

CS 4123 fall 2009: Notes on LTL and Verification

Dan Dougherty
WPI

version of *October 15, 2009 – 15: 00*

Other (more complete) sources: [CGP99] and [BBF+99].

1 Modeling Systems

Informally:

- We think of a system as being in a certain “state” at each moment of time.
- Certain states are identified as initial states.
- From each state we may transition to any of several possible successor states; these transitions are naturally associated with actions, or events.
- At each state we can observe certain properties (for example the current values of the system’s variables) that distinguish it from other states.

Formally: *see Section 3.2.2 of Huth and Ryan.*

2 Temporal Logic

There are a variety of temporal logic, logics that incorporate some notion of time. The currently most successful in systems verification applications are obtained by adding temporal operators to standard propositional logic. Within this class there are still several variations; see [CGP99] or [BBF+99] for more. Here we focus on one particularly important logic, propositional linear temporal logic.

The material in sections 2.1 and 2.2 here is also in Huth and Ryan, Section 3.2

2.1 Syntax

Fix a set Prop of propositions. The formulas of *propositional linear temporal logic* (PLTL) are given by

$$\alpha ::= A \in \text{Prop} \mid \neg\alpha \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \\ \mid X\alpha \mid F\alpha \mid G\alpha \mid \alpha W \beta \mid \alpha U \alpha \mid \alpha R \alpha$$

Informally we read X as “next,” F as “eventually,” G as “always,” U as “until,” R as “releases,” and W as “weak until.” The formal semantics of these connectives is defined, of course, in terms of models.

2.2 Semantics

A *model* of PLTL is a sequence $\pi : \omega \rightarrow 2^{\text{Prop}}$. Here the Greek letter ω , pronounced “omega” stands for the set of natural numbers $\{0, 1, 2, \dots\}$.¹ The notation “ 2^ω ” refers to the set of all subsets of ω . Think of ω as time, so that π assigns a set of propositions to each time point. Think of $\pi(i)$ as declaring which propositions are true at time i .

Note that an execution in a transition system yields a model of PLTL in a canonical way.

Definition 2.1. Fix a set Prop of propositional atoms; let π be a PLTL model over Prop and let α be a formula over Prop.

To define the notion α is true in π , written $\pi \models \alpha$, we first define for each $i \in \omega$ the auxiliary notion *formula α is true in π at index i* , written $\pi, i \models \alpha$. We then treat $\pi \models \alpha$ as a shorthand for $\pi, 0 \models \alpha$.

The definition of $\pi, i \models \alpha$ is as follows.

- for $A \in \text{Prop}$: $\pi, i \models A$ if $A \in \pi(i)$;
- $\pi, i \models \neg\alpha$ if it is not the case that $\pi, i \models \alpha$;
- $\pi, i \models (\alpha \wedge \beta)$ if $\pi, i \models \alpha$ and $\pi, i \models \beta$
- $\pi, i \models (\alpha \vee \beta)$ if $\pi, i \models \alpha$ or $\pi, i \models \beta$
- $\pi, i \models X\alpha$ if $\pi, (i+1) \models \alpha$;
- $\pi, i \models F\alpha$ if there exists $j \geq i$ such that $\pi, j \models \alpha$;
- $\pi, i \models G\alpha$ if for all $j \geq i$, $\pi, j \models \alpha$;
- $\pi, i \models (\alpha U \beta)$ if there exists $k \geq i$ such that
 - $\pi, k \models \beta$, and
 - for all $i \leq j < k$, $\pi, j \models \alpha$;
- $\pi, i \models (\alpha R \beta)$ if either
 - for all j with $i \leq j$, $\pi, j \models \beta$, or
 - there exists k with $k \geq i$ such that
 - * $\pi, k \models \alpha$ and
 - * for all j with $i \leq j \leq k$ we have $\pi, j \models \beta$
- $\pi, i \models (\alpha W \beta)$ if either
 - for all j with $i \leq j$, $\pi, j \models \alpha$, or
 - there exists $k \geq i$ such that
 - * $\pi, k \models \beta$, and
 - * for all j with $i \leq j < k$, $\pi, j \models \alpha$;

¹In certain areas of mathematics this use of ω for the natural numbers is standard notation, and it is commonplace in verification, so we have inherited it . . .

