Formal Specification and Verification

Temporal logic (Part 4)

1.07.2014

Viorica Sofronie-Stokkermans

Markus Bender

e-mail: 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

- $\top, \bot$ 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
  $A \bigcirc F$ and $E \bigcirc F$,
  $A(F \mathcal{U} G)$ and $E(F \mathcal{U} G)$.

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

$A\Box F$ for $\neg E\Diamond\neg F$

$E\Box F$ for $\neg A\Diamond\neg F$

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

CTL: Semantics

Let $T = (S, \to, L)$ 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 $(T, s) \models G$
$(T, s) \models E \bigcirc F$	iff	$(T, t) \models F$ for some $t \in S$ with $s \to t$
$(T, s) \models A \bigcirc F$	iff	$(T, t) \models F$ for all $t \in S$ with $s \to t$
$(T, s) \models A(F \mathcal{U} G)$	iff	for all computations $\pi = s_0 s_1 \ldots$ 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 $k < m$
$(T, s) \models E(F \mathcal{U} G)$	iff	there exists a computation $\pi = s_0 s_1 \ldots$ 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.
- $A\Box(\text{enabled}_1 \wedge \ldots \text{enabled}_k)$
  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 $T = (S, \to, L)$ and $s \in S$, we have

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

Examples:

$\neg A \Diamond F \equiv E \Box \neg F$

$\neg E \Diamond F \equiv A \Box \neg F$

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

$A \Diamond F \equiv A[\top \mathcal{U} F]$

$E \Diamond F \equiv E[\top \mathcal{U} F]$

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 tree).

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., $S' = \{s_0 \ldots s_k \mid s_i \rightarrow s_{i+1} \text{ for all } i < k\}$;
- $\rightarrow' = \{(\pi, \pi') \in S' \times S' \mid \pi = qs, \pi' = \pi s' \text{ for some } s, s' \in S \text{ with } s \rightarrow s'\}$;
- $(P \in L'(\pi) \text{ iff } P \in L(s))$ 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 ($\bigcirc, \mathcal{U}, \Box, \Diamond$) 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 $\bigcirc, \mathcal{U}, \Box, \Diamond$.

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: $A[\Box\Diamond p \to \Diamond q]$

  "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 satifies 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 satifies 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 $F_0, \ldots F_k$ 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

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

in this order (from shorter to longer formulae);

(3) Check whether $S \subseteq sat(F)$.

Model Checking

How to compute $sat(F_i)$

- $p \in \Pi \mapsto sat(p) = \{s \mid L(p, s) = 1\}$
- $sat(\neg F_i) = S \backslash sat(F_i)$
- $sat(F_i \wedge F_j) = sat(F_i) \cap sat(F_j)$
- $sat(F_i \vee F_j) = sat(F_i) \cup sat(F_j)$
- $sat(E \bigcirc F_i) = \{s \mid \exists t \in S : (s \rightarrow t) \wedge t \in sat(F_i)\}$
- $sat(A \bigcirc F_i) = \{s \mid \forall t \in S : (s \rightarrow t) \wedge t \in sat(F_i)\}$
- $sat(E(F_i \mathcal{U} F_j))$ and $sat(A(F_i \mathcal{U} F_j)$ are computed with the following procedures: