15-414: Bug Catching: Automated Program Verification

Lecture Notes on CTL Model Checking

Matt Fredrikson

Carnegie Mellon University
Lecture 21
Tuesday, November 17, 2026

1 Introduction

In the last lecture we introduced Computation Tree Logic (CTL), whose formulas are evaluated at the states of a Kripke structure and quantify over the paths that leave those states. Today we turn to the algorithmic question: given a finite computation structure K and a CTL formula ϕ, how do we decide whether $K \models \phi$?

For LTL, we answered the corresponding question with Büchi automata: we built an automaton for the negated property, formed its product with the structure, and searched for an accepting cycle. That automaton can be exponentially larger than the formula [BKL08]. CTL model checking avoids this construction entirely. Because CTL formulas are state formulas, we can compute the set of states that satisfy each subformula directly, working from the inside out. The only interesting cases are the temporal operators, and for those the set of satisfying states is the least or greatest solution of a simple recursive equation. The resulting algorithm runs in time polynomial in both the structure and the formula [CES83, CGP99].

Learning Goals

After this lecture, you will:

- Understand the expansion laws that unfold each CTL temporal operator by one step.
- Characterize the states satisfying a CTL formula as least and greatest fixpoints of monotone set functions.
- Compute those fixpoints by iteration on a finite computation structure.

- Carry out the CTL model checking algorithm by hand on small examples.

2 The Model Checking Problem

Throughout this lecture, $K = (W, \curvearrowright, v, I)$ is a computation structure as defined in the previous lecture: W is finite, every state has at least one successor, $v(s)$ is the set of atomic propositions true at s, and $I \subseteq W$ is the set of initial states. For a CTL formula ϕ, define its *satisfaction set*

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

Since $K \models \phi$ holds exactly when every initial state satisfies ϕ, the model checking problem reduces to computing $[\![\phi]\!]$ and checking that

$$I \subseteq [\![\phi]\!].$$

We compute $[\![\phi]\!]$ recursively on the structure of ϕ: first the satisfaction sets of the immediate subformulas, then the satisfaction set of ϕ itself. Propositional connectives are easy, since they correspond to set operations. The temporal operators require more thought.

We will use two functions that look one step ahead in the transition relation. For a set of states $Z \subseteq W$, the *existential successor function* $\tau_{\mathbf{EX}}()$ and the *universal successor function* $\tau_{\mathbf{AX}}()$ are defined by

$$\tau_{\mathbf{EX}}(Z) \stackrel{\text{def}}{=} \{s \in W \ : \ t \in Z \text{ for some } t \text{ with } s \curvearrowright t\},$$
$$\tau_{\mathbf{AX}}(Z) \stackrel{\text{def}}{=} \{s \in W \ : \ t \in Z \text{ for all } t \text{ with } s \curvearrowright t\}.$$

Directly from the semantics of the next-state operators,

$$[\![\mathbf{EX}P]\!] = \tau_{\mathbf{EX}}([\![P]\!]) \qquad \text{and} \qquad [\![\mathbf{AX}P]\!] = \tau_{\mathbf{AX}}([\![P]\!]).$$

Both functions can be computed with a single pass over the transition relation. Both are also *monotone*: if $Z \subseteq Z'$, then $\tau_{\mathbf{EX}}(Z) \subseteq \tau_{\mathbf{EX}}(Z')$ and $\tau_{\mathbf{AX}}(Z) \subseteq \tau_{\mathbf{AX}}(Z')$. Because every state of a computation structure has a successor, $\tau_{\mathbf{AX}}(\emptyset) = \tau_{\mathbf{EX}}(\emptyset) = \emptyset$ and $\tau_{\mathbf{AX}}(W) = \tau_{\mathbf{EX}}(W) = W$.

3 Expansion Laws

The key to handling the remaining operators is that each of them can be unfolded by one step. The following lemma uses the fact that every state of a computation structure has a successor, so that a path can always be continued.

