

8. CTL Model Checking



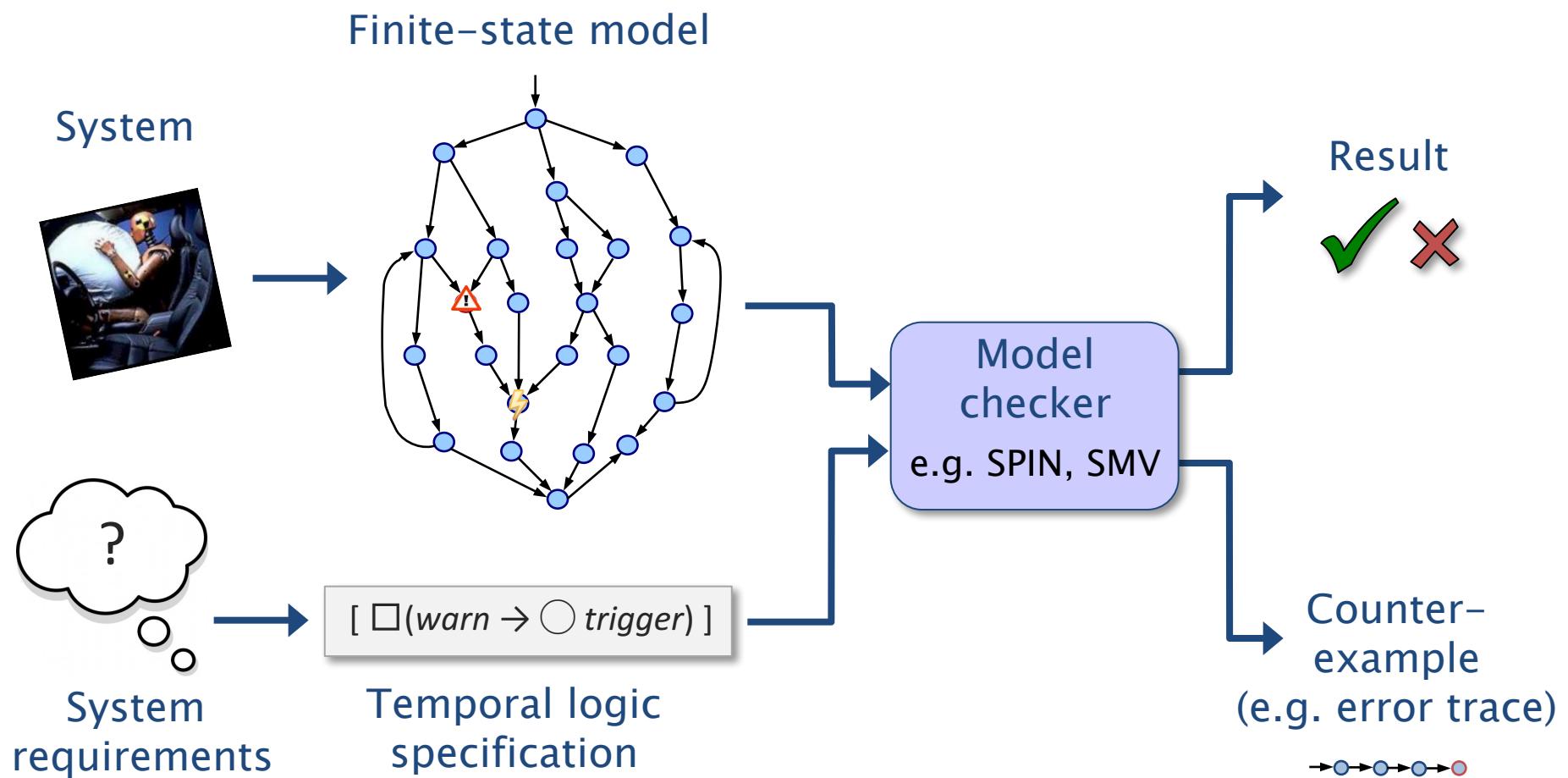
Computer-Aided Verification

Dave Parker

University of Birmingham

2016/17

Verification via model checking



Module syllabus

- Modelling sequential and parallel systems
 - labelled transition systems, linear-time properties
- Temporal logic
 - LTL, CTL and CTL*, etc.
- Model checking
 - CTL model checking
 - automata-theoretic model checking (LTL)
- Quantitative verification
 - probabilistic (and timed) systems
- Software verification
 - bounded model checking via propositional satisfiability
 - symbolic execution
- Verification software

