A

CMSC 452 CTL Lecture Notes: March 27, 2014 and April 1, 2014

Peter Fontana, University of Maryland

For additional information, see [Clarke et al. 1999; Cleaveland 2008].

Contents

1. MOTIVATING EXAMPLE: GATE AND TRAIN

Let us motivate this lecture with an example.

Example 1.1 (*Motivating example*). Consider a gate, given in Figure 1, and a train, given in Figure 2.

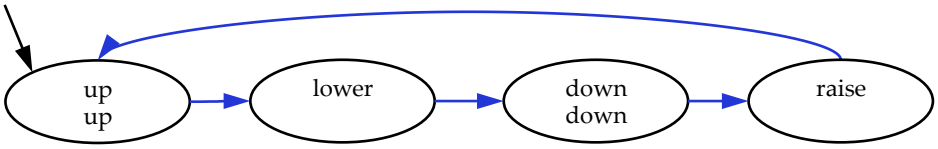

Fig. 1. Gate.

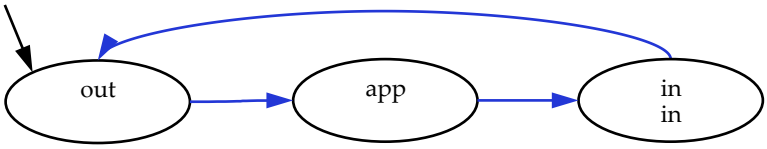

Fig. 2. Train.

We can consider a system of this gate and train modeled together as the automaton in Figure 3. In this model, the states have abbreviations based on the models of the train and gate. The abbreviations corresponding to the gate are:

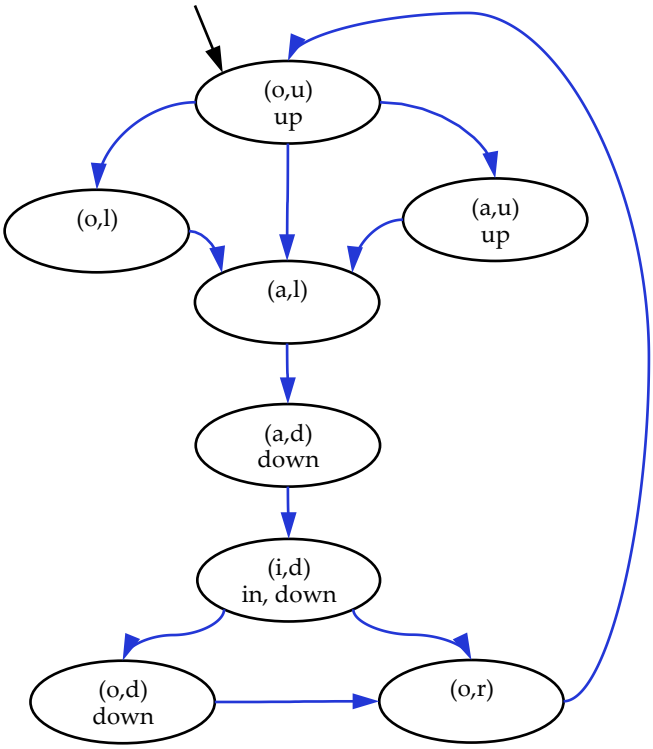

Fig. 3. Model of system with train and gate.

— u: up,
— l: lower,
— d: down, and
— r: raise.

The abbreviations corresponding to the train are:

— o: out,
— a: approach, and
— i: in.

For instance, the state (o, u) means that the train is out and the gate is up.

With this model, we wish to verify two properties:

(1) It is always the case that when the train is in, the gate is down.
(2) It is always the case that if the gate is down, it will inevitably be up.

Furthermore, we wish to have a computer verify that this model satisfies these properties.

We begin this lecture by defining a framework to specify this model and these properties. Then we give an algorithm to formally verify if this model satisfies these properties or not.

2. FRAMEWORK: LABELED AUTOMATA

Notice that the current model is a **finite automaton**. However, rather than labeling actions, we have chosen to label some states (nodes) with atomic propositions. Note that actions can still be labeled (for definitions, every edge has action τ).