Lemma 1 (Expansion laws)**.** *The following equivalences hold in every state of every computation structure:*

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

(AF) $\mathbf{AF}P \leftrightarrow P \vee \mathbf{AXAF}P$

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

(AG) $\mathbf{AG}P \leftrightarrow P \wedge \mathbf{AXAG}P$

(EU) $\mathbf{E}[P\mathbf{U}Q] \leftrightarrow Q \vee \big(P \wedge \mathbf{EXE}[P\mathbf{U}Q]\big)$

(AU) $\mathbf{A}[P\mathbf{U}Q] \leftrightarrow Q \vee \big(P \wedge \mathbf{AXA}[P\mathbf{U}Q]\big)$

Proof. We show (EU); the other cases are similar. Suppose $s \models \mathbf{E}[P\mathbf{U}Q]$, witnessed by a path $s = s_0, s_1, \ldots$ and an index n with $s_n \models Q$ and $s_j \models P$ for all $j < n$. If $n = 0$, then $s \models Q$. Otherwise $s \models P$, and the suffix $s_1, s_2, \ldots$ witnesses $s_1 \models \mathbf{E}[P\mathbf{U}Q]$ with index $n - 1$. Since $s \curvearrowright s_1$, we get $s \models \mathbf{EXE}[P\mathbf{U}Q]$. Conversely, if $s \models Q$, then any path from s (one exists since every state has a successor) is a witness with $n = 0$. If $s \models P$ and $s \curvearrowright t$ for some $t \models \mathbf{E}[P\mathbf{U}Q]$, then prefixing s to a witness path for t gives a witness path for s. □

Read as equations on satisfaction sets, the expansion laws say that $[\![\mathbf{EF}P]\!]$ is a solution Z of $Z = [\![P]\!] \cup \tau_{\mathbf{EX}}(Z)$, that $[\![\mathbf{EG}P]\!]$ is a solution of $Z = [\![P]\!] \cap \tau_{\mathbf{EX}}(Z)$, and so on. These equations do not determine the sets uniquely. For example, every state of a computation structure has a successor in W, so $Z = W$ always solves $Z = [\![P]\!] \cup \tau_{\mathbf{EX}}(Z)$, even when P is false everywhere. Similarly, $Z = \emptyset$ always solves $Z = [\![P]\!] \cap \tau_{\mathbf{EX}}(Z)$, even when P is true everywhere. The meaning of each operator picks out a particular solution: the *smallest* one for the eventually and until operators, and the *largest* one for the always operators.

4 Satisfaction Sets as Fixpoints

A set Z is a *fixpoint* of a function $f : \wp(W) \to \wp(W)$ if $f(Z) = Z$. We write $\mu Z.f(Z)$ for the *least fixpoint* of f, which is contained in every other fixpoint, and $\nu Z.f(Z)$ for the *greatest fixpoint*, which contains every other fixpoint. In the equations below, we use the convention that $\cap$ binds more strongly than $\cup$, just as $\wedge$ binds more strongly than $\vee$:

$$P \vee Q \wedge R \equiv P \vee (Q \wedge R)$$
$$X \cup Y \cap Z = X \cup (Y \cap Z)$$

Theorem 2 (CTL model checking)**.** *In a computation structure, the satisfaction sets of CTL formulas satisfy the following equations:*

1. $[\![p]\!] = \{s \in W \ : \ p \in v(s)\}$ *for atomic propositions* p
2. $[\![\neg P]\!] = W \setminus [\![P]\!]$
3. $[\![P \wedge Q]\!] = [\![P]\!] \cap [\![Q]\!]$
4. $[\![P \vee Q]\!] = [\![P]\!] \cup [\![Q]\!]$

5. $[\![\mathbf{EX}P]\!] = \tau_{\mathbf{EX}}([\![P]\!])$
6. $[\![\mathbf{AX}P]\!] = \tau_{\mathbf{AX}}([\![P]\!])$
7. $[\![\mathbf{E}[P\mathbf{U}Q]]\!] = \mu Z.\big([\![Q]\!] \cup [\![P]\!] \cap \tau_{\mathbf{EX}}(Z)\big)$
8. $[\![\mathbf{A}[P\mathbf{U}Q]]\!] = \mu Z.\big([\![Q]\!] \cup [\![P]\!] \cap \tau_{\mathbf{AX}}(Z)\big)$
9. $[\![\mathbf{EF}P]\!] = \mu Z.([\![P]\!] \cup \tau_{\mathbf{EX}}(Z))$
10. $[\![\mathbf{AF}P]\!] = \mu Z.([\![P]\!] \cup \tau_{\mathbf{AX}}(Z))$
11. $[\![\mathbf{EG}P]\!] = \nu Z.([\![P]\!] \cap \tau_{\mathbf{EX}}(Z))$
12. $[\![\mathbf{AG}P]\!] = \nu Z.([\![P]\!] \cap \tau_{\mathbf{AX}}(Z))$

