15-414: Bug Catching: Automated Program Verification

Lecture Notes on Computations & Computation Tree Logic

Matt Fredrikson André Platzer

Carnegie Mellon University
Lecture 16

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. To get a good sense of how model checking works, we, thus, consider the closely related but different(!) Computation Tree Logic (CTL) instead. Both LTL and CTL are common in model checking even if they have different advantages and downsides.

The main point about LTL is that its semantics fixes a trace and then talks about temporal properties along that particular trace. CTL instead switches to a new trace every time a temporal operator is used. CTL has the advantage of having a pretty simple model checking algorithm.

2 Kripke Structures

Definition 1 (Kripke structure)**.** A *Kripke frame* $(W, \curvearrowright)$ consists of a set W with a transition relation $\curvearrowright \subseteq W \times W$ where $s \curvearrowright t$ indicates that there is a direct transition from s to t in the Kripke frame $(W, \curvearrowright)$. The elements $s \in W$ are also called states. A *Kripke structure* $K = (W, \curvearrowright, v)$ is a Kripke frame $(W, \curvearrowright)$ with a mapping $v : W \to \Sigma \to \{\mathit{true}, \mathit{false}\}$ assigning truth-values to all the propositional atoms in all states.

The program semantics $[\![\alpha]\!]$ which was defined as a relation of initial and final states in Lecture 3 is an example of a Kripke structure.

Definition 2 (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.

3 Computation Tree Logic

Definition 3. 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. $s \models P \wedge Q$ 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 $0 \leq j < i$

4 Definables

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 4. *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.

5 Core Insights

The following lemma exploits the fact that every state has a successor in computation structures, so some next state is always defined.

Lemma 5 (Next remainders)**.** *The following are sound axioms for the computation structures of CTL:*

(EG) $\mathbf{EG}P \leftrightarrow P \land \mathbf{EXEG}P$

(EF) $\mathbf{EF}P \leftrightarrow P \lor \mathbf{EXEF}P$

(EU) $\mathbf{E}P\mathbf{U}Q \leftrightarrow Q \lor P \land \mathbf{EXE}P\mathbf{U}Q$

(AU) $\mathbf{A}P\mathbf{U}Q \leftrightarrow Q \lor P \land \mathbf{AXA}P\mathbf{U}Q$

6 CTL Model Checking Algorithm

The idea behind model checking is to exploit finiteness of the state spaces to directly compute the semantics of the formulas.

Given a *finite* computation structure $K = (W, \curvearrowright, v)$ the CTL model checking algorithm computes the set of all states of K in which CTL formula ϕ is true:

$$[\![\phi]\!] \stackrel{\text{def}}{=} \{s \in W \ : \ s \models \phi\}$$

The CTL model checking algorithm for a computation structure $K = (W, \curvearrowright, v)$ computes this set $[\![\phi]\!]$ by directly following the semantics in a recursive function along the equations in this lemma.

Note that we adopt the same notational convention for set union $\cup$ and intersection $\cap$ as we did for logical disjunction $\lor$ and conjunction $\land$ namely that $\cap$ and $\land$ bind stronger:

$$P \lor Q \land R \equiv P \lor (Q \land R)$$
$$X \cup Y \cap Z = X \cup (Y \cap Z)$$

Theorem 6 (CTL model checking)**.** *In computation structures, the set $[\![\phi]\!]$ of all states that satisfy CTL formula ϕ satisfies the following equations:*

1. $[\![p]\!] = \{s \in W \ : \ v(s)(p) = true\}$ *for atomic propositions* p
2. $[\![\neg P]\!] = W \setminus [\![P]\!]$
3. $[\![P \land Q]\!] = [\![P]\!] \cap [\![Q]\!]$
4. $[\![P \lor Q]\!] = [\![P]\!] \cup [\![Q]\!]$
5. $[\![\mathbf{EX}P]\!] = \tau_{\mathbf{EX}}([\![P]\!])$ *using the existential successor function* $\tau_{\mathbf{EX}}()$ *defined as follows:*
$$\tau_{\mathbf{EX}}(Z) \stackrel{\text{def}}{=} \{s \in W \ : \ t \in Z \textit{ for some state } t \textit{ with } s \curvearrowright t\}$$
6. $[\![\mathbf{AX}P]\!] = \tau_{\mathbf{AX}}([\![P]\!])$ *using the universal successor function* $\tau_{\mathbf{AX}}()$ *defined as follows:*
$$\tau_{\mathbf{AX}}(Z) \stackrel{\text{def}}{=} \{s \in W \ : \ t \in Z \textit{ for all states } t \textit{ with } s \curvearrowright t\}$$
7. $[\![\mathbf{EF}P]\!] = \mu Z.([\![P]\!] \cup \tau_{\mathbf{EX}}(Z))$ *where* $\mu Z.f(Z)$ *denotes the least fixpoint* Z *of the operation* $f(Z)$*, that is, the smallest set of states satisfying* $Z = f(Z)$*.*
8. $[\![\mathbf{EG}P]\!] = \nu Z.([\![P]\!] \cap \tau_{\mathbf{EX}}(Z))$ *where* $\nu Z.f(Z)$ *denotes the greatest fixpoint* Z *of the operation* $f(Z)$*, that is, the largest set of states satisfying* $Z = f(Z)$*.*
9. $[\![\mathbf{AF}P]\!] = \mu Z.([\![P]\!] \cup \tau_{\mathbf{AX}}(Z))$
10. $[\![\mathbf{AG}P]\!] = \nu Z.([\![P]\!] \cap \tau_{\mathbf{AX}}(Z))$
11. $[\![\mathbf{E}P\mathbf{U}Q]\!] = \mu Z.\big([\![Q]\!] \cup ([\![P]\!] \cap \tau_{\mathbf{EX}}(Z))\big)$
12. $[\![\mathbf{A}P\mathbf{U}Q]\!] = \mu Z.\big([\![Q]\!] \cup ([\![P]\!] \cap \tau_{\mathbf{AX}}(Z))\big)$

