Linear-Time Temporal Logic and Büchi Automata*

Madhavan Mukund
SPIC Mathematical Institute
92 G N Chetty Road
Madras 600 017, India
E-mail: madhavan@ssf.ernet.in

Abstract

Over the past two decades, temporal logic has become a very basic tool for specifying properties of reactive systems. For finite-state systems, it is possible to use techniques based on Büchi automata to verify if a system meets its specifications. This is done by synthesizing an automaton which generates all possible models of the given specification and then verifying if the given system refines this most general automaton. In these notes, we present a self-contained introduction to the basic techniques used for this automated verification. We also describe some recent space-efficient techniques which work on-the-fly.

*Tutorial talk, Winter School on Logic and Computer Science, Indian Statistical Institute, Calcutta, January 1997.

Introduction

Program verification is a fundamental area of study in computer science. The broad goal is to verify whether a program is "correct"—i.e., whether the implementation of a program meets its specification. This is, in general, too ambitious a goal and several assumptions have to be made in order to render the problem tractable. In these lectures, we will focus on the verification of *finite-state reactive programs*. For specifying properties of programs, we use linear time temporal logic.

What is a reactive program? The general pattern along which a conventional program executes is the following: it accepts an input, performs some computation, and yields an output. Thus, such a program can be viewed as an abstract function from an input domain to an output domain whose behaviour consists of a transformation from initial states to final states.

In contrast, a reactive program is not expected to terminate. As the name suggests, such systems "react" to their environment on a continuous basis, responding appropriately to each input. Examples of such systems include operating systems, schedulers, discrete-event controllers etc. (Often, reactive systems are complex distributed programs, so concurrency also has to be taken into account. We will not stress on this aspect in these lectures—we take the view that a run of a distributed system can be represented as a sequence, by arbitrarily interleaving concurrent actions.)

To specify the behaviour of a reactive system, we need a mechanism for talking about the way the system evolves along potentially infinite computations. Temporal logic [Pnu77] has become a well-established formalism for this purpose. Many varieties of temporal logic have been defined in the past twenty years—we focus on propositional linear time temporal logic (LTL).

There is an intimate connection between models of LTL formulas and languages of infinite words—the models of an LTL formula constitute an ω-regular language over an appropriate alphabet. As a result, the satisfiability problem for LTL reduces to checking for emptiness of ω-regular languages. This connection was first explicitly pointed out in [SVW87].

Later, in [VW86], the connection between LTL and ω-regular languages has been extended to *model checking*. Unlike the satisfiability problem, which asks if a given formula α has a model, the model-checking problem is one of verification: the task is to verify whether a given finite-state program P satisfies a specification α. This consists of checking that all runs of P constitute models for α. Since finite-state reactive programs can be represented quite naturally as Büchi automata, model-checking also reduces to a problem in automata theory. It suffices to show that no run of P is a model for $\neg\alpha$, which is the same as checking that the intersection of the language accepted by P and the language defined by $\neg\alpha$ is empty.

In recent years, the techniques proposed in [VW86] have moved from the realm of theory to practice. In this context, there has been renewed emphasis on reducing the complexity of the automata-theoretic method. One fruitful approach is to build up the automaton associated with a formula "on-the-fly" so that only as much of the automaton is constructed as is needed to settle the model-checking problem. A first step in this regard is the algorithm proposed in [GPVW95].

This write-up is organized as follows. We begin with a description of the logical language LTL. We then provide a quick introduction to Büchi automata in Section 2. In Section 3 we describe the basic automata-theoretic approach of [VW86]. Next, in Section 4, we present a formal basis for the on-the-fly method of [GPVW95]. We conclude with some pointers to ways in which these approaches can be extended in more general settings.

1 Linear-time temporal logic

Linear time temporal logic is a logic for talking about infinite sequences, where each element in the sequence corresponds to a propositional world.

Syntax

We fix a countable set of *atomic propositions* $\mathcal{P} = \{p_0, p_1, \ldots\}$. Then Φ, the set of formulas of LTL, is defined inductively as follows:

- Every member of $\mathcal{P}$ belongs to Φ.
- If α and β are formulas in Φ, then so are $\neg\alpha$, $\alpha \vee \beta$, $O\alpha$ and $\alpha\mathcal{U}\beta$.

The connectives $\neg$ and $\vee$ correspond to the usual Boolean connectives "not" and "or" respectively. The modality O to be read as "next" while the binary modality $\mathcal{U}$ is to be read as "until". Thus $O\alpha$ is "next α" while $\alpha\mathcal{U}\beta$ is "α until β".

Semantics

A *model* is a function $M : \mathbb{N}_0 \to 2^{\mathcal{P}}$. In other words, a model is an infinite sequence $P_0P_1\ldots$ of subsets of $\mathcal{P}$. The function M describes how the truth of atomic propositions changes as time progresses.

We write $M, i \models \alpha$ to denote that "α is true at time instant i in the model M". This notion is defined inductively, according to the structure of α.

- $M, i \models p$, where $p \in \mathcal{P}$, iff $p \in M(i)$.
- $M, i \models \neg\alpha$ iff $M, i \not\models \alpha$.
- $M, i \models \alpha \vee \beta$ iff $M, i \models \alpha$ or $M, i \models \beta$.
- $M, i \models O\alpha$ iff $M, i{+}1 \models \alpha$.
- $M, i \models \alpha\mathcal{U}\beta$ iff there exists $k \geq i$ such that $M, k \models \beta$ and for all j such that $i \leq j < k$, $M, j \models \alpha$.

A formula α is said to be *satisfiable* if there exists a model M and an instant i such that $M, i \models \alpha$. Since the modalities we have defined only talk about future time-points within a model, it is not difficult to argue that a formula is satisfiable iff in some model it is satisfied at the initial point.

Proposition 1.1 *Let α be a formula in Φ. Then α is satisfiable iff there exists a model M such that $M, 0 \models \alpha$.*

Another simple observation is that in order to satisfy a formula α, a model needs to assign truth values only to those propositions which occur in α. Let $\mathrm{Voc}(\alpha)$, the "vocabulary" of α, denote the subset of $\mathcal{P}$ which is mentioned in α. $\mathrm{Voc}(\alpha)$ can be defined inductively quite easily:

- $\mathrm{Voc}(p) = \{p\}$
- $\mathrm{Voc}(\neg\alpha) = \mathrm{Voc}(\alpha)$
- $\mathrm{Voc}(\alpha \vee \beta) = \mathrm{Voc}(\alpha) \cup \mathrm{Voc}(\beta)$
- $\mathrm{Voc}(O\alpha) = \mathrm{Voc}(\alpha)$
- $\mathrm{Voc}(\alpha \mathcal{U} \beta) = \mathrm{Voc}(\alpha) \cup \mathrm{Voc}(\beta)$

If M is a model and α is a formula, let $M_{\mathrm{Voc}(\alpha)}$ be a new model where for all $i \in \mathbb{N}_0$, $M_{\mathrm{Voc}(\alpha)}(i) = M(i) \cap \mathrm{Voc}(\alpha)$. We then have the following simple fact.

Proposition 1.2 *Let M be a model and α be a formula. Then, for all $i \in \mathbb{N}_0$, $M, i \models \alpha$ iff $M_{\mathrm{Voc}(\alpha)}, i \models \alpha$.*

Derived connectives

As usual, we introduce constants $\top$ and $\bot$ representing "true" and "false". We can write $\top$ as, for instance, $p_0 \vee \neg p_0$ (recall that $\mathcal{P} = \{p_0, p_1, \ldots\}$) and $\bot$ as $\neg\top$.

We can also generate normal Boolean connectives like $\wedge$ ("and"), $\Rightarrow$ ("implies") and $\Leftrightarrow$ ("iff") from the connectives $\neg$ and $\vee$ in the usual way—for instance, $\alpha \Rightarrow \beta = \neg\alpha \vee \beta$.

We also introduce two derived modalities based on $\alpha \mathcal{U} \beta$. We write $\Diamond\alpha$ for $\top \mathcal{U} \alpha$ and $\Box\alpha$ for $\neg\Diamond\neg\alpha$. The modality $\Diamond$ is read as "eventually" while the modality $\Box$ is read as "henceforth". Let M be a model. It is not difficult to verify the following facts:

- $M, i \models \Diamond\alpha$ iff there exists $k \geq i$ such that $M, k \models \alpha$.
- $M, i \models \Box\alpha$ iff for all $k \geq i$, $M, k \models \alpha$.

Examples

Here are some examples of the kinds of assertions we can make in temporal logic.

- The formula $\Diamond\Box\alpha$ says that α is eventually a "stable property" of the system—$M, i \models \Diamond\Box\alpha$ iff for some $j \geq i$, for all $k \geq j$, $M, k \models \alpha$.