2.3 Systems and (LTL) properties

Note the difference between a transition system and an LTL model. A transition system is a directed graph of nodes, with each node functioning as a propositional logic model, while LTL model is a single infinite sequence of such nodes. Each execution of \mathcal{M} , that is, each path in \mathcal{M} , is itself an LTL model.

The approach to system verification that we are developing here proceeds as follows.

Suppose \mathcal{M} is a transition system over the atoms Prop (see Huth and Ryan, Def 3.4). We say that α is true in \mathcal{M} , written $\mathcal{M} \models \alpha$, if for every execution π of \mathcal{M} , considered as a PLTL model, we have $\pi \models \alpha$.

Satisfiability The *satisfiability* problem for PLTL is the following

INPUT: a PLTL formula α

QUESTION: Does there exist a model π such that $\pi \models \alpha$?

Model checking The *model checking* problem for PLTL is the following

INPUT: a transition system \mathcal{M} and a PLTL formula α

QUESTION: Does $\mathcal{M} \models \alpha$?

We are going to show that these problems are each decidable. Let's explore a little to see if we should *expect* them to be decidable.

First note that any LTL model is an infinite object, since it has infinitely many nodes. So in attempting to solve the satisfiability problem for a formula α we cannot hope to resort to exhaustive search as we did with ordinary propositional logic. So it should come as a bit of a surprise to learn that LTL satisfiability is in fact decidable.

On the other hand model checking is about transition systems, which are finite objects, so that seems promising. But to ask whether α is true in a transition system \mathcal{M} is to ask about *all* paths through \mathcal{M} . Now consider the next exercise.

Exercise 2.2.

1. Give an example of a transition system with finitely many executions.
2. Give an example of a transition system with infinitely many executions. If you know about countable and uncountable sets, give two examples, one whose set of executions is countably infinite, and one whose set of executions is uncountably infinite.

Since deciding the truth of a PLTL formula in a transition system involves quantifying over all executions, Exercise 2.2 suggests that this problem might be undecidable.

2.4 The set of executions as a tree

CS 4123 note: this material is optional: you won't be quizzed on Section 2.4

Observe that the set of executions of a transition system naturally forms a tree defined by "unfolding." It is often useful to pay attention to this extra structure on the space of executions.

Let t be an infinite tree whose nodes are associated with data of some sort. For example, t might be the set of executions of a transition system starting with a particular start node. Each node n of t determines a *subtree* of t , namely the set of all descendants of n (including n itself). Two trees t_1 and t_2 are *isomorphic* if there is a bijection between the nodes preserving the labels. Note that since t is an infinite tree it can certainly happen that two trees t_1 and t_2 are isomorphic even if one is a proper subtree of the other. Say that t is *regular* if it has only finitely many different subtrees up to isomorphism.

Exercise 2.3.

1. Give an example of a regular tree.
2. Give an example of a tree (over a finite set of labels) that is not regular.
3. Let \mathcal{M} be a transition system with a single initial state. So the executions of \mathcal{M} naturally form a tree E (where the labels are the assignments to AP). Show that E is regular.

Since deciding the truth of a PLTL formula in a transition system involves quantifying over all executions organized into a regular tree, The last part of Exercise 2.3 suggests that this problem might be decidable.

2.5 Practice

Exercise 2.4. Let P be the transition system in Figure 1 For each system property below decide whether it is satisfied by P . If your answer is “no” give a path which falsifies the property.

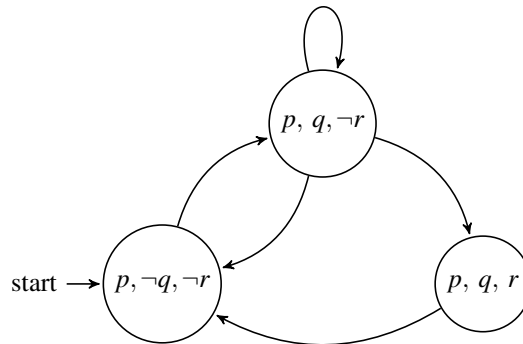


Figure 1: Model for Exercise 2.4

1. $G(p \rightarrow Fq)$
2. $G(p \rightarrow Fr)$
3. $G((\neg r) U q)$
4. $GF\neg r$
5. $FG\neg r$

Exercise 2.5. Let \mathcal{M} be the model (taken from [CGP99]) of the microwave oven found in Figure 2.5. The figure shows the labeling, which you then interpret as giving a truth value to each proposition at each state. For example, at the initial state, all the propositions are false; at the state immediately below that the door is closed, but start has not been requested, the heat is not on and there is no error . . .