Definition 2.1 (*Transition system* $TS = (Q, q_0, \Sigma, \longrightarrow)$). A *transition system* $TS = (Q, q_0, \Sigma, \longrightarrow)$ is a tuple where:

— Q is the set of states.
— $q_0 \in Q$ is the initial state.
— Σ is the set of actions, labels or action symbols.
— $\longrightarrow :\subseteq Q \times \Sigma \times Q$ is the transition relation (need not be a function) that if $(q, a, q') \in \rightarrow$, then the TS can transition from state q to state q' on label a.
Here $q \xrightarrow{a} q'$ is a notation for $(q, a, q') \in \longrightarrow$. A transition system is also called a *labeled transition system* (LTS) or *concrete transition system* (CTS).

Transition systems are the NFA or DFA that you are used to.

We now augment a transition system to a **labeled automaton** or **Kripke Structure** by adding atomic propositions (labels on states).

Definition 2.2 (*Labeled automaton* $LA = (Q, q_0, \Sigma, \longrightarrow, AP, L)$). A *labeled automaton* or *Kripke structure* $LA = (Q, Q_0, \Sigma, \longrightarrow, AP, L)$ is a tuple where:

— Q is the set of states.
— $q_0 \in Q$ is the initial state.
— Σ is the set of actions, labels or action symbols. (For this document, labeled automata will not have any action symbols.)
— $\longrightarrow \subseteq Q \times \Sigma \times Q$ is the transition relation (need not be a function) that if $(q, a, q') \in \rightarrow$, then the TS can transition from state q to state q' on label a.
— AP is the set of atomic propositions (or labels).
— $L : Q \longrightarrow 2^{AP}$ is a labeling function where $L(q)$ gives the set of atomic propositions that q satisfies.

Furthermore, we make a modeling restriction: every state must have some **outgoing transition**.

Atomic propositions are similar to boolean variables in propositional logic.

The restriction of making every state have an outgoing transition makes modeling easier. We will model a **dead state** by giving it a self-loop and no other outgoing transitions. Notice that these automata can be deterministic or nondeterministic.

With this definition we can now label states with propositions that satisfy them. We will take a look.

Example 2.3 (*Gate*). Consider the gate in Figure 1. For this automaton, $AP = \{down, up\}$, $L(up) = \{up\}$, $L(lower) = \emptyset$, $L(down) = \{down\}$, and $L(raise) = \emptyset$. All actions (Σ) are the generic action τ.

Example 2.4 (*Motivating Example*). Consider the model in Figure 3. Here, $AP = \{up, down, in\}$. Here are some of the states of the labeling function L:
$L((o, u)) = \{up\}$,
$L((a, l)) = \emptyset$, and
$L((i, d)) = \{in, down\}$.

3. FRAMEWORK: COMPUTATION TREE LOGIC (CTL)

To specify properties, we encode properties by writing them in a **logic**. For this lecture, we will use a subset of the branching-time logic, Computation Tree Logic (CTL).

3.1. Syntax

We will work with a subset of CTL. We will call this fragment of CTL $CTL_{F,G,X}$.

Notice that there are path quantifiers: E and A, and state quantifiers: F, G, and X. In CTL we require that every path quantifier is followed by a state quantifier, and that every state quantifier is preceded by a temporal operator. Also note that this logic is closed under negation. Our goal is to only model with F and G, but we will use the quantifier X when verifying F and G.

Definition 3.1 ($CTL_{F,G,X}$ *syntax*). A $CTL_{F,G,X}$ *formula* can be constructed with the following grammar:

$$\phi ::= p \mid \texttt{tt} \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid EX(\phi) \mid AX(\phi) \mid EG[\phi_1] \mid EF[\phi_1] \mid AG[\phi_1] \mid AF[\phi_1]$$

Here, $p \in 2^Q$ is an atomic proposition (a subset of locations).

We will also use abbreviations for derived operators, which include: $\texttt{ff}$ for $\neg\texttt{tt}$, $\phi_1 \vee \phi_2$ for $\neg(\neg\phi_1 \wedge \neg\phi_2)$, and $\phi_1 \rightarrow \phi_2$ for $\neg\phi_1 \vee \phi_2$. Notice that this logic is closed under negation.