Each function on the right-hand side is monotone in Z, because $\cup$, $\cap$, $\tau_{\mathbf{EX}}()$, and $\tau_{\mathbf{AX}}()$ are monotone. The next section shows that such functions always have least and greatest fixpoints, so the equations make sense.

Proof. Cases 1–6 follow directly from the semantics of CTL. For each remaining case, we first observe that the satisfaction set is a fixpoint, which is the corresponding expansion law of Lemma 1 combined with cases 5 and 6. For example, for (EU):

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

It remains to show that it is the least or greatest fixpoint, as indicated.

7. Let H be any fixpoint, $H = [\![Q]\!] \cup [\![P]\!] \cap \tau_{\mathbf{EX}}(H)$. We show $[\![\mathbf{E}[P\mathbf{U}Q]]\!] \subseteq H$. Every $s \models \mathbf{E}[P\mathbf{U}Q]$ has a witness: a path $s = s_0, s_1, \ldots$ and an index n with $s_n \models Q$ and $s_j \models P$ for all $j < n$. We show by induction on n that every state with a witness of index n is in H.

   $n = 0$: Then $s \in [\![Q]\!] \subseteq H$.

   $n + 1$: Then $s \models P$, and the suffix $s_1, s_2, \ldots$ is a witness of index n for s_1. By the induction hypothesis, $s_1 \in H$. Since $s \curvearrowright s_1$, we have $s \in [\![P]\!] \cap \tau_{\mathbf{EX}}(H) \subseteq H$.

8. Let H be any fixpoint, $H = [\![Q]\!] \cup [\![P]\!] \cap \tau_{\mathbf{AX}}(H)$. We show $[\![\mathbf{A}[P\mathbf{U}Q]]\!] \subseteq H$. Now each path from s has its own index at which Q first holds, and there may be infinitely many paths, so we need a bound that works for all of them.

   For $s \models \mathbf{A}[P\mathbf{U}Q]$, consider the tree T_s of finite sequences $s = s_0 \curvearrowright s_1 \curvearrowright \ldots \curvearrowright s_k$ with $s_i \not\models Q$ for all $i \leq k$, ordered by extension. Each node has at most $|W|$ children, so T_s is finitely branching. If T_s were infinite, then by König's lemma[1] it would have an infinite branch: an infinite path from s on which Q never holds. That would contradict $s \models \mathbf{A}[P\mathbf{U}Q]$. So T_s is finite, and we let $d(s)$ be its height, the number of states on its longest sequence, with $d(s) = 0$ when T_s is empty, i.e., when $s \models Q$. We show by induction on $d(s)$ that $s \in H$.

[1] König's lemma: every infinite, finitely branching tree has an infinite branch.

$d(s) = 0$: Then $s \in [\![Q]\!] \subseteq H$.

$d(s) > 0$: Then $s \not\models Q$, so every path from s reaches Q at a positive index, and therefore $s \models P$. Let t be any successor of s. Every path $t, t_1, t_2, \ldots$ extends to the path $s, t, t_1, t_2, \ldots$, whose witness index is positive, so dropping s gives a witness for t. Hence $t \models \mathbf{A}[P \mathbf{U} Q]$. Moreover, every sequence in T_t extends a sequence in T_s by prefixing s, so $d(t) < d(s)$. By the induction hypothesis, $t \in H$. Since this holds for every successor t, we get $s \in [\![P]\!] \cap \tau_{\mathbf{AX}}(H) \subseteq H$.

This argument relies on Q eventually holding on every path. It would fail for a weak until, which is also satisfied when P holds forever and Q never does.

9. By case 7, using $\mathbf{EF}P \leftrightarrow \mathbf{E}[\mathit{true}\mathbf{U}P]$ from the previous lecture and $[\![\mathit{true}]\!] \cap \tau_{\mathbf{EX}}(Z) = \tau_{\mathbf{EX}}(Z)$.