- On the other hand, $\Box\Diamond\beta$ asserts that β holds infinitely often.

- If we use the full power of $\mathcal{U}$, we can make some fairly complex statements about a program's behaviour. For instance, consider a system which schedules access to a shared resource among k competing processes which are named, for convenience, $1, 2, \ldots, k$. For each process i, we could have two atomic propositions r_i and g_i to denote the state of i vis-a-vis the shared resource—proposition r_i is true if process i has *requested* the resource but has not yet got access to it, while g_i is true when i is *granted* access to the resource.

  The formula $r_i \mathcal{U} g_i$ asserts that process i continues to be in a requesting state until access to the resource is granted. A desirable property that the system should satisfy is that for all i, the formula $\Box(r_i \Rightarrow (r_i \mathcal{U} g_i))$ is true at the initial state—i.e., the scheduler eventually grants every request.

For a detailed exposition of how to specify properties of reactive systems in temporal logic, the reader is referred to the book [MP91].

2 Büchi automata

Automata on infinite inputs were introduced by Büchi in [Bü60]. A Büchi automaton is a non-deterministic finite-state automaton which takes infinite words as input. A word is accepted if the automaton goes through some designated "good" states infinitely often while reading it.

We begin with some notation for infinite words. Let Σ be a finite alphabet. An infinite word $\alpha \in \Sigma^\omega$ is an infinite sequence of symbols from Σ. We shall represent α as a function $\alpha : \mathbb{N}_0 \to \Sigma$, where $\mathbb{N}_0$ is the set $\{0, 1, 2, \ldots\}$ of natural numbers. Thus, $\alpha(i)$ denotes the letter occurring at the i^{th} position.

In general, if S is a set and σ an infinite sequence of symbols over S—in other words, $\sigma : \mathbb{N}_0 \to S$—then $\mathsf{inf}(\sigma)$ denotes the set of symbols from S which occur infinitely often in σ. Formally, $\mathsf{inf}(\sigma) = \{s \in S \mid \exists^\omega n \in \mathbb{N}_0 : \sigma(n) = s\}$, where $\exists^\omega$ denotes the quantifier "there exist infinitely many".

Automata An *automaton* is a triple $\mathcal{A} = (S, \to, S_{in})$ where S is a set of *states*, $S_{in} \subseteq S$ is a set of *initial states* and $\to \subseteq S \times \Sigma \times S$ is a *transition relation*. Normally, we write $s \xrightarrow{a} s'$ to denote that $(s, a, s') \in \to$.

The automaton is said to be *deterministic* if S_{in} is a singleton and $\to$ is a function from $S \times \Sigma$ to S. The automata we encounter in this paper will, in general, be non-deterministic.

Runs Let $\mathcal{A} = (S, \to, S_{in})$ be an automaton and $\alpha : \mathbb{N}_0 \to \Sigma$ an input word. A *run* of $\mathcal{A}$ on α is a infinite sequence $\rho : \mathbb{N}_0 \to S$ such that $\rho(0) \in S_{in}$ and for all $i \in \mathbb{N}_0$, $\rho(i) \xrightarrow{\alpha(i)} \rho(i{+}1)$.

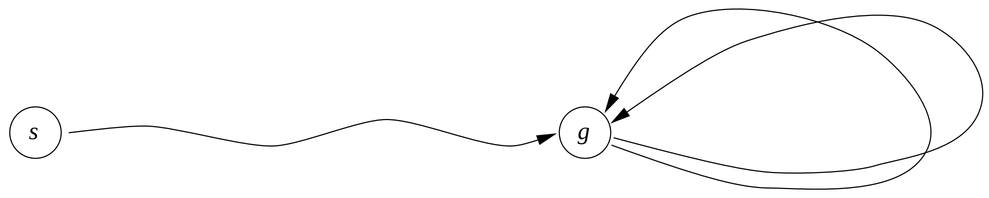

Figure 1: A typical accepting run of a Büchi automaton, with $s \in S_{in}$ and $g \in G$.

So, a run is just a "legal" sequence of states that an automaton can pass through while reading the input. In general, an input may admit many runs because of non-determinism. Since a non-deterministic automaton may have states where there are no outgoing transitions corresponding to certain input letters, it is also possible that an input admits *no* runs—in this case, every potential run leads to a state from where there is no outgoing transition enabled for the next input letter. If the automaton is deterministic, each input admits precisely one run.

Büchi automata A *Büchi automaton* is a pair $(\mathcal{A}, G)$ where $\mathcal{A} = (S, \rightarrow, S_{in})$ and $G \subseteq S$. G denotes a set of *good states*. The automaton $(\mathcal{A}, G)$ *accepts* an input $\alpha : \mathbb{N}_0 \rightarrow \Sigma$ if there is a run ρ of $\mathcal{A}$ on α such that $\mathsf{inf}(\rho) \cap G \neq \emptyset$. The language *recognized by* $(\mathcal{A}, G)$, $L(\mathcal{A}, G)$, is the set of all infinite words accepted by $(\mathcal{A}, G)$. A set $L \subseteq \Sigma^\omega$ is said to be *Büchi-recognizable* if there is a Büchi automaton $(\mathcal{A}, G)$ such that $L = L(\mathcal{A}, G)$.

According to the definition, a Büchi automaton accepts an input if there is a run along which some subset of G occurs infinitely often. Since G is a finite set, it is easy to see that there must actually be a state $g \in G$ which occurs infinitely often along σ. In other words, if we regard the state space of a Büchi automaton as a graph, an accepting run traces an infinite path which starts at some state s in S_{in}, reaches a good state $g \in G$ and, thereafter, keeps looping back to g infinitely often (see Figure 1).

Example 2.1 Consider the alphabet $\Sigma = \{a, b\}$. Let $L \subseteq \Sigma^\omega$ consist of all infinite words α such that there are infinitely many occurrences of a in α. Figure 2 shows a Büchi automaton recognizing L. The initial state is marked by an unlabelled incoming arrow. There is only one good state, which is indicated with a double circle. In this automaton, all transitions labelled a lead into the good state and, conversely, all transitions coming into the good state are labelled a. From this, it follows that the automaton accepts an infinite word iff it has infinitely many occurrences of a.

The complement of L, which we denote $\overline{L}$, is the set of all infinite words α such that α has only finitely many occurrences of a. An automaton recognizing $\overline{L}$ is shown in Figure 3. The automaton guesses a point in the input beyond which it will see no more a's—such a point must exist in any input with only a finite number of a's. Once it has made this guess, it can process only b's—there is no transition labelled a from the second state—so if it reads any more a's it gets stuck.

□

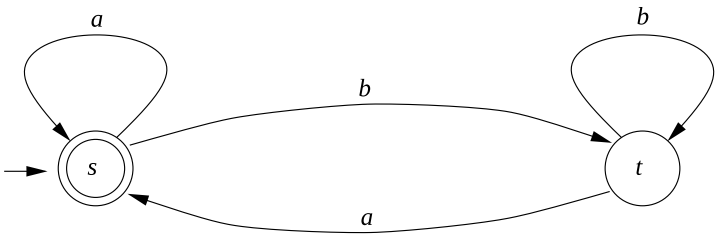

Figure 2: A Büchi automaton for L (Example 2.1)

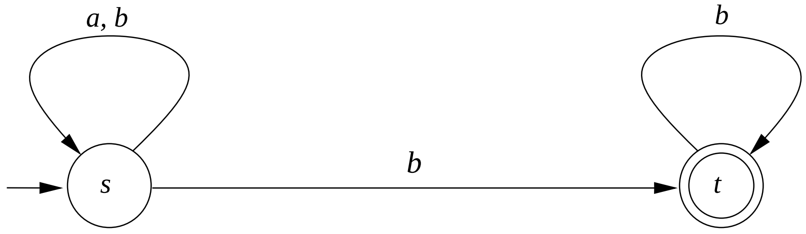

Figure 3: A Büchi automaton for $\overline{L}$ (Example 2.1)

In the example, notice that the automaton recognizing L is deterministic while the automaton for $\overline{L}$ is non-deterministic. It can be shown that the non-determinism in the second case is unavoidable—that is, there is *no* deterministic automaton recognizing $\overline{L}$. This means that Büchi automata are fundamentally different from their counterparts on finite inputs: we know that over finite words, deterministic automata are as powerful as non-deterministic automata.

2.1 Constructions on Büchi automata

It turns out that the class of Büchi-recognizable languages is closed under boolean operations.