Note that given $EX(\phi)$, $EF[\phi]$ and $AF[\phi]$, we can derive the other temporal operators: $AX(\phi)$, $EG[\phi]$ and $AG[\phi]$.

The operators have informal meanings. Notice that the path quantifiers are always next to state quantifiers. This is a property of CTL that makes CTL easy to reason with.

Here is the informal meaning of each operator:

— A**:** For all paths.
— E**:** There exists a path.
— X**:** Next.
— F**:** Eventually.
— G**:** Always.

For computational reasons, we always place an A or an E (path quantifier) immediately before a temporal operator: X, F, or G. This makes the logic verifiable (over labeled automata) in polynomial time. Putting these together, we can think of the formulas with certain acronyms:

— AX: for all next states.
— EX: for some next state.
— AG: always.
— EF: possibly. Notice that $q \models EF[\phi]$ if and only if starting at state q, one can reach a state satisfying ϕ.
— AF: inevitably.
— EG: there is some path such that ... is always true.

3.2. Paths

Now, we can give the formal semantics of this logic. Each formula is interpreted over a state q of the automaton. To formalize this definition, we define paths in an automaton

Definition 3.2 (*Path*). Let LA be a labeled automaton. Then an execution or **path** π in LA is a (countably) infinite sequence of states $q_1, q_2, \ldots$ such that:

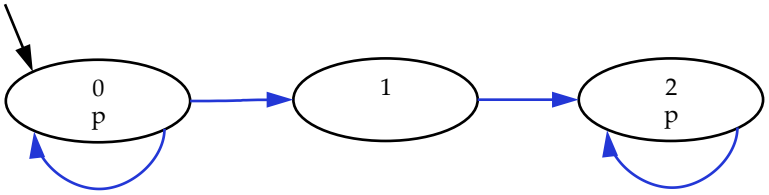

Fig. 4. Additional Automaton A.

— $q_1 = q_0$, the initial state of LA.
— For all i, $q_i \xrightarrow{a} q_{i+1}$

A path π in LA starting at state q is a path where $q_1 = q$.

To not confuse notation, we have paths start at index 1. (Recall that q_0 is the notation for the initial state of the automaton). In the definition, to handle an automaton with a finite path, we utilize the assumption that every state has a transition to allow an infinite sequence. Because every state can take some sequence, any finite sequence can be extended into an infinite sequence by taking that transition. The motivation behind this modeling choice is the intention that desirable models have no dead states.

Let us explore some paths.

Example 3.3 (*Gate*). Consider the gate in Figure 1. It has one path from state up, which is the path $up \longrightarrow lower \longrightarrow down \longrightarrow raise \longrightarrow up \ldots$.

Example 3.4 (*Additional Example*). Consider the automaton in Figure 4. From state 0 there are a variety of paths, but of two kinds. The first path stays in state 0 forever. The second kind stays in state 0 for a finite number of steps (perhaps no transitions) and then goes to state 1, then to state 2, and then stays in state 2 forever. One such path is $\pi = 0 \longrightarrow 0 \longrightarrow 1 \longrightarrow 2 \longrightarrow 2 \ldots$.

Paths are **infinite**. However, when illustrating if a labeled automaton satisfies a formula or not, I will use path segments to illustrate the point. These path segments can be extended to form paths in the labeled automata.

3.3. Semantics

Now with the definition of paths, we can give the formal $CTL_{F,G,X}$ semantics.

Definition 3.5 ($CTL_{F,G,X}$ *semantics*). Given a labeled automaton LA, a $CTL_{F,G,X}$ formula ϕ, and a state $q \in Q$, we say that a state q satisfies ϕ, written as $q \models \phi$ if and only if:

— $q \models p$ iff $p \in L(q)$.
— $q \models \neg\phi$ iff $q \not\models \phi$.
— $q \models \phi_1 \wedge \phi_2$ iff $q \models \phi_1$ and $q \models \phi_2$.
— $q \models \phi_1 \vee \phi_2$ iff $q \models \phi_1$ or $q \models \phi_2$.
— $q \models EX\,(\phi)$ iff there exists a state q' such that $q \longrightarrow q'$ and $q' \models \phi$.
— $q \models EF\,[\phi]$ iff there exists a path π starting at q and there is some index $j \geq 1$ such that $q_j \models \phi$.
— $q \models EG\,[\phi]$ iff there exists a path π starting at q and for all $j \geq 1$, $q_j \models \phi$.
— $q \models AX\,(\phi)$ iff for every state q' such that $q \longrightarrow q'$, then $q' \models \phi$.
— $q \models AF\,[\phi]$ iff for all paths π starting at q, there is some index $j \geq 1$ such that $q_j \models \phi$.
— $q \models AG\,[\phi]$ iff iff for all paths π starting at q, and for all $j \geq 1$, $q_j \models \phi$.

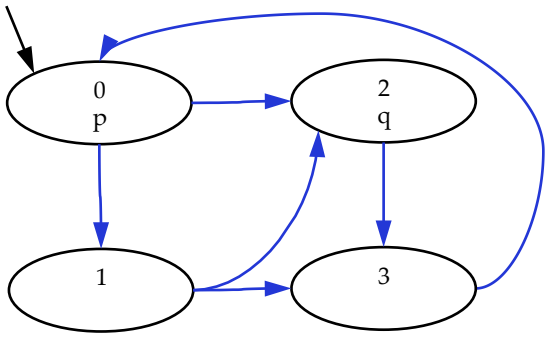

Fig. 5. Additional Automaton B.

A labeled automaton LA satisfies a formula ϕ, denoted as $LA \models \phi$ if and only if $q_0 \models \phi$.

Note that for eventually, the i can be the original state q (Paths where $q_1 = q$).

By definition, this logic is closed: $AF[\phi] \equiv \neg EG[\neg\phi]$ and $AG[\phi] \equiv \neg EF[\neg\phi]$. Additionally, $AX(\phi) = \neg EX(\neg\phi)$ Hence, $AF[\phi]$ is the dual of $EG[\phi]$, $AX(\phi)$ is the dual of $EX(\phi)$, and $AG[\phi]$ is the dual of $EF[\phi]$.

Here are some examples of some properties and their satisfaction.

Example 3.6 (*Satisfaction example 1*). Consider the automaton in Figure 4. Here are some properties and satisfaction examples:

— $0 \models p$, and $1 \not\models p$.
— $0 \models EG[p]$,
— $0 \not\models AG[p]$,
— $2 \models AG[p]$.

Example 3.7 (*Satisfaction example 2*). Consider the automaton in Figure 5. Two properties of interest are: $AF[q]$ and $EF[q]$. Here are some properties and satisfaction examples:

— $0 \models EX(q)$,
— $0 \not\models AX(q)$,
— $3 \models EX(p)$,
— $3 \models AX(p)$
— $0 \models EF[q]$.
— $0 \not\models AF[q]$. One such path where q is not inevitably true

Now with this language, we can write down the properties that we want the model to satisfy.

Example 3.8 (*Motivating example properties*). Recall the two properties we wanted to verify. These are:

(1) It is always the case that when the train is in, the gate is down.
(2) It is always the case that if the gate is down, it will inevitably be up.

The first can be written in $CTL_{F,G,X}$ as $AG[in \rightarrow down]$, and the second property can be written as $AG[down \rightarrow AF[up]]$. As a result, we can express the properties we would like to express.

For model checking purposes, we will often consider $AX(S)$ and $EX(S)$, where $S \subseteq Q$ is a set of states. We say that $q \in AX(S)$ if and only if for every q' such that $q \longrightarrow q'$, then $q' \in S$; and we say that $q \in EX(S)$ if and only if there is some q' such that $q \longrightarrow q'$ and $q' \in S$.

4. VERIFYING THE PROPERTIES: CTL MODEL CHECKING ALGORITHM

Now we want to verify that a model satisfies (or does not satisfy) a given CTL property. Do to this, we need an algorithm. The algorithm is recursive. To handle a temporal operator such as $AF[\phi]$, we first label all the states that satisfy ϕ (not labelling a state means that the state does not satisfy ϕ). Then we verify the temporal operator.