10. By case 8, using $\mathbf{AF}P \leftrightarrow \mathbf{A}[\mathit{true}\mathbf{U}P]$.

11. Let H be any fixpoint, $H = [\![P]\!] \cap \tau_{\mathbf{EX}}(H)$. We show $H \subseteq [\![\mathbf{EG}P]\!]$. Let $s_0 \in H$. We construct a path $s_0, s_1, s_2, \ldots$ that stays in H: if $s_n \in H$, then $s_n \in \tau_{\mathbf{EX}}(H)$, so we can pick a successor $s_{n+1} \in H$. Since $H \subseteq [\![P]\!]$, every state on this path satisfies P, so $s_0 \models \mathbf{EG}P$.

12. Left as an exercise. □

5 Computing Fixpoints by Iteration

Theorem 2 reduces model checking to computing fixpoints of monotone functions on the finite set $\wp(W)$. The Knaster-Tarski theorem guarantees that every monotone function on $\wp(W)$ has a least and a greatest fixpoint, even when W is infinite. When W is finite, we can find them by simple iteration. Let f^n denote the n-fold application of f, so that $f^0(Z) = Z$ and $f^{n+1}(Z) = f(f^n(Z))$.

Theorem 3 (Fixpoint iteration)**.** *Let W be finite and $f : \wp(W) \to \wp(W)$ be monotone. Then*

$$\emptyset \subseteq f(\emptyset) \subseteq f^2(\emptyset) \subseteq \cdots \qquad \textit{and} \qquad W \supseteq f(W) \supseteq f^2(W) \supseteq \cdots,$$

and there is some $m \leq |W|$ such that $\mu Z.f(Z) = f^m(\emptyset)$ and $\nu Z.f(Z) = f^m(W)$.

Proof. We consider the least fixpoint; the greatest fixpoint is symmetric. First, $\emptyset \subseteq f(\emptyset)$ trivially, and if $f^n(\emptyset) \subseteq f^{n+1}(\emptyset)$ then monotonicity gives $f^{n+1}(\emptyset) \subseteq f^{n+2}(\emptyset)$, so the sequence is increasing. Each strict inclusion adds at least one state of W, so there can be at most $|W|$ strict inclusions. Hence $f^m(\emptyset) = f^{m+1}(\emptyset)$ for some $m \leq |W|$, which means $f^m(\emptyset)$ is a fixpoint. To see that it is the least one, let Z be any fixpoint. Then $\emptyset \subseteq Z$, and if $f^n(\emptyset) \subseteq Z$ then $f^{n+1}(\emptyset) \subseteq f(Z) = Z$. By induction, $f^m(\emptyset) \subseteq Z$. □

In practice, we do not need to know m in advance: we iterate until two consecutive sets are equal.

The algorithm. Putting Theorems 2 and 3 together gives a recursive procedure that computes $[\![\phi]\!]$:

1. Recursively compute the satisfaction sets of the immediate subformulas of ϕ.
2. If ϕ is an atomic proposition, a Boolean combination, or a next-state formula, combine those sets using cases 1–6 of Theorem 2.
3. Otherwise, iterate the function given by Theorem 2 starting from $\emptyset$ (for until and eventually) or from W (for always) until the set stops changing.

Finally, $K \models \phi$ if and only if $I \subseteq [\![\phi]\!]$. If the check fails, any initial state $s \in I \setminus [\![\phi]\!]$ is a starting point for a counterexample.

Complexity. Computing $\tau_{\mathbf{EX}}(Z)$ or $\tau_{\mathbf{AX}}(Z)$ takes time $O(|K|)$, where $|K| = |W| + |\curvearrowright|$. A fixpoint needs at most $|W|$ iterations, and the formula has $|\phi|$ subformulas, so the naive procedure runs in time $O(|W| \cdot |K| \cdot |\phi|)$, which is polynomial in both inputs. More careful algorithms bring this down to $O(|K| \cdot |\phi|)$: until formulas can be handled by a single backward search from $[\![Q]\!]$, and $\mathbf{EG}P$ by finding the strongly connected components of the subgraph restricted to $[\![P]\!]$ [CGP99, BKL08]. By contrast, LTL model checking is PSPACE-complete, and the automaton-based algorithm from Lecture 19 takes time exponential in the size of the formula.