For each of the following system properties, paraphrase it in English, then decide whether it holds of \mathcal{M} . This means: decide whether the property is true of all executions of \mathcal{M} . If you answer “no”, define a path falsifying the property.

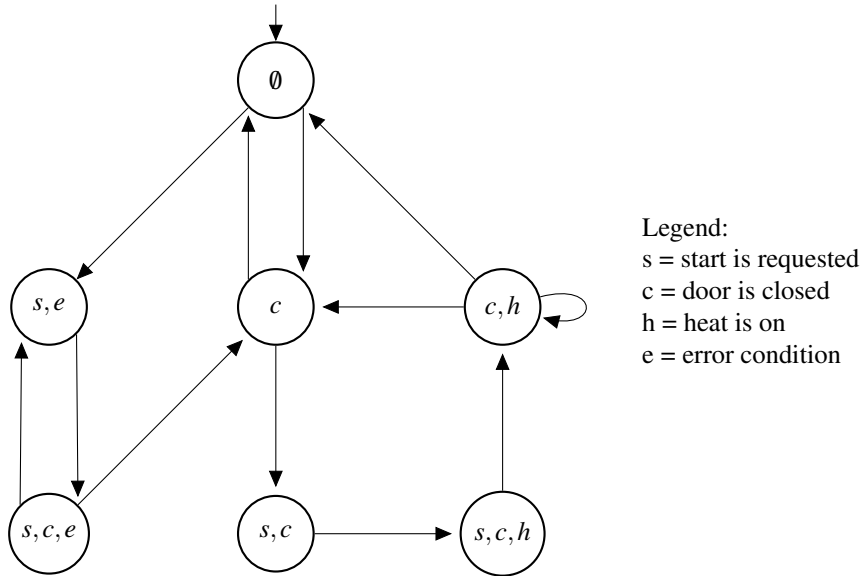


Figure 2: Modeling an oven (from [CGP99])

1. $G(\neg c \rightarrow \neg h)$
2. $G(e \rightarrow \neg h)$
3. $G(s \rightarrow (Fh))$
4. $G((\neg s \wedge c \wedge Xs) \rightarrow (Fh))$
5. $G((\neg s \wedge c) \rightarrow X(s \rightarrow (Fh)))$
6. $G(e \rightarrow (e \cup c))$
7. $G(e \rightarrow ((e \cup c) \vee Ge))$

2.6 From natural language to PLTL

Some examples.

- We can say that P cycles on and off infinitely often: $G(FP \wedge F\neg P)$
- We can say that P and Q alternate, in the sense that they are mutually exclusive and each happens infinitely often. Let α be the pure propositional formula over P and Q that expresses that exactly one of P and Q holds. Then the desired formula is $G\alpha \wedge GFP \wedge GFQ$

- More interestingly we can say that P , Q , and R cycle in the pattern, “ P , then Q , then R , infinitely often” (think about a traffic light). Exercise.
- How might we translate P is true before Q ? This is ambiguous in English: here are some different plausible readings, with their translations into PLTL:
 - if Q ever becomes true then P is true at all times before that: $FQ \rightarrow (P U Q)$
 - or if Q ever becomes true then P is true at some time before that: $(P R \neg Q)$
 - or, Q will be true and P is true at all times before that: $P U Q$
 - or, Q will be true at some time and P is true at some time before that: $F(P \wedge FQ)$
 - or P becomes true while Q is (still) false: $\neq Q W (P \wedge \neg Q)$
- How might we translate P is false after Q ? Different readings:
 - if Q ever becomes false then P will be false thereafter
 - if Q ever becomes false then P will be false at least one time after that
 - Q will become false, and P will be false thereafter
 - Q will become false, and P will be false at least one time after that.
- Can we say in PLTL that P is true at exactly one time? Yes: $F(p) \wedge G(p \rightarrow XG(\neg p))$.
Indeed for each k we can write a formula saying that P holds at exactly k points.
- We can say that there is exactly one *time interval* when P is true: $F(P) \wedge G(P \rightarrow G((\neg P) \rightarrow G(\neg P)))$
Similarly we can say that there are exactly two intervals in which P holds. (Exercise)

