

Chapter 4

CTL

Course “Specification and Verification of Parallel Systems”

Tobias John

Autumn 2023



Section

Overview

Autumn 2023

Plan for today

- introduce CTL
- compare CTL and LTL
- basic CTL model checking algorithm
- complexity of CTL MC



Computation Tree Logic (CTL)

- logic to describe trees of computations
- introduces by Clarke and Emerson in 1981
- deciding model checking for transition systems is very efficient



Edmund M Clarke (1945–)



E. Allen Emerson (1954–)



Overview

Syntax and semantic

CTL vs. LTL

Basic MC algorithm

Sat sets

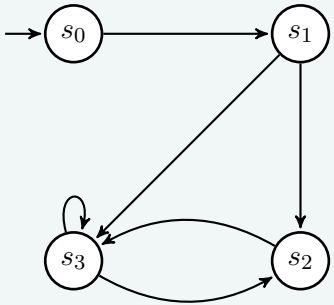
MC algorithm

Fixpoint Calculations

Complexity analysis

Unfolding a transition system to a tree

Transition system

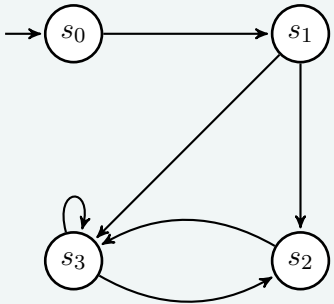


Unfolding

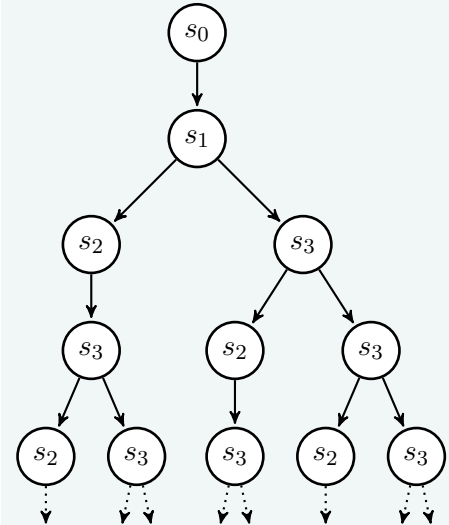


Unfolding a transition system to a tree

Transition system



Unfolding





Section

Syntax and semantic

Autumn 2023

Remember: Syntax of CTL*

Definition (CTL* formulas)

Let AP be a set of atomic propositions and $p \in \mathrm{AP}$. The set of CTL* formulas is defined as the set of all *state formulas* Φ whith

$$\Phi ::= \top \mid \bot \mid p \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \exists\varphi \mid \forall\varphi \qquad \text{state formulas}$$
$$\varphi ::= \Phi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \Diamond\varphi \mid \Box\varphi \mid \varphi\ U\ \varphi \qquad \text{path formulas}$$



Remember: Syntax of CTL*

Definition (CTL* formulas)

Let AP be a set of atomic propositions and $p \in \mathrm{AP}$. The set of CTL* formulas is defined as the set of all *state formulas* Φ whith

$$\Phi ::= \top \mid \bot \mid p \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \exists\varphi \mid \forall\varphi \qquad \text{state formulas}$$
$$\varphi ::= \Phi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \Diamond\varphi \mid \Box\varphi \mid \varphi\ U\ \varphi \qquad \text{path formulas}$$

CTL: restrict to "simple" path formulas. ($\bigcirc\Phi, \Phi\ U\ \Phi, \ldots$)



Syntax of CTL

$$\Phi ::= \top \mid p \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \forall\varphi \quad \text{state formulas}$$
$$\varphi ::= \bigcirc\Phi \mid \Phi_1 \; U \; \Phi_2 \quad \text{path formulas}$$

- note: $\Box\Phi$ and $\Diamond\Phi$ can be defined with the given operators



Example: CTL semantics

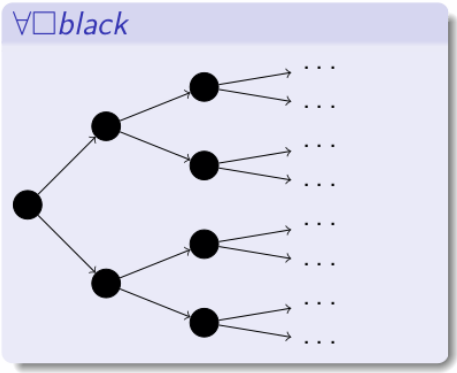
1. $\forall \Box p$
2. $\exists \Box p$
3. $\forall \Diamond p$
4. $\forall (p \ U \ q)$
5. $\exists \bigcirc \forall \Box p$
6. $\forall \Box \exists \bigcirc p$

How do computation trees look like that are models for the given formulas?