Union To show closure under finite union, let $(\mathcal{A}_1, G_1)$ and $(\mathcal{A}_2, G_2)$ be two Büchi automata. To construct an automaton $(\mathcal{A}, G)$ such that $L(\mathcal{A}, G) = L(\mathcal{A}_1, G_1) \cup L(\mathcal{A}_2, G_2)$, we take $\mathcal{A}$ to be the *disjoint* union of $\mathcal{A}_1$ and $\mathcal{A}_2$. Since we are permitted to have a *set* of initial states in $\mathcal{A}$, we retain the initial states from both copies. If a run of $\mathcal{A}$ starts in an initial state contributed by $\mathcal{A}_1$, it will never cross over into the state space contributed by $\mathcal{A}_2$ and vice versa. Thus, we can set the good states of $\mathcal{A}$ to be the union of the good states contributed by both components.

Complementation Showing that Büchi-recognizable languages are closed under complementation is highly non-trivial. One problem is that we cannot determinize Büchi automata. Even if we could work with deterministic automata, the formulation of Büchi acceptance is not symmetric with respect to complementation in the following sense. Suppose $(\mathcal{A}, G)$ is a deterministic Büchi automaton and α is an infinite word which does not belong to $L(\mathcal{A}, G)$. Then, the (unique) run ρ_α of $\mathcal{A}$ on α is such that $\inf(\rho_\alpha) \cap G = \emptyset$. Let $\overline{G}$ denote the complement of G. It follows that $\inf(\rho_\alpha) \cap \overline{G} \neq \emptyset$, since *some* state must occur infinitely often in ρ_α. It would be tempting to believe that the automaton $(\mathcal{A}, \overline{G})$ recognizes $\Sigma^\omega - L(\mathcal{A}, G)$. However, there may be words which admit runs which visit both G and $\overline{G}$ infinitely often. These words will be including both in $L(\mathcal{A}, \overline{G})$ as well as in $L(\mathcal{A}, G)$. So, there is no convenient way to express the complement of a Büchi condition again as a Büchi condition. Fortunately, we shall not need to complement Büchi automata for any of the constructions which we describe here.

Intersection Turning to intersection, the natural way to intersect automata $\mathcal{A}_1$ and $\mathcal{A}_2$ is to construct an automaton whose state space is the cross product of the state spaces of $\mathcal{A}_1$ and $\mathcal{A}_2$ and let both copies process the input simultaneously. For finite words, the input is accepted if each copy can generate a run which reaches a final state at the end of the word.

For infinite inputs, we have to do a more sophisticated product construction. An infinite input α should be accepted by the product system provided both copies generate runs which visit good states infinitely often. Unfortunately, there is no guarantee that these runs will ever visit good states simultaneously—for instance, it could be that the first run goes through a good state after $\alpha(0)$, $\alpha(2)$, ... while the second run enters good states after $\alpha(1)$, $\alpha(3)$, ... So, the main question is one of identifying the good states of the product system.

The key observation is that to detect that both components of the product visit good states infinitely often, one need not record *every* point where the copies visit good states; in each copy, it suffices to observe an infinite subsequence of the overall sequence of good states. So, we begin by focusing on the first copy and waiting for its run to enter a good state. When this happens, we switch attention to the other copy and wait for a good state there. Once the second copy reaches a good state, we switch back to the first copy and so on. Clearly, we will switch back and forth infinitely often iff both copies visit their respective good states infinitely often. Thus, we can characterize the good states of the product in terms of the states where one switches back and forth.

Formally, the construction is as follows. Let $(\mathcal{A}_1, G_1)$ and $(\mathcal{A}_2, G_2)$ be two Büchi automata such that $\mathcal{A}_i = (S_i, \rightarrow_i, S^i_{in})$ for $i = 1, 2$. Define $(\mathcal{A}, G)$, where $\mathcal{A} = (S, \rightarrow, S_{in})$, as follows:

- $S = S_1 \times S_2 \times \{1, 2\}$
- The transition relation $\rightarrow$ is defined as follows:

$$(s_1, s_2, 1) \xrightarrow{a} (s_1', s_2', 1) \text{ if } s_1 \xrightarrow{a}_1 s_1',\ s_2 \xrightarrow{a}_2 s_2' \text{ and } s_1 \notin G_1.$$
$$(s_1, s_2, 1) \xrightarrow{a} (s_1', s_2', 2) \text{ if } s_1 \xrightarrow{a}_1 s_1',\ s_2 \xrightarrow{a}_2 s_2' \text{ and } s_1 \in G_1.$$
$$(s_1, s_2, 2) \xrightarrow{a} (s_1', s_2', 2) \text{ if } s_1 \xrightarrow{a}_1 s_1',\ s_2 \xrightarrow{a}_2 s_2' \text{ and } s_2 \notin G_2.$$

$$(s_1, s_2, 2) \xrightarrow{a} (s_1', s_2', 1) \text{ if } s_1 \xrightarrow{a}_1 s_1',\ s_2 \xrightarrow{a}_2 s_2' \text{ and } s_2 \in G_2.$$

- $S_{in} = \{(s_1, s_2, 1) \mid s_1 \in S_{in}^1 \text{ and } s_2 \in S_{in}^2\}$
- $G = S_1 \times G_2 \times \{2\}$.

In the automaton $\mathcal{A}$, each product state carries an extra tag indicating whether the automaton is checking for a good state on the first or the second component. The automaton accepts if it switches focus from the second component to the first infinitely often. (Notice that we could equivalently have defined G to be the set $G_1 \times S_2 \times \{1\}$.) It is not difficult to verify that $L(\mathcal{A}, G) = L(\mathcal{A}_1, G_1) \cap L(\mathcal{A}_2, G_2)$.

Emptiness In applications, we will need to be able to check whether the language accepted by a Büchi automaton is empty. To do this, we recall our observation that any accepting run of a Büchi automaton must begin in an initial state, reach a final state g and then cycle back to g infinitely often.

If we ignore the labels on the transitions, we can regard the state space of a Büchi automaton $(\mathcal{A}, G)$ as a directed graph $G_{\mathcal{A}} = (V_{\mathcal{A}}, E_{\mathcal{A}})$ where $V_{\mathcal{A}} = S$ and $(s, s') \in E_{\mathcal{A}}$ iff for some $a \in \Sigma$, $s \xrightarrow{a} s'$. Recall that a set of vertices X in a directed graph is a *strongly connected component* iff for every pair of vertices $v, v' \in X$, there is a path from v to v'. Clearly, $L(\mathcal{A}, G)$ is non-empty iff there is a strongly connected component X in $G_{\mathcal{A}}$ such that X contains a vertex g from G and X is reachable from one of the initial states. We thus have the following theorem.

Theorem 2.2 *The emptiness problem for Büchi automata is decidable.*

Notice that it is sufficient to analyze *maximal* strongly connected components in $G_{\mathcal{A}}$ in order to check that $L(\mathcal{A}, G) \neq \emptyset$. Computing the maximal strongly connected components of a directed graph can be done in time linear in the size of the graph [AHU74], where the size of a graph $G = (V, E)$ is, as usual, given by $|V| + |E|$. Checking reachability can also be done in linear time. So, if $\mathcal{A}$ has n states, checking that $L(\mathcal{A}, G) \neq \emptyset$ can be done in time $O(n^2)$.

2.2 Generalized Büchi automata

When expressing the connection between temporal logic and Büchi automata, it is often convenient to work with a slightly more elaborate acceptance condition. A *generalized Büchi automaton* is a structure $(\mathcal{A}, G_1, G_2, \ldots, G_k)$, where $\mathcal{A} = (S, \rightarrow, S_{in})$ and for all $i \in \{1, 2, \ldots, k\}$, $G_i \subseteq S$.

An input α is accepted by the automaton $(\mathcal{A}, G_1, G_2, \ldots, G_k)$ if there is a run ρ of $\mathcal{A}$ on α such that $\mathsf{inf}(\rho) \cap G_i \neq \emptyset$ for all $i \in \{1, 2, \ldots, k\}$. As usual, $L(\mathcal{A}, G_1, G_2, \ldots, G_k)$ denotes the language of all infinite words accepted by the automaton $(\mathcal{A}, G_1, G_2, \ldots, G_k)$. The following observation is immediate.

Proposition 2.3 *Let $(\mathcal{A}, G_1, G_2, \ldots, G_k)$ be a generalized Büchi automaton. Then*

$$L(\mathcal{A}, G_1, G_2, \ldots, G_k) = \bigcap_{i \in \{1,2,\ldots,k\}} L(\mathcal{A}, G_i).$$

In other words, every language which is recognized by a generalized Büchi automaton is also Büchi recognizable. It is not difficult to argue that checking whether the language accepted by a generalized Büchi automaton is empty is no harder, in terms of computational complexity, than the corresponding check for a normal Büchi automaton.

Further reading

The languages recognized by Büchi automata correspond to ω-regular languges. These languages have a syntactic characterization in terms of regular languages of finite strings. They can also be characterized logically using the monadic second order theory of one successor (S1S). For a more detailed introduction to the theory of automata on infinite words, the reader is encouraged to consult the excellent surveys by Thomas [Th90, Th96].