The correctness argument for the verification algorithm uses the axioms EF,EU together with the insight that the respective set of states that they characterize are the *smallest* set satisfying the respective equivalence. The largest set for $\mathbf{EF}P$ satisfying the equivalence in EF would simply be the entire set of states, which is futile. Likewise, the smallest set of states for $\mathbf{EG}P$ satisfying the equivalence in EG would simply be the empty set of states, since every state has a successor in a computation structure.

Proof of Theorem 6. The proof is *not* by induction on the number of states or on the formula because the resulting formulas are not any easier than the original formulas. Instead, the proof proves each equation separately. While the proof was left as an exercise originally [CES83], some cases are already proved in [CGP99], some more in [BKL08], and a much more comprehensive proof including the nontrivial case $\mathbf{A}P\mathbf{U}Q$ that uses König's lemma is in [Sch03].

The first cases immediately follow the semantics of atomic propositions, propositional operators, and $\mathbf{EX}$. The remaining cases separately argue that it is a fixpoint and then that it is the largest or smallest as indicated.

6. By axiom EF and case 5, the formula $\mathbf{EF}P$ satisfies the indicated fixpoint equation:

$$[\![\mathbf{EF}P]\!] = [\![P \lor \mathbf{EXEF}P]\!] = [\![P]\!] \cup \tau_{\mathbf{EX}}([\![\mathbf{EF}P]\!])$$

   Showing that it is the least fixpoint is left as an exercise.

7. By axiom EG and case 5, the formula $\mathbf{EG}P$ satisfies the fixpoint equation:

$$[\![\mathbf{EG}P]\!] = [\![P \land \mathbf{EXEG}P]\!] = [\![P]\!] \cap \tau_{\mathbf{EX}}([\![\mathbf{EG}P]\!])$$

   In order to show that $[\![\mathbf{EG}P]\!]$ is the greatest fixpoint, consider another fixpoint $H = [\![P]\!] \cap \tau_{\mathbf{EX}}(H)$ and show that $H \subseteq [\![\mathbf{EG}P]\!]$ by considering any state $s_0 \in H$ and showing that $s_0 \in [\![\mathbf{EG}P]\!]$. Since $H \subseteq [\![P]\!]$, it is enough to show that there is a path $s_0, s_1, s_2, \ldots$ such that $s_i \in H$ for all i by induction on i, implying $s_i \models P$.

   n=0: The base case follows from $s_0 \in H$.

   n+1: By induction hypothesis $s_n \in H$. Thus, $s_n \in H = [\![P]\!] \cap \tau_{\mathbf{EX}}(H)$, so there is a state s_{n+1} with $s_n \curvearrowright s_{n+1}$ and $s_{n+1} \in H$.

8. By axiom EU and case 5, the formula $\mathbf{E}P\mathbf{U}Q$ satisfies the fixpoint equation:

$$[\![\mathbf{E}P\mathbf{U}Q]\!] = [\![Q \lor P \land \mathbf{EXE}P\mathbf{U}Q]\!] = [\![Q]\!] \cup [\![P]\!] \cap \tau_{\mathbf{EX}}([\![\mathbf{E}P\mathbf{U}Q]\!])$$

   In order to show that $[\![\mathbf{E}P\mathbf{U}Q]\!]$ is also the least fixpoint, consider another fixpoint $H = [\![Q]\!] \cup [\![P]\!] \cap \tau_{\mathbf{EX}}(H)$ and show that $[\![\mathbf{E}P\mathbf{U}Q]\!] \subseteq H$. So consider any $s_0 \in [\![\mathbf{E}P\mathbf{U}Q]\!]$ and show that $s_0 \in H$. By $s_0 \in [\![\mathbf{E}P\mathbf{U}Q]\!]$, there is a path $s_0, s_1, s_2, \ldots$ and an n such that $s_n \models Q$ and $s_j \models P$ for all $0 \le j < n$. We prove that $s_n \in H$ by induction on n (note that we could also do a backwards induction starting at n and going backwards to 0).

