \exists\Box\phi$

- Basic idea: again, backwards search of the LTS
  - $T_0 := \mathrm{Sat}(\phi)$
  - $T_i := T_{i-1} \cap \{ s \in \mathrm{Sat}(\phi) \mid \mathrm{Post}(s) \cap T_{i-1} \neq \emptyset \}$
  - until $T_i = T_{i-1}$
  - $\mathrm{Sat}(\exists\Box\phi) = T_i$
- (i.e. keep removing 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: $M \models \phi$

Model checking ∃□

- More detailed algorithm:

<u>CheckExistsAlways(Sat(ϕ)):</u>

```
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 $\exists\Box$

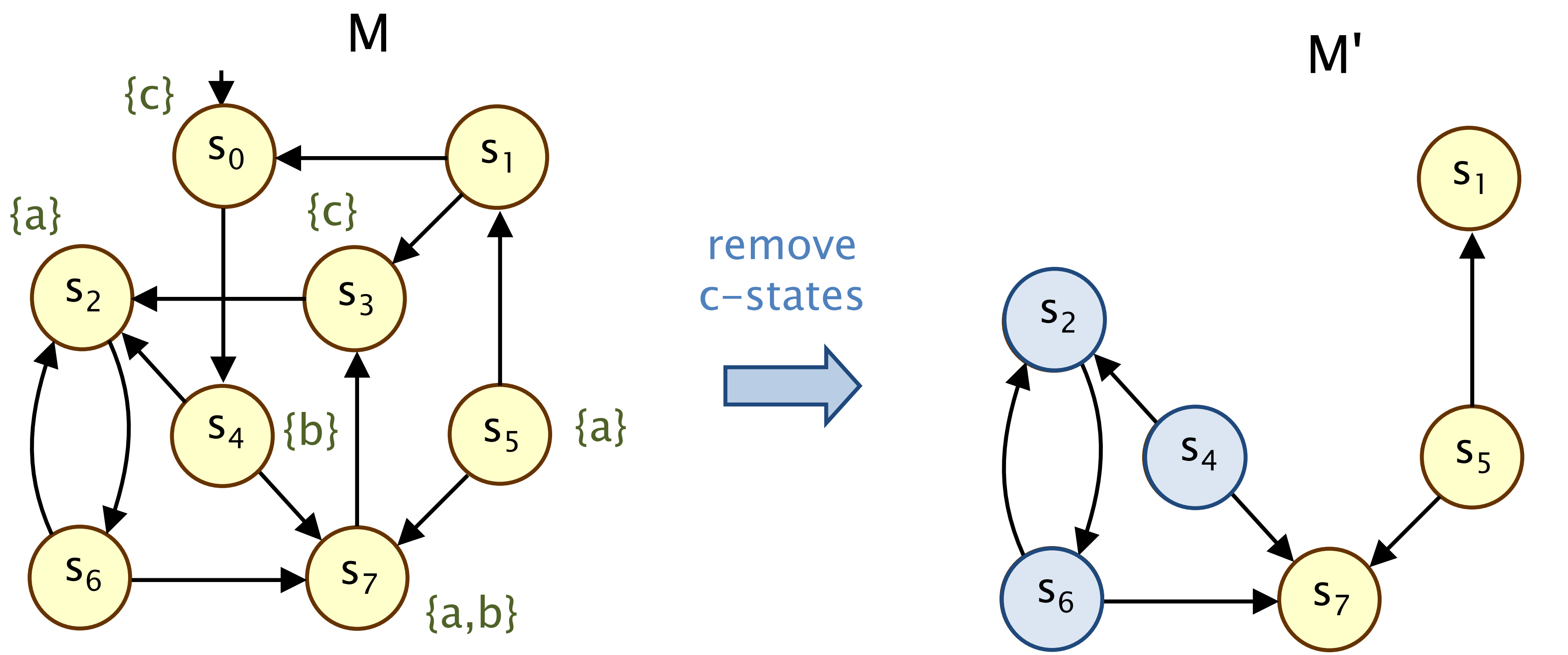
- An alternative algorithm to model check $\exists\Box\phi$ on LTS M
  - based on strongly connected components

- 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 <u>not</u> 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

Summary

- 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

- Graph-based algorithms
  - $\exists\bigcirc$ – check predecessors
  - $\exists$U, $\exists\Box$ – backwards graph traversal
  - $\exists\Box$ – also via strongly connected components

Next lecture

- Automata-based model checking

  - see Sections 4-4.2 of [BK08]