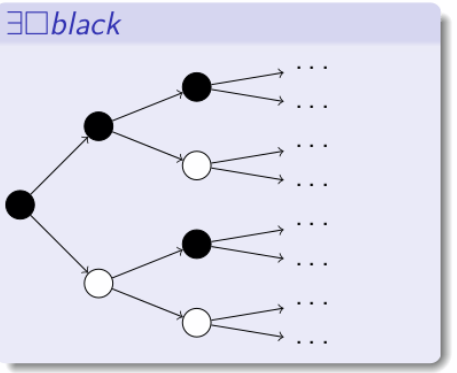


CTL semantics, example 1

$\forall\Box black$

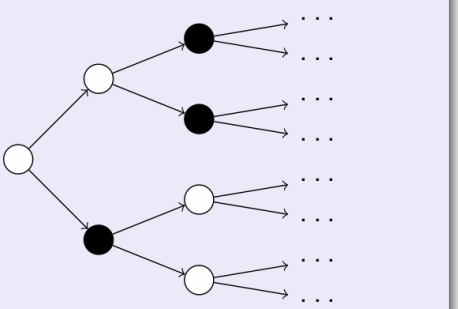


$\exists\Box black$

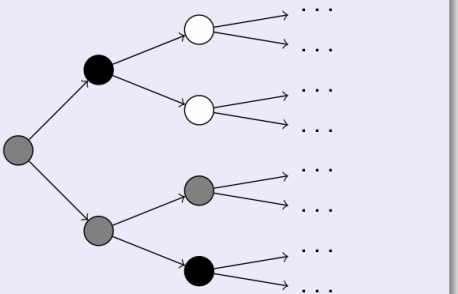


CTL semantics, example 2

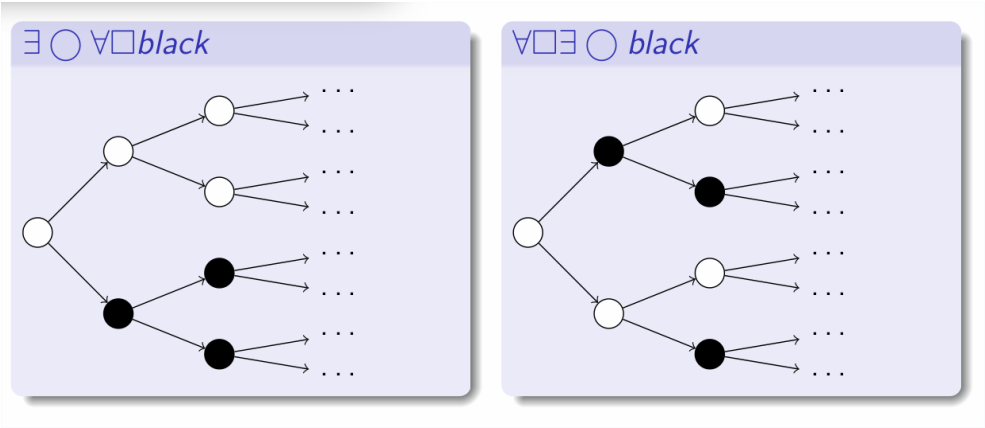
$\forall \Diamond black$



$\forall (gray \mathsf{U} black)$



CTL semantics, example 3





More Examples

- ∞ often *green*?



More Examples

- ∞ often *green*?

$$\forall\Box\forall\Diamond green$$

- Mutex?



More Examples

- ∞ often *green*?

$$\forall\Box\forall\Diamond green$$

- Mutex?

$$\forall\Box(\neg crit_1 \vee \neg crit_2)$$

- Progress (both processes reach critical section infinitely often)?



More Examples

- ∞ often *green*?

$$\forall\Box\forall\Diamond green$$

- Mutex?

$$\forall\Box(\neg crit_1 \vee \neg crit_2)$$

- Progress (both processes reach critical section infinitely often)?

$$(\forall\Box\forall\Diamond crit_1) \wedge (\forall\Box\forall\Diamond crit_2)$$

- Response (each request has eventually a response)?