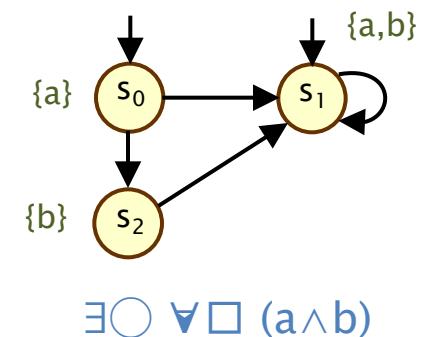
Overview

- CTL model checking
 - the model checking problem
 - basic algorithm
 - existential normal form
 - model checking $\exists U$
 - model checking $\exists \Box$
- See [BK08] Section 6.4

CTL model checking

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

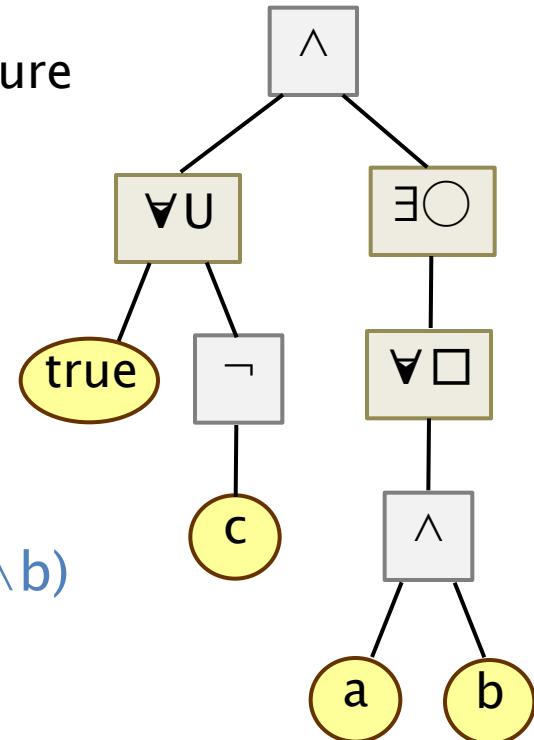
- Assumptions:
 - M is finite and has no terminal states



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

Basic algorithm

- Compute $\text{Sat}(\phi)$, then check if $I \subseteq \text{Sat}(\phi)$
 - so we in fact check if $s \models \phi$ for all states s
 - known as a **global** model checking procedure
- $\text{Sat}(\phi)$ is computed recursively
 - bottom-up traversal
 - of the **parse tree** of formula ϕ
- Example: $\phi = \forall(\text{true} U \neg c) \wedge \exists \bigcirc \forall \square (a \wedge b)$



Computing $\text{Sat}(\phi)$

- Recursive computation of $\text{Sat}(\phi)$:

- $\text{Sat}(\text{true}) = S$
- $\text{Sat}(a) = \{ s \in S \mid a \in L(s) \}$
- $\text{Sat}(\phi_1 \wedge \phi_2) = \text{Sat}(\phi_1) \cap \text{Sat}(\phi_2)$
- $\text{Sat}(\neg\phi) = S \setminus \text{Sat}(\phi)$
- $\text{Sat}(\exists \bigcirc \phi) = \dots$
- $\text{Sat}(\forall \bigcirc \phi) = \dots$
- $\text{Sat}(\exists(\phi_1 \cup \phi_2)) = \dots$
- $\text{Sat}(\forall(\phi_1 \cup \phi_2)) = \dots$
- \dots