3 Automata-theoretic methods

As we saw in the last section, a model for an LTL formula α is a function $M : \mathbb{N}_0 \to 2^{\mathcal{P}}$. We also saw that to check whether α is satisfiable, it suffices to look at models defined over $\mathrm{Voc}(\alpha)$. In other words, we can restrict our attention to models of the form $P_0 P_1 \ldots$ where each P_i is a subset of $\mathrm{Voc}(\alpha)$. Since $\mathrm{Voc}(\alpha)$ is finite, we can treat each model as an infinite word over the finite alphabet $2^{\mathrm{Voc}(\alpha)}$.

The result we shall establish is that the set of all infinite words over $2^{\mathrm{Voc}(\alpha)}$ which are models for α—i.e., the set

$$\mathrm{Mod}(\alpha) = \{M = P_0 P_1 \ldots \mid M, 0 \models \alpha\}$$

actually constitutes a Büchi recognizable language. We shall also demonstrate how to explicit construct a generalized Büchi automaton $(\mathcal{A}_\alpha, G_1, G_2, \ldots, G_k)$ over the alphabet $2^{\mathrm{Voc}(\alpha)}$ such that $L(\mathcal{A}_\alpha, G_1, G_2, \ldots, G_k) = \mathrm{Mod}(\alpha)$.

3.1 Satisfiability

We begin by defining the (Fischer-Ladner) closure of a formula.

Closure Let α be an LTL formula. Then $\mathrm{CL}'(\alpha)$ is the smallest set of formulas such that:

- $\alpha \in \mathrm{CL}'(\alpha)$.
- If $\neg\beta \in \mathrm{CL}'(\alpha)$, then $\beta \in \mathrm{CL}'(\alpha)$.

- If $\beta \vee \gamma \in \mathrm{CL}'(\alpha)$, then $\beta, \gamma \in \mathrm{CL}'(\alpha)$.
- If $O\beta \in \mathrm{CL}'(\alpha)$, then $\beta \in \mathrm{CL}'(\alpha)$.
- If $\beta \mathcal{U} \gamma \in \mathrm{CL}'(\alpha)$, then $\beta, \gamma, O(\beta \mathcal{U} \gamma) \in \mathrm{CL}'(\alpha)$.

Finally, $\mathrm{CL}(\alpha) = \mathrm{CL}'(\alpha) \cup \{\neg\beta \mid \beta \in \mathrm{CL}'(\alpha)\}$, where we identify $\neg\neg\beta$ with β.

Notice that $\mathrm{CL}(\alpha)$ is always finite, even though the clause for $\beta \mathcal{U} \gamma$ throws in a larger formula into the set. In fact, if α contains n symbols, then $|\mathrm{CL}(\alpha)|$ is $O(n)$.

The automaton $\mathcal{A}_\alpha$ that we associate with a formula α will have as its states subsets of $\mathrm{CL}(\alpha)$ which are both propositionally and temporally "consistent". These subsets are called atoms.

Atoms Let α be a formula. Then $A \subseteq \mathrm{CL}(\alpha)$ is an *atom* if:

- $\forall \beta \in \mathrm{CL}(\alpha)$, $\beta \in A$ iff $\neg\beta \notin A$.
- $\forall \beta \vee \gamma \in \mathrm{CL}(\alpha)$, $\beta \vee \gamma \in A$ iff $\beta \in A$ or $\gamma \in A$.
- $\forall \beta \mathcal{U} \gamma \in \mathrm{CL}(\alpha)$, $\beta \mathcal{U} \gamma \in A$ iff $\gamma \in A$ or $\beta, O(\beta \mathcal{U} \gamma) \in A$.

Let $\mathcal{AT}$ be the set of all atoms of α.

Constructing a Büchi automaton for α
We first construct an automaton $\mathcal{A}_\alpha = (S, \rightarrow, S_{in})$ over the alphabet $2^{\mathrm{Voc}(\alpha)}$, where:

- $S = \mathcal{AT}$
- Let $A, B \in \mathcal{AT}$ and $P \subseteq \mathrm{Voc}(\alpha)$. Then $A \xrightarrow{P} B$ iff the following hold.
  - $P = A \cap \mathrm{Voc}(\alpha)$.
  - For all $O\beta \in \mathrm{CL}(\alpha)$, $O\beta \in A$ iff $\beta \in B$.
- $S_{in} = \{A \in \mathcal{AT} \mid \alpha \in A\}$.

Let $\{\beta_1 \mathcal{U} \gamma_1, \beta_2 \mathcal{U} \gamma_2, \ldots, \beta_k \mathcal{U} \gamma_k\}$ be the set of until formulas which appear in $\mathrm{CL}(\alpha)$. We add a generalized Büchi acceptance condition $(G_1, G_2, \ldots, G_k)$ where for each i,

$$G_i = \{A \in \mathcal{AT} \mid \beta_i \mathcal{U} \gamma_i \notin A \text{ or } \gamma_i \in A\}.$$

We then have the following theorem.

Theorem 3.1 *Let M be an infinite word over $2^{\mathrm{Voc}(\alpha)}$. Then*

$$M \in L(\mathcal{A}_\alpha, G_1, G_2, \ldots, G_k) \textit{ iff } M, 0 \models \alpha$$

Proof ($\Rightarrow$) Let $M = P_0 P_1 \ldots$ be an infinite word over $\text{Voc}(\alpha)$ which is accepted by $(\mathcal{A}_\alpha, G_1, G_2, \ldots, G_k)$. Let $A_0 A_1 \ldots$ be an accepting run of $\mathcal{A}_\alpha$ on M. For all $\beta \in \text{CL}(\alpha)$ and for all $i \in \mathbb{N}_0$, we show that $M, i \models \beta$ iff $\beta \in A_i$.

The proof is by induction on the structure of β.

($\beta = p \in \mathcal{P}$)

Then $M, i \models p$ iff $p \in P_i$ iff $p \in A_i$.

($\beta = \neg\gamma$)

Then $M, i \models \beta$ iff $M, i \not\models \gamma$ iff (by the induction hypothesis) $\gamma \notin A_i$ iff $\beta \in A_i$ (by the definition of an atom).

($\beta = \gamma \vee \delta$)

Then $M, i \models \beta$ iff ($M, i \models \gamma$ or $M, i \models \delta$) iff—by the induction hypothesis—($\gamma \in A_i$ or $\delta \in A_i$) iff—by the definition of an atom—$\gamma \vee \delta \in A_i$.

($\beta = O\gamma$)

Then $M, i \models \beta$ iff $M, i{+}1 \models \gamma$ iff (by the induction hypothesis) $\gamma \in A_{i+1}$ iff (by the definition of $A_i \xrightarrow{P_i} A_{i+1}$) $O\gamma = \beta \in A_i$.

($\beta = \gamma\,\mathcal{U}\delta$)

Suppose that $M, i \models \beta$. We must show that $\beta \in A_i$. By the semantics of the modality $\mathcal{U}$, there exists $k \geq i$ such that $M, k \models \delta$ and for all j, $i \leq j < k$, $M, j \models \gamma$. We show that $\beta \in A_i$ by a second induction on $k - i$.

Base case: ($k - i = 0$)

Then $k = i$ and $M, i \models \delta$. By the main induction hypothesis, $\delta \in A_i$, whence, from the definition of atoms, $\gamma\,\mathcal{U}\delta = \beta \in A_i$.

Induction step: ($k - i = \ell > 0$)

By the semantics of the modality $\mathcal{U}$, $M, i \models \gamma$ and $M, i{+}1 \models \gamma\,\mathcal{U}\delta$. Then, by the secondary induction hypothesis, $\gamma\,\mathcal{U}\delta \in A_{i+1}$. From the definition of $A_i \xrightarrow{P_i} A_{i+1}$, we then have $O(\gamma\,\mathcal{U}\delta) \in A_i$ (recall that if $\gamma\,\mathcal{U}\delta \in \text{CL}(\alpha)$ then $O(\gamma\,\mathcal{U}\delta) \in \text{CL}(\alpha)$ as well). By the main induction hypothesis, we also have $\gamma \in A_i$. Combining these facts, from the definition of an atom, $\gamma\,\mathcal{U}\delta = \beta \in A_i$.

Conversely, suppose $\beta \in A_i$. We must show that $M, i \models \beta$. Recall that we have indexed the until formulas in $\text{CL}(\alpha)$ by $1, 2, \ldots, k$. Let m be the index of β.

Since $A_0 A_1 \ldots$ is an accepting run of $(\mathcal{A}_\alpha, G_1, G_2, \ldots, G_k)$, there must exist a $k \geq i$ such that $A_k \in G_m$. Choose the least such k. Once again, we do a second induction on $k - i$ to show that $M, i \models \beta$.