More Examples

- ∞ often *green*?

$$\forall\Box\forall\Diamond green$$

- Mutex?

$$\forall\Box(\neg crit_1 \vee \neg crit_2)$$

- Progress (both processes reach critical section infinitely often)?

$$(\forall\Box\forall\Diamond crit_1) \wedge (\forall\Box\forall\Diamond crit_2)$$

- Response (each request has eventually a response)?

$$\forall\Box(request \rightarrow \forall\Diamond response)$$

- Restart (a (re-)start state is always reachable)?



More Examples

- ∞ often *green*?

$$\forall\Box\forall\Diamond green$$

- Mutex?

$$\forall\Box(\neg crit_1 \vee \neg crit_2)$$

- Progress (both processes reach critical section infinitely often)?

$$(\forall\Box\forall\Diamond crit_1) \wedge (\forall\Box\forall\Diamond crit_2)$$

- Response (each request has eventually a response)?

$$\forall\Box(request \rightarrow \forall\Diamond response)$$

- Restart (a (re-)start state is always reachable)?

$$\forall\Box\exists\Diamond start$$



Semantics of CTL formulas (analog to CTL*)

Definition ($\models$)

Let $T = (Q, \Sigma, \rightarrow, I, \mathrm{AP}, L)$ be a transition system with $q \in Q$. We define that T and q satisfy Φ ($T, q \models \Phi$) as:



Semantics of CTL formulas (analog to CTL*)

Definition ($\models$)

Let $T = (Q, \Sigma, \to, I, \mathrm{AP}, L)$ be a transition system with $q \in Q$. We define that T and q satisfy Φ ($T, q \models \Phi$) as:

$$
\begin{array}{lll}
T, q \models & \top & T, q \not\models \bot \\
T, q \models & p & \text{iff } p \in L(q) \\
T, q \models & \neg\Phi & \text{iff } T, q \not\models \Phi \\
T, q \models & \Phi_1 \wedge \Phi_2 & \text{iff } T, q \models \Phi_1 \text{ and } T, q \models \Phi_2 \\
T, q \models & \exists\varphi & \text{iff there is a path } \pi = (q, q_1, \ldots) \text{ in } T \text{ with } T, \pi \models \varphi \\
T, q \models & \forall\varphi & \text{iff } T, \pi \models \varphi \text{ for all paths } \pi = (q, q_1, \ldots) \text{ in } T \\
T, \pi \models & \bigcirc\Phi & \text{iff } T, \pi^1 \models \Phi \\
T, \pi \models & \Phi_1 \ U \ \Phi_2 & \text{iff } \exists k \in \mathbb{N}.\ (T, \pi^k \models \Phi_2 \wedge \forall 0 \leq l < k.\ T, \pi^l \models \Phi_1)
\end{array}
$$





Section

CTL vs. LTL

Autumn 2023

CTL vs. LTL

CTL and LTL are not equally expressive, as neither is more expressive than the other.



CTL vs. LTL

CTL and LTL are not equally expressive, as neither is more expressive than the other.

Theorem (CTL vs. LTL)

Let Φ be a CTL formula, and φ the LTL formula that is obtained by eliminating all path quantifiers in Φ. Then:

$$\Phi \equiv \varphi \text{ or there is no LTL formula that is equivalent to } \Phi.$$

Example

Consider $\Phi = \forall\Diamond\forall\Box p$. Is it equivalent to $\Diamond\Box p$?



CTL vs. LTL

CTL and LTL are not equally expressive, as neither is more expressive than the other.

Theorem (CTL vs. LTL)

Let Φ be a CTL formula, and φ the LTL formula that is obtained by eliminating all path quantifiers in Φ. Then:

$$\Phi \equiv \varphi \text{ or there is no LTL formula that is equivalent to } \Phi.$$

Example

Consider $\Phi = \forall\Diamond\forall\Box p$. Is it equivalent to $\Diamond\Box p$?
No.



CTL vs. LTL

CTL and LTL are not equally expressive, as neither is more expressive than the other.

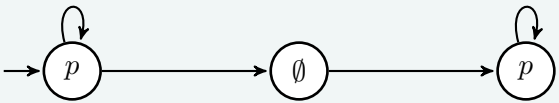
Theorem (CTL vs. LTL)

Let Φ be a CTL formula, and φ the LTL formula that is obtained by eliminating all path quantifiers in Φ. Then:

$$\Phi \equiv \varphi \text{ or there is no LTL formula that is equivalent to } \Phi.$$

Example

Consider $\Phi = \forall\Diamond\forall\Box p$. Is it equivalent to $\Diamond\Box p$?
No.





