

# Formal Specification and Verification

Temporal logic (Part 3)

7.01.2019

Viorica Sofronie-Stokkermans  
e-mail: [sofronie@uni-koblenz.de](mailto:sofronie@uni-koblenz.de)

# Branching Time Logic: CTL

---

When doing model checking, we effectively use LTL in a branching time environment:

Every state in a transition system that has more than a single successor gives rise to a “branching” in time.

This is reflected by the fact that usually, a transition system has more than a single computation.

Branching time logics allow us to explicitly talk about such branches in time.

# CTL: Syntax

---

The class of computational tree logic (CTL) formulas is the smallest set such that

- $\top$ ,  $\perp$  and each propositional variable  $P \in \Pi$  are formulae;
- if  $F, G$  are formulae, then so are  $F \wedge G, F \vee G, \neg F$ ;
- if  $F, G$  are formulae, then so are  
 $A \circ F$  and  $E \circ F$ ,  
 $A(FUG)$  and  $E(FUG)$ .

The symbols  $A$  and  $E$  are called path quantifiers.

# Abbreviations

---

Apart from the Boolean abbreviations, we use:

$A \diamond F$  for  $A(\top \mathcal{U} F)$

$E \diamond F$  for  $E(\top \mathcal{U} F)$

$A \square F$  for  $\neg E \diamond \neg F$

$E \square F$  for  $\neg A \diamond \neg F$

Note that formulas such as  $E(\square q \wedge \diamond p)$  are not CTL formulas.

# CTL: Semantics

---

Let  $T = (S, \rightarrow, L)$  be a transition system. We define satisfaction of CTL formulas in  $T$  at states  $s \in S$  as follows:

$(T, s) \models p$	iff	$p \in L(s)$
$(T, s) \models \neg F$	iff	$(T, s) \models F$ is not the case
$(T, s) \models F \wedge G$	iff	$(T, s) \models F$ and $(T, s) \models G$
$(T, s) \models F \vee G$	iff	$(T, s) \models F$ or $(T, s) \models G$
$(T, s) \models E \circ F$	iff	$(T, t) \models F$ for some $t \in S$ with $s \rightarrow t$
$(T, s) \models A \circ F$	iff	$(T, t) \models F$ for all $t \in S$ with $s \rightarrow t$
$(T, s) \models A(FUG)$	iff	for all computations $\pi = s_0 s_1 \dots$ of $T$ with $s_0 = s$ , there is an $m \geq 0$ such that $(T, s_m) \models G$ and $(T, s_k) \models F$ for all $k < m$
$(T, s) \models E(FUG)$	iff	there exists a computation $\pi = s_0 s_1 \dots$ of $T$ with $s_0 = s$ , such that there is an $m \geq 0$ such that $(T, s_m) \models G$ and $(T, s_k) \models F$ for all $k < m$

# Example of formulae in CTL

---

- $E\Diamond((A = 2) \wedge (B = 2))$

It is possible to reach a state where both processes are in the critical section.

- $A\Box(\text{enabled}_1 \vee \dots \vee \text{enabled}_k)$

freedom from deadlocks (a safety property);

- $A\Box(\text{req} \rightarrow A\Diamond\text{grant})$

every request will eventually be acknowledged (a liveness property);

- $A\Box(A\Diamond\text{enabled}_i)$

process  $i$  is enabled infinitely often on every computation path (unconditional fairness)

- $A\Box(E\Diamond\text{Restart})$

from every state it is possible to get to a restart state

# Equivalence

---

We say that two CTL formulas  $F$  and  $G$  are (globally) equivalent (written  $F \equiv G$ )

if, for all CTL structures  $T = (S, \rightarrow, L)$  and  $s \in S$ , we have

$$T, s \models F \text{ iff } T, s \models G.$$

# Equivalence

---

We say that two CTL formulas  $F$  and  $G$  are (globally) equivalent (written  $F \equiv G$ )

if, for all CTL structures  $T = (S, \rightarrow, L)$  and  $s \in S$ , we have

$$T, s \models F \text{ iff } T, s \models G.$$

## Examples:

$$\neg A \diamond F \equiv E \square \neg F$$

$$\neg E \diamond F \equiv A \square \neg F$$

$$\neg A \bigcirc F \equiv E \bigcirc \neg F$$

$$A \diamond F \equiv A[\top U F]$$

$$E \diamond F \equiv E[\top U F]$$



# CTL

---

Why is CTL called a tree logic?