Base case: ($k - i = 0$)

If $k = i$, then $A_i \in G_m$. Since $\gamma\,\mathcal{U}\delta \in A_i$, the only way for A_i to be in G_m is for δ to also belong to A_i. Then, by the main induction hypothesis, $M, i \models \delta$, whereby $M, i \models \gamma\,\mathcal{U}\delta$ as well.

Induction step: $(k - i = \ell > 0)$

Since $A_i \notin G_m$, $\delta \notin A_i$. From the definition of atoms, both γ and $O(\gamma\mathcal{U}\delta)$ must be in A_i. Since $A_i \xrightarrow{P_i} A_{i+1}$, it must be the case that and $\gamma\mathcal{U}\delta \in A_{i+1}$. By the secondary induction hypothesis, $M, i{+}1 \models \gamma\mathcal{U}\delta$ while by the main induction hypothesis $M, i \models \gamma$. From the semantics of the modality $\mathcal{U}$, it then follows that $M, i \models \gamma\mathcal{U}\delta$ as well.

($\Leftarrow$) Suppose that $M = P_0P_1\ldots$ such that $M, 0 \models \alpha$. We have to show that $M \in L(\mathcal{A}_\alpha, G_1, G_2, \ldots, G_k)$. In other words, we have to exhibit an accepting run of the automaton on M.

For $i \in \mathbb{N}_0$, define A_i to be the set $\{\beta \in \mathrm{CL}(\alpha) \mid M, i \models \alpha\}$. It is easy to verify that each A_i is an atom. We can also verify that each adjacent pair of atoms A_i and A_{i+1} satisfy the conditions specified for the existence of an arrow $A_i \xrightarrow{P_i} A_{i+1}$. Finally, since $M, 0 \models \alpha$, we have $\alpha \in A_0$, so A_0 is an initial state in $\mathcal{A}_\alpha$. From all this, it follows that $A_0A_1\ldots$ is a run of the automaton on M.

To check that it is an accepting run, we have to verify that each good set G_m is met infinitely often. Suppose this is not the case—i.e., for some G_m and some index $k \in \mathbb{N}_0$, for all $j \geq k$, $A_j \notin G_m$. In other words, for all $j \geq k$, the m^{th} until formula in $\mathrm{CL}(\alpha)$, $\gamma_m\mathcal{U}\delta_m$, belongs to A_j and $\delta_m \notin A_j$. Since $A_k = \{\beta \in \mathrm{CL}(\alpha) \mid M, k \models \beta\}$, it follows that $M, k \models \gamma_m\mathcal{U}\delta_m$. But, since $\delta_m \notin A_j$ for all $j \geq k$, it follows that $M, j \models \neg\delta_m$ for all $j \geq k$, which contradicts the fact that $M, k \models \gamma_m\mathcal{U}\delta_m$. □

It follows from the preceding theorem that α is satisfiable iff the language recognized by $(\mathcal{A}_\alpha, G_1, G_2, \ldots, G_k)$ is non-empty. Let the length of α be n. Since the size of $\mathrm{CL}(\alpha)$ is linear in n, it follows that the number of states in $\mathcal{A}_\alpha$ is bounded by $2^{O(n)}$. Since checking for non-emptiness can be done in time $(2^{O(n)})^2 = 2^{O(n)}$, the satisifiability problem for a formula of length n is solvable in time $2^{O(n)}$.

3.2 Model checking

Let us model finite-state reactive programs as (generalized) Büchi automata—a program is a structure $P = ((S, \rightarrow, S_{in}), G_1, G_2, \ldots, G_k)$. The acceptance condition may trivial—i.e., we could have $k = 1$ and $G_1 = S$.

We use atomic propositions to characterize properties satisified by the states of the program. In other words, a program P comes with a function $V : S \rightarrow \mathcal{P}$ which describes the properties of each state.

Each run of P generates a model in the obvious way. Let $\rho = s_0s_1\ldots$ be a run of P—in other words, $s_0 \in S_{in}$ and for all $i \in \mathbb{N}_0$, $s_i \xrightarrow{a} s_{i+1}$. This induces a model $M_\rho : \mathbb{N}_0 \rightarrow \mathcal{P}$ given by $M_\rho(i) = V(\rho(i))$ for all $i \in \mathbb{N}_0$.

The model-checking problem is the following: given a program P and a valuation V together with a specification α, does every run of P satisfy α? In other words, we want to check that the set $\mathrm{Mod}(P) = \{M_\rho \mid \rho \text{ is a run of } P\}$ is a subset of $\mathrm{Mod}(\alpha)$, the set of models of α. This is equivalent to saying that $\mathrm{Mod}(P)$ does not contain any model of $\neg\alpha$.

To check this, we first construct an automaton $(\mathcal{A}_P, G_1, G_2, \ldots, G_k)$ from (P, V) over the alphabet $2^{\mathrm{Voc}(\alpha)}$ as follows. Let $P = ((S, \rightarrow, S_{in}), G_1, G_2, \ldots, G_k)$. Then $\mathcal{A}_P = (S_P, \rightarrow_P, S^P_{in})$, where

- $S_P = S$
- $\rightarrow_P = \{(s, Q, s') \mid \exists a.\ s \xrightarrow{a} s' \text{ and } Q = V(s) \cap \mathrm{Voc}(\alpha)\}$.
- $S^P_{in} = S_{in}$.

The following is then obvious:

Theorem 3.2 *Let (P, V) be a program and α be a formula. Then all models generated by P satisfy α iff $L(\mathcal{A}_P) \cap L(\mathcal{A}_{\neg\alpha}) = \emptyset$.*

Recall that intersecting two Büchi automata is just a simple extension of the product construction. From this it follows that the complexity of settling the model checking problem is $O(m\ 2^{O(n)})$, where m is the the number of states in $\mathcal{A}_P$ and n is the length of α.

4 On-the-fly methods

The construction described in the previous section requires us to construct the entire automaton $\mathcal{A}_{\neg\alpha}$ in order to solve the model-checking problem. In particular, the emptiness check at the end of the procedure assumes that the entire automaton is available in memory in order to search for an accepting run.

A simple way to get around this memory requirement is to explore the state space in an incremental fashion. For instance, in [CVWY92], a depth-first-search (DFS) based strategy is used to detect cycles and hence check for emptiness, rather than looking for strongly connected components directly in the full automaton graph.

Such a strategy can be implemented along with an "on-the-fly" construction of the automaton—when constructing the product of the program automaton and the formula automaton, we generate the states of the product incrementally, as and when they are explored by the DFS-based cycle-detection strategy. In the process, if we find an accepting cycle, we can abort the search without generating the entire state space. Of course, in the worst case (which occurs when the program *does* meet the specification given the formula!), we end up having to explore the entire state space.

To generate the product state space on the fly, we have to first have a method for constructing both the program automaton and formula automaton on-the-fly, individually.

In many applications, the program to be verified is a concurrent system specified in terms of its components. In order to apply the technique discussed in the previous section, we have to generate the global state space of the concurrent program. The obvious on-the-fly approach is to generate these global states as and when the need arises.

More tricky is to find an on-the-fly approach for generating the formula automaton. A procedure for achieving this is described in [GPVW95]. However, the construction presented in [GPVW95] is fairly opaque thanks to the fact that it blurs the distinction between the basic strategy used for on-the-fly generation and some specific optimizations which improve the overall performance.

Here, we present the basis a technically cleaner version of the [GPVW95] algorithm (which is admittedly less efficient overall). However, by separately applying optimizations to our construction, we can improve the complexity bounds. The construction described here is from [DSz96].

Now and next-state requirements

Though LTL formulas are intepreted on infinite runs, the definition of the satisfaction relation for LTL gives rise to what may be called a "two-state" semantics. Every LTL formula α can be thought of as specifying two sets of requirements: one set to be satisfied "now" and the rest to be satisfied in the "next state". For instance, the formula $\alpha\mathcal{U}\beta$ says that either β is true now, or α is true now and $\alpha\mathcal{U}\beta$ is true in the next state.

By repeatedly breaking down a formula, we can put it in a sort of disjunctive normal form, which specifies all possible ways of making the formula true. For instance, the formula $p \wedge (q\mathcal{U}r)$ breaks down into two sets of requirements, $\{p, q, O(q\mathcal{U}r)\}$ and $\{p, r\}$. If either of these sets is true, the original formula is true. Notice that each of these sets internally breaks up into current requirements (e.g., p, q in the first set) and next-state requirements ($O(q\mathcal{U}r)$ in the first set).

We would like to formalize this notion: a formula will generate an automaton whose states are sets like the ones above. Each set represents one way of making some requirement true. A state t is a valid next state for a state s if it is one of the ways of satisfying the next-state formulas in s.