By the way, the fact that the same English sentence can give rise to so many plausible translations into LTL should not make you suspicious about LTL: quite the contrary. A formal language such as LTL can play an important role in disambiguating English “specifications:” two people can say or hear the same English sentence and think it means very different things, but an LTL sentence exactly one thing. This is useful in making sure that the specifier and implementer agree about their goals.

2.7 Entailment, equivalence, etc

As in any logic, if α and β are PLTL formulas, we say that α *entails* β if every model of α is a model of β , and that α and β are *logically equivalent* if they entail each other.

Exercise 2.6. For each of the following pairs of formulas, determine whether they are equivalent or, if not, whether one of them entails the other. Give proofs or countermodels.

1. $F(\alpha \wedge \beta)$ and $(F\alpha \wedge F\beta)$
2. $F(\alpha \vee \beta)$ and $(F\alpha \vee F\beta)$
3. $G(\alpha \wedge \beta)$ and $(G\alpha \wedge G\beta)$
4. $G(\alpha \vee \beta)$ and $(G\alpha \vee G\beta)$
5. $(\alpha \wedge \beta) U \gamma$ and $((\alpha U \gamma) \wedge (\beta U \gamma))$
6. $(\alpha U (\beta \wedge \gamma))$ and $((\alpha U \gamma) \wedge (\beta U \gamma))$
7. ... replace \vee with \wedge in the previous two ...

8. ... now replace U with W, then with R ...

Exercise 2.7. Prove the following equivalences

1. $F\alpha$ is equivalent to $\neg G\neg\alpha$
2. $X\alpha$ is equivalent to $\neg X\neg\alpha$
3. $F\alpha$ is equivalent to $(\top U \alpha)$ where \top stands for any tautology
4. $G\alpha$ is equivalent to $(\alpha W \perp)$ where \perp stands for any unsatisfiable formula
5. $\alpha U \beta$ is equivalent to $\neg(\neg\alpha R \neg\beta)$
6. $\alpha U \beta$ is equivalent to $(\alpha W \beta) \wedge (F\beta)$
7. $\alpha W \beta$ is equivalent to $(\alpha U \beta) \vee (G\alpha)$
8. $\alpha W \beta$ is equivalent to $\beta R (\alpha \vee \beta)$
9. $\alpha R \beta$ is equivalent to $\beta W (\alpha \wedge \beta)$

Exercise 2.8. Show that any PLTL formula is equivalent to one built using only the temporal connectives X and U .

Show that any PLTL formula is equivalent to one built using only the temporal connectives X and W .

(Use exercise 2.7.)

3 Büchi automata

Recall that a non-deterministic finite automaton N is a tuple $N = (Q, \Sigma, \delta, Q_s, F)$ with Q a finite set of states, Σ an input alphabet, $Q_s \subseteq Q$ a set of start states, $F \subseteq Q$ a set of accepting states, and δ a transition relation: $\delta: Q \times \Sigma \rightarrow 2^Q$.

A convenient notation: write $q \xrightarrow{a} q'$ to mean $q' \in \delta(s, a)$.

Given a finite word $x \in \Sigma^*$ there is a natural notion of a *run* ρ of N on x : a run is a sequence of states determined by executing δ on x starting in state of Q_s . Because N is non-deterministic, there can be more than one possible run—or no runs—of N on x . The length of any run ρ is 1 + the length of x . A run ρ is an *accepting* run if it ends in a state in F . The word x is *recognized*, or *accepted*, by N if there is some accepting run on N on x . Then N determines a set $L(N) \subseteq \Sigma^*$ of finite words: the set of words that are recognized by N .