Intuitively, it can talk about branching paths (which exists in a tree), but not about joining path (which do not exist in a

Let  $T = (S, \rightarrow, L)$  be a transition system.

We define a tree-shaped transition systems  $Tree(T) = (S', \rightarrow', L')$  as follows:

- $S'$  is the set of all finite computations of  $T$ , i.e.,  
 $S' = \{s_0 \dots s_k \mid s_i \rightarrow s_{i+1} \text{ for all } i < k\}$ ;
- $\rightarrow' = \{(\pi, \pi') \in S' \times S' \mid \pi = qs, \pi' = \pi s' \text{ for some } s, s' \in S \text{ with } s \rightarrow s'\}$ ;
- $(P \in L'(\pi) \text{ iff } P \in L(s))$  if  $\pi = s\pi'$  for some  $\pi' \in \{\epsilon\} \cup S'$  and  $s \in S$ .

$Tree(T)$  is called the unravelling of  $T$ . Observe that  $Tree(T)$  has no leaves because of the assumption that we have no deadlocks in  $T$ .

# CTL

---

CTL formulas cannot distinguish between a state in a Kripke structure and the corresponding states in the tree-shaped unravelling.

**Lemma** Let  $T$  be a transition system,  $s$  a state of  $T$ ,  $\pi = s_0 \dots s_k$  a state of  $Tree(T)$  such that  $s_k = s$ , and  $F$  a CTL formula.

Then  $(T, s) \models F$  iff  $(Tree(T), \pi) \models F$ .

**Proof.** By induction on the structure of  $F$ .

# CTL\*

---

CTL\* is a logic which combines the expressive powers of LTL and CTL, by dropping the CTL constraint that every temporal operator ( $\bigcirc, \mathcal{U}, \square, \diamond$ ) has to be associated with a unique path quantifier ( $A, E$ ).

# CTL vs LTL

---

We want to compare the expressive power of LTL and CTL.

To do this, we give a branching time reading to LTL formulas that is inspired by our interpretation of LTL formulas in model checking:

we view LTL formulas as implicitly universally quantified.

(in LTL we consider all paths)

LTL formula  $F \mapsto$  CTL\* formula  $AF$

CTL is also a subset of CTL\*, since it is the fragment of CTL\* in which path quantifiers can only be applied to formulae starting with  $\bigcirc, \mathcal{U}, \square, \diamond$ .

# CTL vs LTL

---

**Definition.** We call two CTL\* formulas  $F$  and  $G$  equivalent if, for all transition systems  $T$  and states  $s$  of  $T$ , we have  $(T, s) \models F$  iff  $(T, s) \models G$ .

# CTL vs LTL

---

**Definition.** We call two CTL\* formulas  $F$  and  $G$  equivalent if, for all transition systems  $T$  and states  $s$  of  $T$ , we have  $(T, s) \models F$  iff  $(T, s) \models G$ .

Some (but not all) LTL formulas can be converted into CTL formulas by adding an  $A$  to each temporal operator.

**Theorem.** There exists formulae in LTL which cannot be expressed in CTL and vice-versa.

- In CTL but not in LTL:  $A\Box E\Diamond F$

This expresses: wherever we have got to, we can always get to a state in which  $F$  is true.

This is also useful, e.g., in finding deadlocks in protocols.

- In LTL but not in CTL:  $A[\Box\Diamond p \rightarrow \Diamond q]$

“If there are infinitely many  $p$  along the path, then there is an occurrence of  $q$ .”

This is an interesting thing to be able to say; for example, many fairness constraints are of the form “infinitely often requested implies eventually acknowledged”.

# Model Checking

---

The CTL model checking problem is as follows:

Given a transition system  $T = (S, \rightarrow, L)$  and a CTL formula  $F$ , check whether  $T$  satisfies  $F$ , i.e., whether  $(T, s) \models F$  for all  $s \in S$ .

# Model Checking

---

The CTL model checking problem is as follows:

Given a transition system  $T = (S, \rightarrow, L)$  and a CTL formula  $F$ , check whether  $T$  satisfies  $F$ , i.e., whether  $(T, s) \models F$  for all  $s \in S$ .

## Method (Idea)

- (1) Arrange all subformulas  $F_i$  of  $F$  in a sequence  $F_0, \dots, F_k$  in ascending order w.r.t. formula length: for  $1 \leq i < j \leq k$ ,  $F_i$  is not longer than  $F_j$ ;
- (2) For all subformulas  $F_i$  of  $F$ , compute the set

$$\text{sat}(F_i) := \{s \in S \mid (T, s) \models F_i\}$$

in this order (from shorter to longer formulae);

- (3) Check whether  $S \subseteq \text{sat}(F)$ .