Existential normal form (ENF)

- Syntax for CTL formula ϕ :
 - $\phi ::= \text{true} \mid a \mid \phi \wedge \phi \mid \neg\phi \mid \forall \psi \mid \exists \psi$
 - $\psi ::= \bigcirc \phi \mid \phi \cup \phi \mid \lozenge \phi \mid \square \phi$
- Existential normal form (ENF) for CTL
 - no universal path quantifier (\forall) allowed, and no $\exists \lozenge$ formulae
 - $\phi ::= \text{true} \mid a \mid \phi \wedge \phi \mid \neg\phi \mid \exists \bigcirc \phi \mid \exists(\phi \cup \phi) \mid \exists \square \phi$
- \forall can be removed using path quantifier duality:
 - $\forall \psi \equiv \neg \exists \neg \psi$
 - $\exists \psi \equiv \neg \forall \neg \psi$

Conversion to ENF

- Allowed:
 - $\exists \bigcirc, \exists U, \exists \Box$
- Not allowed:
 - $\exists \Diamond, \forall \bigcirc, \forall U, \forall \Diamond, \forall \Box$
- Can always convert to ENF:
 - $\exists \Diamond \phi \equiv \exists (\text{true} \cup \phi)$
 - $\forall \bigcirc \phi \equiv \neg \exists \bigcirc \neg \phi$
 - $\forall \Diamond \phi \equiv \neg \exists \Box \neg \phi$
 - $\forall \Box \phi \equiv \neg \exists \Diamond \neg \phi \equiv \neg \exists (\text{true} \cup \neg \phi)$
 - $\forall (\phi_1 \cup \phi_2) \equiv \neg \exists ((\neg \phi_2 \cup (\neg \phi_1 \wedge \neg \phi_2))) \wedge \neg \exists (\Box \neg \phi_2)$

Computing $\text{Sat}(\phi)$

- Recursive computation of $\text{Sat}(\phi)$:

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



Propositional logic

Immediate predecessors

Graph algorithms

Example

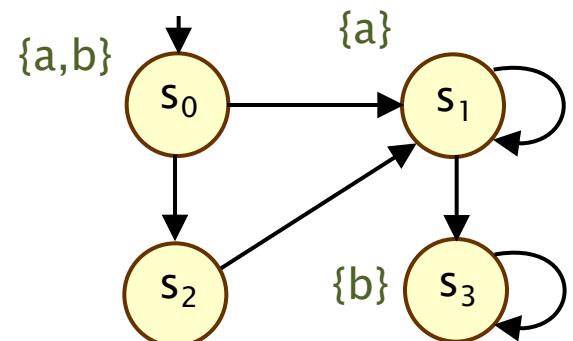
- Model the check CTL formula: $\phi = \forall \bigcirc a \wedge \exists \bigcirc \neg b$

- Convert to ENF:

- $\phi \equiv \neg \exists \bigcirc \neg a \wedge \exists \bigcirc \neg b$

- Evaluate $\text{Sat}(\phi)$ recursively

- $\text{Sat}(a) = \{s_0, s_1\}$
- $\text{Sat}(\neg a) = S \setminus \text{Sat}(a) = S \setminus \{s_0, s_1\} = \{s_2, s_3\}$
- $\text{Sat}(\exists \bigcirc \neg a) = \{s_0, s_1, s_3\}$
- $\text{Sat}(\neg \exists \bigcirc \neg a) = S \setminus \{s_0, s_1, s_3\} = \{s_2\}$
- $\text{Sat}(b) = \{s_0, s_3\}$
- $\text{Sat}(\neg b) = S \setminus \{s_0, s_3\} = \{s_1, s_2\}$
- $\text{Sat}(\exists \bigcirc \neg b) = \{s_0, s_1, s_2\}$
- $\text{Sat}(\phi) = \{s_2\} \cap \{s_0, s_1, s_2\} = \{s_2\} \Rightarrow M \not\models \phi$