Examples: expressiveness

- there is *no* LTL formula for
  - $\forall\Diamond\forall\Box p$
  - $\forall\Box\exists\Diamond p$ (restart)



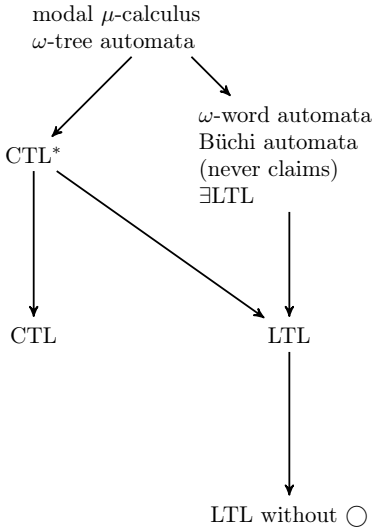
Examples: expressiveness

- there is *no* LTL formula for
  - $\forall\Diamond\forall\Box p$
  - $\forall\Box\exists\Diamond p$ (restart)

- there is *no* CTL formula for
  - $\Diamond\Box p$ (persistence)
  - $\Diamond(p \wedge \bigcirc p)$
  - $\Box\Diamond p \rightarrow \Box\Diamond q$ (strong fairness)



Expressiveness





Example: CTL vs. LTL

Let us consider the following four formulas:

CTL : $\forall \bigcirc \forall \Diamond p \quad \forall \Diamond \forall \bigcirc p$

LTL : $\bigcirc \Diamond p \quad \Diamond \bigcirc p$

Which of the CTL (LTL) formulas are equivalent?



Example: CTL vs. LTL

Let us consider the following four formulas:

CTL : $\forall \bigcirc \forall \Diamond p \quad \forall \Diamond \forall \bigcirc p$
LTL : $\bigcirc \Diamond p \quad \Diamond \bigcirc p$

Which of the CTL (LTL) formulas are equivalent?

CTL : $\forall \bigcirc \forall \Diamond p \not\equiv \forall \Diamond \forall \bigcirc p$
LTL : $\bigcirc \Diamond p \equiv \Diamond \bigcirc p$

Counter example for CTL:



Example: CTL vs. LTL

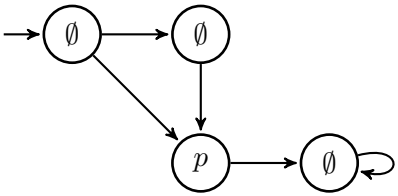
Let us consider the following four formulas:

CTL : $\forall \bigcirc \forall \Diamond p \quad \forall \Diamond \forall \bigcirc p$
LTL : $\bigcirc \Diamond p \quad \Diamond \bigcirc p$

Which of the CTL (LTL) formulas are equivalent?

CTL : $\forall \bigcirc \forall \Diamond p \not\equiv \forall \Diamond \forall \bigcirc p$
LTL : $\bigcirc \Diamond p \equiv \Diamond \bigcirc p$

Counter example for CTL:







Section

Basic MC algorithm

Autumn 2023

Sat-sets

Definition (Sat-set)

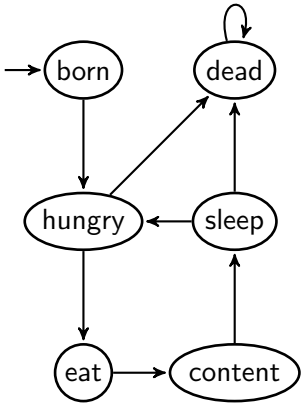
The satisfaction set $\mathrm{Sat}(\Phi)$ is the set of states in a transitions system T that satisfies Φ.

Definition

A transition system $T = (Q, \Sigma, \to, I, \mathrm{AP}, L)$ satisfies Φ, written as $T \models \Phi$, iff all the initial states of TS satisfy Φ: $I \subseteq \mathrm{Sat}(\Phi)$.