6 Example: Mutual Exclusion

Recall the mutual exclusion protocol from the previous lecture, shown again in Figure 1. The states are numbered 0 through 8. In each label, the first letter is the status of process 1 and the second letter is the status of process 2: each is in the **n**oncritical section, **t**rying to enter the critical section, or in the **c**ritical section. For example, tn means that $t_1 \wedge n_2$ holds, and no other atomic propositions do.

Reading off the edges, the successors of each state are:

s	0	1	2	3	4	5	6	7	8
successors	1, 5	2, 3	0, 4	4	5	6, 7	8	0, 8	1

In the previous lecture, we stated two properties of this protocol:

1. Safety: $\neg\mathbf{EF}(c_1 \wedge c_2)$. Since $[\![c_1 \wedge c_2]\!] = \emptyset$, the first iterate of $\mu Z.(\emptyset \cup \tau_{\mathbf{EX}}(Z))$ is $\tau_{\mathbf{EX}}(\emptyset) = \emptyset$, which is already a fixpoint. So $[\![\mathbf{EF}(c_1 \wedge c_2)]\!] = \emptyset$ and the safety property holds in every state.
2. Liveness: $\mathbf{AG}(t_1 \rightarrow \mathbf{AF}c_1) \wedge \mathbf{AG}(t_2 \rightarrow \mathbf{AF}c_2)$.

We check the first conjunct of the liveness property, writing $t_1 \rightarrow \mathbf{AF}c_1$ as $\neg t_1 \vee \mathbf{AF}c_1$. The algorithm starts with the innermost subformulas:

$$[\![t_1]\!] = \{1, 3, 6, 8\} \qquad [\![\neg t_1]\!] = \{0, 2, 4, 5, 7\} \qquad [\![c_1]\!] = \{2, 4\}$$

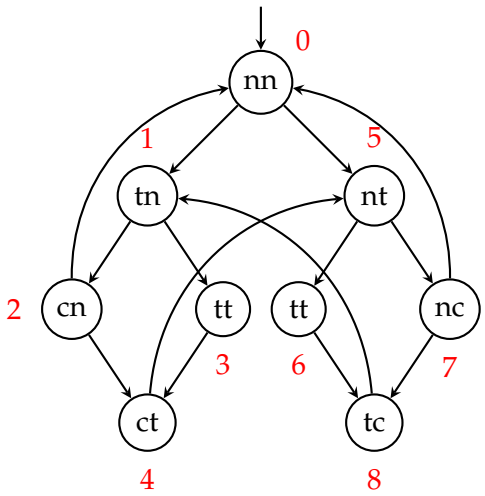

Figure 1: Kripke structure for a mutual exclusion protocol, with initial state 0.

Next, $[\![\mathbf{AF}c_1]\!] = \mu Z.f(Z)$ for $f(Z) = [\![c_1]\!] \cup \tau_{\mathbf{AX}}(Z)$. We iterate from $\emptyset$:

$$
\begin{aligned}
f(\emptyset) &= \{2,4\} \cup \tau_{\mathbf{AX}}(\emptyset) = \{2,4\} \\
f^2(\emptyset) &= \{2,4\} \cup \tau_{\mathbf{AX}}(\{2,4\}) = \{2,3,4\} \\
f^3(\emptyset) &= \{2,4\} \cup \tau_{\mathbf{AX}}(\{2,3,4\}) = \{1,2,3,4\} \\
f^4(\emptyset) &= \{2,4\} \cup \tau_{\mathbf{AX}}(\{1,2,3,4\}) = \{1,2,3,4,8\} \\
f^5(\emptyset) &= \{2,4\} \cup \tau_{\mathbf{AX}}(\{1,2,3,4,8\}) = \{1,2,3,4,6,8\} \\
f^6(\emptyset) &= \{2,4\} \cup \tau_{\mathbf{AX}}(\{1,2,3,4,6,8\}) = \{1,2,3,4,6,8\}
\end{aligned}
$$

