15-414: Bug Catching: Automated Program Verification

Lecture Notes on Branching-Time Properties

Matt Fredrikson

Carnegie Mellon University
Lecture 21

1 Introduction

Linear temporal logic is a very important logic for model checking [Eme90, CGP99, BKL08] but has the downside that its verification algorithms are rather complex. Additionally, the formulas of LTL can only describe a particular type of temporal property over individual traces, with the assumption that an entire system satisfies an LTL formula if and only if *all* its traces satisfy the formula. It might be useful in some situations to describe properties that quantify over possible traces of a system, both universally and existentially, to account for the fact that computations often branch among many possible paths during execution.

Today we will discuss Computation Tree Logic (CTL), another temporal logic that is suited to the goals described above. CTL formulas describe properties that switch to a new trace every time a temporal operator is used, requiring either the existence of a satisfying trace, or universal satisfaction among all possible traces, depending on a *path quantifier*. CTL has the advantage of having a pretty simple model checking algorithm, and it describes an incomparable set of properties from LTL.

2 Review

Recall the definition of a computation structure, which is a transition system describing a set of infinite traces over states labeled with atomic propositions.

Definition 1 (Computation structure)**.** A Kripke structure $K = (W, \curvearrowright, v)$ is called a *computation structure* if W is a finite set of states and every element $s \in W$ has at least one direct successor $t \in W$ with $s \curvearrowright t$. A (computation) *path* in a computation structure is an infinite sequence $s_0, s_1, s_2, s_3, \ldots$ of states $s_i \in W$ such that $s_i \curvearrowright s_{i+1}$ for all i. The

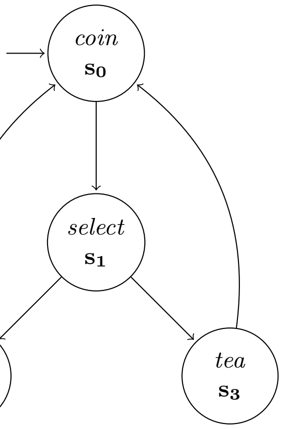


Figure 1: Computation structure describing the operation of a vending machine.

mapping v labels each state with a set of atomic propositions, or basic formulas that are true in the state.

Figure 1 shows an example of a computation structure for a simple beverage vending machine. We saw how to use LTL formulas to describe useful properties of computation structures, so that all of their traces satisfy a given formula (Definitions 3 and 3).

Definition 2 (LTL semantics (traces))**.** The truth of LTL formulas in a trace σ is defined inductively as follows:

1. $\sigma \models F$ iff $\sigma_0 \models F$ for state formula F provided that $\sigma_0 \neq \Lambda$
2. $\sigma \models \neg P$ iff $\sigma \not\models P$, i.e. it is not the case that $\sigma \models P$
3. $\sigma \models P \wedge Q$ iff $\sigma \models P$ and $\sigma \models Q$
4. $\sigma \models \mathbf{X}P$ iff $\sigma^1 \models P$
5. $\sigma \models \Box P$ iff $\sigma^i \models P$ for all $i \geq 0$
6. $\sigma \models \Diamond P$ iff $\sigma^i \models P$ for some $i \geq 0$
7. $\sigma \models P\mathbf{U}Q$ iff there is an $i \geq 0$ such that $\sigma^i \models Q$ and $\sigma^j \models P$ for all $0 \leq j < i$

In all cases, the truth-value of a formula is, of course, only defined if the respective suffixes of the traces are defined.

Definition 3 (LTL semantics (computation structure))**.** Given an LTL formula P and computation structure $K = (W, \curvearrowright, v)$, $K \models P$ if and only if $\sigma \models P$ for all σ where $\sigma_i = v(s_i)$ for some path $s_0, s_1, s_2, \ldots$ in K.

3 Computation Tree Logic

We have seen that LTL universally quantifies over paths in a computation. However, we may also want to quantify existentially over paths, requiring that at least one way of executing the computation satisfies a given property. CTL incorporates this by introducing *path quantifiers* **E** (existential) and **A** (universal). Path quantifiers are evaluated in a particular state of a transition structure, and so we refer to formulas constructed from them as *state formulas*.

- $\mathbf{E}P$ is a state formula where for a given Kripke structure K we have the following:

$$K, s \models \mathbf{E}P \leftrightarrow \text{ there } \textbf{exists} \text{ a path } \pi \text{ starting at } s \text{ where } \pi \models P$$

- $\mathbf{A}P$ is a state formula where for a given Kripke structure K we have the following:

$$K, s \models \mathbf{A}P \leftrightarrow \text{ for } \textbf{all} \text{ paths } \pi \text{ starting at } s, \pi \models P$$

The temporal operators for next, future, globally, and until are **always** paired with a path quantifier in CTL. It is also conventional to use the letter **G** in place of $\Box$, and **F** in place of $\Diamond$, when writing CTL formulas. The semantics of the logic is shown in Definition 4.