Now suppose $x \in \Sigma^\omega$ is an infinite word. There is again a natural notion of run of N on x : a run is again a sequence of states determined by executing δ on x starting in a state of Q_s . Now, of course, ρ is an *infinite* sequence of states. So what should we call an accepting run? It makes no sense to call ρ accepting if the last state is in F since there is no last state of ρ . It turns out that the following idea works very well: we say that an infinite run ρ is accepting if it hits F infinitely often. Then just as before we say that the (infinite) word x is *recognized*, or *accepted*, by N if there is some accepting run on N on x . The NFA N determines a set $L_\omega(N) \subseteq \Sigma^\omega$ of infinite words: the set of words that are recognized by N .

Some examples In class we saw Büchi automata recognizing the following languages.

1. Over the alphabet $\Sigma = \{a, b\}$, the language $L = \{x \mid x \text{ contains infinitely many } as \}$
2. Over the alphabet $\Sigma = \{a, b\}$, the language $L = \{x \mid x \text{ contains only finitely many } as \}$
3. Over the alphabet $\Sigma = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\}$, the language $L = \{x \mid \text{after every input symbol with a 0 in the top row, the next symbol has a 1 in the bottom row} \}$.

A convenient manner of speaking when working with an alphabet such as $\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\}$ is to think of each row as being associated with a propositional variable, let's say p for the top row and q for the bottom row. Then the symbol $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ can be described by saying “ p is false and q is true.” If we do this then the language just previously defined could be described as $L = \{x \mid \text{whenever } p \text{ is false then } q \text{ is true immediately afterwards} \}$.

In this spirit we can construct Büchi automata for the following languages over $\Sigma = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\}$:

1. $L = \{x \mid \text{eventually } p \text{ becomes true} \}$
2. $L = \{x \mid \text{at every occurrence of } \neg p, q \text{ holds until } p \text{ becomes true again (if it ever does)} \}$
3. ... you see where this is going, don't you? See the next section.

3.1 On the essentially nondeterministic nature of Büchi automata.

CS 4123 note: this material is optional: you won't be quizzed on Section 3.1

Definition 3.1. Let $L \subseteq \Sigma^*$. Then $\lim L \stackrel{\text{def}}{=} \{x \in \Sigma^\omega \mid \text{for infinitely many } k, x[0..k] \in L\}$

Lemma 3.2. Let \mathcal{A} be a deterministic automata. Then $L_\omega(\mathcal{A}) = \lim L(\mathcal{A})$.

Corollary 3.3. The following language over $\Sigma = \{a, b\}$ is not recognized by any deterministic Büchi automaton: $L = \{x \mid x \text{ has only finitely many } as \}$

Exercise 3.4. We saw that when \mathcal{A} is a deterministic automaton then $L_\omega(\mathcal{A}) = \lim L(\mathcal{A})$. Give an example to show that this claim can fail if we do not assume \mathcal{A} to be deterministic. Are either of the following claims true? (i) For arbitrary automata $L_\omega(\mathcal{A}) \subseteq \lim L(\mathcal{A})$; (ii) For arbitrary automata $L_\omega(\mathcal{A}) \supseteq \lim L(\mathcal{A})$.

3.2 Closure properties of Büchi automata

Lemma 3.5. Let \mathcal{A}_1 and \mathcal{A}_2 be Büchi automata over the same input alphabet. Then there is a Büchi automaton \mathcal{A} such that $L(\mathcal{A}) = L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$.

Proof. Proof idea: the simple classical construction, essentially taking the “union” of the given automata, works. (Here the fact that allow sets of start states make life easier.)

Formally: write \mathcal{A}_1 as $(Q_1, \Sigma, \delta_1, Q_{s1}, F_1)$, and \mathcal{A}_2 as $(Q_2, \Sigma, \delta_2, Q_{s2}, F_2)$. Define \mathcal{A} as $(Q, \Sigma, \delta, Q_s, F)$, where

- $Q = Q_1 \cup Q_2$;
- $Q_s = Q_{s1} \cup Q_{s2}$;

- $F = F_1 \cup F_2$;
- $\langle q_1, q_2 \rangle \xrightarrow{a} \langle q'_1, q'_2 \rangle$ if $q_1 \xrightarrow{a} q'_1$ or $q_2 \xrightarrow{a} q'_2$.