For instance, in the second step, state 3 is added because its only successor is 4, but state 1 is not, since its successor 3 is not yet in the set. Each iterate adds the states whose successors are *all* already known to reach c_1. Since $f^6(\emptyset) = f^5(\emptyset)$, we have $[\![\mathbf{AF}c_1]\!] = \{1,2,3,4,6,8\}$, and therefore

$$[\![\neg t_1 \vee \mathbf{AF}c_1]\!] = \{0,2,4,5,7\} \cup \{1,2,3,4,6,8\} = W.$$

Finally, $[\![\mathbf{AG}(\neg t_1 \vee \mathbf{AF}c_1)]\!] = \nu Z.(W \cap \tau_{\mathbf{AX}}(Z))$. Iterating from W gives $W \cap \tau_{\mathbf{AX}}(W) = W$ immediately, so every state satisfies the first conjunct, including the initial state 0. By the symmetry of the protocol, the second conjunct holds as well, and the model checker confirms the liveness property.

A greatest fixpoint. Notice that the initial state 0 is *not* in $[\![\mathbf{AF}c_1]\!]$. Is there an execution along which process 1 never enters its critical section? We can answer this by computing $[\![\mathbf{EG}\neg c_1]\!] = \nu Z.g(Z)$ for $g(Z) = [\![\neg c_1]\!] \cap \tau_{\mathbf{EX}}(Z)$, where $[\![\neg c_1]\!] = \{0,1,3,5,6,7,8\}$.

Now we iterate downward from W:

$$
\begin{aligned}
g(W) &= \{0,1,3,5,6,7,8\} \cap \tau_{\mathbf{EX}}(W) = \{0,1,3,5,6,7,8\} \\
g^2(W) &= \{0,1,3,5,6,7,8\} \cap \tau_{\mathbf{EX}}(\{0,1,3,5,6,7,8\}) = \{0,1,5,6,7,8\} \\
g^3(W) &= \{0,1,3,5,6,7,8\} \cap \tau_{\mathbf{EX}}(\{0,1,5,6,7,8\}) = \{0,5,6,7,8\} \\
g^4(W) &= \{0,1,3,5,6,7,8\} \cap \tau_{\mathbf{EX}}(\{0,5,6,7,8\}) = \{0,5,6,7\} \\
g^5(W) &= \{0,1,3,5,6,7,8\} \cap \tau_{\mathbf{EX}}(\{0,5,6,7\}) = \{0,5,7\} \\
g^6(W) &= \{0,1,3,5,6,7,8\} \cap \tau_{\mathbf{EX}}(\{0,5,7\}) = \{0,5,7\}
\end{aligned}
$$

Each step removes the states that can no longer stay inside the set: first 3, whose only successor is 4, then 1, 8, and 6 in turn. The result $[\![\mathbf{EG}\neg c_1]\!] = \{0,5,7\}$ contains the initial state, witnessed by the cycle $0,5,7,0,5,7,\ldots$. On this path, process 1 stays in its noncritical section forever, which is why the liveness property only requires $\mathbf{AF}c_1$ in states where process 1 is trying to enter.

Exercises

1. Prove case 12 of Theorem 2. *Hint:* Use $\mathbf{AG}P \leftrightarrow \neg\mathbf{EF}\neg P$, case 9, and the fact that $W \setminus \tau_{\mathbf{EX}}(W \setminus Z) = \tau_{\mathbf{AX}}(Z)$.

2. Explain why $\mu Z.([\![P]\!] \cap \tau_{\mathbf{EX}}(Z)) = \emptyset$ and $\nu Z.([\![P]\!] \cup \tau_{\mathbf{EX}}(Z)) = W$ in every computation structure. Why does this make them useless for model checking?

3. In the mutual exclusion structure, compute $[\![\mathbf{E}[\neg c_2 \mathbf{U} c_1]]\!]$ and $[\![\mathbf{AF}c_2]\!]$ by fixpoint iteration. Does state 0 satisfy $\mathbf{AG}(t_2 \rightarrow \mathbf{AF}c_2)$?

4. Give a computation structure with n states and a formula $\mathbf{AF}p$ for which the naive fixpoint iteration takes n iterations before it stabilizes.

5. Show how to compute $[\![\mathbf{E}[P\mathbf{U}Q]]\!]$ in time $O(|K|)$, given $[\![P]\!]$ and $[\![Q]\!]$. *Hint:* Search backward from $[\![Q]\!]$.

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.