Verifying the temporal operators are a bit trickier. The easiest are $EX(\phi)$ and $AX(\phi)$, since we either check one next state, or all next state. However, to verify $AF[\phi]$ and $AG[\phi]$, we need to work more carefully.

The benefit of using CTL to specify properties is that the verification is polynomial in the size of the automaton and the size of the formula (linear in the product of them).

4.1. Representing the Formulas Recursively

The key to verifying these formulas is to get **recursive representations** of the formulas.

Definition 4.1 (*Fixpoint*). A **fixpoint** of a function f is an input x such that $f(x) = x$.

Fixpoints are valuable because they provide termination to recursion. In our case, we will make the formulas recursive and terminate over a fixpoint. The recursive formulas we use are:

$$EF[\phi] \stackrel{\mu}{=} \phi \vee EX(EF[\phi]) \tag{1}$$

$$EG[\phi] \stackrel{\nu}{=} \phi \wedge EX(EG[\phi]) \tag{2}$$

$$AF[\phi] \stackrel{\mu}{=} \phi \vee AX(AF[\phi]) \tag{3}$$

$$AG[\phi] \stackrel{\nu}{=} \phi \wedge AX(AG[\phi]) \tag{4}$$

Here, we label a least fixpoint with μ and a greatest fixpoint with ν. If we read these equations, the first one ($EF[\phi]$) reads as, "Possibly ϕ" means that either ϕ is true now or there is some next state where Possibly ϕ is true. For the time being, think of $\equiv$ as equiv, and the symbols ν, μ as keys for initializations for the algorithm.

4.2. Global Algorithm: Examining All States

Rather than using a local on-the-fly algorithm, one can model check a formula for all states of the automaton. This also utilizes the recursive nature. The key is to write the formula using a **variable** to store the current set of states that satisfy

(1) Take a recursive formula (see previous equations) and use a variable Y to store the set of states that currently satisfy the formula
(2) If the formula has a least fixpoint μ, we initialize $Y = \emptyset$. Else, if the formula has a greatest fixpoint ν, we initialize $Y = Q$.
(3) Iterate the formula, changing Y to be the set of states that currently satisfies the formula.
(4) Repeat iterations until we have a fixpoint (Y does not change value within an iteration).

There is quite a bit of mathematics behind the proof of this. Note that to solve a nested formula, we solve the innermost equations first, and then solve the outside formula. Because our space is finite, this is guaranteed to terminate.

Think of recursing over a greatest (ν) fixpoint formula as weeding out states that do not satisfy the property, and think of recursing over a least (μ) fixpoint formula as searching for states that do satisfy the property.

Let's look at some examples.

Example 4.2 (*First example*). Consider the automaton in Figure 4 and the forumla $AG\,[p]$. We use Y as our placeholder for the set of states that satisfy $AG\,[p]$. First, we use the recursive equation, giving us the equation:

$$Y \overset{\nu}{=} p \wedge AX\,(Y)\,. \tag{5}$$

Since we have a greatest fixpoint, we initialize $Y = \{0, 1, 2\}$, which is all possible states.

Step 1: $Y = \{0, 1, 2\}$, so we recurse with the equation

$$Y \overset{\nu}{=} p \wedge AX\,(\{0, 1, 2\}) \tag{6}$$

Since the set of states $\{0, 2\}$ satisfies p and the set of states $\{0, 1, 2\}$ satisfies $AX\,(\{0, 1, 2\})$, we conjunct them (because of the $\wedge$ and get $Y = \{0, 2\}$.

Step 2: $Y = \{0, 2\}$, so we recurse with the equation

$$Y \overset{\nu}{=} p \wedge AX\,(\{0, 2\}) \tag{7}$$

Since the set of states $\{0, 2\}$ satisfies p and the set of states $\{1, 2\}$ satisfies $AX\,(\{0, 2\})$ (0 can transition to state 1), we conjunct them (because of the $\wedge$ and get $Y = \{2\}$.