# Model Checking

---

How to compute  $sat(F_i)$

- $p \in \Pi \mapsto sat(p) = \{s \mid L(p, s) = 1\}$
- $sat(\neg F) = S \setminus sat(F)$
- $sat(F \wedge G) = sat(F) \cap sat(G)$
- $sat(F \vee G) = sat(F) \cup sat(G)$
- $sat(E \circ F) = \{s \mid \exists t \in S : (s \rightarrow t) \wedge t \in sat(F)\}$
- $sat(A \circ F) = \{s \mid \forall t \in S : (s \rightarrow t) \text{ implies } t \in sat(F)\}$
- $sat(E(FUG))$  and  $sat(A(FUG))$  are computed as explained in what follows.

# Model Checking

---

**Lemma.**  $\text{sat}(E(F\mathcal{U}G))$  is the smallest set  $T$  with

(1)  $\text{sat}(G) \subseteq T$

(2)  $s \in \text{sat}(F)$  and  $\text{Post}(s) \cap T \neq \emptyset$  implies  $s \in T$

**Proof: 1. Show that  $T = \text{sat}(E(F\mathcal{U}G))$  satisfies (1) and (2).**

This follows from the fact that

$$E(F\mathcal{U}G) = G \vee (F \wedge E \circ E(F\mathcal{U}G)).$$

(1)  $\text{sat}(G) \subseteq T$

(2)  $s \in \text{sat}(F)$  and  $\text{Post}(s) \cap T \neq \emptyset$  implies  $s \in T$

# Model Checking

---

**Lemma.**  $\text{sat}(E(F\mathcal{U}G))$  is the smallest set  $T$  with

(1)  $\text{sat}(G) \subseteq T$

(2)  $s \in \text{sat}(F)$  and  $\text{Post}(s) \cap T \neq \emptyset$  implies  $s \in T$

**Proof: 2. Show that for any  $T$  satisfying (1) and (2),  $\text{sat}(E(F\mathcal{U}G)) \subseteq T$**

Let  $s \in \text{sat}(E(F\mathcal{U}G))$

**Case 1:**  $s \in \text{sat}(G)$ . Then by (1),  $s \in T$ .

**Case 2:**  $s \notin \text{sat}(G)$ .

Then there exists a path  $\pi = s_0 \dots s_k \dots$  with  $s_0 = s$  such that  $\pi \models F\mathcal{U}G$ .

Let  $n \geq 0$  such that

$$s_i \models F \text{ for } 0 \leq i \leq n$$

$$s_{n+1} \models G.$$

# Model checking

---

Proof: 2. Show that for any  $T$  satisfying (1) and (2),  $\text{sat}(E(FUG)) \subseteq T$

....continued

Then

$s_{n+1} \in \text{sat}(G) \in T$ ,

$s_n \in \text{sat}(F)$  and  $s_{n+1} \in \text{Post}(s_n) \cap T$ , so  $s_n \in T$ .

$s_{n-1} \in \text{sat}(F)$  and  $s_n \in \text{Post}(s_{n-1}) \cap T$ , so  $s_{n-1} \in T$ .

...

$s_0 = s \in \text{sat}(F)$  and  $s_1 \in \text{Post}(s_0) \cap T$ , so  $s_0 = s \in T$ .

# Model checking

---

**Remark:**  $E(FUG)$  is a fixpoint of the equation  $\Phi \equiv G \vee (F \wedge E \circ \Phi)$ .

Since  $\text{sat}(E(FUG))$  is the smallest set  $T$  with

(1)  $\text{sat}(G) \subseteq T$

(2)  $s \in \text{sat}(F)$  and  $\text{Post}(s) \cap T \neq \emptyset$  implies  $s \in T$

it can be computed iteratively as follows:

$$T_0 := \text{sat}(G)$$

$$T_{i+1} := T_i \cup \{s \in \text{sat}(F) \mid \text{Post}(s) \cap T_i \neq \emptyset\}$$

Then:  $T_0 \subseteq T_1 \subseteq \dots \subseteq T_j \subseteq T_{j+1} \subseteq \dots \subseteq \text{sat}(E(FUG))$ .

Since  $S$  is finite, there exists  $j$  such that  $T_j = T_{j+1} = \dots$

This  $T_j$  will be  $\text{sat}(E(FUG))$ .