///

The (finite-word) regular languages are closed under intersection: this can be proved by the classical product construction on automata. In contrast to the situation with union, the classical construction does *not* work for Büchi automata. (*Exercise:* show this by giving two Büchi automata \mathcal{A}_1 and \mathcal{A}_2 such that when you build \mathcal{A}_3 according to the standard product construction, $L(\mathcal{A}_3) \neq L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$). This is an easy exercise, since almost any \mathcal{A}_1 and \mathcal{A}_2 you write down will suffice! Try it.)

But we can recover, by a relatively simple trick.

Lemma 3.6. *Let \mathcal{A}_1 and \mathcal{A}_2 be Büchi automata over the same input alphabet. Then there is a Büchi automaton \mathcal{A} such that $L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$.*

Proof. Proof idea: do a classical product construction augmented with a flag that keeps track of which automaton's accepting states were last visited.

Formally: write \mathcal{A}_1 as $(Q_1, \Sigma, \delta_1, Q_{s1}, F_1)$, and \mathcal{A}_2 as $(Q_2, \Sigma, \delta_2, Q_{s2}, F_2)$. Define \mathcal{A} as $(Q, \Sigma, \delta, Q_s, F)$, where

- $Q = Q_1 \times Q_2 \times \{1, 2\}$;
- $Q_s = Q_{s1} \times Q_{s2} \times \{1\}$;
- $F = F_1 \times Q_2 \times \{1\}$;
- $\langle q_1, q_2, i \rangle \xrightarrow{a} \langle q'_1, q'_2, j \rangle$ if $q_1 \xrightarrow{a} q'_1$ and $q_2 \xrightarrow{a} q'_2$ and $j = \begin{cases} 2 & \text{if } i = 1 \text{ and } q_1 \in F_1 \\ 1 & \text{if } i = 2 \text{ and } q_2 \in F_2 \\ i & \text{otherwise} \end{cases}$

///

Lemma 3.7. *Let \mathcal{A}_1 and \mathcal{A}_2 be Büchi automata over the same input alphabet. Then there is a Büchi automaton \mathcal{A} such that $L(\mathcal{A}) = \overline{L(\mathcal{A}_1)}$.*

Proof. This is a hard theorem; the proof is omitted here.

///

4 Automata and Temporal Logic

It turns out that there is an amazing connection between LTL formulas and Büchi automata, which we now outline. Roughly speaking:

- first we observe that LTL models can be thought of as ω -words; then
- we show that given any LTL formula α we can build a Büchi automaton \mathcal{A}_α which captures α in the sense that the language that \mathcal{A}_α recognizes “is” the space of all models where α is true.

Once we have this we can think of Büchi automata as being a clever data structure for representing LTL formulas, leading to clear algorithms for satisfiability and validity, and ultimately for model-checking.

See [Var96] (available at the course web page) for a complete treatment of PLTL model-checking, via Büchi automata.

4.1 Building automata for PLTL formulas

Fix a set $AP = \{p_1, \dots, p_n\}$ of atomic propositions. Let $2^{AP} = \{m \mid m : AP \rightarrow \{0, 1\}\}$ be the set of all truth assignments on AP , considered as an alphabet. For example when $AP = \{p_1, p_2\}$ we can picture 2^{AP} as the set

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\}.$$

Then a PLTL model $x : \omega \rightarrow 2^{AP}$ can also be viewed as an ω -word $x \in (2^{AP})^\omega$ when 2^{AP} is considered as an alphabet.

Theorem 4.1. *Fix a set $AP = \{p_1, \dots, p_n\}$ of atomic propositions. There is an algorithm that, given any PLTL formula α over AP , constructs a Büchi automaton \mathcal{A}_α with input alphabet 2^{AP} such that for any $x : \omega \rightarrow 2^{AP}$*

$$x \in \mathcal{A}_\alpha \quad \text{if and only if} \quad x \models \alpha$$

Proof. not given here.

///

The proof of Theorem 4.1 gives a generic construction for building an automaton from a formula, but you need not use such a construction to do the following exercises. It's better for building your intuitions if you attack these directly (and the automata will almost certainly be much smaller when you build them "by hand").