Step 3: $Y = \{2\}$, so we recurse with the equation

$$Y \overset{\nu}{=} p \wedge AX\,(\{2\}) \tag{8}$$

Since the set of states $\{0, 2\}$ satisfies p and the set of states $\{1, 2\}$ satisfies $AX\,(\{2\})$ (0 can transition to state 1), we conjunct them (because of the $\wedge$ and get $Y = \{2\}$. Since Y is unchanged, we stop.

We now know that $0 \not\models AG\,[p]$, $1 \not\models AG\,[p]$ and $2 \models AG\,[p]$.

Example 4.3 (*Motivating example*). Recall the automaton in Figure 3, and the $CTL_{F,G,X}$ $AG\,[in \rightarrow down]$. We will model check this with the global algorithm.

Equation. We use a predicate variable Y to represent the set of states that satisfy the formula $AG\,[in \rightarrow down]$. Using the recursion, we have the equation

$$Y \overset{\nu}{=} (in \rightarrow down) \wedge AX\,(Y) \tag{9}$$

Initialization: $Y = \{(o, u), (o, l), (a, u), (a, l), (a, d), (i, d), (o, d), (o, r)\}$.

Step 1: We apply the equation. Each state satisfies $(in \rightarrow down)$, and each state has some outgoing transition to Y. Hence Y is unchanged and we have a fixpoint.

Conclusion: Hence, the set of states satisfying $AG\,[in \rightarrow down] = \{(o, u), (o, l), (a, u), (a, l), (a, d), (i, d), (o, d), (o, r)\}$. Notice that the initial state, (o, u) is in this set.

Now we model check the formula $AG\,[down \rightarrow AF\,[up]]$. To do this, we first find all the states that satisfy $AF\,[up]$. Here is the equation:

$$Y \overset{\nu}{=} up \vee AX\,(Y) \tag{10}$$

Since we have a least fixpoint μ, we initialize $Y = \emptyset$.

Step 1: $Y = \emptyset$ and the current equation is

$$Y \overset{\mu}{=} up \vee AX\,(\emptyset) \tag{11}$$

Since every state has some transition, no state is in the set $AX(\emptyset)$, but some states satisfy up. Hence, $Y = \{(o,u),(a,u)\}$.

Step 2: $Y = \{(o,u),(a,u)\}$ and the current equation is

$$Y \stackrel{\mu}{=} up \vee AX(\{(o,u),(a,u)\}) \tag{12}$$

We have that $\{(o,r))\}$ satisfies $AX((o,u),(a,u)))$. Since we have an $\vee$, we union this with the set of states that satisfy up. Hence, $Y = \{(o,u),(a,u),(o,r)\}$.

Step 3: $Y = \{(o,u),(a,u),(o,r)\}$ and the current equation is

$$Y \stackrel{\mu}{=} up \vee AX(\{(o,u),(a,u),(o,r)\}) \tag{13}$$

We compute this formula in a similar fashion to previous steps, yielding $Y = \{(o,u),(a,u),(o,d),(o,r)\}$.

Step 4: $Y = \{(o,u),(a,u),(o,d),(o,r)\}$ and the current equation is

$$Y \stackrel{\mu}{=} up \vee AX(\{(o,u),(a,u),(o,d)(o,r)\}) \tag{14}$$

We compute this formula in a similar fashion to previous steps, yielding $Y = \{(o,u),(a,u),(i,d),(o,d),(o,r)\}$.

If we repeat this process (more steps), we will eventually get $Y = \{(o,u),(o,l),(a,u),(a,l),(a,d),(i,d),(o,d),(o,r)\}$ which is the entire state space, and will get a fixpoint. Hence we know that every state satisfies $AF[up]$.

Now, we treat $AF[up]$ as a formula satisfied by every state, and use another variable Y_2 to track the set of states that satisfies $AG[down \rightarrow AF[up]]$. We have the equation:

$$Y_2 \stackrel{\nu}{=} (down \rightarrow AF[up]) \wedge AX(Y_2) \tag{15}$$

Since we have a greatest fixpoint, we initialize $Y_2 = \{(o,u),(o,l),(a,u),(a,l),(a,d),(i,d),(o,d),(o,r)\}$.

