Lecture 18: CTL Model Checking (cont.)
Intro to timed automata

Huan Zhang
huan@huan-zhang.com
Logistics

HW 1 graded - contact TA Sanil Arun Chawla <schawla7@illinois.edu> by **Saturday (3/24)** for regrade request

Midterm project presentation: **3/26** and **3/28**. (Next week!)

- **5-min** presentations for each team. (5% of final grade) + 2 min Q/A & Feedback
- Submit your slides on Canvas (due on **3/25**). We will compile all slides into a single file for fast switching.
- Presentation includes **problem setting**, proposed **methodology**, and **initial results**.

In addition: each person should give feedback for 3 projects that interest you most on each day. (total 6 feedbacks; count towards the **5% class participation** grades), due **3/29**

Feedback will be submitted to Canvas and also shared to the class (use the **template** on Canvas)
Review Computation tree logic (CTL)

**Unfolding** the automaton

We get a tree, representing all possible computations

A **CTL formula** allows us to specify subsets of paths in this tree
Review: CTL quantifiers

**Path quantifiers**
- E: Exists some path
- A: All paths

**Temporal operators**
- X: Next state
- U: Until ("p U q" means "p holds until q holds")
- F: Eventually (some time in future)
- G: Globally (always)
Visualizing CTL semantics

Path quantifiers
- E: Exists some path
- A: All paths

Temporal operators
- X: Next state
- U: Until
- F: Eventually
- G: Globally

$q \models ?$
Visualizing CTL semantics

$q \models EF \text{ red}$

$q \models EG \text{ red}$

$q \models A[\text{red } U \text{ green}]$

$q \models AX \text{ red}$

$q \models AF \text{ red}$

$q \models AG \text{ red}$

$q \models E[\text{red } U \text{ green}]$

$q \models EX \text{ red}$

Slides adapted from Prof. Sayan Mitra’s slides in Fall 2021
Universal CTL operators

\(X, U, G\) can be used to derive other operators

\[true \ U \ f \equiv F \ f\]

\[Gf \equiv \neg F(\neg f)\]

All combinations can be expressed using \(EX, EU, EG\)

\[
\begin{array}{cccc}
AXf & AGf & AFf & A[ f_1 U f_2 ] \\
\neg EX(\neg f) & \neg EF(\neg f) & \neg EG(\neg f) & \neg (E[\neg f_1] \neg (f_1 \lor f_2) \lor EG(\neg f_2)) \\
EX & EG & EF & EU \\
EX & EG & E(true \ U \ f) & EU
\end{array}
\]

Slides adapted from Prof. Sayan Mitra's slides in Fall 2021
Algorithm for deciding $\mathcal{A} \models f$

Since all CTL operators can be expressed by EX, EU, EG, we just need to figure out how to check these operators.

$\mathcal{A}$

CTL: e.g., $AG(a \Rightarrow AFb)$
**CheckEG**($f_1, Q, T, L$)

From $\mathcal{A}$ we construct a new automaton $\mathcal{A}' = \langle Q', T', L' \rangle$ such that

$Q' = \{ q \in Q \mid f_1 \in \text{label}(q) \}$

$T' = \{ (q_1, q_2) \in T \mid q_1 \in Q' \}$

$L' : Q' \rightarrow 2^{AP} \ \forall \ q' \in Q', L'(q') : = L(q')$

**Claim.** $q \models EGf_1$ iff

(1) $q \in Q'$

(2) $\exists \alpha \in \text{Execs}_\mathcal{A}$, with $\alpha.f\text{state} = q$ and $\alpha.l\text{state}$ is in a nontrivial *Strongly Connected Components* $C$ of the graph $\langle Q', T' \rangle$

Slides adapted from Prof. Sayan Mitra’s slides in Fall 2021
Check  EG  ! Heat

Start oven

Start oven

Open door

Open door

Open door

Open door

Close door

Close door

Close door

Close door

Reset

Warmup

Start cooking

Start cooking

Start cooking

Start cooking

Start cooking

Start cooking

Start cooking

Start cooking
Set of states in $\mathcal{A}'$ that can reach the nontrivial SCC of $\neg\text{Heat}$
Claim. $\mathcal{A}, q \models EGf_1$ iff

(1) $q \in Q'$ and

(2) $\exists \alpha \in \text{Execs}_{\mathcal{A}}$ with $\alpha.fstate = q$ and $\alpha.lstate$ is in a nontrivial SCC $C$ of the graph $\langle Q', T' \rangle$

Proof. Suppose $\mathcal{A}, q \models EGf_1$

Consider any execution $\alpha$ with $\alpha.fstate = q$. Obviously, $q \models f_1$ and so, $q \in Q'$. Since $Q$ is finite $\alpha$ can be written as $\alpha = \alpha_0 \alpha_1$ where $\alpha_0$ is finite and every state in $\alpha_1$ repeats infinitely many times.