Exercises 4.2. For each property α below, construct a Buchi automaton \mathcal{A}_α over the alphabet 2^{AP} which accepts precisely those ω -sequences for which the property is true. (Note that when AP consists of only one proposition, 2^{AP} is just $\{0, 1\}$.)

1. $G p$ Here $AP = \{p\}$.
2. $F p$ Here $AP = \{p\}$.
3. $F p_2$ Here $AP = \{p_1, p_2\}$.
4. $X p$ Here $AP = \{p\}$.
5. $p_1 W p_2$ Here $AP = \{p_1, p_2\}$.
6. $p_1 U p_2$ Here $AP = \{p_1, p_2\}$.
7. $p_1 \rightarrow (X p_2)$ Here $AP = \{p_1, p_2\}$.
8. $G(p_1 \rightarrow (X p_2))$ Here $AP = \{p_1, p_2\}$.
9. $p_1 \rightarrow (F p_2)$ Here $AP = \{p_1, p_2\}$.
10. $G(p_1 \rightarrow (F p_2))$ Here $AP = \{p_1, p_2\}$.
11. $p_1 U (G p_2)$ Here $AP = \{p_1, p_2\}$.
12. $F p_1 \vee F p_2$ Here $AP = \{p_1, p_2\}$.
13. $F p_1 \wedge F p_2$ Here $AP = \{p_1, p_2\}$.
14. $G p_1 \vee G p_2$ Here $AP = \{p_1, p_2\}$.
15. $G p_1 \wedge G p_2$ Here $AP = \{p_1, p_2\}$.
16. $F p_1 \vee G p_2$ Here $AP = \{p_1, p_2\}$.

17. $F p_1 \wedge G p_2$ Here $AP = \{p_1, p_2\}$.

18. $F G p$ Here $AP = \{p\}$.

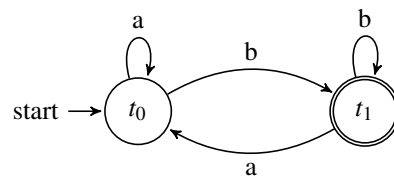
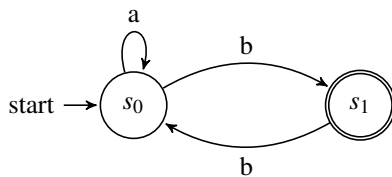
19. $G F p$ Here $AP = \{p\}$.

20. $F G p_1 \rightarrow F G p_2$ Here $AP = \{p_1, p_2\}$.

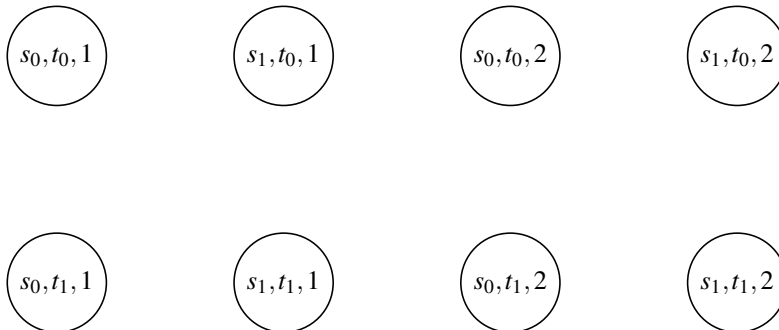
Hint. Use the fact that $\alpha \rightarrow \beta$ is equivalent to $\neg\alpha \vee \beta$.

4.2 Büchi automata constructions

Exercise 4.3. Let A and B be the following Büchi automata.



We want to build a Büchi automaton P such that $L(P) = L(A) \cap L(B)$. Below is the state space used in the Büchi product construction that we studied:



1. What is the set of initial (start) states of P ?
2. What is the set of accepting states of P ?
3. Draw and label the transitions of P .

Exercise 4.4. Let A_1 and A_2 be Büchi automata over the same input alphabet, and further suppose that every state of A_1 is accepting. Prove that in this case the ordinary cross-product \otimes from finite automata theory works, that is

$$L(A_1 \otimes A_2) = L(A_1) \cap L(A_2)$$

Exercise 4.5. We have pointed out that both ordinary NFAs and Büchi automata comprise the same components: states, initial states, accepting states, an input alphabet, and a transition relation. So given an “automaton” we can choose to view it as and NFA processing finite words *or* as a Büchi automaton for processing ω -words.