Positive formulas

We modify the syntax of LTL so that all negations appear only at the level of atomic propositions. Formally, we begin with a set of atomic propositions $\mathcal{P}$, together with predefined constants $\top$ and $\bot$. Let $\overline{\mathcal{P}}$ denote the set $\{\neg p \mid p \in \mathcal{P}\}$. Then Φ^+, the set of LTL formulas in *positive form*, is defined inductively as follows:

- $\top$ and $\bot$ belong to Φ^+.
- Every member of $\mathcal{P}$ and $\overline{\mathcal{P}}$ belongs to Φ^+.
- If α and β are formulas in Φ^+, then so are $\alpha \vee \beta$, $\alpha \wedge \beta$, $O\alpha$, $\alpha\mathcal{U}\beta$ and $\alpha\mathcal{V}\beta$.

The semantics is $\vee, \wedge, O$ and $\mathcal{U}$ is as before. The formula $\alpha\mathcal{V}\beta$ is an abbreviation for $\neg(\neg\alpha\mathcal{U}\neg\beta)$. It is easy to check that

- $M, i \models \alpha \mathcal{V} \beta$ iff either for all $k \geq i$, $M, k \models \beta$ or there is a $j \geq i$ such that $M, j \models \alpha$ and for all ℓ, $i \leq \ell \leq j$, $M, \ell \models \beta$.

Notice that any LTL formula can be converted into a positive formula by pushing all negations inwards to the level of atomic propositions. The resulting formula may be longer, but is still linear in the size of the original formula. Henceforth, we assume that all formulas we encounter are from Φ^+.

For a set of formulas X, let $\mathsf{next}(X) = \{O\alpha \mid O\alpha \in X\}$ and $\mathsf{snext}(X) = \{\alpha \mid O\alpha \in X\}$. In other words, $\mathsf{snext}(X)$ consists of all O-formulas in X stripped of the O modality. We use $\bigwedge X$ to denote the conjunction of formulas in X and $\bigvee X$ to denote the disjunction of formulas in X. We adopt the convention that $\bigwedge \emptyset = \top$ and $\bigvee \emptyset = \bot$.

Disjunctive normal form To formalize the notion of breaking down a formula into all possible ways of satisfying it, we introduce a version of disjunctive normal form.

Let

$$\mathsf{dnf} : \Phi^+ \rightarrow \text{Sets of subsets of } \mathcal{P} \cup \overline{\mathcal{P}} \cup \mathsf{next}(\Phi^+)$$

be given by:

$$\begin{array}{lcl}
\mathsf{dnf}(\top) & = & \{\emptyset\} \\
\mathsf{dnf}(\bot) & = & \emptyset \\
\mathsf{dnf}(x) & = & \{\{x\}\}, \text{ for } x = p, \neg p, O\alpha \\
\mathsf{dnf}(\alpha \vee \beta) & = & \mathsf{dnf}(\alpha) \cup \mathsf{dnf}(\beta) \\
\mathsf{dnf}(\alpha \wedge \beta) & = & \{C \cup D \mid C \in \mathsf{dnf}(\alpha), D \in \mathsf{dnf}(\beta), C \cup D \text{ is propositionally consistent}\} \\
\mathsf{dnf}(\alpha \mathcal{U} \beta) & = & \mathsf{dnf}(\alpha \wedge O(\alpha \mathcal{U} \beta)) \cup \mathsf{dnf}(\beta) \\
\mathsf{dnf}(\alpha \mathcal{V} \beta) & = & \mathsf{dnf}(\alpha \wedge \beta) \cup \mathsf{dnf}(\beta \wedge O(\alpha \mathcal{V} \beta))
\end{array}$$

Thus the function dnf breaks up a formula into a set of sets of formulas. Each lower level set represents a clause in an extended version of disjunctive normal form, where propositions, their negations and next formulas are treated as literals. The modalities $\mathcal{U}$ and $\mathcal{V}$ are interpreted as disjunctions over their "two-state semantics".

We can prove the following by induction on the structure of formulas.

Lemma 4.1 *For all formulas $\alpha \in \Phi^+$,*

$$\alpha \Leftrightarrow \bigvee_{X \in \mathsf{dnf}(\alpha)} (\bigwedge X)$$

Let us fix a formula $\alpha \in \Phi^+$. We extend the notion of $\mathrm{CL}(\alpha)$ to cover the new modality $\mathcal{V}$ in the obvious way: if $\beta \mathcal{V} \gamma \in \mathrm{CL}(\alpha)$, then $\beta, \gamma \in \mathrm{CL}(\alpha)$.

In our on-the-fly construction, we will not work with fully specified atoms. Instead, we normally work with "unsaturated" subsets $X \subseteq \mathrm{CL}(\alpha)$. For such a subset X, we define $\mathsf{consq}(X)$, the *logical consequences of* X as follows.

Let $X \subseteq \mathrm{CL}(\alpha)$. Then $\mathsf{consq}(X)$ is the smallest subset of $\mathrm{CL}(\alpha)$ such that:

- $X \subseteq \mathsf{consq}(X)$
- If $\top \in \mathrm{CL}(\alpha)$ then $\top \in \mathsf{consq}(X)$.
- For all $\beta \vee \gamma \in \mathrm{CL}(\alpha)$, if $\beta \in \mathsf{consq}(X)$ or $\gamma \in \mathsf{consq}(X)$ then $\beta \vee \gamma \in \mathsf{consq}(X)$.
- For all $\beta \wedge \gamma \in \mathrm{CL}(\alpha)$, if $\beta, \gamma \in \mathsf{consq}(X)$ then $\beta \wedge \gamma \in \mathsf{consq}(X)$.
- For all $\beta \mathcal{U} \gamma \in \mathrm{CL}(\alpha)$, if $\beta, O(\beta \mathcal{U} \gamma) \in \mathsf{consq}(X)$ or $\gamma \in \mathsf{consq}(X)$ then $\beta \mathcal{U} \gamma \in \mathsf{consq}(X)$.
- For all $\beta \mathcal{V} \gamma \in \mathrm{CL}(\alpha)$, if $\beta, \gamma \in \mathsf{consq}(X)$ or $\beta, O(\beta \mathcal{V} \gamma) \in \mathsf{consq}(X)$ then $\beta \mathcal{V} \gamma \in \mathsf{consq}(X)$.

The following properties of consq are not difficult to prove.

Lemma 4.2 *Let $\beta \in CL(\alpha)$ and $X, Y \subseteq CL(\alpha)$. Then*

(i) If $X \subseteq Y$ then $\mathsf{consq}(X) \subseteq \mathsf{consq}(Y)$.

(ii) If $X \in \mathsf{dnf}(\beta)$ then $\beta \in \mathsf{consq}(X)$.

(iii) If $\beta \in X$ and $Y \in \mathsf{dnf}(\bigwedge X)$, then $\beta \in \mathsf{consq}(Y)$.

(iv) Let $M = P_O P_1 \ldots$ be a model. If $M, 0 \models \bigwedge X$ then $M, 0 \models \bigwedge \mathsf{consq}(X)$.

We can now define a generalized Büchi automaton corresponding to a formula α. The automaton $\mathcal{A}_\alpha = (S, \rightarrow, S_{in})$ over $2^{\mathrm{Voc}(\alpha)}$ is given by:

- $S =$ Subsets of $\mathcal{P} \cup \overline{\mathcal{P}} \cup \mathsf{next}(\mathrm{CL}(\alpha))$.
- $S_{in} = \mathsf{dnf}(\alpha)$.
- $X \xrightarrow{P} Y$ iff the following are true:
  - $X \cap \mathcal{P} \subseteq P$.
  - $\{p \mid \neg p \in X\} \cap P = \emptyset$.
  - $Y \in \mathsf{dnf}(\bigwedge \mathsf{snext}(X))$.

Notice that we could now have many different arrows $X \xrightarrow{P} Y$ between a pair of states X and Y, since X and Y are no longer atoms. In general, a P labelled-arrow exists provided P does not contradict the propositional assertions in X.

As before let $\{\beta_1 \mathcal{U} \gamma_1, \beta_2 \mathcal{U} \gamma_2, \ldots, \beta_k \mathcal{U} \gamma_k\}$ be the set of until formulas which appear in $\mathrm{CL}(\alpha)$. We add a generalized Büchi acceptance condition $(G_1, G_2, \ldots, G_k)$ where for each i, $G_i = \{X \in S \mid \beta_i \mathcal{U} \gamma_i \notin \mathsf{consq}(X) \text{ or } \gamma_i \in \mathsf{consq}(X)\}$.

We want to show that $M = P_0 P_1 \ldots$ is accepted by $(\mathcal{A}_\alpha, G_1, G_2, \ldots, G_k)$ iff $M, 0 \models \alpha$. We break the proof up into two parts.