Definition 4. In a fixed computation structure $K = (W, \curvearrowright, v)$, the truth of CTL formulas in state s is defined inductively as follows:

1. $s \models p$ iff $v(s)(p) = \mathit{true}$ for atomic propositions p
2. $s \models \neg P$ iff $s \not\models P$, i.e. it is not the case that $s \models P$
3. $s \models P \land Q$ iff $s \models P$ and $s \models Q$
4. $s \models \mathbf{AX}P$ iff all successors t with $s \curvearrowright t$ satisfy $t \models P$
5. $s \models \mathbf{EX}P$ iff at least one successor t with $s \curvearrowright t$ satisfies $t \models P$
6. $s \models \mathbf{AG}P$ iff all paths $s_0, s_1, s_2, \ldots$ starting in $s_0 = s$ satisfy $s_i \models P$ for all $i \geq 0$
7. $s \models \mathbf{AF}P$ iff all paths $s_0, s_1, s_2, \ldots$ starting in $s_0 = s$ satisfy $s_i \models P$ for some $i \geq 0$
8. $s \models \mathbf{EG}P$ iff some path $s_0, s_1, s_2, \ldots$ starting in $s_0 = s$ satisfies $s_i \models P$ for all $i \geq 0$
9. $s \models \mathbf{EF}P$ iff some path $s_0, s_1, s_2, \ldots$ starting in $s_0 = s$ satisfies $s_i \models P$ for some $i \geq 0$
10. $s \models \mathbf{A}P\mathbf{U}Q$ iff all paths $s_0, s_1, s_2, \ldots$ starting in $s_0 = s$ have some $i \geq 0$ such that $s_i \models Q$ and $s_j \models P$ for all $0 \leq j < i$
11. $s \models \mathbf{E}P\mathbf{U}Q$ iff some path $s_0, s_1, s_2, \ldots$ starting in $s_0 = s$ has some $i \geq 0$ such that $s_i \models Q$ and $s_j \models P$ for all $0 \leq j < i$

While LTL formulas describe linear-time properties (single paths), CTL formulas describe branching-time properties and can describe multiple possible futures. We can visualize LTL formulas as a sequence of states in a single line where CTL corresponds to a transition of states in a tree. Figure 2 shows the visualization of the LTL formula $P\mathbf{U}Q$, whereas Figure 3 shows the visualization of the CTL formula $\mathbf{A}[P\mathbf{U}Q]$.

3.1 Useful equivalences

Some of the CTL formulas are redundant in the sense that they are definable with other CTL formulas already. But the meaning of the original formulas is usually much easier to understand than the meaning of its equivalent.

Lemma 5. *The following are valid CTL equivalences:*

1. $\mathbf{EF}P \leftrightarrow \mathbf{E}[\mathit{true}\mathbf{U}P]$
2. $\mathbf{AF}P \leftrightarrow \mathbf{A}[\mathit{true}\mathbf{U}P]$
3. $\mathbf{EG}P \leftrightarrow \neg\mathbf{AF}\neg P$
4. $\mathbf{AG}P \leftrightarrow \neg\mathbf{EF}\neg P$
5. $\mathbf{AX}P \leftrightarrow \neg\mathbf{EX}\neg P$
6. $\mathbf{A}[P\mathbf{U}Q] \leftrightarrow \neg\mathbf{E}[\neg Q\mathbf{U}(\neg P \wedge \neg Q)] \wedge \neg\mathbf{EG}\neg Q$

Most of these cases except the last are quite easy to prove. So as not to confuse ourselves, we will definitely make use of the finally and globally operators in applications. But thanks to these equivalences, when developing reasoning techniques we can simply pretend next and until would be the only temporal operators to worry about. In fact, we can even pretend only the existential path quantifier $\mathbf{E}$ is used, never the universal path quantifier $\mathbf{A}$, but this reduction in the number of different operators comes at quite some expense in the size and complexity in the resulting formulas.

3.2 Comparison with LTL

LTL and CTL are incomparable in terms of expressiveness. There are formulas in both logics that cannot be expressed in the other. Consider the LTL formula $\Diamond\Box P$, which is satisfied by the automaton in Figure 5. What is the equivalent CTL formula? One idea is to take the fact that LTL formulas "implicitly" have universal path quantifiers before every temporal operator, and propose the formula $\mathbf{AFAG}P$. However, this formula illustrates that the "quantifiers" that our intuition might tell us are implicit in LTL do not mean the same thing as the semantics of CTL, because this automaton does not satisfy $\mathbf{AFAG}P$. Note that we can visualize the unrolling of Figure 5 in Figure 6, where we can see that there is a run in which the system will always be in the state from which a run finally goes in a non P state. This corresponds to staying in the initial state of