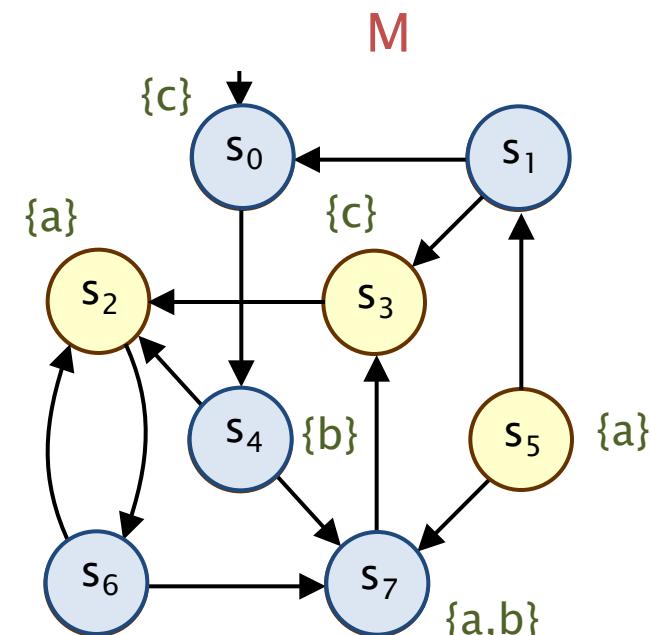


Model checking $\exists U$

- Procedure to compute $\text{Sat}(\exists(\phi_1 \cup \phi_2))$
 - given $\text{Sat}(\phi_1)$ and $\text{Sat}(\phi_2)$
- Basic idea: backwards search of the LTS from ϕ_2 -states
 - $T_0 := \text{Sat}(\phi_2)$
 - $T_i := T_{i-1} \cup \{ s \in \text{Sat}(\phi_1) \mid \text{Post}(s) \cap T_{i-1} \neq \emptyset \}$
 - until $T_i = T_{i-1}$
 - $\text{Sat}(\exists(\phi_1 \cup \phi_2)) = T_i$
- (i.e. keep adding predecessors of states in T_{i-1})
- Based on expansion law
 - $\exists(\phi_1 \cup \phi_2) \equiv \phi_2 \vee (\phi_1 \wedge \exists \bigcirc \exists(\phi_1 \cup \phi_2))$
 - (can be formulated as a fixed-point equation)

Example – $\exists U$

- Model the check CTL formula: $\phi = \exists (\neg a \cup b)$
 - $\text{Sat}(\neg a) = S \setminus \{s_2, s_5, s_7\} = \{s_0, s_1, s_3, s_4, s_6\}$
 - $\text{Sat}(b) = \{s_4, s_7\}$
- Backwards search
 - $T_0 := \text{Sat}(b) = \{s_4, s_7\}$
 - $T_1 := T_0 \cup \{s_0, s_4, s_6\} = \{s_0, s_4, s_6, s_7\}$
 - $T_2 := T_1 \cup \{s_0, s_1, s_4, s_6\} = \{s_0, s_1, s_4, s_6, s_7\}$
 - $T_3 := T_2 \cup \{s_0, s_1, s_4, s_6\} = \{s_0, s_1, s_4, s_6, s_7\}$
 - $T_3 = T_2$
 - $\text{Sat}(\phi) = \{s_0, s_1, s_4, s_6, s_7\}$
- So: $M \models \phi$



Model checking $\exists U$

- More detailed algorithm:

CheckExistsUntil(Sat(ϕ_1), Sat(ϕ_2)):