   $n = 0$: The base case where $n = 0$ follows from $s_n \in [\![Q]\!] \subseteq [\![Q]\!] \cup [\![P]\!] \cap \tau_{\mathbf{EX}}(H) = H$.

   n+1: By induction hypothesis, $s_1 \in H$. In order to show that $s_0 \in H = [\![Q]\!] \cup [\![P]\!] \cap \tau_{\mathbf{EX}}(H)$, we use that we know $s_0 \models P$ and that s_0 has a successor $s_1 \in H$. Thus, $s_0 \in [\![P]\!] \cap \tau_{\mathbf{EX}}(H) \subseteq H$.

9. By axiom AU and case 6, the formula $\mathbf{A}P\mathbf{U}Q$ satisfies the fixpoint equation:

$$[\![\mathbf{A}P\mathbf{U}Q]\!] = [\![Q \lor P \land \mathbf{AXA}P\mathbf{U}Q]\!] = [\![Q]\!] \cup [\![P]\!] \cap \tau_{\mathbf{AX}}([\![\mathbf{A}P\mathbf{U}Q]\!])$$

   In order to show that $[\![\mathbf{A}P\mathbf{U}Q]\!]$ is also the least fixpoint, consider another fixpoint $H = [\![Q]\!] \cup [\![P]\!] \cap \tau_{\mathbf{AX}}(H)$ and show that $[\![\mathbf{A}P\mathbf{U}Q]\!] \subseteq H$. So consider any $s_0 \in [\![\mathbf{A}P\mathbf{U}Q]\!]$ and show that $s_0 \in H$. By $s_0 \in [\![\mathbf{A}P\mathbf{U}Q]\!]$, all paths $s^i_0, s^i_1, s^i_2, \ldots$ starting in $s^i_0 = s_0$ have an n_i such that $s^i_{n_i} \models Q$ and $s^i_j \models P$ for all $0 \le j < n_i$. Could there be infinitely many such paths?

Only the prefix of a path till n_i matters (because no statement is made beyond n_i). Each such prefix is finite, because the strong until requires Q to eventually happen and cannot be postponed forever. Without loss of generality, the smallest respective n_i can be assumed on each path, though. So it can be shown that only finitely many such paths exist as follows. By König's lemma[1], there can only be finitely many paths till the respective n_i, because if there were infinitely many finite paths of length at most n_i, then there would have to be infinite branching so infinitely many states, but W is finite. For example, there can only be finitely many paths of length, say, $n_i = 10$ in a finite Kripke structure.

Consequently, since there are only finitely many such paths, the maximum n_i is still a finite natural number $n \in \mathbb{N}$, as well (the supremum of infinitely many finite numbers can be infinite). So we will prove the conjecture by induction on n.

We prove that $s_0 \in H$ by induction on n.

$n = 0$: The base case where $n = 0$ follows from $s_n \in [\![Q]\!] \subseteq [\![Q]\!] \cup [\![P]\!] \cap \tau_{\mathbf{AX}}(H) = H$.

n+1: By induction hypothesis, all of the *finitely many(!)* path numbers i satisfy $s_1^i \in H$. Since we also have $s_0 \models P$ and that i ranges over *all* successors of s_0 that $s_0 \in [\![P]\!] \cap \tau_{\mathbf{AX}}(H) \subseteq H$.

Note that this correctness proof crucially depends on the until condition Q eventually happening, so each of the paths is actually finite. The proof does not work for the weak until, which is also true if Q never becomes true as long as P is true all the time then. $\square$

Since the successor function can be computed by checking off the corresponding states along the computation structure, the only remaining question is how the least and greatest fixpoints can be computed. The first good news is that all the functions in Theorem 6 are monotone, in the sense that if their parts are true in more states then the expressions themselves are true in more states, too.

7 How to Compute Monotone Fixpoints

Let $\wp(W)$ denote the set of all subsets of W and let f^n by the n-fold composition of function f so f^{n+1} is the function mapping Z to $f(f^n(Z))$ and f^1 is f. For example, f^3 is the function mapping Z to $f(f(f(Z)))$ The particular special case of the seminal Knaster-Fixpoint theorem we need here is the following:

Theorem 7 (Knaster-Tarski)**.** *Every monotone function $f : \wp(W) \to \wp(W)$ has a least and a greatest fixpoint and both can be found by iteration:*

$$\mu Z.f(Z) = \bigcup_{n \geq 1} f^n(\emptyset) \qquad \nu Z.f(Z) = \bigcap_{n \geq 1} f^n(W)$$

[1] König's lemma says: every infinite tree has an infinite path or a node with infinitely many branches.