Let $C$ be the states in $\alpha_1. C \in Q'$.

Consider any two $q_1$ and $q_2$ states in $C$, we observe that $q_1 \Leftrightarrow q_2$, and therefore $C$ is a SCC.

Consider (1) and (2). We construct a path $\alpha = \alpha_0 \alpha_1$ such that $\alpha_0.fstate = q$ and $\alpha_0 \in Q'$ and $\alpha_1$ visits some states infinitely often.
Let $Q' = \{q \in Q \mid f_1 \in \text{label}(q)\}$
Let $\mathbb{C}$ be the set of nontrivial SCCs of $\langle Q', T' \rangle$
$T = \bigcup_{C \in \mathbb{C}} \{q \mid q \in C\}$
for each $q \in T$

\[
\text{label}(q) := \text{label}(q) \cup \{EGf_1\}
\]
while $T \neq \emptyset$

\[
\text{for each } q \in T
\]

\[
T := T \setminus \{q\}
\]

\[
\text{for each } q' \in Q' \text{ such that } (q', q) \in T'
\]

\[
\text{if } EGf_1 \notin \text{label}(q') \text{ then}
\]

\[
\text{label}(q') := \text{label}(q') \cup \{EGf_1\}
\]

\[
T := T \cup \{q'\}
\]

\[
\text{Proposition. For any state } \text{label}(q) \ni EGf_1 \text{ iff } q \models EGf_1.
\]

\[
\text{Proposition. Finite } Q \text{ therefore terminates and in } O(|Q| + |T|) \text{ steps.}
\]
**CheckEU**($f_1, f_2, Q, T, L$)

Let $S = \{ q \in Q \mid f_2 \in \text{label}(q) \}$

for each $q \in S$

\[
\text{label}(q) := \text{label}(q) \cup \{ E[f_1Uf_2] \}
\]

while $S \neq \emptyset$

for each $q' \in S$

$S := S \setminus \{ q' \}$

for each $q \in T^{-1}(q')$

if $f_1 \in \text{label}(q)$ then

\[
\text{label}(q) := \text{label}(q) \cup \{ E[f_1Uf_2] \}
\]

$S := S \cup \{ q \}$

**Proposition.** For any state $\text{label}(q) \ni E[f_1Uf_2]$ iff $q \models E[f_1Uf_2]$.

**Proposition.** Finite $Q$ therefore terminates and in $O(|Q| + |T|)$ steps.

Slides adapted from Prof. Sayan Mitra’s slides in Fall 2021
Putting it all together

Explicit model checking algorithm input $\mathcal{A} \models f$?
Structural induction over CTL formula: starting with APs at lowest depth and keep updating labels for each state

\[
\begin{align*}
f &= p, & \text{for some } p \in AP, \forall q, \text{ label}(q) := \text{label}(q) \cup \{p\} \\
f &= \neg f_1 & \text{if } f_1 \notin \text{label}(q) \text{ then } \text{label}(q) := \text{label}(q) \cup f \\
f &= f_1 \land f_2 & \text{if } f_1, f_2 \in \text{label}(q) \text{ then } \text{label}(q) := \text{label}(q) \cup f \\
f &= EX f_1 & \text{if } \exists q' \in Q \text{ such that } (q, q') \in T \text{ and } f_1 \in \text{label}(q') \text{ then } \\
& \quad \quad \quad \text{label}(q) := \text{label}(q) \cup f \\
f &= E[f_1 U f_2] & \text{CheckEU}(f_1, f_2, Q, T, L) \\
f &= EG f_1 & \text{CheckEG}(f_1, Q, T, L)
\end{align*}
\]
AG (Start => AF Heat)
\[ EF (\text{Start} \land \text{EG} \land \text{Heat}) \]
\[ \text{[True EU (Start} \land \text{EG} \land \text{Heat})] \]
! EF (Start ∧ EG ! Heat)

! [True EU (Start ∧ EG ! Heat)]
Goal: \( ! \text{EF} (\text{Start } \land \text{EG} ! \text{Heat}) \)

\text{Start, ! Heat}

\text{EG ! Heat}
Goal: ! EF (Start \land EG ! Heat)
Start, ! Heat
EG ! Heat
Start \land EG ! Heat

New label added for node 2

Set of states that can reach the nontrivial SCC of ! Heat
Goal: $! EF (Start \land EG \land !Heat)$