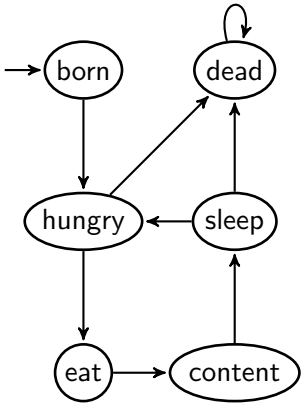


Example: Sat-sets



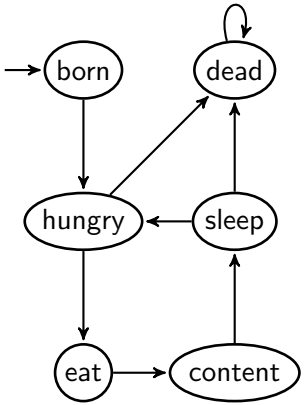
- $\forall\Diamond$dead?

Example: Sat-sets



- $\forall\Diamond$dead?
- $\mathrm{Sat}(\forall\Diamond\text{dead}) = \{\text{dead}\}$
- $\exists \bigcirc$ hungry?

Example: Sat-sets



- $\forall\Diamond$dead?
- $\mathrm{Sat}(\forall\Diamond\text{dead}) = \{\text{dead}\}$
- $\exists \bigcirc$ hungry?
- $\mathrm{Sat}(\exists \bigcirc \text{hungry}) = \{\text{born}, \text{sleep}\}$



MC algorithm: idea

- focus on ENF ($\exists\bigcirc$, $\exists\ U$, $\exists\Box$)
  - state-formulas only
  - interesting for us: existential NF

ENF

$$\Phi ::= \top \mid p \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists \bigcirc \Phi \mid \exists\Phi_1\ U\ \Phi_2 \mid \exists\Box\Phi \mid$$



MC algorithm: idea

- focus on ENF ($\exists\bigcirc$, $\exists\ U$, $\exists\Box$)
  - state-formulas only
  - interesting for us: existential NF

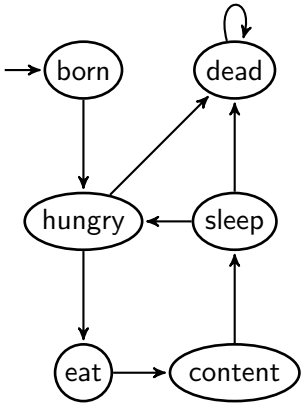
ENF

$$\Phi ::= \top \mid p \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists \bigcirc \Phi \mid \exists\Phi_1\ U\ \Phi_2 \mid \exists\Box\Phi \mid$$

- calculate $\mathrm{Sat}(\Phi)$
- check $I \subseteq \mathrm{Sat}(\Phi)$
- **recursive over structure of formula**



Example: MC algorithm

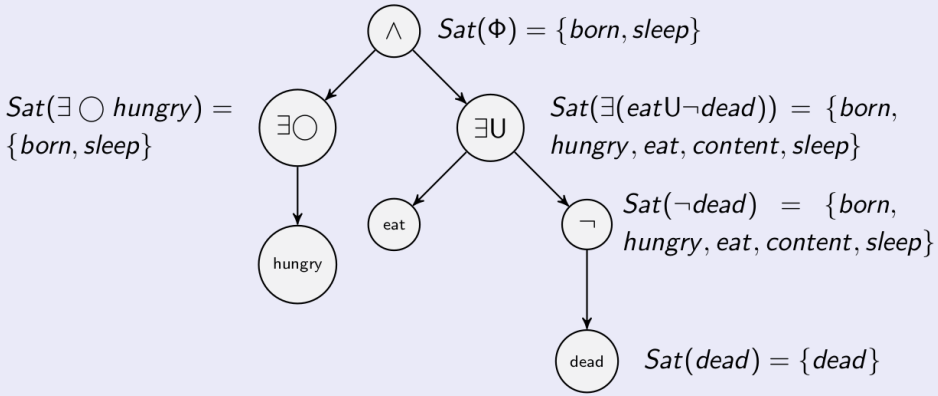


- $\Phi = \exists \bigcirc \text{hungry} \wedge \exists(\text{eat } U \neg\text{dead})$

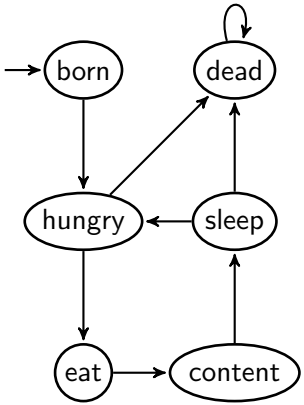
How to calculate $\mathrm{Sat}(\Phi)$ recursively?