For complicated functions on infinite sets, the above unions and intersections range over more than just all natural numbers. But model checking is typically done when the computation structure is finite. In that case, it is entirely obvious that the union and intersection only range over finitely many natural numbers. Every time we consider an additional iteration $f^n(\emptyset)$, we either find a new state that was not in the union yet. Or we do not find such a state but then, since nothing changed, the iterate $f^{n+1}(\emptyset)$ will not find anything new either. Since there are only finitely many different states in a finite state set W of a finite computation structure, we can only find new states finitely often so that the computation terminates. The argument for the intersection is correspondingly.

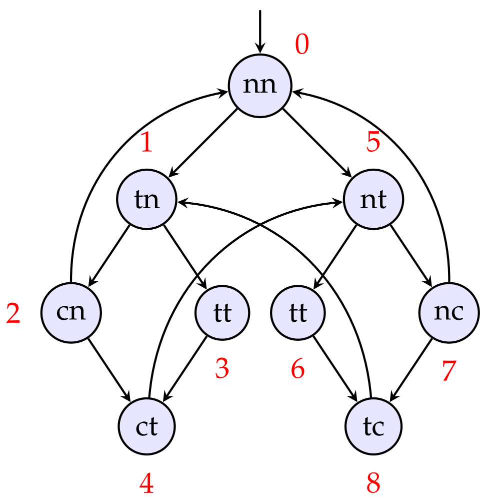
Theorem 8 (Complexity)**.** *The CTL model checking problem is linear in the size of the state space $K = (W, \curvearrowright, v)$ and in the size of the formula ϕ in the sense that it is in $O(|K| \cdot |\phi|)$ where $|K| = |W| + |\curvearrowright|$.*

8 Example: Mutual Exclusion

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: $\mathbf{AG}(t_1 \to \mathbf{AF} c_1) \wedge \mathbf{AG}(t_2 \to \mathbf{AF} c_2)$

Checking $1 \models t_1 \to \mathbf{AF}c_1$ alias $1 \models \neg t_1 \lor \mathbf{AF}c_1$ first computes subformulas.

$$
\begin{aligned}
[\![t_1]\!] &= \{1,3,6,8\}\\
[\![c_1]\!] &= \{2,4\}\\
[\![\neg t_1]\!] &= \{0,2,4,5,7\}\\
[\![\mathbf{AF}c_1]\!] &= \mu Z.([\![c_1]\!] \cup \tau_{\mathbf{AX}}(Z)) =: \mu Z.f(Z)\\
& \qquad f^1(\emptyset) = [\![c_1]\!] && = \{2,4\}\\
& \qquad f^2(\emptyset) = [\![c_1]\!] \cup \tau_{\mathbf{AX}}(\{2,4\}) && = \{1,2,3,4\}\\
& \qquad f^3(\emptyset) = [\![c_1]\!] \cup \tau_{\mathbf{AX}}(\{1,2,3,4\}) && = \{1,2,3,4,8\}\\
& \qquad f^4(\emptyset) = [\![c_1]\!] \cup \tau_{\mathbf{AX}}(\{1,2,3,4,8\}) && = \{1,2,3,4,6,8\}\\
& \qquad f^5(\emptyset) = [\![c_1]\!] \cup \tau_{\mathbf{AX}}(\{1,2,3,4,6,8\}) && = \{1,2,3,4,6,8\} = f^4(\emptyset)\\
[\![\mathbf{AF}c_1]\!] &= \{1,2,3,4,6,8\}\\
[\![\neg t_1 \lor \mathbf{AF}c_1]\!] &= \{0,1,2,3,4,5,6,7,8\}
\end{aligned}
$$

Since $1 \in [\![\neg t_1 \lor \mathbf{AF}c_1]\!]$ CTL model checking confirms $1 \models \neg t_1 \lor \mathbf{AF}c_1$. Since every state $[\![\neg t_1 \lor \mathbf{AF}c_1]\!]$ equals the set of all states, it is easy to see that model checking will also eventually find $0 \in [\![\mathbf{AG}(\neg t_1 \lor \mathbf{AF}c_1)]\!]$. Consequently it confirms that the initial state 0 satisfies $0 \models \mathbf{AG}(\neg t_1 \lor \mathbf{AF}c_1)$.

References

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

[CES83] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications: A practical approach. In *POPL*, pages 117–126, 1983.

[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.

[Sch03] Peter H. Schmitt. Nichtklassische Logiken. Vorlesungsskriptum Fakultät für Informatik , Universität Karlsruhe, Mai 2003.