Soundness

Let $M = P_0P_1\ldots$ be accepted by $(\mathcal{A}_\alpha, G_1, G_2, \ldots, G_k)$. We need to show that $M, 0 \models \alpha$. Let $X_0X_1\ldots$ be an accepting run for M. We show the following:

Lemma 4.3 *$\forall \beta \in CL(\alpha)$, $\forall i \in \mathbb{N}_0$, if $\beta \in \mathsf{consq}(X_i)$ then $M, i \models \beta$.*

Proof By induction on the structure of β.

$(\beta = p \in \mathcal{P})$

If $p \in \mathsf{consq}(X_i)$ then $p \in X_i$ which implies that $p \in P_i$ which implies that $M, i \models p$.

$(\beta = \neg p, p \in \mathcal{P})$

If $\neg p \in \mathsf{consq}(X_i)$ then $\neg p \in X_i$ which implies that $p \notin P_i$ which implies that $M, i \models \neg p$.

$(\beta = \gamma \vee \delta)$

Since X_i consists of formulas from $\mathcal{P} \cup \overline{\mathcal{P}} \cup \mathsf{next}(\Phi^+)$, it cannot be the case that $\gamma \vee \delta \in X_i$. Thus, we must have derived the fact that $\gamma \vee \delta \in \mathsf{consq}(X_i)$ using our inductive definition of $\mathsf{consq}(X_i)$. From this, it follows that either $\gamma \in \mathsf{consq}(X_i)$ or $\delta \in \mathsf{consq}(X_i)$. By the induction hypothesis, either $M, i \models \gamma$ or $M, i \models \delta$, so $M, i \models \gamma \vee \delta$.

$(\beta = \gamma \wedge \delta)$

Once again, it cannot be the case that $\gamma \wedge \delta \in X_i$. From the fact that $\gamma \wedge \delta \in \mathsf{consq}(X_i)$, it follows that $\gamma \in \mathsf{consq}(X_i)$ and $\delta \in \mathsf{consq}(X_i)$. By the induction hypothesis, $M, i \models \gamma$ and $M, i \models \delta$, so $M, i \models \gamma \wedge \delta$.

$(\beta = O\gamma)$

If $O\gamma \in \mathsf{consq}(X_i)$ then $O\gamma \in X_i$. So $\gamma \in \mathsf{snext}(X_i)$. Since $X_{i+1} \in \mathsf{dnf}(\mathsf{snext}(X_i))$, it follows from Lemma 4.2 (ii) that $\gamma \in \mathsf{consq}(X_{i+1})$. By the induction hypothesis, $M, i{+}1 \models \gamma$, so $M, i \models O\gamma$.

$(\beta = \gamma\,\mathcal{U}\,\delta)$

In general if $X \xrightarrow{P} Y$ and $\gamma\,\mathcal{U}\,\delta \in \mathsf{consq}(X)$, then either $\delta \in \mathsf{consq}(X)$ or $\gamma \in \mathsf{consq}(X)$ and $\gamma\,\mathcal{U}\,\delta \in \mathsf{consq}(Y)$. So, if $\gamma\,\mathcal{U}\,\delta \in \mathsf{consq}(X_i)$, it follows either that for all $k \geq i$, $\gamma\,\mathcal{U}\,\delta \in \mathsf{consq}(X_k)$ and $\delta \notin \mathsf{consq}(X_k)$ or that there exists $k \geq i$ such that $\delta \in \mathsf{consq}(X_k)$ and for all j such that $i \leq j < k$, $\gamma \in \mathsf{consq}(X_j)$.

The first case is ruled out by the fact that $X_0X_1\ldots$ is an accepting run. By the induction hypothesis, the second case yields $M, k \models \delta$ and $M, j \models \gamma$ whereby $M, i \models \gamma\,\mathcal{U}\,\delta$.

$(\beta = \gamma\,\mathcal{V}\,\delta)$

In general if $X \xrightarrow{P} Y$ and $\gamma\,\mathcal{V}\,\delta \in \mathsf{consq}(X)$, then either $\gamma \wedge \delta \in \mathsf{consq}(X)$ or $\delta \in \mathsf{consq}(X)$ and $\gamma\,\mathcal{V}\,\delta \in \mathsf{consq}(Y)$. It follows that either for all $k \geq i$, $\gamma\,\mathcal{V}\,\delta \in \mathsf{consq}(X_k)$

and $\delta \in \mathsf{consq}(X_k)$ or there exists $k \geq i$ such that $\gamma \wedge \delta \in \mathsf{consq}(X_k)$ and for all j such that $i \leq j < k$, $\gamma \in \mathsf{consq}(X_j)$.

If the first case holds, by the induction hypothesis, we have $M, k \models \delta$ for all $k \geq i$, so $M, i \models \gamma \mathcal{V} \delta$ by the semantics of the modality $\mathcal{V}$. If the second case holds, from the definition of consq, it follows that $\gamma, \delta \in \mathsf{consq}(X_k)$. Hence, by the induction hypothesis, $M, k \models \gamma$ and $M, k \models \delta$, so $M, k \models \gamma \wedge \delta$. Also, by the induction hypothesis, for j such that $i \leq j < k$, $M, j \models \gamma$. From the semantics of the modality $\mathcal{V}$, it then follows that $M, i \models \gamma \mathcal{V} \delta$.

□

Since $X_0 \in \mathsf{dnf}(\alpha)$, we have $\alpha \in \mathsf{consq}(X_0)$ (Lemma 4.2 (ii)). It then follows that $M, 0 \models \alpha$.

Completeness

Let $M = P_0 P_1 \ldots$ be a model such that $M, 0 \models \alpha$. We have to show that M is accepted by $(\mathcal{A}_\alpha, G_1, G_2, \ldots, G_k)$. We begin with an auxiliary lemma.

Lemma 4.4 *Let X be a state of $\mathcal{A}_\alpha$ such that for some i, $M, i \models \bigwedge X$. Then there exists a state Y of $\mathcal{A}_\alpha$ such that:*

- $X \xrightarrow{P_i} Y$.
- $M, i{+}1 \models Y$.
- *For all $\gamma \mathcal{U} \delta \in CL(\alpha)$, if $\gamma \mathcal{U} \delta \in \mathsf{consq}(X)$ and $\delta \notin \mathsf{consq}(X)$ and $M, i{+}1 \models \delta$, then $\delta \in \mathsf{consq}(Y)$.*

Proof Let $\mathsf{next}_1 = \{\gamma \mathcal{U} \delta \in \mathsf{consq}(X) \mid \delta \notin \mathsf{consq}(X) \text{ and } M, i{+}1 \models \delta\}$. Note that $\mathsf{next}_1 \subseteq \mathsf{snext}(X)$ for, in general, if $\gamma \mathcal{U} \delta \in \mathsf{consq}(X')$ and $\delta \notin \mathsf{consq}(X')$ then it must be the case that $O(\gamma \mathcal{U} \delta) \in \mathsf{consq}(X')$. Let $\mathsf{next}_2 = \mathsf{snext}(X) - \mathsf{next}_1$.

Since $M, i \models \bigwedge X$, it must be the case that $M, i{+}1 \models \bigwedge \mathsf{snext}(X)$. Hence $M, i{+}1 \models \bigwedge \mathsf{next}_1$ and $M, i{+}1 \models \bigwedge \mathsf{next}_2$.

Let $\Delta = \{\delta \mid \gamma \mathcal{U} \delta \in \mathsf{next}_1\}$. From the definition of next_1, $M, i{+}1 \models \delta$ for each $\delta \in \Delta$. For each $\delta \in \Delta$, since we know that $\delta \Leftrightarrow \bigvee_{Z \in \mathsf{dnf}(\delta)} \bigwedge Z$, there must be some $Z_\delta \in \mathsf{dnf}(\delta)$ such that $M, i{+}1 \models Z_\delta$. Let $Z_1 = \bigcup_{\delta \in \Delta} Z_\delta$.

Since $M, i{+}1 \models \bigwedge Z_1$, Z_1 must be consistent. Further, from the way dnf is defined on $\mathcal{U}$, it follows that $Z_1 \in \mathsf{dnf}(\bigwedge \mathsf{next}_1)$.

Similarly, since $M, i{+}1 \models \bigwedge \mathsf{next}_2$, there must be a set $Z_2 \in \mathsf{dnf}(\bigwedge \mathsf{next}_2)$ such that $M, i{+}1 \models \bigwedge Z_2$. We choose Y to be $Z_1 \cup Z_2$.

To show that $X \xrightarrow{P_i} Y$, we note that $\mathsf{dnf}(\bigwedge \mathsf{snext}(X)) = \mathsf{dnf}((\bigwedge \mathsf{next}_1) \wedge (\bigwedge \mathsf{next}_2))$. We know that $Z_1 \in \mathsf{dnf}(\bigwedge \mathsf{next}_1)$ and $Z_2 \in \mathsf{dnf}(\bigwedge \mathsf{next}_2)$. Since $M, i{+}1 \models \bigwedge Z_1$ and $M, i{+}1 \models \bigwedge Z_2$, $Z_1 \cup Z_2$ is consistent. Thus, $Y = Z_1 \cup Z_2 \in \mathsf{dnf}(\mathsf{snext}(X))$. To check that

P_i is a valid label, we just use the fact that $M, i \models X$. So, if $p \in X$, then $p \in P_i$ and if $\neg p \in X$ then $p \notin P_i$.