Example: MC algorithm

$\Phi = \exists \bigcirc hungry \wedge \exists(eat \, \mathsf{U} \, \neg dead)$



Example: MC algorithm

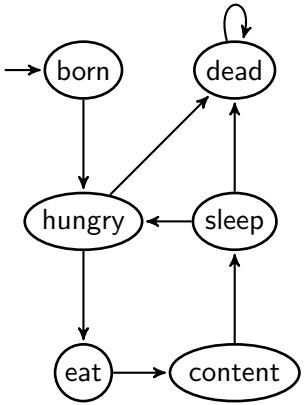


- $\Phi = \exists \bigcirc \text{hungry} \wedge \exists(\text{eat } U \neg\text{dead})$

How to calculate $\mathrm{Sat}(\Phi)$ recursively?

- $\mathrm{Sat}(\Phi) = \{\text{born}, \text{sleep}\}$

Example: MC algorithm



- $\Phi = \exists \bigcirc \text{hungry} \wedge \exists(\text{eat } U \neg\text{dead})$

How to calculate $\mathrm{Sat}(\Phi)$ recursively?

- $\mathrm{Sat}(\Phi) = \{\text{born}, \text{sleep}\}$
- $I = \{\text{born}\}$
- $I \subseteq \mathrm{Sat}(\Phi)$
- $\implies T \models \Phi$



IN5110 – Specification and Verification of Parallel Systems

Tobias John

Overview

Syntax and semantic

CTL vs. LTL

Basic MC algorithm

Sat sets

MC algorithm

Fixpoint Calculations

Complexity analysis

Recursive definition of Sat

1. $\mathrm{Sat}(\top) =$



Recursive definition of Sat

1. $\mathrm{Sat}(\top) = Q$.
2. $\mathrm{Sat}(p) =$



Recursive definition of Sat

1. $\mathrm{Sat}(\top) = Q$.
2. $\mathrm{Sat}(p) = \{s \in Q \mid p \in L(s), \text{ for any } p \in \mathrm{AP}\}$.
3. $\mathrm{Sat}(\Phi_1 \wedge \Phi_2) =$



Recursive definition of Sat

1. $\mathrm{Sat}(\top) = Q$.
2. $\mathrm{Sat}(p) = \{s \in Q \mid p \in L(s), \text{ for any } p \in \mathrm{AP}\}$.
3. $\mathrm{Sat}(\Phi_1 \wedge \Phi_2) = \mathrm{Sat}(\Phi_1) \cap \mathrm{Sat}(\Phi_2)$.
4. $\mathrm{Sat}(\neg\Phi) =$



Recursive definition of Sat

1. $\mathrm{Sat}(\top) = Q$.
2. $\mathrm{Sat}(p) = \{s \in Q \mid p \in L(s), \text{ for any } p \in \mathrm{AP}\}$.
3. $\mathrm{Sat}(\Phi_1 \wedge \Phi_2) = \mathrm{Sat}(\Phi_1) \cap \mathrm{Sat}(\Phi_2)$.
4. $\mathrm{Sat}(\neg\Phi) = Q \setminus \mathrm{Sat}(\Phi)$.
5. $\mathrm{Sat}(\exists \bigcirc \Phi) =$



Recursive definition of Sat

1. $\mathrm{Sat}(\top) = Q$.
2. $\mathrm{Sat}(p) = \{s \in Q \mid p \in L(s), \text{ for any } p \in \mathrm{AP}\}$.
3. $\mathrm{Sat}(\Phi_1 \wedge \Phi_2) = \mathrm{Sat}(\Phi_1) \cap \mathrm{Sat}(\Phi_2)$.
4. $\mathrm{Sat}(\neg\Phi) = Q \setminus \mathrm{Sat}(\Phi)$.
5. $\mathrm{Sat}(\exists \bigcirc \Phi) = \{s \in Q \mid \exists s'.s \to s' \wedge s' \in \mathrm{Sat}(\Phi)\}$
6. $\mathrm{Sat}(\exists \Box \Phi)$ is