Start, $! Heat$

$EG \land !Heat$

Start $\land EG \land !Heat$

---

Start oven

Open door

Close door

Start cooking

Warmup

Done
**Goal:** $\neg EF(Start \land \neg EG \neg Heat)$

$Start, \neg Heat$

$EG \neg Heat$

$Start \land EG \neg Heat$

$EF(Start \land EG \neg Heat)$

---

Diagram:

1. $!Start$
2. $Start$
3. $!Start$
4. $!Start$
5. $Start$
6. $Start$
7. $Start$

- **1:** !Start, Close, !Heat, !Error
- **2:** Start
- **3:** !Start, Close, !Heat, !Error
- **4:** !Start, Close, Heat, !Error
- **5:** Start, Close, !Heat, Error
- **6:** Start, Close, !Heat, !Error
- **7:** Start, Close, Heat, !Error

Actions:
- Start oven
- Open door
- Close door
- Warmup
- Start cooking
Goal: ! EF (Start ∧ EG ! Heat)

Start, ! Heat

EG ! Heat

Start ∧ EG ! Heat

EF (Start ∧ EG ! Heat)
Goal: ! EF (Start ∧ EG ! Heat)
Start, ! Heat
EG ! Heat
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

Set of states that can reach
Start ∧ EG ! Heat

1
!Start
!Close
!Heat
!Error

2
Start
!Close
!Heat
Error

3
! Start
Close
!Heat
!Error

4
!Start
Close
Heat
!Error

5
Start
Close
!Heat
Error

6
Start
Close
!Heat
!Error

7
Start
Close
Heat
!Error

Set of states that can reach
Start ∧ EG ! Heat

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)

EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)
None of the states are labeled with
! EF (Start ∨ EG ! Heat)
• We have studied hybrid automaton

**automaton** Bouncingball(c,h,g)
**variables:** x: Reals := h, v: Reals := 0
**actions:** bounce
**transitions:**
bounce
  **pre** $x = 0 \land v < 0$
  **eff** $v := -cv$
**trajectories:**
  freefall
  **evolve** $d(x) = v; d(v) = -g$
**invariant** $x \geq 0$
Next topic: Timed Automata & Reachability

- We have studied hybrid automaton
- However, verification for general hybrid automaton is in general difficult

- Special classes of hybrid automaton:
  - (Alur-Dill’s) Timed Automata
  - Rectangular initialized hybrid automata
  - Linear hybrid automata

- Verification is feasible for these classes
  - New techniques: abstraction (will be covered after ~2 weeks)
Clocks and Clock Constraints

• A clock variable $x$ is a continuous (analog) variable of type real such that along any trajectory $\tau$ of $x$, for all $t \in \tau.\ dom$, $(\tau \downarrow x)(t) = t$.

• In other words, $d(x) = 1$

• For a set $X$ of clock variables, the set $\Phi(X)$ of integral clock constraints are expressions defined by the syntax:

$$g ::= x \leq q | x \geq q | \neg g | g_1 \land g_2$$
where $x \in X$ and $q \in \mathbb{Z}$

• Examples: $x = 10; x \in [2, 5]$ are valid clock constraints

• What do clock constraints look like?
Example: “smart” light switch

**Automaton** Switch

**Variables** \( x, y: \text{Real} := 0, \ loc: \{\text{on,off}\} := \text{off} \)

**Transitions**

**Push**

\[ \text{pre } x \geq 2 \]

\[ \text{eff if } \text{loc} = \text{off} \text{ then } x,y := 0; \ \text{loc} := \text{on} \]

\[ \text{else } x := 0 \]

**Pop**

\[ \text{pre } y = 15 \land \text{loc} = \text{on} \]

\[ \text{eff } x := 0; \ \text{loc} = \text{off} \]

**Integral clock constraints**

**Trajectories**

**Invariant** \( \text{loc} = \text{off} \lor y \leq 15 \)

**Evolve** \( \text{d}(x) = 1; \ \text{d}(y) = 1 \)

**Clock variables**

**Description**
Switch can be turned on whenever at least 2 time units have elapsed since the last turn off. Switches can be turned off 15 time units after the last on.
Integral Timed Automata

• **Definition.** A integral timed automaton \( \mathcal{A} = \langle V, \Theta, A, \mathcal{D}, \mathcal{T} \rangle \) where
  - \( V = X \cup \{l\} \), where \( X \) is a set of \( n \) clocks and \( l \) is a discrete state variable of finite type \( L \)
  - \( A \) is a finite set
  - \( \mathcal{D} \) is a set of transitions such that
    - The preconditions are described by clock constraints \( \Phi(X) \)
    - \( \langle x, l \rangle \rightarrow \langle x', l' \rangle \) implies either \( x' = x \) or \( x' = 0 \) (time is reset to 0, or no change)
    - \( \mathcal{T} \) set of clock trajectories for the clock variables in \( X \)