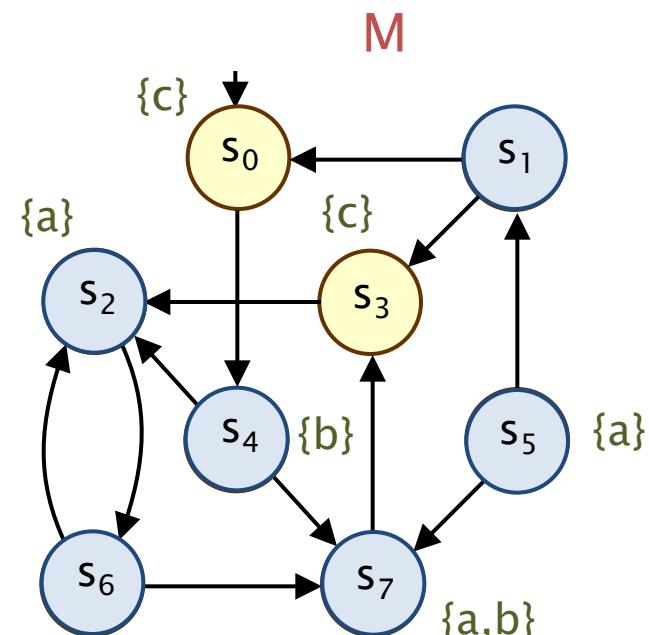
```
E := Sat( $\phi_2$ )
T := E
while (E  $\neq \emptyset$ ) do
    let s'  $\in$  E
    E := E  $\setminus$  {s'}
    for all s  $\in$  Pre(s') do
        if s  $\in$  Sat( $\phi_1$ )  $\setminus$  T then E := E  $\cup$  {s} ; T := T  $\cup$  {s} fi
    od
od
return T
```

Model checking $\exists \Box$

- Procedure to compute $\text{Sat}(\exists \Box \phi)$
 - given $\text{Sat}(\phi)$
- It again helps to consider expansion laws:
 - $\exists(\phi_1 \cup \phi_2) \equiv \phi_2 \vee (\phi_1 \wedge \exists \circ \exists(\phi_1 \cup \phi_2))$
 - $\exists \Box \phi \equiv \phi \wedge \exists \circ \exists \Box \phi$
- Basic idea: again, backwards search of the LTS
 - $T_0 := \text{Sat}(\phi)$
 - $T_i := T_{i-1} \cap \{ s \in \text{Sat}(\phi) \mid \text{Post}(s) \cap T_{i-1} \neq \emptyset \}$
 - until $T_i = T_{i-1}$
 - $\text{Sat}(\exists(\phi_1 \cup \phi_2)) = T_i$
- (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: $\forall \Diamond c \equiv \neg \exists \Box \neg c$
 - $\text{Sat}(\neg c) = S \setminus \{s_0, s_3\} = \{s_1, s_2, s_4, s_5, s_6, s_7\}$
- Backwards search
 - $T_0 := \text{Sat}(\neg c) = \{s_1, s_2, s_4, s_5, s_6, s_7\}$
 - $T_1 := T_0 \cap \{s_2, s_4, s_5, s_6\} = \{s_2, s_4, s_5, s_6\}$
 - $T_2 := T_1 \cap \{s_2, s_4, s_6\} = \{s_2, s_4, s_6\}$
 - $T_3 := T_2 \cap \{s_2, s_4, s_6\} = \{s_2, s_4, s_6\}$
 - $T_3 = T_2$
 - $\text{Sat}(\exists \Box \neg c) = \{s_2, s_4, s_6\}$
 - $\text{Sat}(\phi) = S \setminus \{s_2, s_4, s_6\} = \{s_0, s_1, s_3, s_5, s_7\}$



- So: $M \models \phi$

Model checking $\exists \Box$

- More detailed algorithm:

CheckExistsAlways(Sat(ϕ)):