Recursive definition of Sat

1. $\mathrm{Sat}(\top) = Q$.
2. $\mathrm{Sat}(p) = \{s \in Q \mid p \in L(s), \text{ for any } p \in \mathrm{AP}\}$.
3. $\mathrm{Sat}(\Phi_1 \wedge \Phi_2) = \mathrm{Sat}(\Phi_1) \cap \mathrm{Sat}(\Phi_2)$.
4. $\mathrm{Sat}(\neg\Phi) = Q \setminus \mathrm{Sat}(\Phi)$.
5. $\mathrm{Sat}(\exists \bigcirc \Phi) = \{s \in Q \mid \exists s'.s \to s' \wedge s' \in \mathrm{Sat}(\Phi)\}$
6. $\mathrm{Sat}(\exists\Box\Phi)$ is the *largest* subset R of Q such that
   - 6.1 $R \subseteq \mathrm{Sat}(\Phi)$ and
   - 6.2 $s \in R$ implies $\exists s' \in R.s \to s'$.
7. $\mathrm{Sat}(\exists(\Phi_1 \ U \ \Phi_2))$ is



Recursive definition of Sat

1. $\mathrm{Sat}(\top) = Q$.
2. $\mathrm{Sat}(p) = \{s \in Q \mid p \in L(s), \text{ for any } p \in \mathrm{AP}\}$.
3. $\mathrm{Sat}(\Phi_1 \wedge \Phi_2) = \mathrm{Sat}(\Phi_1) \cap \mathrm{Sat}(\Phi_2)$.
4. $\mathrm{Sat}(\neg\Phi) = Q \setminus \mathrm{Sat}(\Phi)$.
5. $\mathrm{Sat}(\exists \bigcirc \Phi) = \{s \in Q \mid \exists s'.s \to s' \wedge s' \in \mathrm{Sat}(\Phi)\}$
6. $\mathrm{Sat}(\exists\Box\Phi)$ is the *largest* subset R of Q such that
   - 6.1 $R \subseteq \mathrm{Sat}(\Phi)$ and
   - 6.2 $s \in R$ implies $\exists s' \in R.s \to s'$.
7. $\mathrm{Sat}(\exists(\Phi_1 \ U \ \Phi_2))$ is the *smallest* subset R of Q such that
   - 7.1 $\mathrm{Sat}(\Phi_2) \subseteq R$ and
   - 7.2 $s \in \mathrm{Sat}(\Phi_1)$ and $\exists s' \in R.s \to s'$ implies $s \in R$.



Basic algorithm

- recursive algorithm on structure of Φ
- $\exists\Box$ and $\exists\ U$ need special treatment (fixpoint algorithms)

```
1  input:  finite transition system T and Φ
2  output: Sat(Φ)
3  ------------------------------------------------------------
4  switch Φ
5      ⊤              => return Q
6      p              => return {s ∈ Q | p ∈ L(s)}
7      ¬Φ             => return Q \ Sat(Φ)
8      Φ1 ∧ Φ2        => return Sat(Φ1) ∩ Sat(Φ2)
9      ∃ ◯ Φ          => return {s ∈ Q | post(s) ∩ Sat(Φ) ≠ ∅}
10     ∃(Φ1 U Φ2)     => ``lfp for ∃ U''
11     ∃□Φ            => ``gfp for ∃□''
```

- Remark: $post(s) = \{s' \in Q \mid s \to s'\}$

Fixpoint algorithms (intuition)

- goal: compute largest (smallest) set of states, based on a *recursive* definition
- we start with minimal (maximal) set
- iteratively: add (remove) states from the set based on the definition
- terminate: if the set does not change anymore
- the current set is largest (smallest) solution



Fixpoint algorithm for $\exists\Box$

- Remember:
- $\mathrm{Sat}(\exists\Box\Phi)$ is the *largest* subset R of Q such that
  1. $R \subseteq \mathrm{Sat}(\Phi)$ and
  2. $s \in R$ implies $\exists s' \in R. s \to s'$.

```
1  R := Sat(Φ);
2  while {s ∈ R | post(s) ∩ R = ∅} ≠ ∅ do
3      let s ∈ {s ∈ R | post(s) ∩ R = ∅};
4      R := R \ {s};
5  od;
6  return R;
```



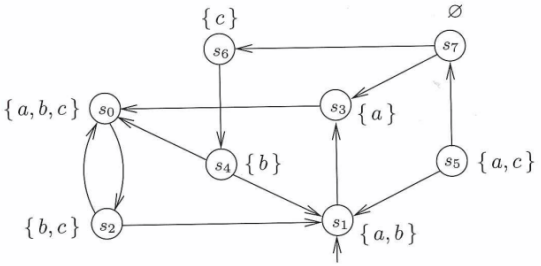
Fixpoint algorithm for $\exists\ U$

- Remember:
- $\mathrm{Sat}(\exists(\Phi_1\ U\ \Phi_2))$ is the *smallest* subset R of Q such that
  1. $\mathrm{Sat}(\Phi_2) \subseteq R$ and
  2. $s \in \mathrm{Sat}(\Phi_1)$ and $\exists s' \in R. s \to s'$ implies $s \in R$.

```
1  R := Sat(Φ2);
2  while {s ∈ Sat(Φ1) \ R | post(s) ∩ R ≠ ∅} ≠ ∅ do
3     let s ∈ {s ∈ Sat(Φ1) \ R | post(s) ∩ R ≠ ∅};
4     R := R ∪ {s};
5  od;
6  return R;
```

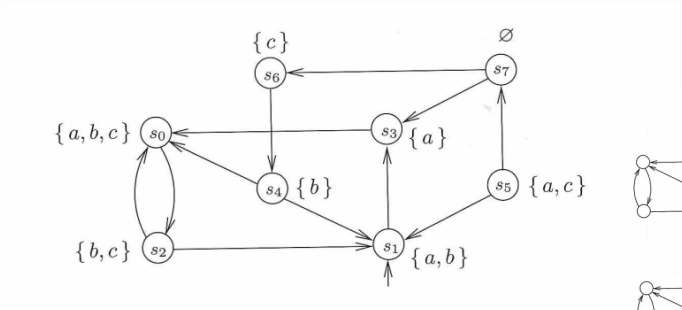


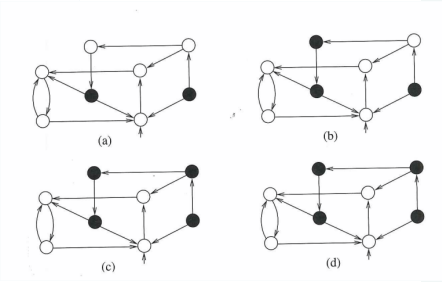
Example: $\exists(\top\ U\ (a = c) \wedge (a \neq b))$




```
1  R := Sat(Φ2);
2  while {s ∈ Sat(Φ1) \ R | post(s) ∩ R ≠ ∅} ≠ ∅ do
3     let s ∈ {s ∈ Sat(Φ1) \ R | post(s) ∩ R ≠ ∅};
4     R := R ∪ {s};
5  od;
6  return R;
```

Example: $\exists(\top\ U\ (a = c) \wedge (a \neq b))$






```
1  R := Sat(Φ2);
2  while {s ∈ Sat(Φ1) \ R | post(s) ∩ R ≠ ∅} ≠ ∅ do
3     let s ∈ {s ∈ Sat(Φ1) \ R | post(s) ∩ R ≠ ∅};
4     R := R ∪ {s};
5  od;
6  return R;
```



Section

Complexity analysis

Autumn 2023

Complexity of CTL MC

- remember: algorithm acts recursively on structure of Φ
- for each subformula: computation is



Complexity of CTL MC

- remember: algorithm acts recursively on structure of Φ
- for each subformula: computation is polynomial in size of TS
  - also for the fixpoint computations
- therefore: CTL MC is in



Complexity of CTL MC

- remember: algorithm acts recursively on structure of Φ
- for each subformula: computation is polynomial in size of TS
  - also for the fixpoint computations
- therefore: CTL MC is in P (!)
- Why should we use LTL at all?



Complexity of CTL MC

- remember: algorithm acts recursively on structure of Φ
- for each subformula: computation is polynomial in size of TS
  - also for the fixpoint computations
- therefore: CTL MC is in P (!)
- Why should we use LTL at all?
  - some properties not in CTL (e.g. strong fairness)
  - for some properties: CTL formulas are exponentially larger than equivalent LTL formulas
  - sometimes: CTL MC methods can be used to efficiently do LTL MC (if property is also in CTL)



Summary

- introduce CTL
- compare CTL and LTL
- basic CTL model checking algorithm
- complexity of CTL MC: P
- next lecture:
  - symbolic model checking
  - efficient data structures



References I

Bibliography



Overview

Syntax and semantic

CTL vs. LTL

Basic MC algorithm

Sat sets

MC algorithm

Fixpoint Calculations

Complexity analysis