Clearly, $M, i{+}1 \models \bigwedge(Z_1 \cup Z_2)$. Further, for all $\delta \in \Delta$, $\delta \in \mathsf{consq}(Z_1 \cup Z_2)$, since $\delta \in \mathsf{consq}(Z_\delta) \subseteq \mathsf{consq}(Z_1) \subseteq \mathsf{consq}(Z_1 \cup Z_2)$.

□

We use this lemma to produce an accepting run as follows. First, note that $\alpha \Leftrightarrow \bigvee_{X \in S_{in}} \bigwedge X$ holds. Since $M, 0 \models \alpha$, we have $M, 0 \models \bigvee_{X \in S_{in}} \bigwedge X$, so S_{in} is non-empty—there must exist $X_0 \in S_{in}$ such that $M, 0 \models \bigwedge X_0$.

From the previous lemma, given $M, 0 \models \bigwedge X_0$, we can find X_1 such that $X_0 \xrightarrow{P_0} X_1$ and $M, 1 \models \bigwedge X_1$. Repeatedly applying the lemma, we extract a run $X_0 X_1 \ldots$ of $\mathcal{A}_\alpha$.

Suppose that this run is not an accepting run. Then there is some formula $\gamma \mathcal{U} \delta \in \mathrm{CL}(\alpha)$ and an index $k \in \mathbb{N}_0$ such that for all $i \geq k$, $\gamma \mathcal{U} \delta \in \mathsf{consq}(X_i)$ and $\delta \notin \mathsf{consq}(X_i)$. But $\gamma \mathcal{U} \delta \in \mathsf{consq}(X_{k+1})$, so $M, k{+}1 \models \gamma \mathcal{U} \delta$ by Lemma 4.2 (iv). So, there exists $j \geq k{+}1$ such that $M, j \models \delta$ and for all ℓ, $k{+}1 \leq \ell < j$, $M, \ell \not\models \delta$. Since $\gamma \mathcal{U} \delta \in \mathsf{consq}(X_{j-1})$ and $\delta \notin \mathsf{consq}(X_{j-1})$ and $M, j \models \delta$, by the preceding lemma we must have $\delta \in \mathsf{consq}(X_j)$ which is a contradiction.

On-the-fly traversal

We have shown that the automaton defined in terms of the function dnf contains all possible models of the formula α. This automaton can be constructed "on-the-fly". We begin by constructing the states corresponding to $\mathsf{dnf}(\alpha)$. After that, whenever we need to find the successors of a node X, we apply the function dnf to $\bigwedge \mathsf{snext}(X)$. With some bookkeeping, we can avoid duplicating nodes. This is essentially what the algorithm in [GPVW95] computes, though they also combine some low-level optimization with the basic algorithm

A straightforward implementation of dnf yields an automaton which could potentially take time $2^{O(n^2)}$ for an input formula of length n, even though the total number of states is bounded by $2^{O(n)}$. In contrast, the algorithm of [GPVW95] runs in $2^{O(n)}$ time [DSz96].

On the other hand, it is possible to build in optimizations when implementing dnf which could bring down the bound. The main point is that the dnf construction clearly brings out the formal basis for the on-the-fly construction, which is somewhat obscured in the direct algorithmic reasoning of [GPVW95].

5 Extensions

The temporal logic we have considered ignores the labels on the transitions of a program—the model generated by a run is just the sequence of states it goes through. It is often useful to be able to talk about which transitions were used as well—for instance, in the setting of concurrent programs, the automaton is often specified as a synchronized product of sequential automata, where the synchronization mechanism is encoded via the actions performed. Normally, each sequential component comes equipped with a local set of actions. If an action is common to more than one component, those components must synchronize to perform the action. Actions which involve disjoint sets of processes can be performed concurrently.

It is not difficult to extend the syntax of LTL to include a modality which talks about actions—for each $a \in \Sigma$, we introduce the modality $\langle a \rangle$. A model is no longer just a sequence of propositional valuations $P_0 P_1 \ldots$, but instead a sequence of transitions of the form $P_O \xrightarrow{a_0} P_1 \xrightarrow{a_1} \cdots$, where each P_i is a subset of $\mathcal{P}$, as before, and each a_i belongs to Σ. The semantics of the new modality is the obvious one. Let $M = P_O \xrightarrow{a_0} P_1 \xrightarrow{a_1} \cdots$. Then

$$M, i \models \langle a \rangle \alpha \text{ iff } a_i = a \text{ and } M, i{+}1 \models \alpha.$$

Let us denote this extended version of LTL as LTL(Σ).

Both the global automata-theoretic construction of Section 3 as well as the on-the-fly approach of [GPVW95] can be extended smoothly to LTL(Σ) [Mad96].

Having extended LTL to talk about actions, the next challenge is to exploit the concurrency present in the program specification to further reduce the complexity of model-checking. As we mentioned in the Introduction, here we have assumed that all program runs are represented as infinite sequences. However, when the program involves concurrent actions, this approach generates many equivalent interleavings of the same stretch of concurrent behaviour. It would be extremely useful if we can identify just one representative sequence to be verified for each such class of interleavings.

There is a considerable amount of work in progress on designing so-called "partial order methods" for verifying temporal properties of concurrent programs [GW94, Val90]. Another challenging task is to identify subclasses of formulas in LTL(Σ) which have the following property: if a formula is true in one representative interleaving of a concurrent run, then it is true in all interleavings. Though some subsets of LTL(Σ) which have this property have been identified, the search for a full characterization of this class is elusive [MT96].

Acknowledgments I thank Deepak D'Souza and P Madhusudan for shedding light on the subtleties of the on-the-fly method. In particular, the dnf based presentation of the [GPVW95] construction is due to Deepak.

References

[AHU74] A.V. Aho, J.E. Hopcroft and J.D. Ullman: *The Design and Analysis of Algorithms*, Addison-Wesley, Reading (1974).

[Bü60] J.R. Büchi: On a decision method in restricted second order arithmetic, *Z. Math. Logik Grundlag. Math*, **6** (1960) 66–92

[CVWY92] C. Courcoubetis, M. Vardi, P. Wolper and M. Yannakakis: Memory efficient algorithms for the verification of temporal logic properties, *Formal Methods in System Design 1* (1992) 275–288.

[DSz96] Deepak D'Souza: On-the-fly techniques for linear time temporal logic, *Manuscript* (1996).

[GPVW95] R. Gerth, D. Peled, M. Vardi and P. Wolper: Simple on-the-fly automatic verification of linear temporal logic, *Proc. IFIP/WG6.1 Symp. on Protocol Specification, Testing and Verification*, Warsaw, Poland, June 1995.

[GW94] P. Godefroid and P. Wolper: A partial approach to model checking, *Inform. and Comput.*, **110** (1994) 305–326.

[Mad96] P. Madhusudan: An on-the-fly algorithm for linear time temporal logic, *M.Sc. Thesis*, Anna University, Madras, India (1996).

[MP91] Z. Manna and A. Pnueli: *The Temporal Logic of Reactive and Concurrent Systems (Specification)*, Springer-Verlag, Berlin (1991).

[MT96] M. Mukund and P.S. Thiagarajan: Linear time temporal logics over Mazurkiewicz traces, *Proc MFCS '96*, *LNCS* **1113** (1996) 62–92.

[Pnu77] A. Pnueli: The temporal logic of programs, *Proc. 18th IEEE FOCS* (1977) 46–57.

[SVW87] A.P. Sistla, M. Vardi and P. Wolper: Reasoning about infinite computation paths, *Proc. 24th IEEE FOCS* (1983) 185–194.

[Th90] W. Thomas: Automata on infinite objects, in J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science, Volume B*, North-Holland, Amsterdam (1990) 133–191.

[Th96] W. Thomas: Languages, automata and logic, Report 9607, Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität, Kiel, Germany, May 1996. (Preliminary version of a chapter to appear in G. Rozenberg, A. Salomaa (eds.), *Handbook of Formal Language Theory*, Springer-Verlag.)

[Val90] A. Valmari: Stubborn sets for reduced state space generation, *LNCS* **483** (1990) 491–515.

[VW86] M. Vardi and P. Wolper: An automata theoretic approach to automatic program verification, *Proc. 1st IEEE LICS*, (1986) 332–345.

[VW94] M. Vardi and P. Wolper: Reasoning about infinite computations, *Inform. Comput.*, **115**(1) (1994) 1–37.