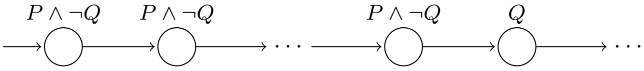


Figure 2: Visualization of a LTL formula: $P\mathbf{U}Q$

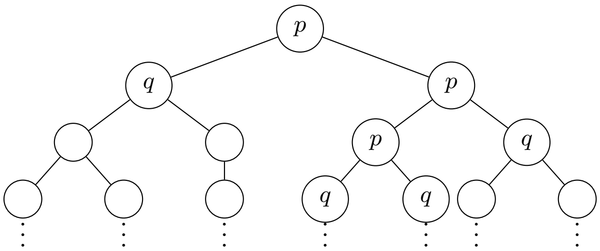


Figure 3: Visualization of a CTL formula: $\mathbf{A}[P\mathbf{U}Q]$

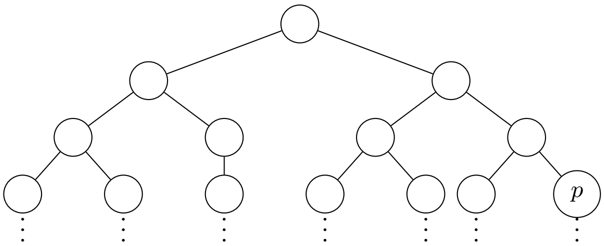


Figure 4: Visualization of a CTL formula: $\mathbf{EF}P$

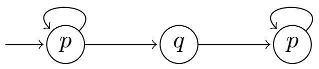


Figure 5: An example of an automaton that satisfies $\Diamond\Box P$

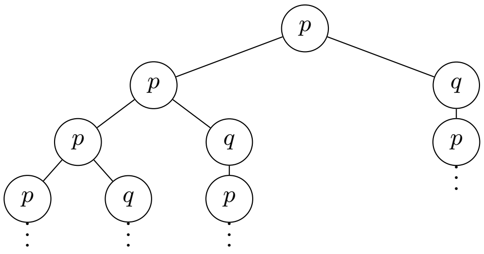


Figure 6: Unrolling the branches of Figure 5

Figure 5 for arbitrarily long, maintaining the existence of an alternate path in which p does not hold globally.

An example of a CTL formula that cannot be expressed in LTL is $\mathbf{AG}(\mathbf{EF}P)$. This formula states that there is always the possibility that a state can be reached during a run, even if it is never actually reached. However, the natural corresponding LTL formula $\Box\Diamond P$ states that at all times, P will eventually be reached. Note that this formula is stronger than the previous one since we just need the possibility of returning to P.

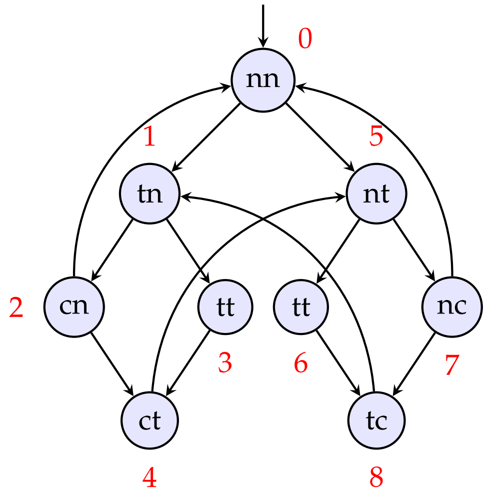
4 Example: Mutual Exclusion

Temporal logic is particularly helpful to verify properties of distributed systems. For example, we may want to reason about safety or liveness. Safety properties state that "nothing bad would ever happen", whereas liveness properties state that "something good always happens". We will how we can encode safety and liveness using CTL for a mutual exclusion protocol.

The notation in the following transition diagram is nt for: the first process is in the noncritical section while the second process is trying to get into its critical section.

n	noncritical section of an abstract process
t	trying to enter critical section of an abstract process
c	critical section of an abstract process

Those atomic propositional letters are used with suffix 1 to indicate that they apply to process 1 and with suffix 2 to indicate process 2. For example the notation nt indicates a state in which $n_1 \wedge t_2$ is true (and no other propositional letters). Consider Kripke structure



1. Safety: $\neg\mathbf{EF}(c_1 \wedge c_2)$ is trivially true since there is no state labelled ccx.
2. Liveness: $\mathbf{AG}(t_1 \rightarrow \mathbf{AF}c_1) \wedge \mathbf{AG}(t_2 \rightarrow \mathbf{AF}c_2)$

References

[BKL08] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of Model Checking*. MIT Press, 2008.

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, Cambridge, 1999.

[Eme90] Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 995–1072. MIT Press, 1990.