From doing the comuptation, you will find that Y_2 is unchanged in the first iteration, and we have a fixpoint. Therefore, every state, including the initial state (o,u), satisfies $AG[down \rightarrow AF[up]]$.

5. CMSC 452 LECTURE QUICK FEEDBACK FOR MAR 27, 2014

Here is a non-graded exercise for me to understand your understanding of the class.

Name:

Consider the automaton in Figure 6.

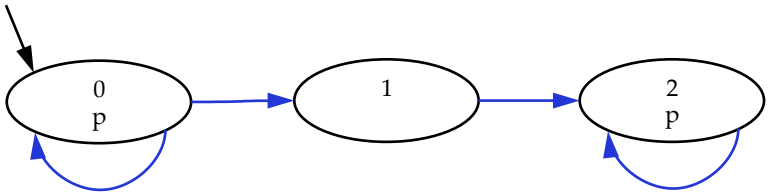

Fig. 6. A simple automaton.

Exercise 1 (*Labeling function*). Assuming that $AP = \{p\}$, please give the complete labeling function L:

$L(0) =$
$L(1) =$
$L(2) =$

Exercise 2 (*Formula satisfaction*). Does state 0 satisfy $AF\,[p]$?

Exercise 3 (*Formula satisfaction*). Does state 0 satisfy $AG\,[p]$?

6. $CTL_{F,G,X}$ LAB EXERCISES FOR APRIL 1, 2014

Exercise 4 (*A familiar automaton*). Consider the automaton in Figure 7.

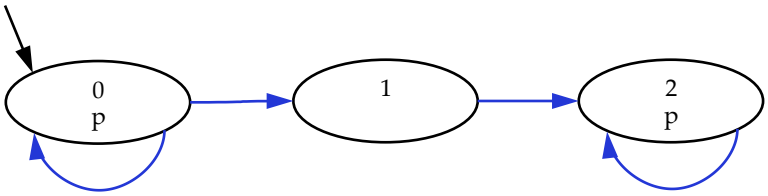

Fig. 7. A simple automaton.

Using the algorithm, compute the set of states that satisfies $AF\,[p]$.

Exercise 5 (*Anamolies in satisfaction*). Consider the two automata G_1 and G_2 in Figure 8.

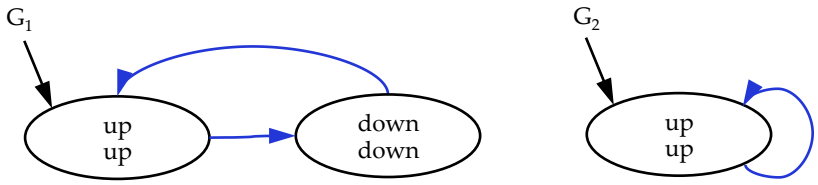

Fig. 8. Two gate models G_1 and G_2.

First, compute the set of states in G_1 that satisfy the formula $AG\,[down \rightarrow AF\,[up]]$. Now show that in model G_2, $up \models AG\,[down \rightarrow AF\,[up]]$.

Next, give a $CTL_{F,G,X}$ property that up in G_1 satisfies but up in G_2 does not satisfy.

Exercise 6 (*Examining another automaton*). Consider the automaton in Figure 9.

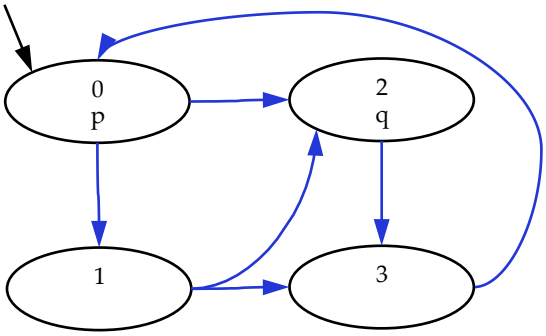

Fig. 9. Another automaton.

Compute the set of states that satisfies $AF\,[q]$. Now compute the set of states that satisfies $EF\,[q]$.

REFERENCES

Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. 1999. *Model Checking*. The MIT Press, Cambridge, MA, USA.

Rance Cleaveland. 2008. CMSC 630 Spring 2008 Lecture Slides. (May 2008). Class Notes.