1. Show two automata \mathcal{A}_1 and \mathcal{A}_2 which are equivalent as NFAs (*i.e.* they accept the same finite words) but inequivalent as Büchi automata (*i.e.* they differ on at least one ω -word).
2. Show two automata \mathcal{A}_1 and \mathcal{A}_2 which are equivalent as Büchi automata but inequivalent as NFAs.

5 Satisfiability and model-checking

5.1 LTL satisfiability and validity

Once we have Theorem 4.1 we can test satisfiability, and can test validity. Here’s how. Since a formula α is valid if and only if $\neg\alpha$ is not satisfiable, it suffices to show how to decide satisfiability. So let α be given. We have seen in the previous sections how to construct an automaton \mathcal{A}_α that accepts precisely the models of α . In particular, α will be satisfiable if and only if the automaton \mathcal{A}_α accepts *some* words, that is, if and only if $L(\mathcal{A}_\alpha) \neq \emptyset$.

So we have reduced the LTL-satisfiability problem to the following problem:

INPUT: a Büchi automaton \mathcal{A}

QUESTION: $L(\mathcal{A}_\alpha) = \emptyset$?

So how do we solve this problem? Recall the algorithm for testing whether an ordinary finite-word NFA has a non-empty language: an NFA N has $L(N) \neq \emptyset$ if and only if there is a path from the start state to some accepting state. For Büchi automata things are just a little more subtle. In order that the Büchi automaton $\mathcal{A} = (Q, \Sigma, \delta, Q_s, F)$ accept some word it is necessary and sufficient that there exist a state $q \in F$ such that (i) there is a path from a state in Q_s to q , and (ii) there is a cycle from q to itself. This can be checked in time linear in $|Q|$.

5.2 LTL model checking

Here we are given a formula α and a transition system \mathcal{M} and we want to decide whether α is true on every path through \mathcal{M} . Surprisingly, this can be done by a variation on the satisfiability algorithm. The idea in outline is this:

1. Suppose we have a transition system $\mathcal{M} = (S, \rightarrow, L)$ over the set Prop, and let $s \in S$ be a state. We can view this a Büchi automaton $\mathcal{A}_{\mathcal{M}} = (S, \Sigma, \delta, Q)$ where Σ is 2^{Prop} and δ is defined by:

$$s_2 \in \delta(s_1, a) \text{ iff } L(s_1) = a \text{ and } s_1 \rightarrow s_2 \text{ in } \mathcal{M}$$

Note that every state in this automaton is an accepting state. It is not hard to check that the set of words accepted by this automaton is precisely the set of executions of the original system \mathcal{M} . starting at s .

2. Now we make a Büchi automaton $\mathcal{A}_{\neg\alpha}$ corresponding to the *negation* of our original formula α .

3. Using the product construction we developed earlier we build the automaton \mathcal{A}_\cap that has the property that

$$L(\mathcal{A}_\cap) = L(\mathcal{A}_\mathcal{M}) \cap L(\mathcal{A}_{\neg\alpha})$$

4. Finally we observe that $L(\mathcal{A}_\cap) \neq \emptyset$ precisely if there is a word x that (i) is an execution of \mathcal{M} , since it is in $L(\mathcal{A}_\mathcal{M})$, and (ii) satisfies $\neg\alpha$, since it is in $L(\mathcal{A}_{\neg\alpha})$. This means that $L(\mathcal{A}_\cap) \neq \emptyset$ precisely when there is an execution of \mathcal{M} that does not satisfy α . In other words, α is true on all executions of \mathcal{M} starting at s if and only if $L(\mathcal{A}_\cap)$ is empty. And as we have seen, we know how to decide that.

References

- [BBF⁺99] B. Bérard, M. Bidoit, A. Finkel, F Laroussine, A. Petit, L. Petrucci, and Ph. Schnoebelen with P. McKenzie. *Systems and Software Verification*. Springer, 1999.
- [CGP99] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999.
- [Var96] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag, 1996.