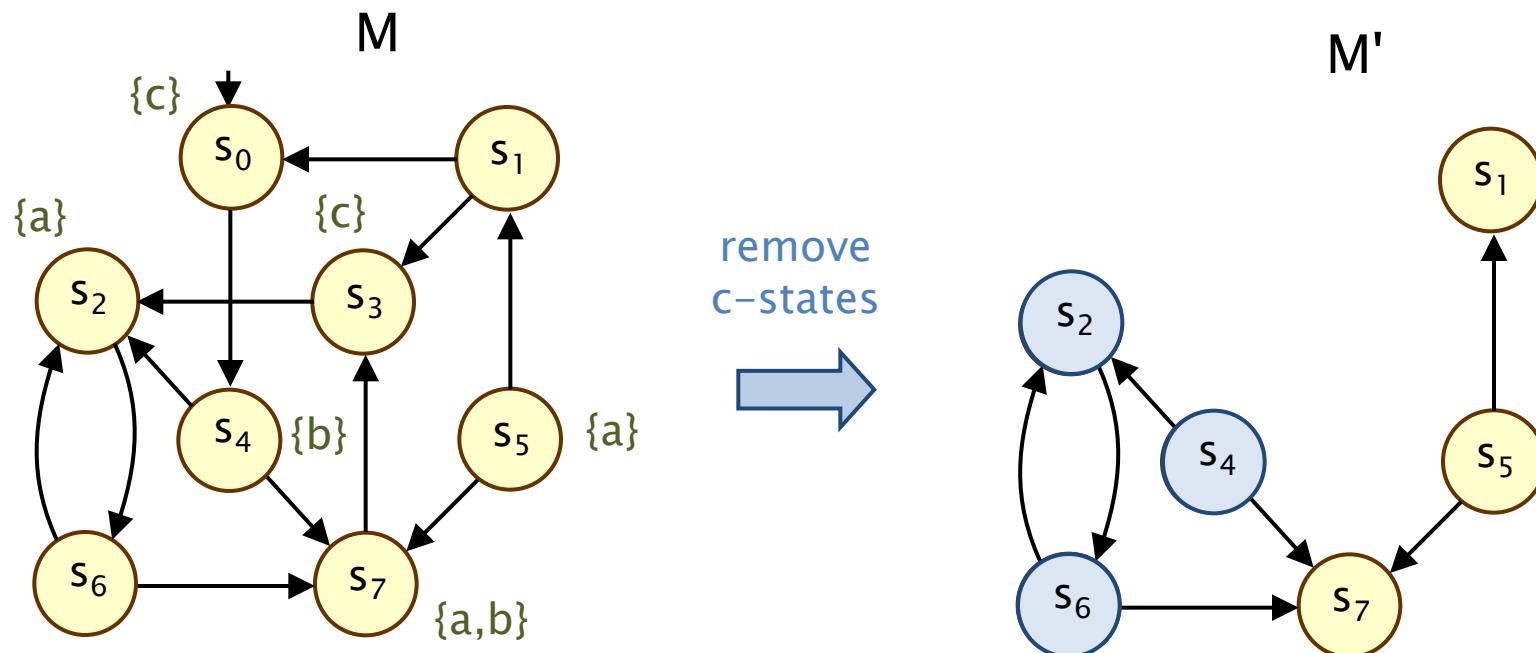
```
E := S \ Sat( $\phi$ )
T := Sat( $\phi$ )
for all s  $\in$  Sat( $\phi$ ) do count[s] := |Post(s)| od
while (E  $\neq$   $\emptyset$ ) do
    let s'  $\in$  E
    E := E \ {s'}
    for all s  $\in$  Pre(s') do
        if s  $\in$  T then
            count[s] := count[s] - 1
            if (count[s] = 0) then T := T \ {s}; E := E  $\cup$  {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 $S \setminus \text{Sat}(\phi)$
 - and removing all transitions to/from those states
 2. find the non-trivial strongly connected components (SCCs) in M'
 3. $\text{Sat}(\phi)$ is the set of states that can reach an SCC in M'

Example revisited - $\exists \Box \neg c$

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



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 $\text{Sat}(\phi)$
 - 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]